Impact of *Scratch* on the Achievements of First-year Computer Science Students in Programming in Some Nigerian Polytechnics

by

OLADELE OLADUNJOYE CAMPBELL

submitted in accordance with the requirements for
the degree of

DOCTOR OF PHILOSOPHY

In the subject

MATHEMATICS, SCIENCE & TECHNOLOGY EDUCATION

with specialisation in

COMPUTING EDUCATION

at the

UNIVERSITY OF SOUTH AFRICA

SUPERVISOR: PROF H .I. ATAGANA

MARCH 2022

Declaration

Name:	OLADELE OLADUNJOYE CAMPBELL
Student number:	51898772
Degree:	PhD

I declare that this study "**Impact of *Scratch* on the Achievements of First-year Computer Science Students in Programming in Some Nigerian Polytechnics**" is my own work and that all the sources that I have used or quoted have been indicated and acknowledged by means of complete references.

I further declare that I have not previously submitted this work, or part of it, for examination at UNISA for another qualification or at any other higher education institution.

[illegible]

SIGNATURE

06-11-21

DATE

Dedication

I dedicate this academic work to God, my heavenly Father who gave me the will to accomplish this feat. To my Saviour Jesus Christ, who influenced the way I went about this research work. To the Holy Spirit, for divine intellectual inputs that moved this work when human wits failed.

I dedicate this thesis also, to Nigerian polytechnic students in Akwanga, Bida, Bwari, Lafia, Nasarawa, and Zungeru who actively welcomed our teaching in both modes and provided the data for this research work.

Acknowledgements

An academic project of this kind, like any other, has benefited from several sources. Apart from those acknowledged by citations and references, numerous sources, some which cannot be mentioned due to space, deserve my gratitude. My appreciations go to:

- My supervisor, Prof H .I. Atagana, thank you for the intellectual and moral supports. I troubled you via calls and emails making requests even at odd times. You have also shared delightful moments of watching me make progress under your supervision. When I consider the many correspondences between us, I think that makes an interesting trove of data for another study. Thank you, Sir.
- Prof U.I. Ogbonnaya, whom Prof H.I Atagana called the first time we met face-to-face in his office, to listen to my research proposal. He encouraged me in this journey when I was naïve about many aspects of computing education research.
- My Rectors at Niger State Polytechnic Zungeru, present and former: Dr Umar Egbako, Dr Garba Kamaye Mohammed, Dr Francis Gana, and late Engr Sani Ango who all have provided opportunities to serve, learn and provide solutions to students' problems.
- Mal Yahaya Zakari, my Director of Academic Planning at NSP, who facilitated and ensured I got the financial support provided by TETFUND for my PhD.
- UNISA for Master and Doctoral students' bursary that provided financial support (after several failed attempts) during the final year of my PhD when TETFUND financial tap had long run dry few years back. Prof Patrick Ngulube and Prof Jimi Adesina provided helpful insights when I was in dilemma about registering, getting the bursary and receiving the link for thesis submission. To the great staff at UNISA who work tirelessly to ensure students get critical support, I say thank you.
- Concerned colleagues in Niger State Polytechnic Zungeru who foresaw me attaining a doctorate, calling me "Doc" when I have not attained it. Thus, motivating me to finish the programme and make them proud. I say, thank you for your 'prophetic' word.

- Dr Ibrahim Kontagora (my HOD, when this journey began); Isah Adamu Dagah (my HOD, when the journey was rough and provided motivation); Yemi Oyetola (our then visiting lecturer, who always had practical ideas to help); Madam Aishetu Mohammed and Madam Hakila Yahuza Wushishi. These were my research assistants, who acted as observers and interviewers during and after exposing the students to *Scratch* at Niger State Polytechnic. Yemi Oyetola marked scripts of students that took part in this study.
- Some of my brethren: Mr. and Mrs. Adewara and Mr. and Mrs. Madu who have contributed moral, spiritual, and financial support; Mr. Gabriel Rahim was morally and spiritually supportive; Dr Enoch Gana who was concerned that I may lose my studentship raised an outcry and rallied financial support from a cooperative; Mr. Paul Gana the chairman of the cooperative; Mr. Elijah Santali, whose fatherly help and understanding was tremendous; Mr. Obadiah Jiya (my brother, friend, prayer partner and burden-sharer of almost three decades) together with his wife contributed moral and spiritual support.
- Profs. Chukunoye Enunuwe Ochonogor, Keshnee Padayachee, Nosisi Feza, and others at ISTE who critiqued and contributed to the birth of this PhD. Prof Ochonogor through PG seminars and personal tutoring contributed immensely to my knowledge of educational research. He and his wife gave me a hand of fellowship and accommodated a stranger in their home one night in 2014 when I visited South Africa.
- Prof. David Mogari shared research knowledge during PG seminars.
- Professors Mark Guzdial (University of Michigan, USA), Sally Fincher (University of Kent, UK) and Orit Hazzan (Technion, Israel) are among my mentors in computing education research, who provided frank peer reviews and supported when controversy arose about the proposal at the early stage.
- Ms Mathapelo Lamola, helped me while she was at ISTE, sometimes with registration problem. At other time, she ensured I connected from Nigeria to Pretoria to avoid missing out on important research events like the PG seminars at the early stage of this PhD.

- Dr. Caleb Akintade, a brother and a friend who shared in the sorrows and joys of this journey with me while we were in SA and back in Nigeria.
- Dr. Caleb Falode, a brother who endured my interruptions of his time at FUT Minna and sometimes at home. He contributed intellectually to my research.
- Dr. Caleb Alade supported the research at the early stage, helping to get permission for the study at FPB, and mobilized his first-year students to participate and provide research data for understanding this CS1 problem. Through Dr. Alade, I also got the support of other lecturers in his department for the study.
- Dr. Joseph Yacim (Federal Poly Nasarawa), his wife and lovely kids provided accommodation and food whenever I was in Nasarawa for data collection.
- Dr. and Dr. Mrs. Abolarinwa and their kids, a family of friendly souls in whose home I stayed often while transiting between Zungeru and Nasarawa. The Abolarinwas have been true friends and brethren, providing moral, spiritual and financial support towards the attainment of my doctorate.
- Prof UG Akpan provided space in his office at the Federal University of Technology, Minna Nigeria for me to present my research seminar to the audience in Pretoria at a time.
- My spiritual leaders, Pastor Josiah Oladesu and his wife, Pastor Ahmed Baro, Pastor Abraham Pyata, Pastor Francis Owojori and Pastor Dauda Madaki who prayed for a breakthrough for me during this programme. God has answered your prayers.
- My in-laws: Mr Tunde Bello and his wife, Dr. Mrs. Aduke Bello for their relentless moral, spiritual and financial support. Deacon Ogunlola and his wife, Deaconess Ogunlola, for their goodwill, moral and spiritual support to our family; Bro. Bayo Alabi and his wife, Aunty Elizabeth, for every kindness shown to me and my family over the years.
- Dr Mathew contributed and extended my ability at data analysis.
- Dr Adam, who in a painful way “stirred up” my moral energy towards improving my ability at doing the literature review.

- Pastor Hosea Eshofonie, my secondary school fellowship president, proved to be a friend indeed - providing financial and moral support when he got to know my continued UNISA studentship for PhD was in danger last year.
- My mother-in-law and a mother in the Lord – Mrs. Felicia Ogunlola, is a pillar of support in prayers and faith.
- My late parents, Henry Abiodun & Dorcas Adepate Campbell, provided support for me while growing up and during this journey. Dad as a taxi driver ensured his boy went to Command Secondary School, Ibadan, where some of the highly placed in Nigeria had their wards. There, my academic ability and desired for knowledge got further boost. Dad kept encouraging my pursuit of attaining the doctorate before he passed on last year. My elder brother, Oyeyemi Campbell and his wife have shown me and my family kindness during this journey. My junior brother Gbenga Campbell, who have expressed concern and desire for me to finish this PhD. Olaitan Campbell and his family whose understanding and kindness make their home welcoming to my family and I.
- To my wife, Vic, and our children– Joshua, Paul, Comfort and Miracle for your patience. Joshua empowered me with access to free academic database resources that were game changers for me. Vic, Paul, Comfort and Miracle were also my teaching assistants, helping dad to do other academic work to lessen burdens.
- To individuals who voiced concerns and others who didn’t, yet had genuine concern and prayed for my success in this programme, I say thank you all.

Publications

1. Campbell, O. O., & Atagana, H. I. (2022). Impact of a Scratch programming intervention on student engagement in a Nigerian polytechnic first-year class: verdict from the observers. *Heliyon*, 8(3), e09191. https://doi.org/10.1016/j.heliyon.2022.e09191.

2. Oladele Campbell. Improving novice programming education in Nigerian polytechnics: A work in progress. A research paper presented at the First Annual National Conference of the School of Natural and Applied Sciences, Niger State Polytechnic, Zungeru. 22nd February 2017.

List of abbreviations

ACM	Association for Computing Machinery
AI	Artificial Intelligence
ANCOVA	Analysis of Covariance
CA	Continuous Assessment
CEM	Coarsened Exact Matching
CS	Computer Science
CS1	Computer Science One
CSPROQ	CS1 Student Profile Questionnaire
CT	Computational Thinking
FPB	Federal Polytechnic, Bida
FPN	Federal Polytechnic, Nasarawa.
IBM	International Business Machine
IPAT	Introductory Programming Achievement Test
IT	Information Technology
NATO	North Atlantic Treaty Organization
NBTE	National Board for Technical Education
ND	National Diploma
NSPL	Nasarawa State Polytechnic, Lafia
NSPZ	Niger State Polytechnic, Zungeru.
NSSE	National Survey of Student Engagement
ODL	Open Distance Learning
OECD	Organisation for Economic Co-operation and Development
SA	South Africa
SCOP	Scratch Class Observation Protocol
SPIEI	*Scratch* Post Intervention Evaluation Interview
TETFUND	Tertiary Education Trust Fund
UNISA	University of South Africa
UTME	Unified Tertiary Matriculation Exam

Abstract

To support the advancement of modern civilisation, our institutions of higher learning must produce the right pool of professionals, who can develop innovative software. However, the teaching and learning of the first programming language (CS1) remains a great challenge for most educators and novice computer students. Indicators such as failure and attrition rates, and CS1 student engagement, continue to show that conventional pedagogy does not adequately meet the needs of some beginning CS students. For its ease in introducing novices to programming, *Scratch*—a visual programming environment following the constructionism philosophy of Seymour Papert—is now employed even in some higher education CS1 classes with mixed evidence of its impact. *Scratch* captures the constructionist agenda by its slogan: “Imagine, Program, Share.”

Therefore, this study explored the impart of using a constructionist *Scratch* programming pedagogy on higher education CS1 students’ achievements. This study also sought to compare the impacts of the two CS1 modes: the conventional class - involving textual programming language, lectures and labs, and the constructionist *Scratch* inquiry-based programming class. It further aims to discover if gender, academic level, age, prior programming, and visual artistic abilities moderate the effects of programming pedagogy on students’ achievements.

To realize the study’s aims, the study employed a quasi-experimental pretest-posttest nonequivalent groups design, involving four intact CS1 classes of polytechnic students (N = 418) in north-central Nigeria. The investigation was conducted in phases: a pilot (n = 236) and main (n=182) studies lasting two academic sessions, with each study comprising one experimental and one control group. In each session, learning in both modes lasted for six weeks. In both studies, purposive sampling was employed to select institutions, and selected institutions were randomly assigned to treatment groups. Instruments employed included CS1 Student Profile Questionnaire (CSPROQ) and Introductory Programming Achievement Test (IPAT). To strengthen the research design, I employed *Coarsened Exact Matching* (CEM) algorithm—after conducting a priori power analysis—to generate matched random samples of cases from both studies. Thus, research data employed in the analysis include: from the pilot, 41 cases in each treatment group; from the main study, 42 cases in each treatment group. Descriptive and inferential statistics were employed to find answers to research questions and test the research hypothesis. Data from both studies satisfied the requirements for statistical tests employed, i.e., t-test and ANCOVA. The alpha level used in testing hypotheses was p = 0.05. The dependent variable is the IPAT post-test score, while the independent variables are treatment, gender, age, academic achievement level, prior programming, and prior visual art. The covariate was the IPAT pretest score. Statistical analyses were conducted using SPSS version 23.

The t-test results from both pilot and main studies indicated that, both programming pedagogies had significant effects on student IPAT scores, although the effect of the constructionist *Scratch* intervention was higher.

Results from the one-way ANCOVA analysis of both pilot and main study data—while controlling for students' IPAT pretest scores—yielded the same outcome: There was significant main effect of treatment on students' IPAT posttest scores, although the impact was moderate. Controlling for pre-test scores, analysis of the main studies data yielded no significant main effects of: gender, age, academic level, prior programming and prior visual artistic ability. The result from the main study also reveals no interaction effect of treatment, gender, academic level, age, prior programming, and prior artistic ability.

While the quality of CS1 students' performance in each session varies as their IPAT achievements show, yet the results of this research revealed a consistent pattern: Students in the constructionist *Scratch* class outperformed those in the conventional class, although the impart was moderate.

This finding implies college students without prior programming experience can perform better in a class following a constructionist *Scratch* programming pedagogy. The study recommends the use of Scratch, following a constructionist pedagogy with first-year students in colleges, especially those without prior background in programming.

Keyword: *Scratch*; constructionism; blocks-based programming environment; introductory programming; *Coarsened Exact Matching*; quasi-experiment.

Table of Contents

List of Figures

List of Tables

Chapter 1

1 The CS1 Problem

"The most disastrous thing that you can ever learn is your first programming language" – Alan Kay

"One of the central topics in computing education research (CEdR) is the exploration of how a person learns their first programming language.." (Robins, 2019, p. 327)

This chapter presents the background to the study, the statement of the problem, the research questions as well as the hypotheses tested in the study. This is followed by the contributions of the study and the outline of the rest of the thesis. The chapter ends with definitions of terms to give operational meanings to some words used in the thesis.

1.1 Background of the Study

The ability to develop programs for Information and Communication Technology (ICT) devices and platforms is a highly priced skill (Boljat et al., 2019; Pérez-Marín et al., 2020). Such knowledge enables professionals to provide software solutions in a world increasingly dependent on automation and innovation. That dependence makes software "drivers" on various platforms which modern life revolves, such as mobile phones, cloud computing, the web, personal computers, social networks, transportation systems, banking systems, healthcare systems, electricity grids, military installations, etc. Therefore, availability of programming knowledge has become critical for continued development of modern societies. The increasing realization of the economic, social, political, and technological impacts of software and ICT continues to drive the demand for software developers. This rising employment prospect is taking place amid a rising global unemployment (International Labour Office & International Labour Organisation, 2017). For instance, by 2024 this category of IT professionals will be among the fastest growing jobs in the USA (Bureau of Labor Statistics, 2017) as citizens continue to demand for services requiring mobile computing, cloud computing, big data analytics as devices are being added. For these reasons, the USA, governments in developed, emerging and developing countries, and private organizations are campaigning rigorously to motivate more students to study computing (Rubio et al., 2015)

However, the journey to gaining such programming ability for the numerous students (and, of course, their teachers) is burden with many difficulties and disappointments from first course, popularly called Computer Science one (CS1). The experience of many teachers and the impression novice computing students have, is that, learning programming is hard (Robins, 2019; Sharmin, 2021). Global average failure rate of students in CS1 stands at about 30% (Bennedsen & Caspersen, 2019; Falkner & Sheard, 2019). This suggests 3 students in a class of ten fails their CS1. That appears not to be "alarmingly" high, however, because many computer science educators in various countries are witnessing much higher failure rates, than the estimate provided by Bennedsen and Caspersen (2019) – a fact Robins (2019) referred to – the CS1 problem is not going away from computer science education research discourse. For instance, Chetty and Barlow-Jones (2014) reported that

60% of students in a South African university failed their CS1 after six months of teaching. Liénardy et al.(2021) reported a higher failure rate of 70% for CS1 students in Belgium University. While many students struggle through their program of study and lack the expected level of programming ability at the end, others drop out from the program, although they started with great enthusiasm. Bennedsen and Caspersen (2019) in their article aimed at correcting what they perceived as wrong notion that CS1 failure rate is high, admitted, "It appears that introducing students to computing is still one of computing education's grand challenges and that we as a community have a huge challenge in developing more inclusive and effective learning environments and instructional methods for CS1." (pg. 35).

Is this lacklustre picture of students' performance in CS1 different in Nigeria? To answer this question, I gathered CS1 assessments from three polytechnics located in the North central Nigeria. Data gathered are presented in Tables 1, 2 and 3. I observed in the available results from the three institutions that students' continuous assessment scores were high while most of these students fail the final examination. Amidst widespread examination malpractices in Nigeria, institutions provide better watch on examinations than continuous assessments (CA), which usually consist of marks from tests, projects, laboratory reports and assignments where students' works are less stringently supervised. In Nigerian polytechnics, the CA accounts for 40% while the final examination accounts for 60% of the final marks. The recommended minimum pass mark is 40% of the final score. However, to take care of margin of errors introduced into final score through irregularities on the part of the students and the lecturers' bonus marks in case of mass failures, which is usually the case with CS1, I used 50% as the pass mark.

Table 1-1 shows the available data from Federal Polytechnic Bida (FPB). The results show failure rates for the four sessions reported were 58.8%, 66.7%, 58.3%, and 28.3%, giving an average failure rate of 53.5%.

Table 1-1 Federal Polytechnic Bida Students Performance in CS1

Session	**No. of Students Examined**	**Scored 80 and above No. (%)**	**Scored 70 and above No (%)**	**Scored 50 and above No (%)**	**Scored bet. 40 and 49 No. (%)**	**Scored less than 40 No. (%)**	**Clear Pass No. (%)**	**Poor (?) or Failed No. (%)**
2003/2004	34	3 (8.8)	6 (17.7)	14 (41.2)	17 (50.0)	3 (8.8)	14 (41.2)	20 (58.8)
2006/2007	75	0 (0.0)	4 (5.3)	25 (33.3)	20 (26.7)	30 (40.0)	25 (33.3)	50 (66.7)
2007/2008	72	1 (1.4)	3 (4.2)	30 (41. 7)	28 (38.9)	14 (19.4)	30 (41.7)	42 (58.3)
2008/2009	65	0 (0.0)	5 (7.7)	29 (44.6)	24 (36.9)	12 (18.5)	29 (44.6)	36 (55.4)
2011/2012	60	3 (5.0)	14 (23.3)	43 (71.7)	10 (16.7)	7 (11.7)	43 (71.7)	17 (28.3)

SOURCE: Dept. of Computer Science, Federal Polytechnic, Bida. Nigeria, 2015

Table 1-2 presents the performance of CS1 students from Federal Polytechnic Nasarawa (FPN). Though FPN is in Nasarawa State, it is situated in the same region in central Nigeria as FPB (see Figure 1-1). The result shows that for the six sessions reported students' failure rates were 69.8%, 82.2%, 44.2%, 46.3%, 48.3% and 27.7%. The average failure rate for the six session was 53.1% just about same performance with students of FPB where average failure rate was 53.5%.

Table 1-2 Federal Polytechnic Nasarawa Students' Performance in CS1

Session	**No. of Students Examined**	**Scored 80 and above No. (%)**	**Scored 70 and above No (%)**	**Scored 50 and above No (%)**	**Scored bet. 40 and 49 No. (%)**	**Scored less than 40 No. (%)**	**Clear Pass No. (%)**	**Poor (?) or Failed No. (%)**
2006/2007	96	1 (1.0)	4 (4.2)	29 (30.2)	49 (51.0)	18 (18.8)	29 (30.2)	67 (69.8)
2007/2008	152	0 (0.0)	2 (1.3)	27 (17.8)	63 (41.4)	62 (40.8)	27 (17.8)	125 (82.2)
2008/2009	95	2 (2.1)	5 (5.3)	53 (55.8)	30 (31.6)	12 (12.6)	53 (55.8)	42 (44.2)
2009/2010	136	0 (0.0)	3 (2.2)	73 (53.7)	52 (38.2)	11 (8.1)	73 (53.7)	63 (46.3)
2010/2011	87	0 (0.0)	0 (0.0)	45 (51.7)	36 (41.4)	6 (6.9)	45 (51.7)	42 (48.3)
2012/2013	159	0 (0.0)	5 (3.1)	115 (72.3)	23 (14.5)	21 (13.2)	115 (72.3)	44 (27.7)

SOURCE: Dept of Computer Science, Federal Polytechnic, Nasarawa. Nigeria, 2015.

Interestingly, the failure rate for 2013 was the lowest. To probe further into the reason for this apparent improvement, I went through the examination scores of this set of students and discovered that only 45 students out of 159 that sat for the examinations scored 50% and above. This makes the success rate in the examination to be 28.3%. This shows that the initial success rate of 72.3% is doubtful. This result was probably influenced by grade inflation: The high Continuous Assessment (CA) marks for take-home assignments, tests and labs awarded to the students. We can reasonably conclude with the available data that generally, students' performance in CS1 is poor in this polytechnic.

Table 1-3 presents the available CS1 data from Niger State Polytechnic, Zungeru (NSPZ). The table shows that failure rates for the six sessions reported were 49.1%, 63.8%, 50.4%, 41.7%, 68.9% and 70.7%. The average failure rate for the six sessions was 57.4%. we can then conclude that performance of students from this polytechnic is same as the two previous schools. However, NSPZ which happens to be a state government-owned institution unlike FPN and FPB had some of the poorest performances in the sessions reported. State institutions in Nigeria are often less well-funded or equipped unlike their federal counterparts.

Table 1-3 Niger State Polytechnic Zungeru Students' Performance in CS1

Session	No. of Students Examined	Scored 80 and above No. (%)	Scored 70 and above No (%)	Scored 50 and above No (%)	Scored between 40 and 49 No. (%)	Scored less than 40 No. (%)	Clear Pass No. (%)	Poor (?) or Failed No. (%)
2005/2006	108	0 (0.0)	6 (5.6)	55 (50.9)	46 (42.6)	7 (6.5)	55 (50.9)	53 (49.1)
2007/2008	152	5 (3.3)	18 (11.8)	55 (36.2)	63 (41.4)	34 (22.4)	55 (36.2)	97 (63.8)
2008/2009	133	4 (3.0)	12 (9.0)	66 (49.6)	39 (29.3)	28 (21.1)	66 (49.6)	67 (50.4)
2010/2011	48	1 (2.1)	5 (10.4)	28 (58.3)	12 (25.0)	8 (16.7)	28 (58.3)	20 (41.7)
2011/2012	90	2 (2.2)	1 (1.1)	28 (31.1)	45 (50.0)	17 (18.9)	28 (31.1)	62 (68.9)
2012/2013	92	0 (0.0)	5 (5.4)	27 (29.4)	40 (43.5)	25 (27.2)	27 (29.4)	65 (70.6)

SOURCE: Dept of Computer Science, Niger State Polytechnic, Zungeru. Nigeria. 2015

Schoeman (2015) corroborates the incidence of high CS1 failures in South Africa, reported earlier by Chetty and Barlow-Jones (2014). Schoeman's findings are presented in Table 1-4. The two studies differ in research subjects. Those of Chetty and Barlow-Jones (2014) were full-time university students while Schoeman (2015) were students from an Open Distance Learning (ODL) university. The average pass rate for these eight sets of CS1 students is 30.3%, indicating a failure rate of 69.7%. This higher failure rate compared to 60% found by Chetty and Barlow-Jones (2014) may be due to the additional challenge of studying in an ODL university.

Table 1-4 CS1 Registration, pass rate, distinctions and dropout from University of South Africa

Examination Year Sitting	Registration Module Count***	Examination Sitting Admitted	Normal Wrote*	Normal Pass*	Normal* Pass Rate Percentage (Passed /written %)	Number of Distinctions (Percentage of written in brackets)**	Dropout ***	Dropout rate (Dropout / Enrollment %)
2011 Jun	2381	2183	1923	549	**28.5%**	198 (10%)	458	19.2%
2011 Nov	1457	1197	1191	346	**29.1%**	111 (9%)	266	18.3%
2012 Jun	2213	2061	1827	582	**31.9%**	197 (11%)	386	17.4%
2012 Nov	1528	1417	1267	352	**27.8%**	106 (8%)	261	17.1%
2013 Jun	1180	1083	988	337	**34.1%**	138 (14%)	192	16.2%
2013 Nov	1145	1020	920	276	**30.0%**	83 (9%)	225	19.7%
2014 Jun	976	889	809	258	**31.9%**	101 (13%)	167	17.1%
2014 Nov	1054	1017	894	263	**29.4%**	78 (9%)	160	15.1%

(**Legend**: * – The term 'Normal' refers to students who were registered for the specific semester. ** – The percentage of students who wrote the examination and obtained distinctions appears in brackets after the number of distinctions. *** – 'Dropout' refers to students who were registered at the end of the semester but did not write the examination. It does not include cancellations, which will raise the dropout rate considerably when taken into account.) Similarly, 'Registration Module Count' indicates the number of students still registered by the end of the semester and therefore excludes cancellations.
SOURCE: Extract from Schoeman (2015, p. 4)

The impression from the above studies and other research in the literature is clear: the problem of learning and teaching programming to novice undergraduates is widespread. For this reason, the problem remains one of the grand challenges of computer science education (Bruce, 2018; McGettrick et al., 2005). This makes this study a significant research endeavour.

Several reasons have been identified for this high failure rate in CS1. These include but not limited to, incorrect mental models, or misconceptions that students bring into the CS1 class (Qian & Lehman, 2017), a fixed mindset that believes that the ability to learn programming is inborn, even among students who were high achievers before enrolment (Scott & Ghinea, 2014); stereotypical belief, especially among women and men who see computing or IT as a male's field (Rubio et al., 2015); poor or weak mathematics background (Qian & Lehman, 2017)

Proponents of *Scratch*, a visual Educational Programming Language (EPL), claimed that the program could make the introduction of programming concepts to novice students easy (Maloney et al., 2010; Resnick et al., 2009). It reduces the cognitive load on novice students since they do not have to grapple with syntax errors in their programs. In addition, students construct programs by using multimedia components such as text, audio, videos, graphic images, or pictures that are of interest to the students. The initial vision of the authors of *Scratch* was for after-school novice programming club members in the age range of 8 – 16 years (Maloney et al., 2008). To address the problem of high failure rates and increase novice students engagement, introductory programming classes (CS0 or CS1) in some colleges and higher institutions now employ some forms of *Scratch* instruction (Becker, 2019; Cárdenas-Cobo et al., 2021; Hijón-Neira et al., 2021; Malan & Leitner, 2007; Papadakis & Kalogiannakis, 2019; Tijani et al., 2020; Topalli & Cagiltay, 2018)

To the best of my knowledge, at the beginning of this research, no empirical study of this scale involving CS1 students in post-secondary institution has been undertaken in Nigeria. The study took place within two sessions (2015-2016) in four Nigerian polytechnics involving five cohorts of CS1 students. Therefore, the aim of this study is to assess the impact of a constructionist *Scratch* programming instruction on the achievements of first-year computer science in programming in some Nigerian polytechnics.

1.2 Statement of Problem

The predominant programming languages of choice in most CS1 classes are, textual programming languages. Examples of such include Java, Python, Visual Basic, and C#. In these languages, the students must master the syntax and semantics of some programming statements and write programs in words. Prior research suggests that learning and using correct programming language syntax adds to the cognitive load of novice students (Medeiros et al., 2019; Sands, 2019).

To address this problem, *Scratch*, a visual programming language, has been suggested as an easier alternative language for introducing novice students to programming (Carlos Begosso et al., 2020; Pérez-Marín et al., 2020; Tijani et al., 2020). However, the following gaps exist:

- While there is extensive research on impact of *Scratch* programming environment on novice students, most of these studies are limited to K-12 (i.e. primary and secondary) schools, and informal settings like after-school computer clubs. However, gaps exist of its impact on undergraduate CS1 students
- There is need for research on the suitability, acceptance and effectiveness of *Scratch* in higher education CS1 classes (Arpaci et al., 2019).
- Though *Scratch* was born out of Seymour Papert's Theory of constructionism, *Scratch* classes as reported in most studies are hardly constructionist. They are pre-eminently lecture-based in nature thereby failing to explore fully the idea behind the program.
- In addition, while *Scratch* has been used and investigated in other countries, it has, to the best of my knowledge, never been used nor its impact investigated in Nigerian higher education CS1 classes, except for a recent study by Tijani et al.(2020) involving preservice student teachers.

In the light of prior research, the argument of this thesis is that exposing novice CS1 undergraduates to programming in a constructionist *Scratch* class will lead to positive affective and cognitive achievement in programming. The reasons for such improvement I suspect, are due to these factors:

- Low barrier to programming that *Scratch* affords
- Low cognitive load during programming
- Increased motivation in a constructionist programming class
- Increased engagement with programming artefacts of interest
- Increased self-efficacy. Self-efficacy has been reported to correlate with programming success in CS1.
- Positive change in attitude towards programming
- Connecting with student's interests or values.
- Collaborative atmosphere in a constructionist class
- Opportunities for experimentation, tinkering and bricolage that such class engenders.

Therefore, the problem of this study is whether exposing a novice undergraduate computer science student to programming following a constructionist pedagogy in a *Scratch* programming class, results in any meaningful learning, and whether such learning is comparatively better than what same or similar students learn in a traditional class. In addition, the problem is, whether learning in a constructionist CS1 class is moderated by these variables: gender, age, prior programming experience, prior visual art experience and academic achievement level.

The aim of this study, therefore, is to assess the impact of a *Scratch* programming class on polytechnic computer science students' achievements in introductory programming.

1.3 Research Questions

The following research questions guided the study:

1 Is there a **significant difference** between the **pre** and **post Introductory Programming Achievement Test Scores of** first-year polytechnic **CS students, after a six-week *Scratch* programming instruction?**

2. Is there **a significant difference** in terms of **Introductory Programming Achievement Test Scores** between first-year polytechnic **CS students** in **a *Scratch* programming class and** those in **the conventional class?**

3. Under what **conditions** will learning programming with *Scratch* have an **effect** on the **Introductory Programming Achievement Test Scores** of first-year polytechnic CS students compared to learning in conventional class?

   3.1. Will **gender** have an **effect** on the **Introductory Programming Achievement Test scores of** first-year polytechnic CS students between those in **a *Scratch* class** and those in **the conventional class?**

   3.2. Will **academic background** have an **effect** on the **Introductory Programming Achievement Test scores** of first-year polytechnic CS students between those in a ***Scratch* class** and those in **conventional class**?

   3.3. Will **prior programming experience** have an **effect** on **the Introductory Programming Achievement Test scores** of first-year polytechnic CS students between those in a ***Scratch* class** and those in **conventional class**?

   3.4. Will **prior visual art experience** have a**n effect** on the **Introductory Programming Achievement Test scores** of first-year polytechnic CS students between those in a ***Scratch* class** and those in **conventional class**?

   3.5. Will **gender**, **age**, **academic background**, **prior programming experience** and **prior visual art** have **effect** on **the Introductory Programming Achievement Test scores** of first-year polytechnic CS students between those in a *Scratch* class and those in the conventional class?

1.4 Research Hypotheses

The following null hypotheses in ***Table 1-5*** were tested at the 0.05 level of significance in the study:

Table 1-5 Research Hypothesis

Research Hypotheses	**Research Study**
$H_0$1: There is no significant difference between the mean scores of the pre- and post- Introductory Programming Achievement Test (IPAT) of first-year polytechnic CS students, after a six-week *Scratch* programming instruction. **$H_0$1:** $\mu_{diff} = 0$ **H_a1: $\mu_{diff} \neq 0$** **Where $\mu_{diff} = \mu_{posttest} - \mu_{pretest}$**	A paired sample t test
H_o2: There is no significant difference in the mean scores of the post Introductory Programming Achievement Test (IPAT) of first-year polytechnic CS students in a constructionist *Scratch* class (experimental group) and those in the conventional class (control group), while controlling for their pretest scores. **Or** **$H_o2{:}\mu_{SC} = \mu_{CC}$** **$H_a2{:}\mu_{SC} \neq \mu_{CC}$** **(SC –*Scratch* Class** **CC – Conventional Class)**	A one-way between-groups Analysis of Covariance. **Variables**: **Independent Variable**: CS1 instruction (Constructionist *Scratch* vs Conventional) **Dependent Variable**: IPAT Posttest scores. **Covariates**: Pretest
H_o3: Gender has no **effect** on **the mean scores of the post Introductory Programming Achievement Test (IPAT)** of **first-year polytechnic CS students** in a **constructionist *Scratch* class (experimental group)** and those in the **conventional class (control group),** while controlling for their **pretest scores.** **$H_o3{:}\mu_M = \mu_F$** **$H_a3{:}\mu_M \neq \mu_F$** **M - Male** **F – Female**	A two-way between-groups Analysis of Covariance. **Variables**: **Independent Variable(IV)** : **Primary IV**: CS1 instruction (Constructionist *Scratch* vs Conventional) **Secondary IV**: Gender (male, female) **Dependent Variable**: Posttest scores. **Covariates**: Pretest

H_o4: Age has no **effect** on **the mean score of the post Introductory Programming Achievement Test (IPAT)** of **first-year polytechnic CS students** in a **constructionist *Scratch* class (experimental group)** and those in the **conventional class (control group),** while controlling for their **pretest scores.** **$H_o4: \mu_{Ag1} = \mu_{Ag2}\ \mu_{Ag3} = \mu_{Ag4}$** **Ag1 – Age 16-18** **Ag2 – Age 19-21** **Ag3 – Age 22-24** **Ag4 – Age above 24**	A two-way between-groups analysis of covariance **Variables**: **Independent Variable (IV)** : **Primary IV**: CS1 instruction (Constructionist *Scratch* vs Conventional) **Secondary IV**: Age (16-18, 19-21, 22-24, Above 24) **Dependent Variable**: Posttest scores. **Covariates**: Pretest
H_o5: Academic background has no **effect** on **the mean score of the post Introductory Programming Achievement Test (IPAT)** of **first-year polytechnic CS students** in a **constructionist *Scratch* class (experimental group)** and those in the **conventional class (control group),** while controlling for their **pretest scores.** **$H_o5: \mu_H = \mu_A = \mu_L$** **H-** High-achieving **A** – Average-achieving **L** – Low-achieving	A two-way between-groups analysis of covariance **Variables**: **Independent Variable (IV)**: **Primary IV**: CS1 instruction (Constructionist *Scratch* vs Conventional) **Secondary IV**: Academic Background (High, Average, Low) **Dependent Variable**: Posttest scores. **Covariates**: Pretest
H_o6: Prior programming experience has no **effect** on **the mean score of the post Introductory Programming Achievement Test (IPAT)** of **first-year polytechnic CS students** in a **constructionist *Scratch* class (experimental group)** and those in the **conventional class (control group),** while controlling for their **pretest scores.** **$H_o6: \mu_{NP} = \mu_{SP}$** NP- No Prior Programming Experience SP- Some Prior Programming Experience	A two-way between-groups Analysis of Covariance. Variables: **Independent Variable(IV)** : **Primary IV**: CS1 instruction (*Scratch* vs Conventional) **Secondary IV**: Prior program writing (none, some) **Dependent Variable**: Posttest scores. **Covariates**: Pretest

H_o7: Prior visual art experience has no **effect** on **the mean score of the post Introductory Programming Achievement Test (IPAT)** of **first-year polytechnic CS students** in a **constructionist *Scratch* class (experimental group)** and those in the **conventional class (control group),** while controlling for their **pretest scores.** **$H_o7: \mu_{NV} = \mu_{SV}$** **NV-** No Prior Visual Art Experience **SV-** Some Prior Visual Art Experience	A two-way between-subject Analysis of Covariance. **Variables**: **Independent Variable (IV)**: **Primary IV**: CS1 instruction (*Scratch* vs Conventional) **Secondary IV**: Prior visual art (none, some) **Dependent Variable**: Posttest scores. **Covariates**: Pretest
H_o8: Treatment, **Gender**, academic **background**, **prior programming experience** and **prior visual art** have no **interaction** on **the mean score of the post Introductory Programming Achievement Test (IPAT)** of **first-year polytechnic CS students** in a **constructionist *Scratch* class (experimental group)** and those in the **conventional class (control group),** while controlling for their **pre-test scores.**	A two-way between-subject Analysis of Covariance. **Variables**: **Independent Variable (IV)**: **Primary IV**: CS1 instruction (*Scratch* vs Conventional) **Secondary IVs**: Gender, Age, Academic Background, Prior Programming and Prior visual art **Dependent Variable**: Posttest scores. **Covariates**: Pre-test

1.5 The Scope of the Study

In this section, the boundaries of the study include:

- Concepts or topics to be covered in the intervention
- the geographical location of the study
- the study cases or samples
- the variables of interest

Concepts covered in the study are those taught in the first six weeks, of the first semester of first year studies in Nigerian polytechnics, as contained in the National Board for Technical Education (NBTE) curriculum. With the present lack of rigorous computer science education in the K-12 (primary and secondary) education in Nigeria, many of these study cases, will be confronted in CS1 class with little or wrong ideas of these basic concepts. Research findings also revealed that many of these topics are those which CS1 students grapple with

(Medeiros et al., 2019). These topics include programming, program, algorithms, variables, initialization, pseudo code, and control structures (i.e., sequence, selection, and iteration)

The study area was limited to Nigeria's north central region otherwise called the Middle Belt. It is a region made up of six states (Benue, Nasarawa, Niger, Kogi, Kwara, and Plateau) and Nigeria's Federal Capital territory. A quasi-experimental study was conducted in two federal and two state government-owned polytechnics located in Nasarawa and Niger States generating research data from four cohorts of students (see ***Figure 1-1***).

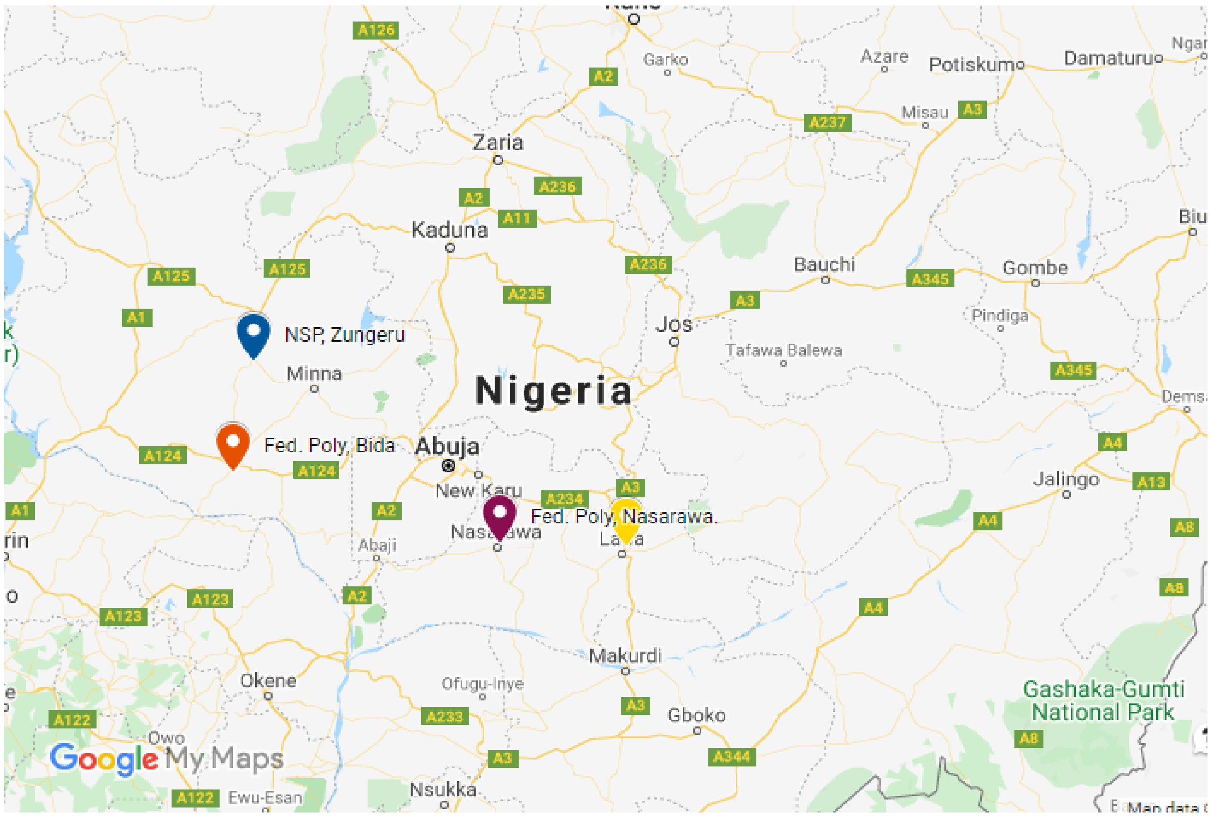


Figure 1-1 The Study Area in North Central Nigeria

1.6 Contributions of the Study

The result from the quantitative pre and post test data suggests a moderate impact on the students' achievements. However, this moderate impact needs to be interpreted in the context of the short duration of the exposure to *Scratch*, and educationally challenging situations of the environments of the study. The study took place in resource-constrained environments. Such resources include computing devices and regular electricity supply. Nevertheless, this study contributes to ongoing computer science education research, as exploring ways of engaging novice computer science remains a grand challenge (Sharmin, 2021).

The global economy increasingly relies on IT to function. Hence, the demand for computing and IT professionals is growing. Addressing the problem of CS1 with better pedagogical tools or approach, will likely boost students' interest and self-efficacy, provide important grounding in foundational programming concepts and motivate them to continue in computer science studies. Thus, with this boost in students' interest and engagement, it is likely to lead improved success rate in CS1 and reduced number of dropouts. This will contribute greatly in meeting the current and future global need for IT work force. This study provides empirical evidence of an engaging pedagogy, especially for novice students without prior programming experience (Campbell & Atagana, 2022) .

This study further provides empirical evidence of *Scratch* impact on CS1 students programming achievement. Thus, this research contributes to the global knowledge on the impact of *Scratch* programming as a programming learning tool in higher institutions. The results from this research will add to the Pedagogical Content Knowledge (PCK) of CS1 teachers intending to use or already using *Scratch* in their CS1 classes. This is one worthwhile pursuit of computer science education research as computer science teachers' PCK has been found to correlate with their students' content knowledge (McKlin et al., 2019)

This study contributes empirical evidence on the application of the constructionist approach to programming education of novice CS1 students.

While this study did not find causality, it provides empirical evidence that prior programming experience correlates with CS1 students programming achievement. Being so, and since some seeds of the problem or advantage in CS1 are sown in prior education levels, this finding suggests the importance of exposing students to programming education during primary and secondary school education.

1.7 The Outline of the Thesis

The next chapter presents results of review of scholarship on computing, computational thinking, novice programming education, constructionism theory, constructionist programming learning, past studies on *Scratch* in Higher Education CS1 classes and other related themes.

Chapter 3 provides the research blueprint as well as activities followed in addressing the research problem. This consists of the research design, the sampling method, instruments, as well as the data collection techniques and procedure employed in the study.

Chapter 4 presents the data analysis, presentations and interpretations of results, and discussions.

Chapter 5 presents the conclusions and recommendations of the study.

1.8 Definitions of Terms

The following definitions present their intended meanings in this thesis:

Achievement level: This is a derived variable. We used it to classify students based on their aggregate scores in the Unified Tertiary Matriculation Examination and their final secondary school examination grades in three compulsory subjects for admission into computer science. These subjects are physics, mathematics, and English language. The achievement level index was computed automatically from the data supplied by the students in the questionnaire. The index goes from 1 to 3, for low, average, and high achievement levels respectively.

Algorithm: This is a computational solution to a programming problem. It is a finite ordered list of steps that solves a problem or for performing a task.

Algorithmic Thinking: It is the ability of a student to think logically to understand and solve computational (or programming) problems.

Computational thinking: This is synonymous to algorithmic thinking. It is a regarded as one of the 21st century skills needed by students to function in a globalized knowledge economy.

Constructionism: A theory of learning propounded by the South-African born American mathematician and computer scientist Seymour Papert. Having worked with Jean Piaget, Papert took the former's theory of constructivism to another level by propounding that student not only construct their own knowledge, rather than being spoon-fed by teachers, but also that they learn as they construct artefacts of interest in collaboration with their peers. In a constructionist class the students take responsibility for their learning while the teachers act as facilitators.

CS0: The name given to the computer science course that novice students take prior to CS1. It is a remedial or an appreciation course aimed at developing the novice students' interest for further computer science studies.

CS1: A label given globally by computer science educators to the first introductory programming course that novice students are taught. It is a foundational course for fresh college or undergraduate students in computer science and other fields.

Educational Programming Languages: These programming environments are developed mainly to introduce novice programmers to programming. It provides a visual environment containing pictures and graphics for creating program unlike the textual programming languages where words or numbers (i.e., text) are used.

Novice programmers: these refer to those who are just learning to develop programs. These could be students in K-12 (i.e., primary, and secondary school) or higher educational institutions such as universities, colleges, or polytechnics.

Pedagogy: This refers to methods and manners teachers employ in the education of the learners in a particular domain. It is the process of accompanying learners in the pursuit of education to acquire knowledge and skills in such domain.

Programming: This is the process of developing programs or applications that drive automated devices such as computers, phones, robots etc.

Programming Achievement: A measure (or the result of assessment) of student's programming learning, knowledge, conceptual understanding and skills or ability to write correct programming constructs or detect programming errors. This is synonymous to programming aptitude or programming ability. In this study, a language-independent assessment was adapted to measure this construct.

Programming Languages: These are codes in form of words, numbers, or icons (pictures or graphic symbols) for writing programs, software or apps.

Prior Programming Learning: This is level of students' exposure to computer programs in formal or informal school settings. In this study prior programming learning was computed from participants answers to question in the questionnaire asking whether they have learnt programming before. An index of 0 or 1 representing none or some experience.

Prior Program Writing: This is level of students' background in developing programs. Computed from a participant's questionnaire responses, it is an index of 0 or 1 representing none or some experience writing.

Scratch: A visual educational programming learning environment developed by MIT for introducing novice programmers to programming concepts through the building of artefacts using multimedia components such as pictures, audio, and video clips.

Textual programming: The traditional way of developing programs with the use of words and numbers to forms statements (or instructions) in the program.

Visual Art Background: A level of students' prior experience in developing creative arts such as games, drawings, art works, video editing etc where their creativity leads to production of tangible artefacts in offline and online contexts. Computed from a participant's questionnaire responses, it is an index of 0 or 1 representing none or some experience in visual arts.

Visual Programming: is a way of developing or building programs majorly with the use of icons, pictures, or graphics as building blocks instead of words.

Chapter 2

2 Review of Literature

Chapter 1 has set the stage for exploring the impact of a constructionist *Scratch* instruction on the achievement of students in their first programming course (CS1). This chapter situates the current study in historical and ongoing global discourse. It also provides a theoretical and conceptual framework for the study. Other topics covered in this review include concepts of computing, computational thinking, computer science education, engaging computer science education, *Scratch*, learning theories, and the relationship between students' gender, age, academic background, prior programming writing, prior visual artistic ability, and their programming achievement. Simply put, the review will be done both theoretically and empirically.

2.1 Theoretical Review

2.2 Concept of Computing

Most people will agree that knowledge of computing is vital for life in modern societies. However, the question is, what is computing? The answer is not an easy one; it means different things to different people (Aho, 2012). The writers of current undergraduate ACM/IEEE computing curricula captures this problem of defining computing thus: "Although computing as a discipline has been around for more than eighty years, many population groups are still not clear about the subject area or what it means. The philosophy underpinning the CC2020 report is to treat computing as a meta-discipline—a collection of disciplines having a central focus of computing"(CC2020 Task Force, 2020). So the question remains, what is computing?

Denning and Martel (2015)—in their book, *Great Principles of Computing*—describe computing as not just "a tool for analysing data but also as a method of thought and discovery" (p. 1). This suggests both the use as well as the practice of computing as a field of knowledge. Association for Computing Machinery (ACM) Computing curricula 2005 provides a concise but comprehensive definition of computing as "any goal-oriented activity requiring, benefiting from, or creating computers" (ACM Computing Curricula, 2005, p. 9). This definition divides individuals or societies into two groups: consumers and producers of computing technologies. Yet, computing is such a hydra-headed and developing concept that probably no single definition may describe it. Szentgyörgyi (1999) reiterated that idea when she opined that Information Technology (IT) or computing "is not a state but is really a process, so IT is never ending unless it is interrupted." (p. 57)

Denning and Martel (2015) outlined the historical origin of computing from the time Alan Turing and other pioneers laid the foundations for the field in the 1930s. Before then, the terms "computation" and "computer" were already in use. Computation referred to mechanical steps followed to execute mathematical functions then while computers were human beings, mostly women, who did computations(Ceruzzi, 1991; Copeland, 2017; Denning & Martell, 2015). Erwig (2017) described computation simply as "a systematic way of problem-solving" (p. viii). The vision of Turing and other pioneers was the development of a machine with the capacity to perform automatic computation and to show intelligence, that is, Artificial intelligence (AI) (Copeland, 2017; Denning & Martell, 2015). This need for automation and AI in modern society remains an active driving force behind the demands for education, research, and developments in computing (Wing, 2008). In the next section, we will consider an aspect of this global drive for computing knowledge for all – popularly called computational thinking.

2.3 The Concept of Computational Thinking

One indication of a worldwide awakening for computing knowledge in modern society is the global call for computational thinking (Guzdial, 2015; Wing, 2017). Several authors have declared that computational thinking is a required skill for living and working in this digital 21st-century society (Haseski et al., 2018; Shute et al., 2017). While there is consensus on the need for students in K-16 (from primary to postsecondary) education to be exposed to computational thinking, nevertheless, there is no widely accepted definition for the concept.

In a 3-page viewpoint section in March 2006 edition of Communication of the Association for Computing Machinery (CACM), Jeannette Wing declared her vision of computing knowledge for all (Wing, 2006). Though she gave several definitions, analogies and anecdotes in that article to arouse widespread interest in computing, the often-cited definition is: "Computational thinking involves solving problems, designing systems, and understanding human behaviour, by drawing on the concepts fundamental to computer science."(Wing, 2006, p. 33). This definition appears far-reaching to have included the idea of the ability to understand human behaviour in computational thinking. However, in the light of developments in computational biology and artificial intelligence, Wing's viewpoint coming after successful use of computing in human genome sequencing in 2003, is understandable. From that 2006 article, it could be deduced that computational thinking is a skill set, as well as a mindset involving fundamental computing ideas, that everyone can acquire and apply to solve computational problems in any area of human endeavour.

Despite the emotional appeal and the manner with which (Wing, 2006) viewpoints on computational thinking resonated with many, there were also criticisms of this modern perspective on the concept from some who held a traditional view of CT (Denning, 2017; Hemmendinger, 2010). For instance, four years after Wing's article, Hemmendinger(2010) disagreed with some of the views expressed in that article, and adjudged them to be over-reaching, ambiguous, and exaggerated the need for people from all walks of life to think like computer scientists. The author, however, agreed that the use of computational thinking and methods of computing in all fields empowers individuals beyond what is possible without such applications. Similarly, in the same viewpoint section of Communication of the ACM, eleven years after the influential Wing's article, Denning (2017), in one of the most articulated and reasoned critiques of Wing and other modern conception of CT, pointed out that their definition of CT was vague. In his view, this was the reason behind the confusion amongst K-12 teachers who were in doubt about what constitutes the body of knowledge they are to teach in their classes.

Borne out of continued questioning by many educators, research collaborations with other computing educators, and in response to criticisms, Jeannette Wing has provided further definitions to clarify what she meant in 2006 by computational thinking. Table 2-1 captures the development of Wing's definition of computational thinking. You will observe her definitions of CT in 2006 and 2008 includes "understanding human behaviour" – a thing that looks tangential. However, as she grew in her understanding of what CT is, we observe that, that aspect of her initial conceptualisation is no longer included in subsequent definitions.

Table 2-1 Progression in Jeannette Wing's Conceptualisation of Computational Thinking

Definition	Source
"Computational thinking involves solving problems, designing systems, and understanding human behaviour, by drawing on the concepts fundamental to computer science."	(Wing, 2006, p. 33)
"Computational thinking is taking an approach to solving problems, designing systems and understanding human behaviour that draws on concepts fundamental to computing."	(Wing, 2008, p. 3717)
"the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent"	(Wing, 2010, p. 1)
"Computational thinking is the thought processes involved in formulating a problem and expressing its solution(s) in such a way that a computer-human or machine—can effectively carry out."	(Wing, 2014, "What is computational thinking?")
"Computational thinking is the thought processes involved in formulating a problem and expressing its solution(s) in such a way that a computer-human or machine—can effectively carry out."	(Wing, 2017, p. 8)

Now for a look at definitions by other authors, there are several, but we will consider a definition developed from a literature review, and another from a much extensive historical review. Selby and Woollard (2013) conducted a review of literature on definitions of computational thinking. They developed criteria for an acceptable definition and proposed a definition using the same terminologies found in the literature as stated in Table 2-2. Two well-known professors in computing education wrote the second definition in Table 2-2, taken from their book *Computational Thinking*. Denning and Tedre (2019) did a historical review of similar terms for computational thinking in use from the 1950s by Alan Perlis, who referred to it as algorithmic thinking, to Seymour Papert who in 1980 called it procedural thinking. In that book, they provided a historical root or context as well as broader perspectives to the concept of computational thinking. This perspective, according to Denning (2017) in his earlier critique of Wing's definition, is the traditional view of computational thinking while those of Wing and other similar definitions represent the modern view.

Table 2-2 Definition of Computational Thinking by Other Authors

Definition	Source
"Computational thinking is an activity, often product-oriented, associated with, but not limited to, problem-solving. It is a cognitive or thought process that reflects the ability to think in abstractions, the ability to think in terms of decomposition, the ability to think algorithmically, the ability to think in terms of evaluations, and the ability to think in generalizations."	(Selby & Woollard, 2013, p. 5)
"Computational thinking is the mental skills and practices for • designing computations that get computers to do jobs for us, and • explaining and interpreting the world as a complex of information processes."	(Denning & Tedre, 2019, p. 17)

The following remarks could be made from the above conceptualizations of computational thinking in Table 2-1 and Table 2-2:

- There is no consensus on the definition of computational thinking, and there may be none anytime soon. It is a developing definition shaped by many forces at different moments in its history. (Denning & Tedre, 2019; Selby & Woollard, 2013).
- Computational thinking deals with computational problems.
- Computational thinking produces solutions to these problems for a computing agent – man or machine – to follow.
- Computational thinking employs some core concepts or "thoughts processes" derived from computer science.

Apart from the definition of computational thinking, a question that has bothered researchers and educators is - what are the thought processes that are core to computational thinking? The answer to this depends on what perspective or definition anyone gives to the term. Therefore, as there are nuances in the definitions of computational thinking, so there are various submissions in the literature on what constitutes the core concepts or thought processes, that students need to learn while developing their computational thinking ability. Let us consider what some authors identified as core to computational thinking in Table 2-3.

Table 2-3 Computational thinking Core Concepts

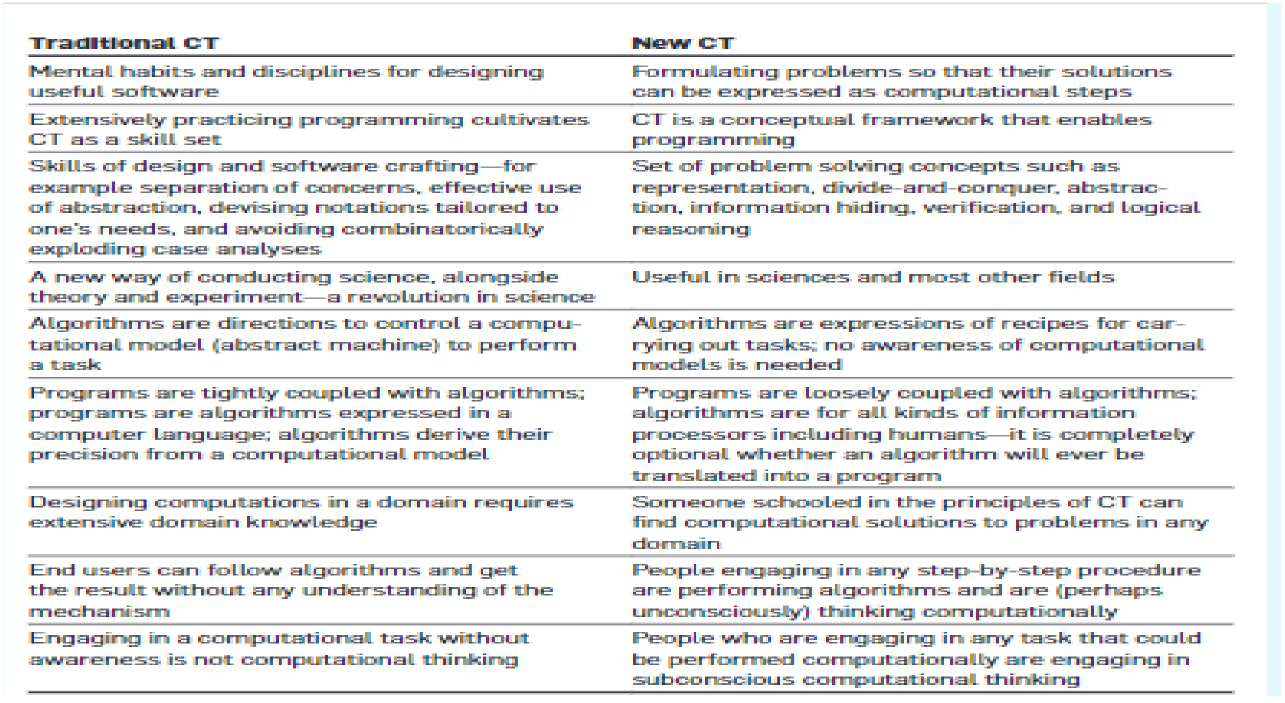
Concepts	Source
“the most important and high-level thought process in computational thinking is the abstraction process”	(Wing, 2017, p. 8)
Abstractions, decomposition, algorithmic thinking, evaluations, generalizations.	(Selby & Woollard, 2013, p. 5)
Logical thinking, algorithmic thinking, decomposition, generalisation and pattern recognition, modelling, abstraction, evaluation.	(Beecher, 2017, p. 11)
Decomposition, abstraction, algorithms, debugging, iteration and generalization.	(Shute et al., 2017, p. 151)
“Skills of design and software crafting—for example, separation of concerns, effective use of abstraction, devising notations tailored to one’s needs, and avoiding combinatorically exploding case analyses.”	(Denning, 2017, p. 37)

We can make the following deductions from these submissions:

- There is consensus that abstraction is key in computational thinking
- Decomposition, algorithmic thinking, and evaluation are also core concepts.

Amid unresolved questions and debates, as well as consensus bothering on the hot topic of computational thinking, Denning (2017) - a veteran computer science educator who holds a traditional view of computational thinking - provides an articulation of the comparisons and contrasts between the traditional view and the modern view (***Table 2-4***).

Table 2-4 Traditional Versus Modern Perspectives on Computational Thinking (CT)



Traditional CT	New CT
Mental habits and disciplines for designing useful software	Formulating problems so that their solutions can be expressed as computational steps
Extensively practicing programming cultivates CT as a skill set	CT is a conceptual framework that enables programming
Skills of design and software crafting—for example separation of concerns, effective use of abstraction, devising notations tailored to one's needs, and avoiding combinatorically exploding case analyses	Set of problem solving concepts such as representation, divide-and-conquer, abstraction, information hiding, verification, and logical reasoning
A new way of conducting science, alongside theory and experiment—a revolution in science	Useful in sciences and most other fields
Algorithms are directions to control a computational model (abstract machine) to perform a task	Algorithms are expressions of recipes for carrying out tasks; no awareness of computational models is needed
Programs are tightly coupled with algorithms; programs are algorithms expressed in a computer language; algorithms derive their precision from a computational model	Programs are loosely coupled with algorithms; algorithms are for all kinds of information processors including humans—it is completely optional whether an algorithm will ever be translated into a program
Designing computations in a domain requires extensive domain knowledge	Someone schooled in the principles of CT can find computational solutions to problems in any domain
End users can follow algorithms and get the result without any understanding of the mechanism	People engaging in any step-by-step procedure are performing algorithms and are (perhaps unconsciously) thinking computationally
Engaging in a computational task without awareness is not computational thinking	People who are engaging in any task that could be performed computationally are engaging in subconscious computational thinking

Extract from: (Denning, 2017, p. 37)

While both Traditional and Modern CT agrees on some common core concepts of computational thinking such as abstraction, decomposition, and evaluation, they disagree in what domain these concepts can be applied. The traditional view sees CT as a skill for developing software, the modern view sees it as a skill for developing any computational solutions other than software. Both disagree on the relationship between programming ability and computational thinking. The traditional view says programming will lead to the development of CT, while the modern view sees learning or exercising CT as leading to programming ability. The question of causality arises here. What causes what? Alternatively, one can ask, which one comes before which? It is like the question - which comes before which, the chicken or the egg? Experimenting can provide answers to these questions. However, the setup or design of the experiment will be determined by whichever view one holds. Those who hold the traditional view are likely to expose students without prior computational thinking skills to programming or software development instruction to test for computational thinking. On the other hand, those who hold the modern view will expose students with no prior programming skill to computational thinking instruction (which may not involve the development of the software) and then test their programming ability.

Like a jury who has listened to both parties in the computational thinking schools of thought, Curzon et al. (2019) in their contribution to *The Cambridge Handbook of Computing Education Research*, provided a well-articulated and excellent characterisation of the two perspectives (Figure 2-1). One can infer that the parties differ in three major ways. Firstly, disagreement between the two borders on the question of ***careers*** that need computational thinking. The traditional view believes that computational thinking is only applicable in computing and some related fields. While the modern view says, all fields of human endeavour in this 21st century will require computational thinking. Secondly, the two groups differ in their views on the question of ***context*** for the application of computational solutions. The traditional view claims that computational solutions such as algorithms or programs are meant for computers. The modern view says computational solutions are not limited to the machine but can be for humans acting as a computational agent. Thirdly, they differ on the question of ***causality*** between programming and computational thinking, that is how computation thinking develops or means for developing computational thinking. The traditional view holds that programming leads to computational thinking while the modern view says computational thinking can be developed by other means apart from programming.

Nevertheless, as indicated in Figure 2-1, several beliefs unite the two schools. We can categorise this area of agreement in these ways. Firstly, both agree on the **common core** of computational thinking. These constitute essential skills anyone who possesses computational thinking ability will demonstrate. This is very essential for uniformity in assessments of computational thinking whatever views educators or researchers may hold. Secondly, both agree on the **capacity** of computational thinking for universal impact. Therefore, both schools of thought are championing the cause for students to acquire computational thinking skills.

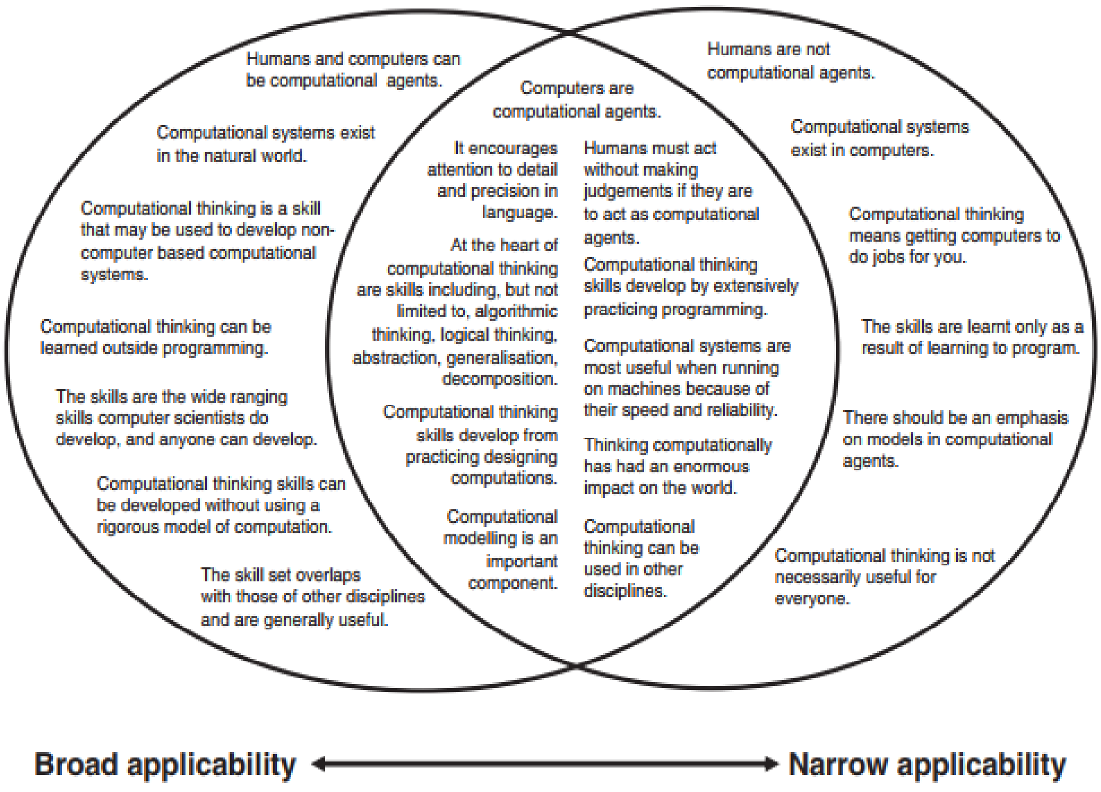

Humans and computers can be computational agents.
Computational systems exist in the natural world.
Computational thinking is a skill that may be used to develop non-computer based computational systems.
Computational thinking can be learned outside programming.
The skills are the wide ranging skills computer scientists do develop, and anyone can develop.
Computational thinking skills can be developed without using a rigorous model of computation.
The skill set overlaps with those of other disciplines and are generally useful.
Computers are computational agents.
It encourages attention to detail and precision in language.
At the heart of computational thinking are skills including, but not limited to, algorithmic thinking, logical thinking, abstraction, generalisation, decomposition.
Computational thinking skills develop from practicing designing computations.
Computational modelling is an important component.
Humans must act without making judgements if they are to act as computational agents.
Computational thinking skills develop by extensively practicing programming.
Computational systems are most useful when running on machines because of their speed and reliability.
Thinking computationally has had an enormous impact on the world.
Computational thinking can be used in other disciplines.
Humans are not computational agents.
Computational systems exist in computers.
Computational thinking means getting computers to do jobs for you.
The skills are learnt only as a result of learning to program.
There should be an emphasis on models in computational agents.
Computational thinking is not necessarily useful for everyone.
Broad applicability
Narrow applicability

Figure 2-1 A Characterisation of traditional and modern perspectives on CT

Source:(Curzon et al., 2019)

Given the above claims and hypotheses concerning computational thinking, the need for computer science educators and researchers to conduct well-designed and rigorous experimental research becomes necessary. Such studies can provide evidence that will lay to rest questions and debates. Convincing empirical results will encourage uniformity in educational training and assessment of students in computational thinking. Computer Science education research has a history of verifying such claims (Guzdial & du Boulay, 2019). What we need is more of empirical investigations instead of peddling unfounded claims. This sentiment for empirical proof is reiterated in the seminal work *Cambridge Handbook of Computing Education* (Blikstein & Moghadam, 2019; Fincher et al., 2019). This study seeks to contribute some empirical evidence to the question of causality between programming and computational thinking.

2.4 Computer Science Education

As nations realise the value of computing, the need for computer science education is growing. This global interest is seen in the level of commitment by governments, business organisations, and private individuals to computer science educational programmes. So much is said in the news and various fora about computing science education. However, the question is what is computer science education?

2.4.1 Defining Computer Science Education

Computer science education is a field of study that is concerned with developing students' computational thinking ability. It is a discipline that empowers learners by transforming them into solution providers to today and tomorrow's computational problems, in a world increasingly dependent on information and communication technologies. In other words, computer science education is the teaching and learning of the science and art of computing. However, the concept is a nuanced term having various forms and meanings in the literature and different parts of the world. Some of these other terms include computing education, informatics education (commonly used in Europe), IT education, and ICT education (commonly used in Europe and South Africa). Tedre, Simon and Malmi (2018) described computing education as a field whose goal is "to facilitate the learning of what the computing community believes fresh graduates should know about computing."(p.22).

In a report *Informatics Education in Europe: Are We All In The Same Boat?*, a joint committee of ACM Europe Council and Informatics Europe described informatics education as "a distinct scientific discipline, characterised by its concepts, methods, a body of knowledge, and open issues. It covers the foundations of computational structures, processes, artefacts and systems; and their software designs, their applications, and their impact on society." (Vahrenhold et al., 2017, pp. 1–3)

According to UNESCO's International Standard Classification of Education (ISCED), ICT education includes ICT fields such as:

- Computer use (e.g., training in the use of application software and internet)
- Database and network design and administration (e.g., Computer administration and management, Computer network installation and maintenance, Database administrator studies, Information technology administration, Web design, etc)
- Software and applications development and analysis (e.g., Computer science, system analysis, system design, software engineering, etc)
- Artificial intelligence
- Inter-disciplinary programmes and qualifications involving Information and Communication Technologies. (e.g., Bioinformatics or computational biology, computational mathematics, computational physics, etc)(UNESCO Institute for Statistics, 2015)

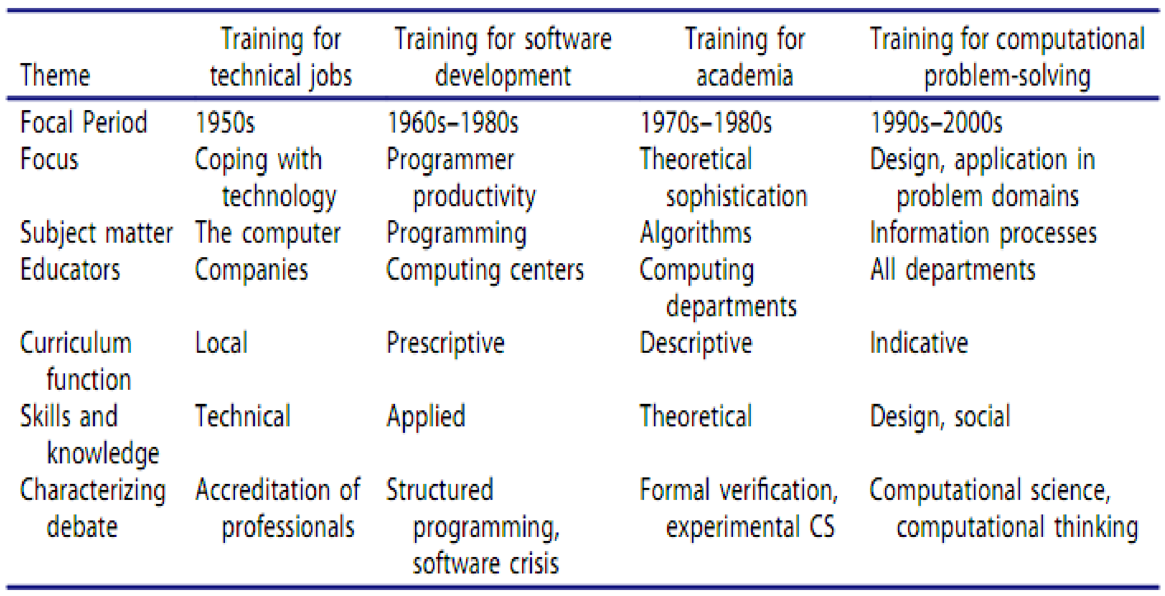
An ACM-IEEE curriculum task force involving IT professionals and educators defines IT education as: "the study of systemic approaches to select, develop, apply, integrate, and administer secure computing technologies to enable users to accomplish their personal, organizational, and societal goals."(Task Group on Information Technology Curriculum, 2017)

The products of the above educational programmes are staples of modern societies. Think of the world without the internet, Facebook, Instagram, WhatsApp, Twitter, Google, Baidu, Yandex, WeChat and several other IT applications and devices that modern life depends on. Most of these ICT tools were developed by former students of computing. Considering the benefits, the world derives today from these technologies, and the growing demands for more sophisticated technologies, it is imperative for computer science educators to produce more inventors, scientists, engineers, entrepreneurs, developers, and other computing professionals the world needs today and tomorrow. To achieve this objective, it becomes necessary to approach this business of motivating and educating newbies into computer science by employing empirically proven pedagogical strategies.

2.4.2 Directions from the Past, and Developments in Computing Education

With the advent of electronic computers in the 1940s came the need for manpower that will operate or program them, which birth the need for computing professionals. Tedre, Simon and Malmi (2018) provides a comprehensive historical review of the beginning of these educational programmes in computing. The summary of the history of computing education as stated by Tedre, Simon and Malmi (2018) is presented in Table 2-5. You will observe that computing education has undergone four eras. It started with training for technical jobs by different companies in the 1950s. One such prominent company was International Business Machine (IBM) in the US. It manufactured and exported computers, trained computing manpower and had offices in different parts of the world. For instance, from the early 1960s IBM computers were already in use in Nigeria (Anyanwu, 1978; Nwachukwu, 1994). To address technical manpower needs in host countries, IBM established training institutes. The first such training centre for West Africa was established in Nigeria at the University of Ibadan in 1963 (Anyanwu, 1978; Nwachukwu, 1994).

Table 2-5 A historical characterisation of computing education



Theme	Training for technical jobs	Training for software development	Training for academia	Training for computational problem-solving
Focal Period	1950s	1960s–1980s	1970s–1980s	1990s–2000s
Focus	Coping with technology	Programmer productivity	Theoretical sophistication	Design, application in problem domains
Subject matter	The computer	Programming	Algorithms	Information processes
Educators	Companies	Computing centers	Computing departments	All departments
Curriculum function	Local	Prescriptive	Descriptive	Indicative
Skills and knowledge	Technical	Applied	Theoretical	Design, social
Characterizing debate	Accreditation of professionals	Structured programming, software crisis	Formal verification, experimental CS	Computational science, computational thinking

Source: (Tedre et al., 2018)

In the first era, with no computing programmes in the universities nor in the US, the only providers of computing education were companies. Each company taught their trainees with localised or customised syllabus in various countries. The same was the case in 1960 in Nigeria. Developments that took place in the 1950s in the US, began in Nigeria in the 1960s. Most sub-Saharan African nations have always lagged in computing technologies and computing education knowledge.

With the growing global distribution of computers, came increasing demands for software. The second era spanning the next decade was devoted to addressing the need for broadening participation of other computing professionals. Computer programming was not the preserve of the few gurus anymore. Note that in the table by Tedre et al.(2018), the second era was (it seems wrongly written as) the 1960s -1980s. Computing education was no longer limited to private companies' training institutes or public institutions laboratories. Universities programmes in computing science started springing. The first of these programmes in the US commenced at Purdue University in October 1962. Soon, many other computing programmes in the US and different continents of the world began during this era. Programming was the main preoccupation of computing education during this era. The first definitive curricula for computing education was produced in 1968 by the ACM. The same year saw the coming together of computing professionals and educators in a NATO-sponsored conference in Germany, to address the software crisis of large industrial software development. That conference fuelled the enthusiasm that led to the birth of a separate field in computing called software engineering.

Challenged by open questions in computer programming, and excited by the events of the previous decade, the next two decades were occupied with endeavours to address issues that affect the production of quality software. According to Tedre, Simon and Malmi (2018), this period marked the third era of computing education. Some of the events of the previous decade included the production of a curriculum for a degree programme in computing and the NATO-sponsored software engineering conferences. For instance, Prof. Edsger Dijkstra, the renowned Dutch Computer Scientist, got motivated by discussions at the NATO Software Engineering conference. He devoted research efforts that produced methods for formal verification of programs (Randell, 2018). Thus, further fuelling the shift to the popular programming paradigm during this era called structured programming, an influential idea he earlier propounded to address the problem of poor programming in the second era. To address criticisms of the definitive ACM curricula 1968, a new descriptive curriculum 1978 was developed which left guidelines for an individual institution to customise computing education, as they deem fit while preserving some common core.

We are in the fourth era of computing education according to Tedre, Simon and Malmi's characterisation. This era is marked by the need for universal application of computing in virtually all fields of human endeavours. This makes it mandatory that some forms of educational programmes in computing are now provided for all, from kindergarten to university, irrespective of learner's field of interest. To address differential computing educational needs, from ACM curricula 1991 to the present ACM curricula 2013, five computing fields, namely computer science, information science, information technology, software engineering and computer engineering, are recognised and each is provided with a separate curriculum. From country to country, curricula for K-12 are also being developed which gives emphasis not only to digital literacy but also computational thinking. It is the view of computing educators in various nations that their citizens should not only be consumers, but also producers of technologies.

2.4.3 Computer Science Education in the US

The National Academies of Sciences, Engineering, and Medicine in a report *Assessing and Responding to the Growth of Computer Science Undergraduate Enrolments* provides another historical view of computing education in US universities from 1965 to 2015 (National Academies of Sciences, Engineering, 2018) (Figure 2-2). As you will observe in the figure, interest in degree programmes in computing education has witnessed two eras of boom and sharp declines each. From 2005, we are witnessing the third era of increasing interest for a degree in computing. This is also an era when computing knowledge is being needed by other fields.

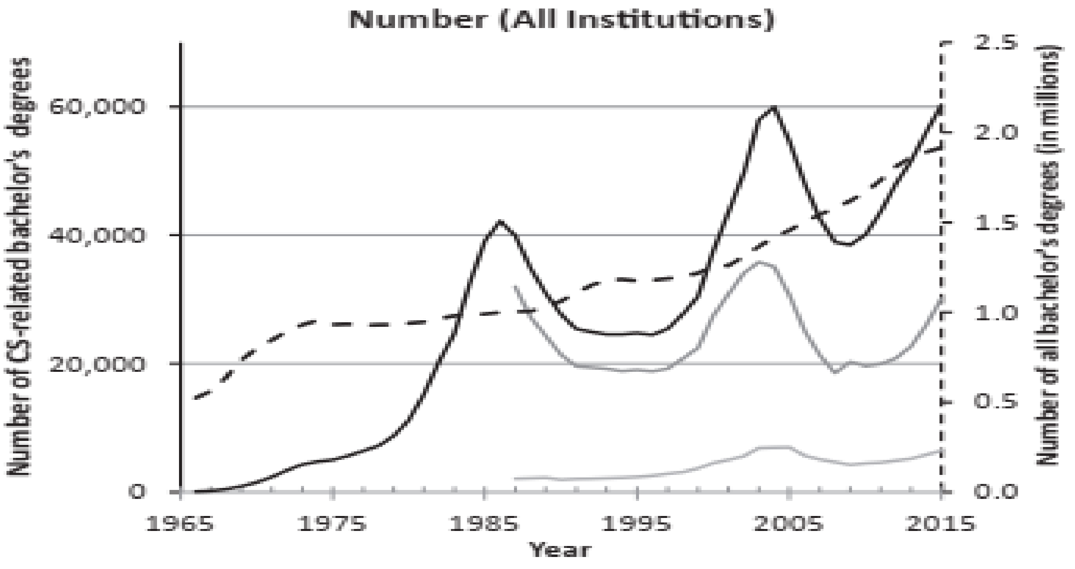


Figure 2-2 Historical view of computing degrees awarded in US universities

Source: (National Academies of Sciences, Engineering, 2018, p. 27)

Looking at ***Figure 2-3*** you will observe that the introduction to computing or programming classes is witnessing increasing enrolments even from students who do not intend to major in computing. This makes it worthwhile to research best ways to instruct the various novices enrolling in our introductory programming classes.

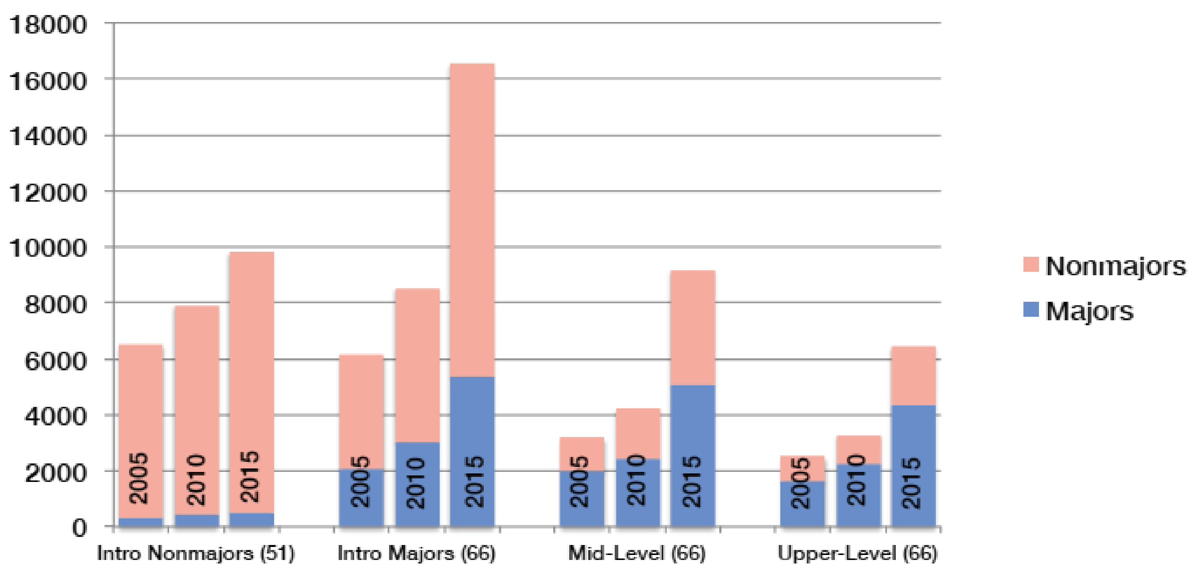


Figure 2-3 Computing course enrolments in US universities

Source: (National Academies of Sciences, Engineering, 2018, p. 46)

2.4.4 Computer Science Education Trends: Portrait from Europe

To find out if a similar trend in the US is observable in other parts of the globe, we will consider data (Figure 2-4) taken from an European report prepared by Informatics Europe (Tikhonenko & Pereira, 2019). In the diagram, some nations provided data for enrolments into what was called “research universities” (RU) and “universities of Applied Sciences” (UAS). The RUs are mainly higher education institutions (HEIs) devoted not only to teaching, but also to research in informatics whereas UASs in parts of Europe are simply institutes of technology or polytechnics devoted to vocational training in informatics. So, while PhDs are awarded by RUs, not all UAS award PhDs.

Trends in CS enrolments into informatics degree programmes in European HEIs provide a mixed picture as shown in the figure. While increased interest is observed from Germany, Italy, Lithuania, Netherlands, Poland, Portugal, Romania, Switzerland, and the UK. Interest in CS seems to decline or fluctuate in some countries like the Czech Republic, Finland, Ireland, Latvia, and Spain.

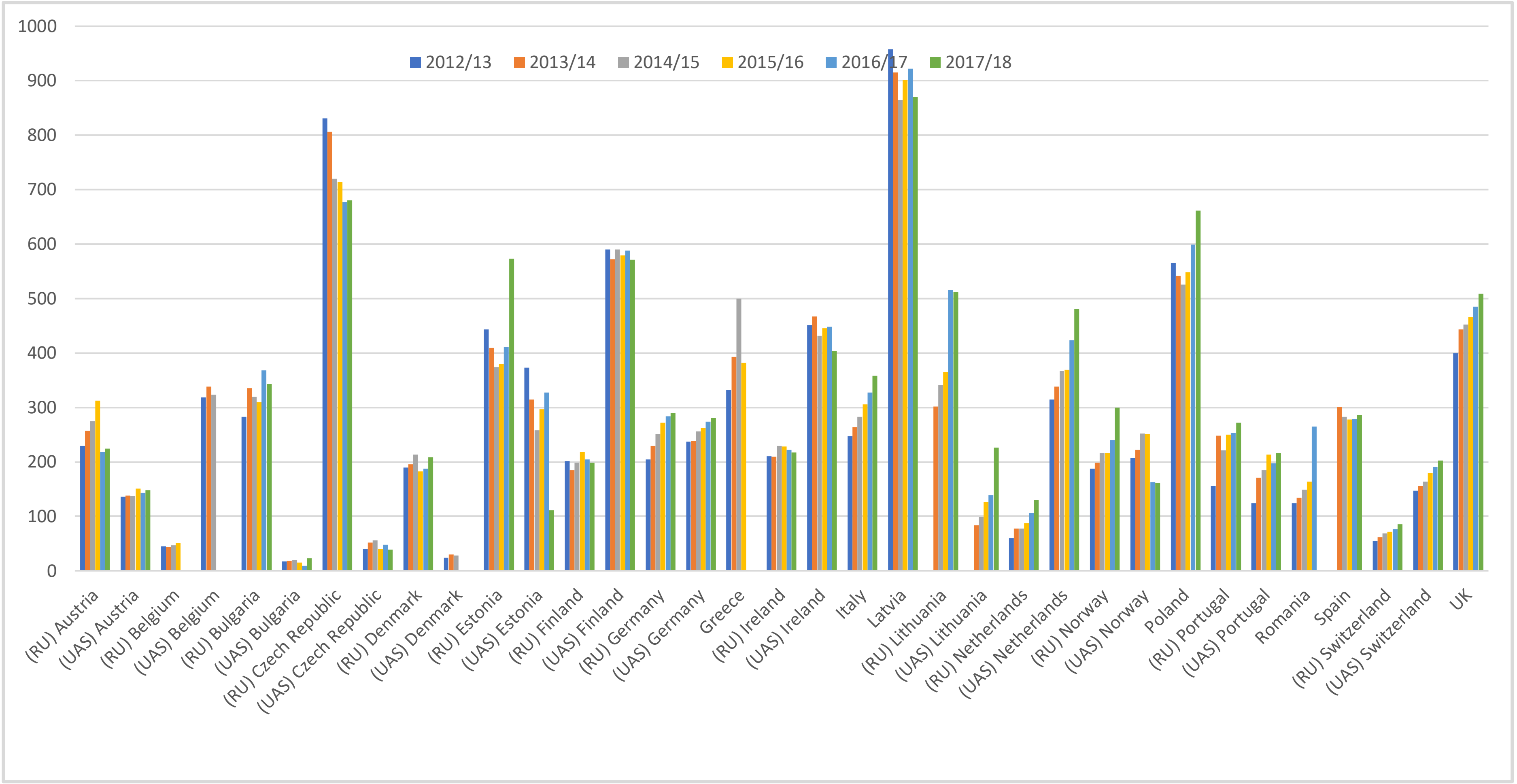


Figure 2-4 Enrolment of first-year students in bachelor's degrees in Informatics per 1,000,000 inhabitants in European Countries

Source: (Tikhonenko & Pereira, 2019, p. 35)

The problem with informatics education in Ireland has been reported in the literature (Piggot & Frawley, 2019). In the Irish Higher Education Authority report, the issue of high dropout rate of students in computing was given a singular mention. Of the students being tracked in the study from the 2007/2008 session, computing had the lowest completion rate of 55% (Figure 2-5). Another disturbing finding from the report is that among all dropout rates occurring after the first year of study in colleges or universities in Ireland, computing has the highest. This may provide further insight into the lack of enrolment into computing in Ireland, although the country is an IT hub in Europe being a leading exporter of software (Becker, 2019) and several vacancies are unfilled with needed IT expertise.

From the foregoing, we conclude that while there is growing interest computing in most European countries as in the US, however learning to program remains a challenge, continuing experience large failure or drop out rates (Becker, 2019).

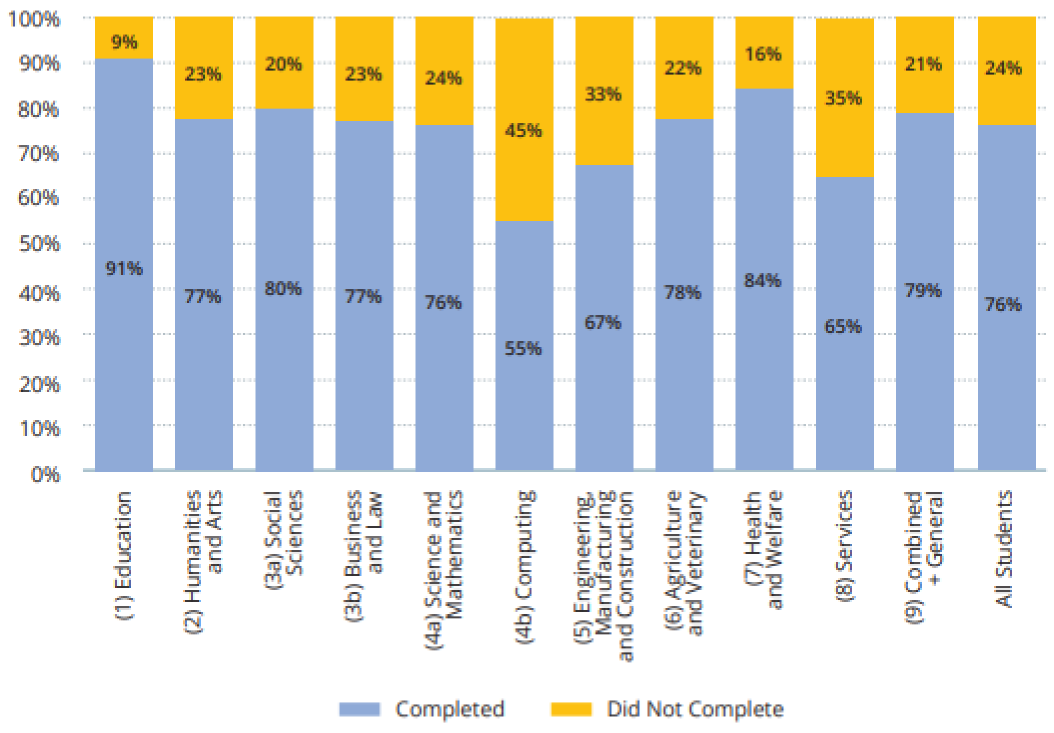


Figure 2-5 Degree Programmes completion rate, Ireland

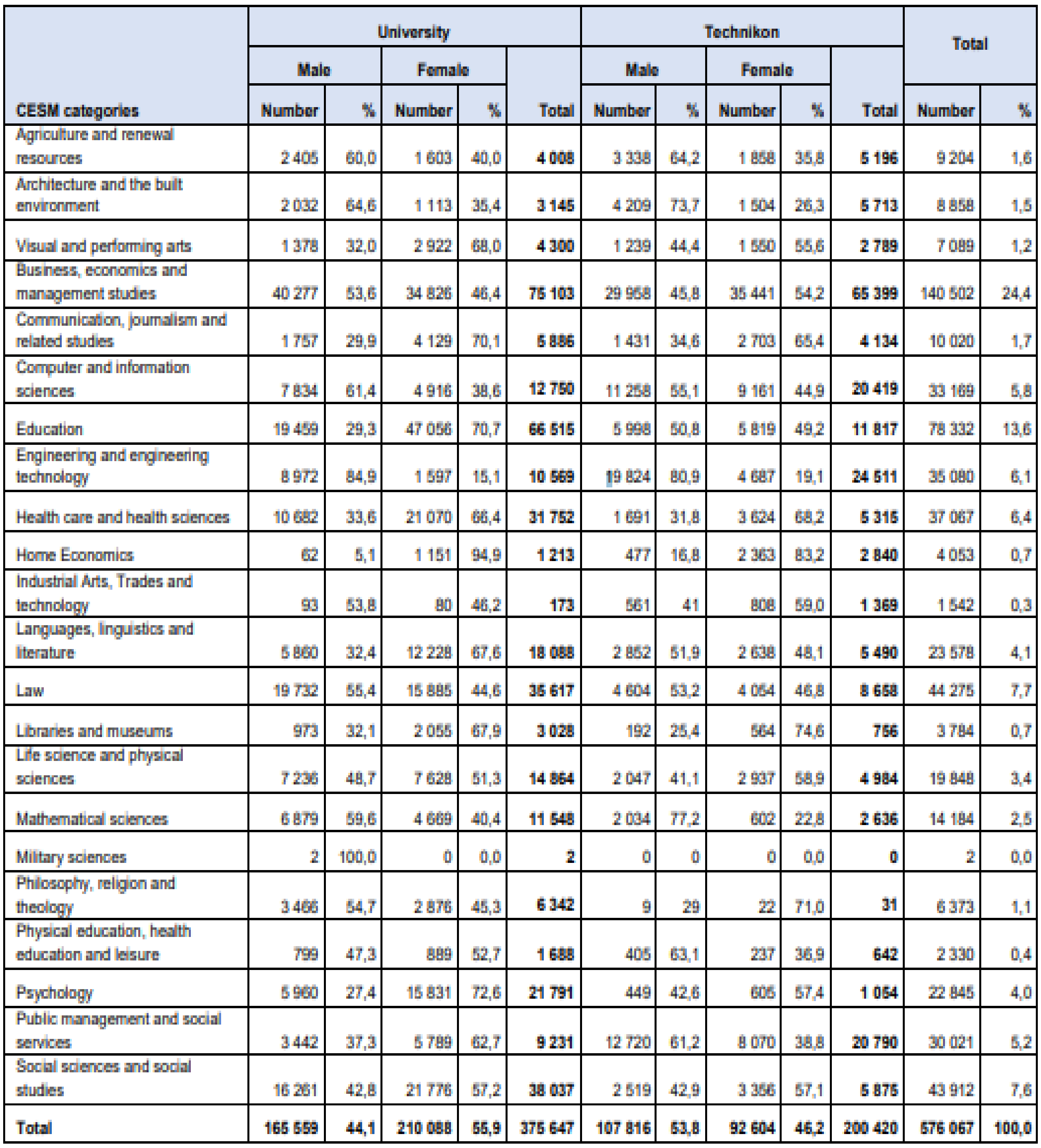
Source: (Piggot & Frawley, 2019, p. 59)

2.4.5 Computer Science Education in Africa

To comprehend interest trends in computing education in sub-Saharan Africa, we consider data from a report from South Africa. Statistics South Africa (2019) provides enrolments data from universities and other HEI referred to as Technikons. Table 2-6 indicates that a total of 12,750 applicants (representing 3.4% of applicants into South African universities) were admitted to study computing courses in 2000. By 2016 the percentage of universities admissions into computing courses rose slightly to 4.3% (Table 2-7). Surprisingly, although South

Africa is one of the most developed and industrialised countries in Africa, and highly in need of computing skills, there is slight increase in interest in computing. Looking at the percentage of combined enrolment into universities and Technikons in Table 2-6 which was 5.8, and the graduation rates in Figure 2-6 suggests that many who enrol into computing drop out or did not complete their studies at the normal time. This also suggests a problem with engaging or sustaining students' interest in computing.

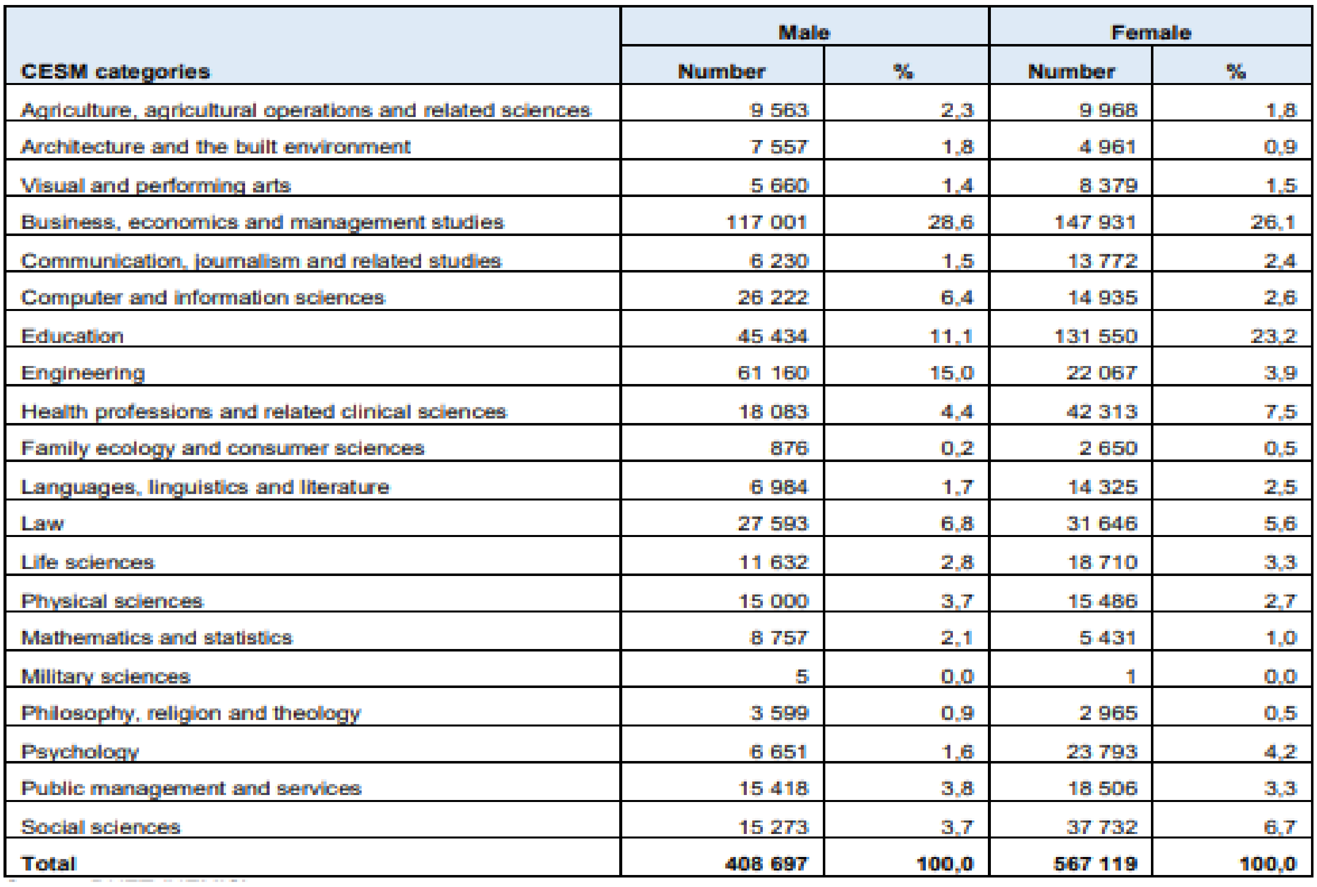
Table 2-6 Enrolments into SA Universities and Technikons in 2000



	University					Technikon					Total	
	Male		Female			Male		Female				
CESM categories	Number	%	Number	%	Total	Number	%	Number	%	Total	Number	%
Agriculture and renewal resources	2 405	60,0	1 603	40,0	**4 008**	3 338	64,2	1 858	35,8	**5 196**	9 204	1,6
Architecture and the built environment	2 032	64,6	1 113	35,4	**3 145**	4 209	73,7	1 504	26,3	**5 713**	8 858	1,5
Visual and performing arts	1 378	32,0	2 922	68,0	**4 300**	1 239	44,4	1 550	55,6	**2 789**	7 089	1,2
Business, economics and management studies	40 277	53,6	34 826	46,4	**75 103**	29 958	45,8	35 441	54,2	**65 399**	140 502	24,4
Communication, journalism and related studies	1 757	29,9	4 129	70,1	**5 886**	1 431	34,6	2 703	65,4	**4 134**	10 020	1,7
Computer and information sciences	7 834	61,4	4 916	38,6	**12 750**	11 258	55,1	9 161	44,9	**20 419**	33 169	5,8
Education	19 459	29,3	47 056	70,7	**66 515**	5 998	50,8	5 819	49,2	**11 817**	78 332	13,6
Engineering and engineering technology	8 972	84,9	1 597	15,1	**10 569**	19 824	80,9	4 687	19,1	**24 511**	35 080	6,1
Health care and health sciences	10 682	33,6	21 070	66,4	**31 752**	1 691	31,8	3 624	68,2	**5 315**	37 067	6,4
Home Economics	62	5,1	1 151	94,9	**1 213**	477	16,8	2 363	83,2	**2 840**	4 053	0,7
Industrial Arts, Trades and technology	93	53,8	80	46,2	**173**	561	41	808	59,0	**1 369**	1 542	0,3
Languages, linguistics and literature	5 860	32,4	12 228	67,6	**18 088**	2 852	51,9	2 638	48,1	**5 490**	23 578	4,1
Law	19 732	55,4	15 885	44,6	**35 617**	4 604	53,2	4 054	46,8	**8 658**	44 275	7,7
Libraries and museums	973	32,1	2 055	67,9	**3 028**	192	25,4	564	74,6	**756**	3 784	0,7
Life science and physical sciences	7 236	48,7	7 628	51,3	**14 864**	2 047	41,1	2 937	58,9	**4 984**	19 848	3,4
Mathematical sciences	6 879	59,6	4 669	40,4	**11 548**	2 034	77,2	602	22,8	**2 636**	14 184	2,5
Military sciences	2	100,0	0	0,0	**2**	0	0	0	0,0	**0**	2	0,0
Philosophy, religion and theology	3 466	54,7	2 876	45,3	**6 342**	9	29	22	71,0	**31**	6 373	1,1
Physical education, health education and leisure	799	47,3	889	52,7	**1 688**	405	63,1	237	36,9	**642**	2 330	0,4
Psychology	5 960	27,4	15 831	72,6	**21 791**	449	42,6	605	57,4	**1 054**	22 845	4,0
Public management and social services	3 442	37,3	5 789	62,7	**9 231**	12 720	61,2	8 070	38,8	**20 790**	30 021	5,2
Social sciences and social studies	16 261	42,8	21 776	57,2	**38 037**	2 519	42,9	3 356	57,1	**5 875**	43 912	7,6
Total	**165 559**	**44,1**	**210 088**	**55,9**	**375 647**	**107 816**	**53,8**	**92 604**	**46,2**	**200 420**	**576 067**	**100,0**

Source: (Statistics South Africa, 2019, p. 34)

Table 2-7 Enrolments into SA Universities in 2016



CESM categories	Male		Female	
	Number	%	Number	%
Agriculture, agricultural operations and related sciences	9 563	2,3	9 968	1,8
Architecture and the built environment	7 557	1,8	4 961	0,9
Visual and performing arts	5 660	1,4	8 379	1,5
Business, economics and management studies	117 001	28,6	147 931	26,1
Communication, journalism and related studies	6 230	1,5	13 772	2,4
Computer and information sciences	26 222	6,4	14 935	2,6
Education	45 434	11,1	131 550	23,2
Engineering	61 160	15,0	22 067	3,9
Health professions and related clinical sciences	18 083	4,4	42 313	7,5
Family ecology and consumer sciences	876	0,2	2 650	0,5
Languages, linguistics and literature	6 984	1,7	14 325	2,5
Law	27 593	6,8	31 646	5,6
Life sciences	11 632	2,8	18 710	3,3
Physical sciences	15 000	3,7	15 486	2,7
Mathematics and statistics	8 757	2,1	5 431	1,0
Military sciences	5	0,0	1	0,0
Philosophy, religion and theology	3 599	0,9	2 965	0,5
Psychology	6 651	1,6	23 793	4,2
Public management and services	15 418	3,8	18 506	3,3
Social sciences	15 273	3,7	37 732	6,7
Total	**408 697**	**100,0**	**567 119**	**100,0**

Source: (Statistics South Africa, 2019, p. 35)

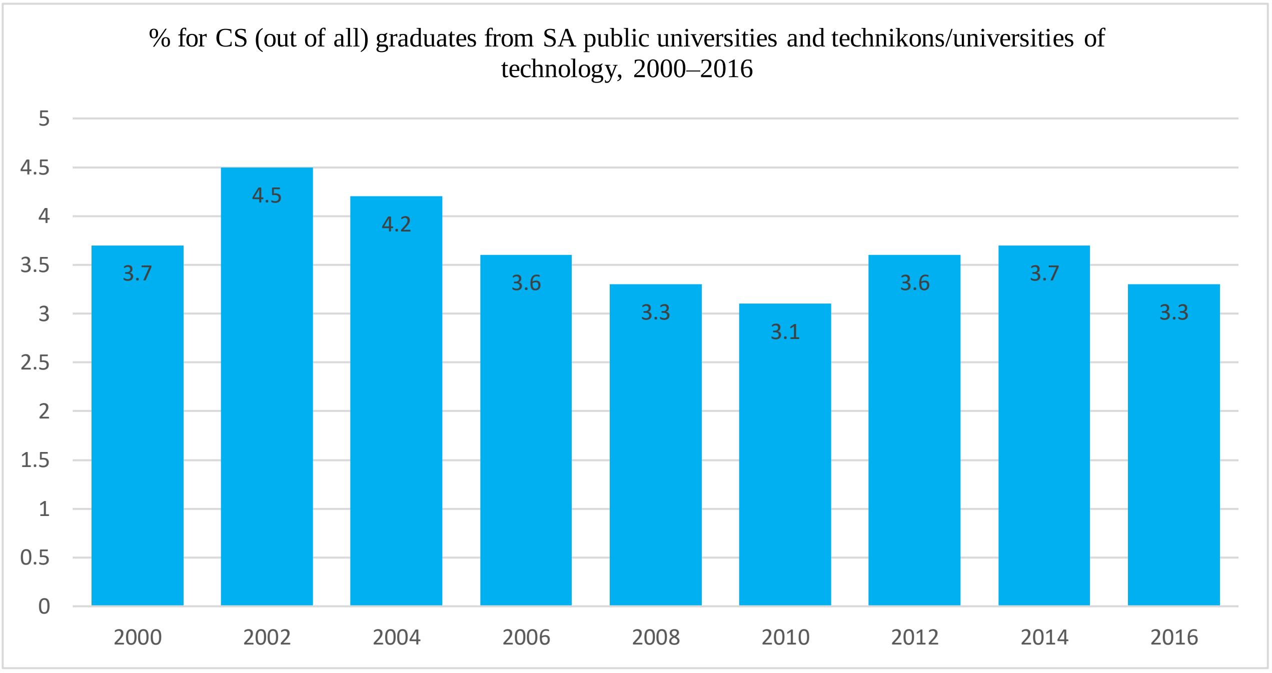


Figure 2-6 Percentage of Computing out of all Graduates from SA HEIs

Source: (Statistics South Africa, 2019, p. 57)

2.4.6 Computer Science Education: Its Driving Forces

To understand some of the forces behind the current era of increasing worldwide interest in computing, we will consider data from industries. For instance, McKinsey and Company carried out a survey of organisations from the US and fourteen European countries about the two historic drivers for computing technologies - automation and Artificial Intelligence (AI). Their finding is provided in Figure 2-7. As you will observe organisations are increasingly doing away with jobs that require physical, manual and basic cognitive skills. Such roles are being replaced by technologies like robots, employing automation and AI. However, some old and new jobs that require higher cognitive, social and emotional, as well as technological skills are witnessing increasing demands. Looking at the skills being demanded by organisations and governments in various nations, technological skills have the highest demands. Technological skills include digitals as wells as advanced IT and programming skills. A reason for increase in enrolments in institutions and other training avenues becomes apparent. Those skills can only be developed through computing education and training programmes, irrespective of whether the worker is a computing professional or not. This makes improved novice programming education that engages the students' passion for computing imperative – if we are to avoid the impending crisis resulting from the inability to supply computing skills needed for continued global development.

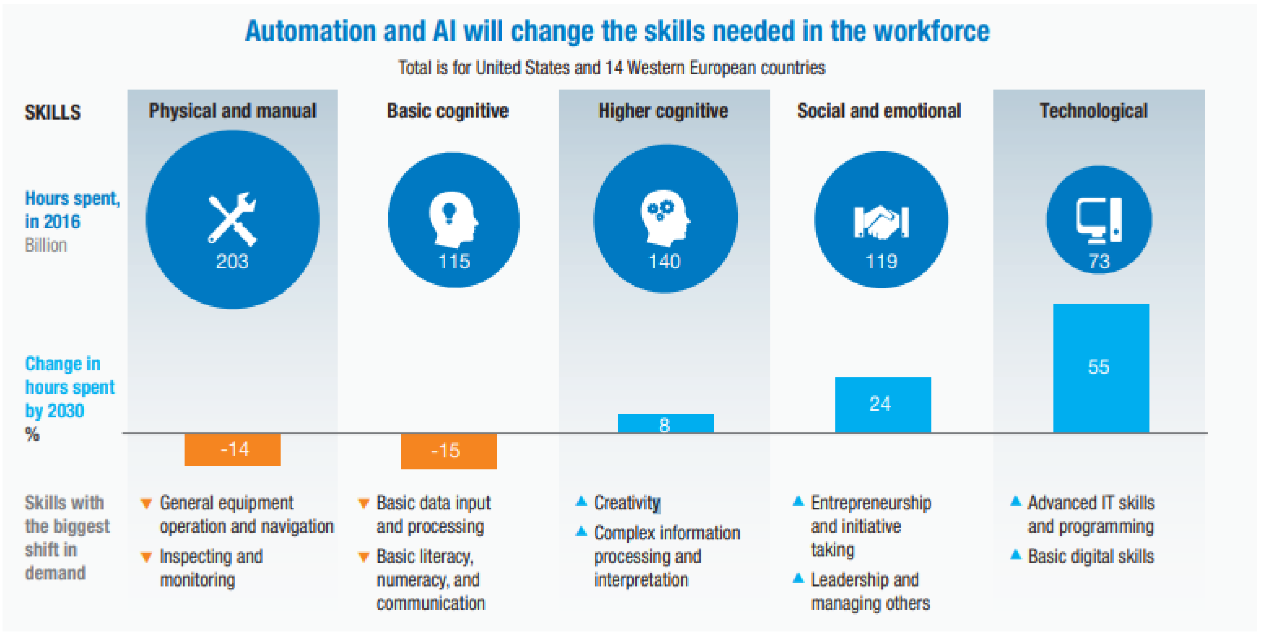


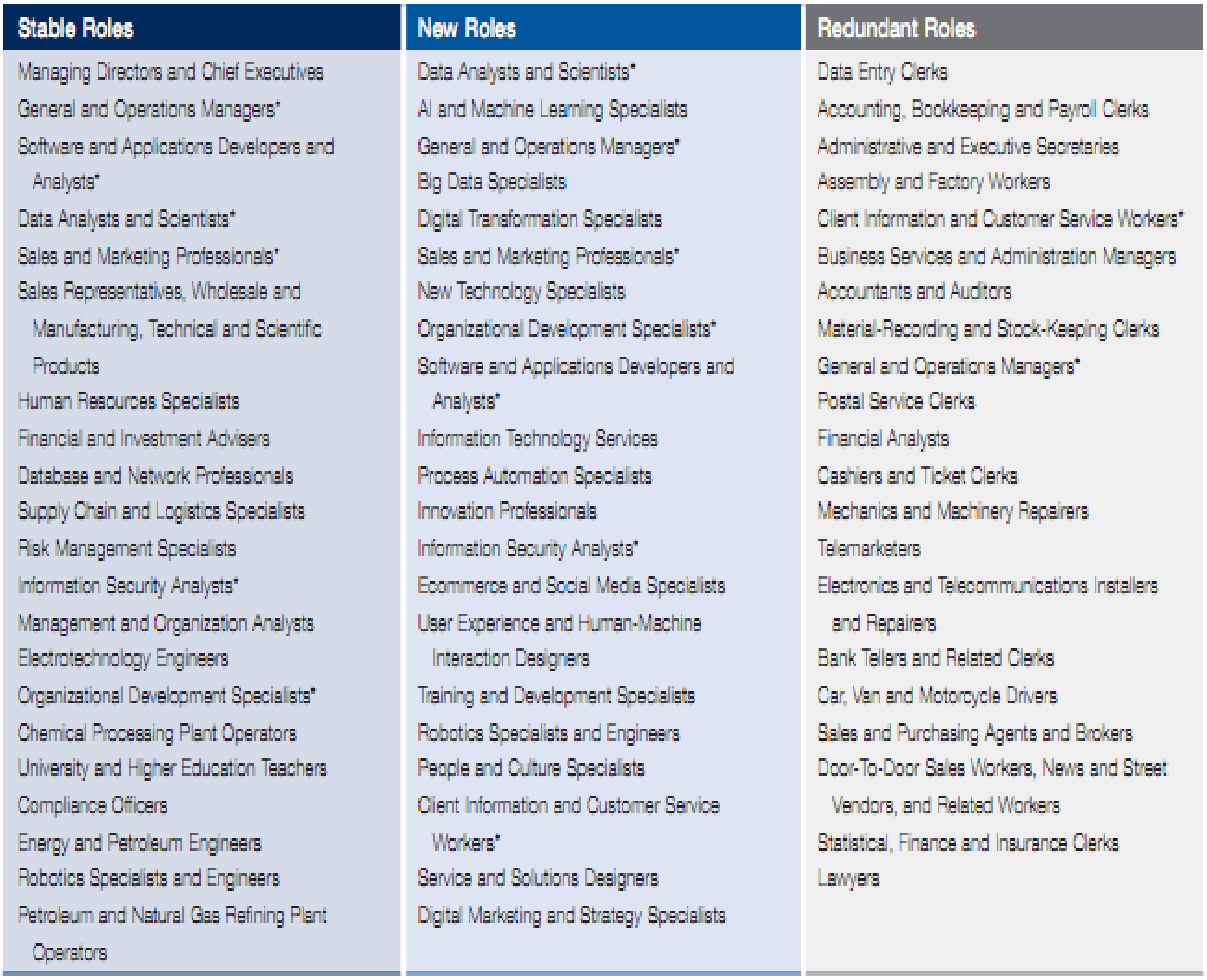
Figure 2-7 Demands for Skills: A survey of organisations in the US and Europe

Source: (Bughin et al., 2018, p. 5)

Findings by World Economic Forum corroborate what we now know about the changing nature of demands for workers in this era (World Economic Forum, 2018). Let us consider ***Table*** 2-8 taking from that report. Findings in the reports are based on survey data from a larger number of organisations and countries compared to those of McKinsey above. As you can observe, various roles requiring manual and basic cognitive skills such as bank tellers and clerks are becoming useless in modern banks. Technologies or machines are replacing such roles. For anyone in those roles to be relevant, it requires switching to new roles. This makes retraining necessary for such workers. The demand for new roles also drives interest in such fields in our training institutions. For those roles that are stable, anyone who wants to remain relevant in the face of continual changes and disruptive technologies being introduced, continued education becomes necessary.

World Economic Forum (2018) not only provides information that are life experiences of workers and management, they also give a view of projection describing how future place of work will look like (***Table 2-9***). Taking a serious consideration of these projections, makes it mandatory for computing educators to reorganize educational strategies to provide educational opportunities and environments for diverse students, to develop skills that adequately prepares them not only for today but also for future jobs.

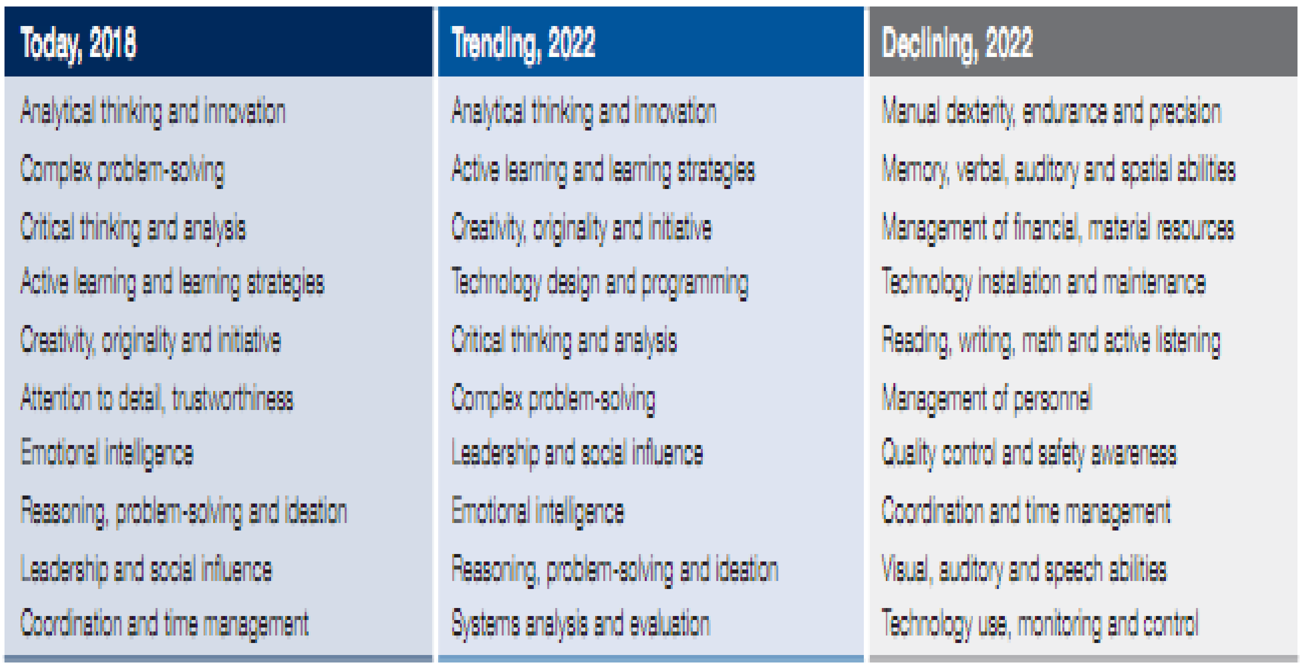
Table 2-8: Changing roles in modern workplaces



Stable Roles	New Roles	Redundant Roles
Managing Directors and Chief Executives	Data Analysts and Scientists*	Data Entry Clerks
General and Operations Managers*	AI and Machine Learning Specialists	Accounting, Bookkeeping and Payroll Clerks
Software and Applications Developers and Analysts*	General and Operations Managers*	Administrative and Executive Secretaries
Data Analysts and Scientists*	Big Data Specialists	Assembly and Factory Workers
Sales and Marketing Professionals*	Digital Transformation Specialists	Client Information and Customer Service Workers*
Sales Representatives, Wholesale and Manufacturing, Technical and Scientific Products	Sales and Marketing Professionals*	Business Services and Administration Managers
Human Resources Specialists	New Technology Specialists	Accountants and Auditors
Financial and Investment Advisers	Organizational Development Specialists*	Material-Recording and Stock-Keeping Clerks
Database and Network Professionals	Software and Applications Developers and Analysts*	General and Operations Managers*
Supply Chain and Logistics Specialists	Information Technology Services	Postal Service Clerks
Risk Management Specialists	Process Automation Specialists	Financial Analysts
Information Security Analysts*	Innovation Professionals	Cashiers and Ticket Clerks
Management and Organization Analysts	Information Security Analysts*	Mechanics and Machinery Repairers
Electrotechnology Engineers	Ecommerce and Social Media Specialists	Telemarketers
Organizational Development Specialists*	User Experience and Human-Machine Interaction Designers	Electronics and Telecommunications Installers and Repairers
Chemical Processing Plant Operators	Training and Development Specialists	Bank Tellers and Related Clerks
University and Higher Education Teachers	Robotics Specialists and Engineers	Car, Van and Motorcycle Drivers
Compliance Officers	People and Culture Specialists	Sales and Purchasing Agents and Brokers
Energy and Petroleum Engineers	Client Information and Customer Service Workers*	Door-To-Door Sales Workers, News and Street Vendors, and Related Workers
Robotics Specialists and Engineers	Service and Solutions Designers	Statistical, Finance and Insurance Clerks
Petroleum and Natural Gas Refining Plant Operators	Digital Marketing and Strategy Specialists	Lawyers

Source: Future of Jobs Survey 2018 as presented in (World Economic Forum, 2018, p. 9)

Table 2-9 Today and Tomorrow's Skills for workplaces



Today, 2018	Trending, 2022	Declining, 2022
Analytical thinking and innovation	Analytical thinking and innovation	Manual dexterity, endurance and precision
Complex problem-solving	Active learning and learning strategies	Memory, verbal, auditory and spatial abilities
Critical thinking and analysis	Creativity, originality and initiative	Management of financial, material resources
Active learning and learning strategies	Technology design and programming	Technology installation and maintenance
Creativity, originality and initiative	Critical thinking and analysis	Reading, writing, math and active listening
Attention to detail, trustworthiness	Complex problem-solving	Management of personnel
Emotional intelligence	Leadership and social influence	Quality control and safety awareness
Reasoning, problem-solving and ideation	Emotional intelligence	Coordination and time management
Leadership and social influence	Reasoning, problem-solving and ideation	Visual, auditory and speech abilities
Coordination and time management	Systems analysis and evaluation	Technology use, monitoring and control

Source: Future of Jobs Survey 2018 as cited in (World Economic Forum, 2018, p. 12)

In conclusion, it could be observed that various factors determine the type of computing education that students receive. It could be rapid technological change as we are currently witnessing (Fee et al., 2017); company demands for computing particular IT skills (Tedre et al., 2018); national policy (Zhang & Yan, 2010). We are witnessing all of these factors today, making the need for engaging computer science education more critical for the continued development and sustenance of modern societies.

2.5 Engaging Computer Science Education

Computer science education is faced with a dilemma. There is a growing demand for computing skills, yet an alarming shortfall in the supply of these skills from educational and training institutions. This situation leaves business organisations and nations competing for available skills from anywhere in the world. Consequently, there is a growing interest in computer science education. However, the global problem of high failure or dropout rate in introductory computer science or programming class (CS1) remains a worrisome matter for computer science educators (Butler et al., 2016). Students come into computer science programmes with high interest, this enthusiasm in a significant number of them is soon eroded in CS1, and they drop out. This situation begs that the need for computing skills for introductory programmming instruction be made engaging to our teeming students. Butler et al. (2016) take this sentiment further, advocating for computing education that "engages and support students throughout their undergraduate studies" (p.1). In this section, we want to consider these questions: What is engaging computing instruction? What makes a learning session engaging for the diverse population of CS students? Is there empirical evidence to support the case for engaging computing education?

2.5.1 Defining an engaging computer science education.

Defining engaging computing instruction will require considering definitions of the construct of engagement in the literature. Unfortunately, there is no consensus on the conceptualisation, forms, and measurement of student engagement (Bond et al., 2020; Butler et al., 2016; Eccles & Wang, 2012; Lam et al., 2012; Reschly & Christenson, 2012). Another problem with defining engagement is the conflating or confusing motivation with engagement. One thing applicable to both constructs of motivation and engagement is: authors often present narrow or broad views in their conceptualizations of the two concepts (Eccles & Wang, 2012). For instance, while Skinner and Pitzer (2012) provide a holistic definition of engagement as "energized, directed, and sustained action, or the observable qualities of students' actual interactions with academic tasks"(p. 24), Reschly and Christenson(2012) identified it "as multidimensional, involving aspects of students' emotion, behaviour (participation, academic learning time), and cognition" (p. 3). Similarly, while Reeve (2012)defines engagement narrowly as "the extent of a student's active involvement in a learning activity" (p. 150), Trowler (2010) provides a comprehensive definition saying:" Student engagement is concerned with the interaction between the time, effort and other relevant resources invested by both students and their institutions intended to optimise the student experience and enhance the learning outcomes and development of students and the performance, and reputation of the institution."(p. 3). McCormick & Kinzie (2014) agree with the later perspective by asserting that student engagement "refers to two critical features…... The first is the amount of time and effort students put into their studies and other educationally purposeful activities. The second is how the institution's resources, curricula and other learning opportunities support and promote student experiences that lead to success (e.g., persistence, learning, satisfaction, graduation)." (p. 14). An excellent definition that captures some features relevant to this study defines:

> "*Student engagement is the energy and effort that students employ within their learning community, observable via any number of behavioural, cognitive or affective indicators across a continuum. It is shaped by a range of structural and internal influences, including the complex interplay of relationships, learning activities and the learning environment. The more students are engaged and empowered within their learning community, the more likely they are to channel that energy back into their learning, leading to a range of short and long term outcomes, that can likewise further fuel engagement*" (Bond et al., 2020)

Several facts are clear from these conceptualisations: an engaging pedagogy is observable, as it awakens active participation of the student, and it is positively rewarding for the student, the teacher, and the institution (Reschly & Christenson, 2012). Another thing about engagement is that it can be measured in its general or specific forms. Skinner and Pitzer (2012) summed up these general and specific dimensions by defining the general construct of engagement as "the quality of a student's involvement with school", and the specific aspects as "behavioural, emotional, cognitive, and psychological engagement." (p. 22).

As there is no consensus yet on the general definition of engagement, so there is no consensus on the number of distinguishable features or types of engagement that can be observed or measured in the class(Bond et al., 2020). Let us consider the various conceptualisations of the specific forms of engagement in ***Table 2-10*** and observe how far the authors agree.

Table 2-10 Dimensions of Engagement

Types of engagement	**Source**
Emotion, behaviour and cognition	(Reschly & Christenson, 2012, p. 3)
Behavioural engagement, emotional engagement, and cognitive engagement	(Fredricks et al., 2004, p. 60)
Affective, behavioural, and cognitive dimensions	(Lam et al., 2012, p. 405)
behavioural engagement, emotional engagement, cognitive engagement, and agentic engagement	(Reeve, 2012, p. 150)
Cognitive, affective, behavioural, academic, and social engagement.”	(Parsons & Taylor, 2011, p. 4)
Behavioural, emotional and cognitive engagements	(Lei et al., 2018)
Affective, cognitive, behavioural engagements	(Bond et al., 2020)

From ***Table 2-10*** you will observe that three types of engagements are common: behavioural, affective, and cognitive engagements.

Behavioural engagement refers to what a student does with learning opportunities, facilities and situations which is believed to predict his or her learning outcome in the school. This is seen in the efforts or time a student invests in activities that lead to a meaningful learning experience, or achievement. Such activities include notetaking, attempting class exercises, assignments, or projects, reading, and studying. Affective engagement, which some authors call emotional engagement, refers to the student’s attitude towards learning opportunities or situations. On the positive side, it refers to students’ interest, self-efficacy, sense of belonging or identity with the school, course of study or subject. On the negative side, it refers to traits such as boredom, withdrawal, or lack of attention. Cognitive engagement refers to the mental strategies and understanding the student employs during learning sessions in or out of class. This refers to the student’s learning styles such as deep or surface learning approaches.

Other variants of the above forms of engagements have been mentioned in literature such as agentic engagement, social engagement, collaborative engagement, ongoing engagement, and reaction to challenge.

Agentic engagement refers to student's ability to take control of their learning by employing proactive actions without instruction from teachers (Reeve, 2012). Social engagement is the level of contribution of efforts the students make towards their collective learning. This is like collaborative engagement. Ongoing engagement is a combination of the behavioural, affective and cognitive engagements demonstrated by the student towards his or her studies. Reaction to challenges refers to what the students do, in the face of challenges to learning. While a student with a high level of engagement on this later measure will persist, taking appropriates steps to overcome his or her learning difficulties, another student low on this engagement will seek to delay doing or completely avoid such difficult learning tasks.

2.5.2 Drivers of an engaging computing instruction

Several anecdotal and empirical evidence from literature present elements that make an instruction engaging for students.

Table 2-11 Essential Factors in an engaging class

A conceptualisation of Engagement Factors (or Facilitators)	**Source**
1. Enriching collaboration and educational experiences among peers; 2. student-teacher interaction; 3. levels of academic challenge; 4. supportive classroom environment; 5. supportive family environment.	(DeVito, 2016, p. 3)
1. Meaningful instruction 2. A sense of competence 3. Autonomy support 4. Collaborative learning 5. Positive teacher-student relationships	(Pino-James, 2018), (Pino-James et al., 2019)
1. Real, relevant and interdisciplinary instruction. 2. Rich in appropriate educational technologies 3. Risk-taking encouraged where mistakes are allowed in an atmosphere of open, challenging, and supportive instruction 4. Respectful relationships between students and teachers 5. Resources are focused on learning and mastery first, and achievement second.	(Parsons & Taylor, 2011)
1. Connecting CS to the students' day-to-day activities. 2. Creating open and safe environment for students to express themselves. 3. Confronting social issues of concern to students using CS.	(Ryoo, 2019)

While it could be said from Table 2-11 that authors are re-echoing what previous writers have said about what makes a learning session engaging, it also reveals some agreement in what they believe are key for engaging students. A synthesis of these common factors in literature that can be identified in an engaging computing class are:

- Cordial supportive relationship between teachers and students. (Klem & Connell, 2004; Reschly & Christenson, 2012)
- Collaborative active learning sessions with peers.
- Challenging computing lesson or exercise commensurate with students' capacity or scaffolded to support students' learning
- Contents tailored to learners' interest and needs.
- Commitment to learning, not just grade.
- Curiosity and a can-do attitude towards computing.

There are compelling reasons for making computing instruction engaging particularly for novice computer science students. Some of the reasons are:

- To address the global problem of high failure and dropout rate for the first programming course (CS1)
- Computer science together with other physical sciences has been reported to be less engaging for undergraduates by the National Survey of Student Engagement in the USA (Morgan et al., 2017). While questions have been raised about the validity or reliability of the survey for computer science (Butler et al., 2016), yet, with the global CS1 problem, well known to computer science educators, could it be said that our computing instructions have been engaging enough for the teeming population of CS students that fail or dropped out? In fact, in a Communication of the ACM opinion page, professors Mark Guzdial and Elliot Soloway, admitted and lamented the inadequacy of the then CS1 pedagogy by raising a question that remains pertinent till date: "Why are we doing such a poor job at getting and keeping students in computer science?" (Guzdial & Soloway, 2002, p. 17). It has been suggested that there is a circular relationship between engagement and achievement (Eccles & Wang, 2012). As a kind of engagement feedback loop, this implies engagement increases achievement. And positive improvement in achievement will likely lead to more engagement(Bond et al., 2020).
- In many countries, enrolment data indicate computer science has been less appealing to women being seen to be stereotypically a male field, though the need for innovation that comes from diversity in computing workforce makes it compelling that women enrol into the field.
- Vacancies for skilled computing professionals continue to be left unfilled globally threatening further development in computing.
- There are promising empirical results and "success stories" of computing educational interventions that are engaging. Evidence from literature suggests CS educators found peer instruction and paired programming to be engaging(Hanks et al., 2011; Porter et al., 2016). The story of David Malan, a professor of computer science at Harvard, who following exposure to a CS1 instruction in his first year as an undergraduate switched to computing science, is an excellent example of the power of engaging instruction(Orbey, 2020). When the same CS1 course was becoming less enrolled and the failure rate was concerning, David then a faculty at the same institution, was assigned to redesign that CS1 course, and by employing an engaging pedagogy which included a week of *Scratch* lessons and projects, turned around the course that attracted him into computing, into a popular CS1 course not

just in Harvard but among students even outside the USA(Malan & Leitner, 2007). Another example is the Media Computation introduced by Mark Guzdial at Georgia Institute of Technology for undergraduates in other degree programmes who needed to learn relevant computing concepts (Guzdial, 2015, 2013). Harvey Mudd also turned her CS programme to an engaging course that attracted more women into the field of computer science(Alvarado et al., 2012).

2.5.3 Students' Engagement and Achievement

Evidence from a growing body of literature on engagement, suggests, a positive association exists between students' engagement and learning outcomes such as achievement(Lei et al., 2018; Phuntsho & Dendup, 2021; Schnitzler et al., 2021). While research suggests a less consistent or indirect impact of some types of engagement, for instance, affective engagement (Finn & Zimmer, 2012), most findings present a picture of growing consensus among researchers of the direct impact of students' overall engagement on their achievements.

2.6 *Scratch*

Scratch was launched in May 2007 (Fields et al., 2017; Kafai & Fields, 2018). According to TIOBE (https://www.tiobe.com/tiobe-index/) programming language popularity January 2022 index, *Scratch* retains its position as the most popular block-based programming language.Of the three hundred programming languages monitored by TIOBE, *Scratch* sits on the index as the 23rd most popular language. In the same period, *Scratch* has over 83million registered participants (or Scratcher) on the *Scratch* website, and more than 90million projects shared by users from over 200 countries worldwide. Both the Scratch online and offline editors are available for users to code in 70 world languages.

A product of research by the Lifelong Kindergarten Media Lab at MIT in conjunction with Yasmin Kafai at UCLA, *Scratch* was initially developed for use by young people in informal setups of homes and afterschool computer clubs (Maloney et al., 2008) but has now become the staples of primary and secondary schools (Rich et al., 2019; Szabo et al., 2019). Higher educational institutions now employ some forms of *Scratch* to introduce novice students to programming (Becker, 2019; Cárdenas-Cobo et al., 2021; Hijón-Neira et al., 2021).

The vision of its developers was to create an engaging constructionist programming environment as part of the strategies for increasing participation of women and other minorities in computing, by providing an environment for users to design, create or remix multimedia programs (called scripts) of their interest (Fields et al., 2017; Maloney et al., 2008). That is, to motivate users so that autonomy given to them, inspires creative expressions leading to their computational thinking and programming ability development. That original constructionist philosophy behind *Scratch* is captured by the slogan on the offline as well as online environments – *Imagine, Program, Share.* The aim was to raise youths who are not just consumers but producers of computing technologies. Students or Scratchers can develop animated stories, games and robotic applications with *Scratch*.

Scratch is a visual programming language with program statements usually written as texts, replaced by coding blocks so that a program looks like a jig-saw puzzle, rather than sequence of texts. Earlier versions of *Scratch* include version 1.4 and 2.0. The current version of Scratch is Scratch 3.0 which was released on January 2, 2019. Scratch 3.0 refers to both the offline program and the website providing facilities for collaborating with millions of Scratchers (shown in Figure 2-8 and Figure 2-9)

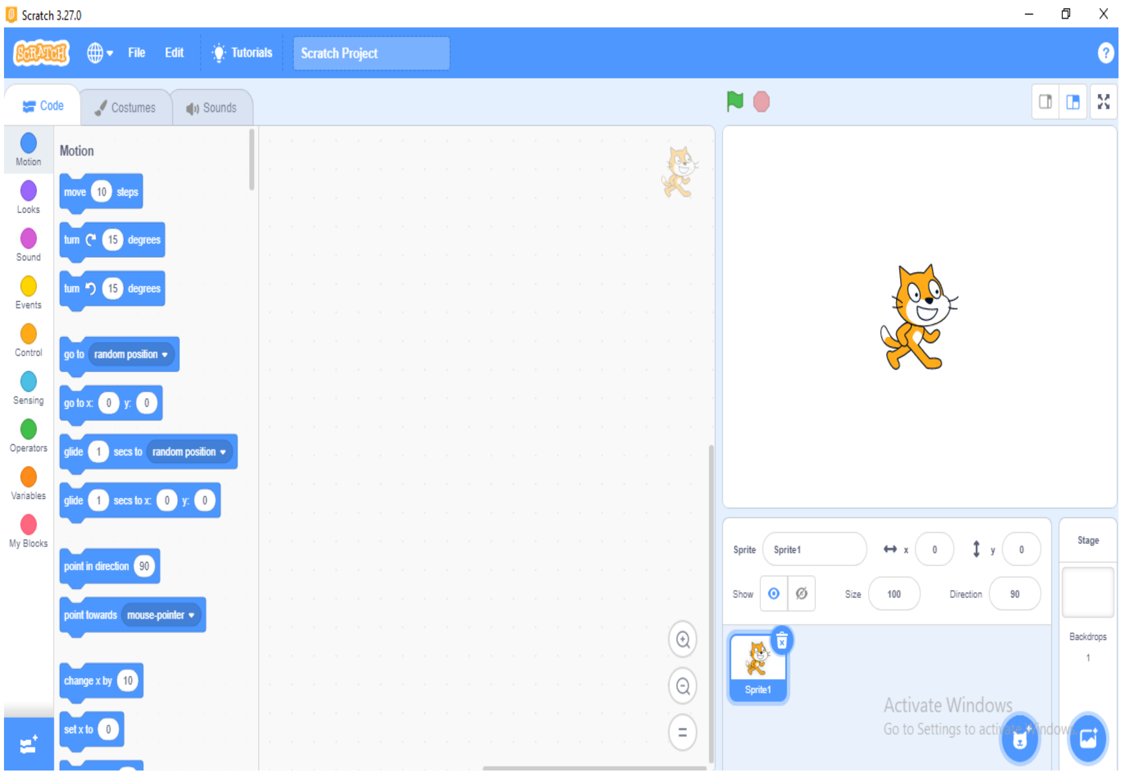


Figure 2-8 Scratch 3.0 environment

- Some parts of Scratch interface include the stage – at the right side of the editor. This is where you see effects of the code you create in the code area.
- Sprite Pane located bottom right under the stage. Where Scratch's objects or characters called sprites are located. A Scratch project contains one or more sprites.
- The code area – located at the center of the interface. This is where you create and edit Scratch codes (called scripts).
- Block pallet – left of the code area. for choosing or creating code blocks. There are nine types of blocks: motion, looks, sounds, events, control, sensing, operators, variables and MyBlocks.
- Paint editor – *Scratch* area for creating and editing your images called costumes.
- Sound editor – For creating and editing sounds for your sprites.

Scratch 3.0 came with features for developing AI programs. Such AI facilities include Translate, Text-to-Speech, Micro:bit, and Lego Mindstorms EV3(Merino et al., 2021)

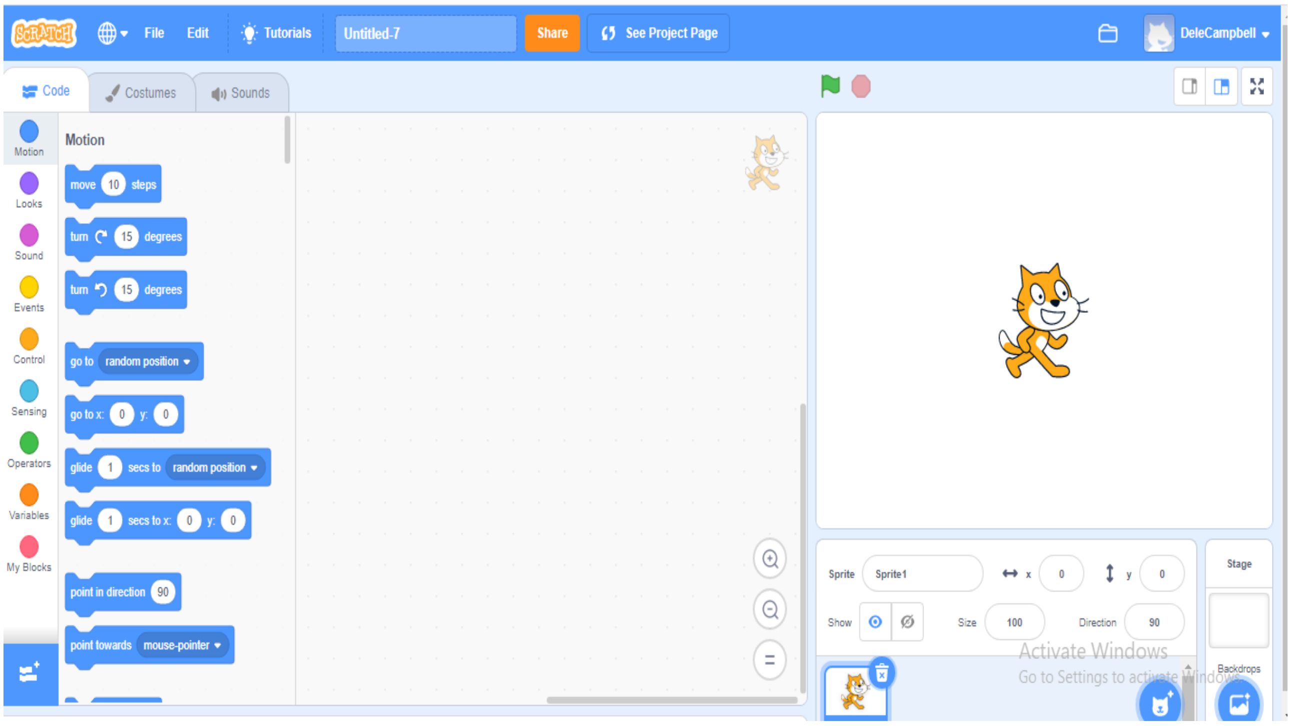


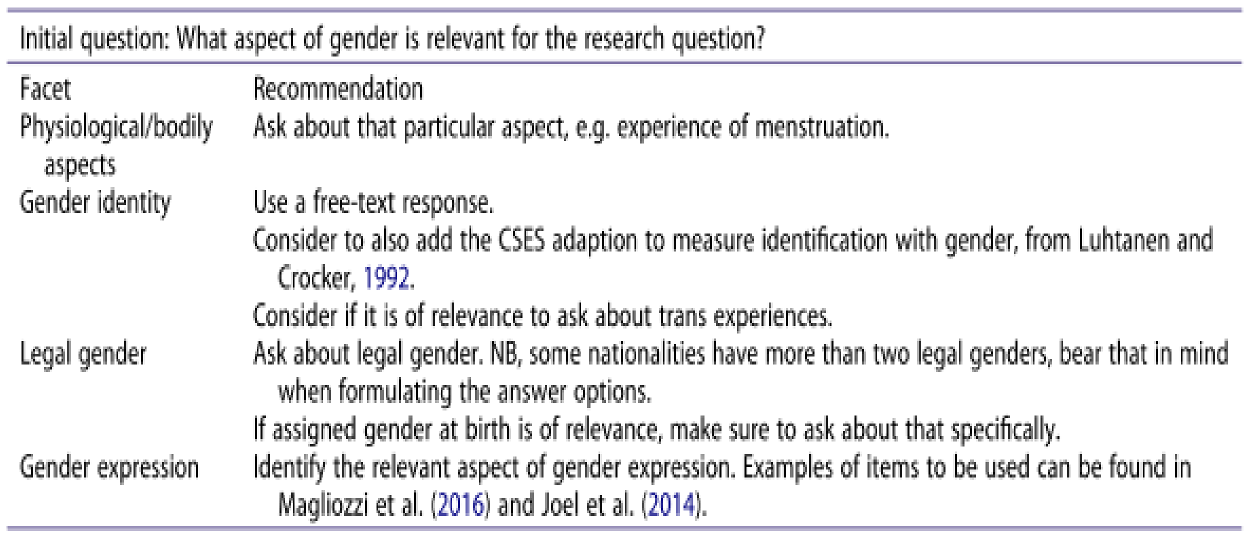
Figure 2-9 The Scratch *online programming environment*

2.7 Concept of gender

John Money an American psychologist was credited for "inventing" or popularizing the term gender (Goldie, 2014). Money's idea of gender is reflected by this statement he made in 1955: "By the term, gender role, we mean all those things that a person says or does to disclose himself or herself as having the status of boy or man, girl or woman, respectively. "as cited in (Goldie, 2014, p. 45). This implies gender is a self-chosen identity by a person to declare their masculinity or femininity. However, this gender identity may also be influenced by society. For instance, World Health Organisation (Organisation, 2018) defines gender as "the characteristics of women, men, girls and boys that are socially constructed. This includes norms, behaviours and roles associated with being a woman, man, girl or boy, as well as relationships with each other. As a social construct, gender varies from society to society and can change over time." WHO definition reveals another idea about this concept of gender: its malleability. That means a person may be born a boy and opt to change their gender identity to a girl. However, there is growing call for departure from such binary or dichotomous grouping of gender as there are many that identify themselves neither as male nor female today(Lindqvist et al., 2021; Rushton et al., 2019). To allow for this, as in the questionnaires used in this study, participants are given the opportunity to self-select their gender. It has been argued such allowance of other ways of operationalizing gender will ensure more accurate data and research results (Rushton et al., 2019).

Gender variable is included in most studies as it is believed to influence certain dependent variables such as attitudes, behaviours, programming achievement, among others(Lindqvist et al., 2021). However, operationalizing this concept with the varied and growing gender identities in research poses a problem. To address this, Lindqvist et al. (2021) argues for conceptualizing gender as being made of four aspects: "physiological/ bodily aspects (sex)"; "gender identity or self-defined gender"; "legal gender"; and "social gender in terms of norm-related behaviours and gender expressions". They recommend as shown in Table 2-12 that researchers should look at their research questions to identify which aspect is relevant to the study, to ask the participants in the questionnaires.

Table 2-12: Suggested guideline for operationalising gender variable



Initial question: What aspect of gender is relevant for the research question?

Facet	Recommendation
Physiological/bodily aspects	Ask about that particular aspect, e.g. experience of menstruation.
Gender identity	Use a free-text response. Consider to also add the CSES adaption to measure identification with gender, from Luhtanen and Crocker, 1992. Consider if it is of relevance to ask about trans experiences.
Legal gender	Ask about legal gender. NB, some nationalities have more than two legal genders, bear that in mind when formulating the answer options. If assigned gender at birth is of relevance, make sure to ask about that specifically.
Gender expression	Identify the relevant aspect of gender expression. Examples of items to be used can be found in Magliozzi et al. (2016) and Joel et al. (2014).

Source:(Lindqvist et al., 2021, p. 11)

Research results on the relationship between gender and CS1 performance is mixed. While some findings indicated that gender have effect on CS1 student achievement (Quille & Bergin, 2019); other found no effect(Gjelsten et al., 2021; Lishinski & Rosenberg, 2021; Veerasamy et al., 2019). Gjelsten et al.(2021) identified the reason for disappearance of gendered difference in CS1 achievement: prior programming experience in high school of female CS1 students. That suggests prior programming experience is a mediating variable between gender and CS1 achievement.

Nevertheless, some gender differences have been identified in CS1 classes. For instances, CS1 assignments focusing on people rather than on things, were found to be preferable to female CS1 students (Marcher et al., 2021). This suggests teachers will give room for flexibility in class exercises or projects assigned to students so that the interest of females who prefer people themes will be supported. Doing so will likely increase interest, self-efficacy, retention and the final outcome – variables that often have been found to be skewed against them (Beyer, 2014; Lehman et al., 2016).

2.8 Concept of age

Several definitions for age exist, depending on the perception or emphasis of the author. In this study, our focus is on the lifespan and developmental stage of first-year CS students as it affects their achievement in programming. Therefore, in this section we will consider definitions or descriptions relating to that. OECD glossary of term defines age as "the interval of time between the day, month and year of birth, and the day and year of occurrence of the event expressed in the largest completed unit of solar time such as years for adults and children and months, weeks, days, hours or minutes of life, as appropriate, for infants under one year of age"(*OECD Gloss. Stat. Terms*, 2008). That is what is popularly called chronological age. Russo and Brian

(2012) go beyond that to define age as “a state of mind, however the physical length of time one inhabits the earth and is alive” Two things are stated in this latter definition: reference to psychological age – “a state of mind” - and chronological age – length of existence. Another popular classification is biological age. While chronological age is the actual life span from the time of birth looking at the record of birth, biological age refers to the perceived age of an individual looking at the growth or decline in the body as indicated by its DNA markers (Jazwinski & Kim, 2019). Our focus in this study is the chronological age, though we hypothesize that CS1 students’ achievement may be moderated or mediated by their psychological age. This is apparent from the theoretical explanations by various theories of human cognitive or psychological development.

Although not without criticisms, Piaget (2008) provides some of the most popular theoretical explanations for age-related human cognitive performances from birth to adolescence. According to Jean Piaget, human cognitive developmental stages from birth to adolescence include sensorimotor, preoperational, concrete operational, and formal hypothetical reasoning abilities. The sensorimotor stage is the reasoning ability from birth before a child learns to speak, that is from age 0 to about 2years. The preoperational stage goes from age 2 to 6 or 7 years. The concrete operational stage in which a person can reason only with concrete objects, goes from age 7 or 8 to 11 or 12. The formal hypothetical, a stage where a person can think with abstract objects, forms and tests hypotheses, goes from age 11 or 12 to age 14 or 15. Although this succession of cognitive growth appears to be constant in most societies, there are variations in the speed due to environmental factors in which an individual grows (Piaget, 2008). This suggests why some individuals reach the formal reasoning level at about age 14 or 15, others, retarded by some factors, may or may not reach it at age 15 to 20. He also identified varied aptitudes or area of interests as other factors that may differentiate how individuals demonstrate maturity via cognitive abilities. In summary, Piaget's provides three reasons for differentiated performances of individuals in various fields, programming inclusive: natural process of cognitive development, talents, and the individual’s domain of study, training, or interest.

In line with speed of human development being witnessed in modern times, Sawyer(2018) in an article advocating for new designation of the age of adolescence, provides some age-related nomenclatures to classify human developmental stages. In Figure 2-10, we observe various classifications under the three human developmental stages of childhood, adolescence, and adulthood. In many countries, from school-aged children in K-12 to senior citizens in retirement, these various groups are being introduced to programming education in formal and informal settings, though problem of access or digital divide is huge.

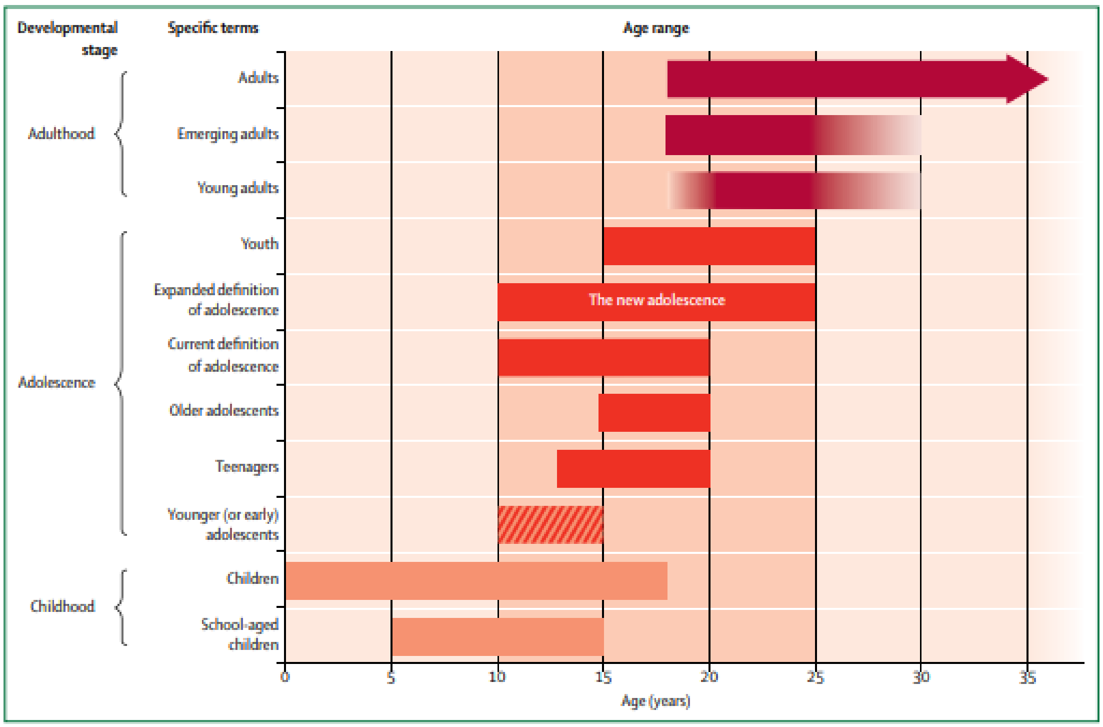


Figure 2-10: A characterisation of age and human developmental stages

Source: (Sawyer et al., 2018)

Evidence shows that not all ages of learners receives programming instruction equally well (Kong et al., 2018; Lambić et al., 2020). In other words, the age of a learner, among other variables, may influence reception of instruction or the outcome (Quille & Bergin, 2019).

Kong et al (2018) found age-related differences in reception towards programming instruction among323 Hong Kong primary school children in grades 4-6 who participated in the study. They found a direct link between interest or positive attitude, perception of meaningfulness, the impact of instruction and self- efficacy. However, they found older students showing less positive attitude to programming than their younger ones. This probably suggests age may not be the only variable influencing attitude towards programming. Almost half of those in study had prior experience. So prior experience may be a factor too.

Lambić et al.(2020) investigated the effects of a course appropriate for grades 1-4 on the popular site Code.org on children's attitude towards programming. The study involved 293 Serbian children between age 7-10 years in grades 1-4. In contrast to Kong et al (2018), they found that the older ones (age 9-10) had more positive attitude and solved more problem than the younger ones (age 7-8). This indicates programming attitude may invariably affect achievements. Tailoring programming instructions and environments to appropriate ages of the learners becomes necessary to achieve positive educational outcomes.

In an earlier investigation on age-related differences and programming knowledge, Morrison & Murphy-Hill (2013) conducted a study of members on the StackOverflow site, exploring the relationship between programmers' age and programming knowledge. They found a direct correlation between programmers' age and their programming knowledge, at least well into their 50s.

In a similar study, Kock et al (2018) explored the hypothesis that older programmers are less effective than younger ones, invariably expecting a negative effect of age on programming performance. University students (n=140) with varying degree of prior programming experience were given a software to develop within a specified time. Following this experiment with students, they conducted a simulation to see what the results will look like, if the participants were professional programmers. Corroborating earlier study by Morrison & Murphy-Hill (2013), they found a positive effect of age on programming experience and perception of stress. They also found programmers' experience is directly related to their performance. They found out that stress had a negative effect on performance. However, regardless of age, as programming experience increases, they found stress loses its negative effect on performance.

A profile of ages of Scratchers (i.e., participants on the *Scratch* online) as of January 2022 is given in Figure 2-11.

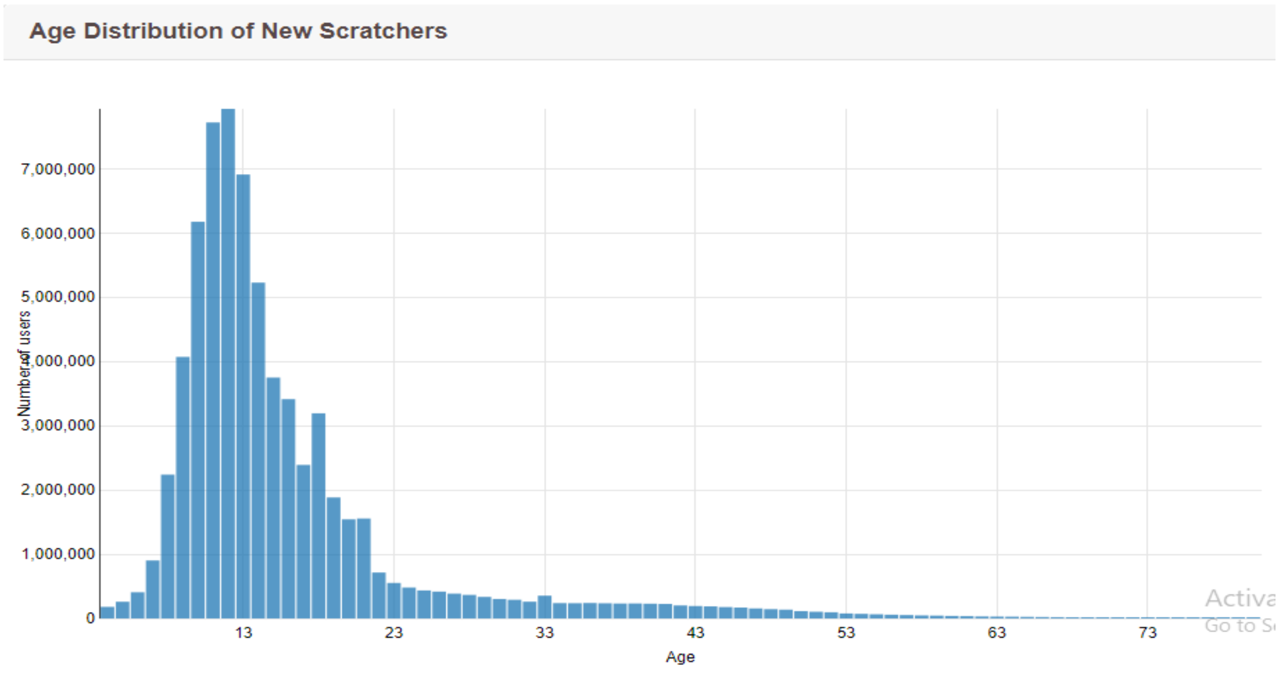


Figure 2-11: Age of Online Scratch Users

Source: (*Scratch - Imagine, Program, Share.*, 2022)

You will observe a mix of age from as young as 4, to as old as 80.

Does age matter in CS1? Quille and Bergin (2019) found that including age as a factor increase the accuracy of a CS1 predictive model, suggesting that age has a positive relation to CS1 performance.

2.9 Concept of academic background

By academic background, we mean the prior achievement level attained by a student in the K-12 education or in a previous course. Students prior academic achievement belongs to one of those cognitive and non-cognitive variables (like self-efficacy, gender and Socio-Economic Status) that are believed to influence their academic achievements in higher ed (Al-Sheeb et al., 2019; Asarta & Schmidt, 2017; Parker et al., 2018; Van den Broeck et al., 2019). The need to investigate the effect of prior academic achievement of CS1 students, hinges on the evidence from past research that, it can predict other educational outcome variables(Ramos, 2018). It has been found that prior academic achievement influences students' motivation, which invariably influences their engagement with new learning situations (Rodríguez et al., 2019). Students' engagement in learning directly relates to their future achievements(Ferrer et al., 2020). Self-belief theory explains this phenomenon. Students who have prior positive achievement, are likely to believe they will learn and do well in later studies (Schöber et al., 2018). This belief in their ability to learn or perform certain skills is called self-efficacy, and has been found to have positive effect on students' academic achievements or programming performance(Al-Sheeb et al., 2019; Lishinski & Rosenberg, 2021; Quille & Bergin, 2019)

Educators and researchers employ various variables to operationalise students' prior academic achievement. Table 2-13 presents some examples.

Table 2-13: Operationalising students' prior academic achievement

Operational variable(s)	**Levels**	**Source**
One variable derived from: prior average academic grades in Spanish, math and foreign language (English)	Five levels: from 1 to 5 (1 = insufficient, 2 = sufficient, 3 = good, 4 = notable, 5 = outstanding).	(Rodríguez et al., 2019, p. 4)
One variable derived from primary schools' national examination scores	2 levels: higher academic achievement (HA) and lower academic achievement (LA) students	(Prayitno et al., 2017, p. 268)
Three variables were used: GPA, a math quiz score and grade in a calculus class	Three levels: - Using GPA: Below 2.80, 2.80-3.34, 3.35 – 4.00 - Using math quiz: Below 9, 9-12, 13-16 - Using Calculus grade: Below 7, 7-9, 10-13	(Asarta & Schmidt, 2017, p. 13)
One variable: high School grade	None	(Al-Sheeb et al., 2019)
One variable: High mathematical exit grade	None	(Quille & Bergin, 2019)
One variable: University entrance exam score (Stanine)	Nine levels: Using Stanine score: Level 1 to 9 (From lowest to highest)	(Ramos, 2018)

Does prior academic achievement matter in CS1 students' performance? Studies have found correlation or causation between prior academic achievement and CS1 final grade. For instance, Ramos(2018) in a study involving Filipino university computer science students found a correlation between students entrance exam (their stanine level) and their CS1 grades.

2.10 Concept of prior programming writing

Increasingly, CS1 classes are known to comprise students with various levels of prior programming experiences (Mohamed, 2021; Rahman, 2020). Possible reasons for this diverse CS1 students' abilities may be the nature of K-12 computer science education in a particular country, informal programming learning opportunities available, the department the students belong, computer science education advocacies ongoing in the country, previous employment, and the demand for computing jobs leading to the need to enrol or switch from one career to another. In developing nations especially, students come into CS1 classes with the larger percentage having little or no background in programming (Agapito & Rodrigo, 2018).

While there is more evidence suggesting that having prior programming experience births better CS1 achievement, regardless of the nature of programming paradigm or environments the students are exposed to (Alvarado et al., 2018; Armoni et al., 2015; Chen et al., 2019; Gjelsten et al., 2021; Holden & Weeden, 2004; Liao et al., 2021; Veerasamy et al., 2018; Wilcox & Lionelle, 2018), yet there are studies that found no evidence (Ventura, 2005). Also, Alvarado et al (2018), despite finding evidence for final CS1 course grade, found that prior programming experience makes no difference in their pre-test scores. Another study by Marling & Juedes (2016) found that students without prior programming experience obtained significantly better grades in CS1 than those with prior experience, after both groups had undergone a CS0 course. That implies the CS0 probably helped those with no prior experience to catch up with their experienced counterparts. This makes it interesting to investigate whether students' prior programming experiences in this study affect their CS1 achievements.

Table 2-14 presents various ways for characterisingCS1 students prior programming levels.

Table 2-14 Defining levels of CS1 students' prior programming experience

Grouping by	Levels	Source
Using a formula that produces an experience index from a series of previous educational or work experiences stated in survey responses.	No experience, Minimal experience, Medium experience, Very experienced	(Holden & Weeden, 2004)
Students' self-selection	Prior Programming experience (P), No programming experience(N)	(Wilcox & Lionelle, 2018)
Student self-assessed responses to survey questions	A fair amount, A little, Absolutely none Formal experiences Informal experiences	(Alvarado et al., 2018)
Calculated an index (0-2) from Students' survey responses	No knowledge (0) Basic knowledge (1) Good Knowledge (2)	(Veerasamy et al., 2018)
Derived from student self-reporting of whether they have written program in any language	No experience (0) Has experience (1)	(Gjelsten et al., 2021)

2.11 Concept of prior visual art

Oxford English online dictionary, defines visual arts as "creative art whose products are to be appreciated by sight, such as painting, sculpture, and film-making (as contrasted with literature and music)".Esaak(2019) echoes the same characterisation by defining visual arts as "those creations that we can see rather than something like the auditory arts, which we hear. ….. [they] include mediums such as drawing, painting, sculpture, architecture, photography, film, and printmaking". Similarly, Encyclopaedia Britannica describes visual art as "a visual object or experience consciously created through an expression of skill or imagination." (Britannica, 2020). We deduce from these definitions that visual arts are forms of art distinguished by their visual appeals. To create such works requires the author to employ cognitive skill or talent.

In this study, by prior visual art we mean past creative artistic portfolios of the students. That is, ways that students have expressed their creative abilities by producing visual works of art with or without the computer. Research has often explored the impact of visual or creative arts activities on computational thinking ability of novices (Kafai et al., 2019). In this study, I am interested in the impact of the visual art backgrounds CS1 students bring into the class on their computational thinking or programming ability. The aim is to explore if innate visual or creative artistic ability relate to their CS1 programming ability.

Donald Knuth in his 1974 Turing award lecture titled '*Computer programming as an art'*, referring to medieval meaning of the word, defines art as "something devised by man's intellect, as opposed to activities derived from nature or instinct"(Knuth, 2007). Drawing a comparison between programming and creative art, Knuth further asserts "when we prepare a program, it can be like composing poetry or music". That suggests both programming and artistic composition have some things in common, one of which is creativity.

The question is, what is creativity or creative ability? While there is no consensus on defining this construct (Said-Metwaly et al., 2017), there are several definitions, each emphasising one, two or several essential properties of creativity. For instance, providing a one-criteria definition, Vygotsky (2004) defines it as "Any human act that gives rise to something new is referred to as a creative act, regardless of whether what is created is a physical object or some mental or emotional construct that lives within the person who created it and is known only to him"(pg). Runco and Jaeger (2012), after extensive review of literature from the beginning of the twentieth century, identified a "standard definition" that says creativity is the ability of an individual to produce something which others judge as original and useful. Other definitions mention three criteria for counting an idea, product, or response as expression of the creativity of its author. For instance, Simonton(2017) reiterating a definition in line with the three criteria employed by the US patent office for judging an invention as creative, he defined creativity as being a product of an idea that is original, useful and surprising.

Is there a relationship between creative artistic ability of a student and their achievement in CS1? That question appears to have found limited attention in the CSE research literature (Sharmin, 2021). That is one secondary question this study is investigating. However, evidence from the few studies suggests there is no significant correlation between students' creative achievement and their CS1 achievement (Gestwicki & Ahmad, 2010; Sharmin, 2021)

2.12 Theoretical Framework:

This section presents a review of some of the theoretical underpinnings for this study, namely constructivism, constructionism, and Keller's ARCS model.

2.12.1 Constructivism Theory

What does constructivism entail? In the midst of ongoing discourse, claims and counterclaims, nuances, jargons or hear-says surrounding the term constructivism, defining it becomes a challenge (Taylor, 2015) and care is needed to avoid contributing to the confusions around this ubiquitous theory, that is influencing various fields today. Raskin(2002) alluded to this confusion when he remarked, "One comes across so many varieties of constructivist psychology that even the experts seem befuddled."(pg. 1). Some of these popular variants of constructivism include cognitive constructivism attributed to Jean Piaget(Powell & Kalina, 2009); radical constructivism by Ernst von Glasersfeld; and social constructivism by Leo Vygotsky to mention a few. In this section we want to define some of these variants, outline a brief history of constructivism, what critics say about it, as well as how this theory relate to CS in general and CS1. Let us examine a few definitions of constructivism.

Cognitive constructivism says knowledge is constructed in a person following a process of assimilation (i.e., adding knowledge to existing knowledge structure (schema) and accommodation (replacing an existing schema with new ideas based on prior experiences, leading to shifts in the cognitive structures of the individual). This emphasizes the individual.

Not satisfied by this Jean Piaget's view of constructivist philosophy, earlier characterisation from the horse's mouth defines radical constructivism as: "… an unconventional approach to the problems of knowledge and knowing. It starts from the assumption that knowledge, no matter how it be defined, is in the heads of persons, and that the thinking subject has no alternative but to construct what he or she knows on the basis of his or her own experience." (Glasersfeld, 1996, p. 1)

Glasersfeld (2007) goes on to present these seven fundamental principles of radical constructivism:

1. Knowledge is what we experience. We experience knowledge. We know from our experiences. Our experiences influence what we know.
2. There is knowledge beyond human reach.
3. However, existence is not knowable except it is experienced.
4. Knowledge is constructed.
5. Human knowledge is subjective, not objective. It is not equal to objective reality.
6. Even a viable knowledge or solution is only a construction, there are other alternative viable constructions.

7. Radical constructivism is only a model of knowing, it is not the only way of explaining how knowledge or learning happens. It is only viable in contexts where it is fit for purpose.

While sharing some of the previous constructivist views, Leo Vygotsky and others, addressed what they saw as gap in earlier constructivist theories: the missing emphasis of the social input to the individual construction of knowledge. That is why their brand is called social constructivism.

While some have attributed constructivism to the theory of cognitive development by Jean Piaget(Ben-Ari, 1998; Malik & Coldwell-Neilson, 2018), others argued that the origin, dates back to the times before the great Greek teacher Socrates (Glasersfeld, 1996, 2007; Llanas, 2018). Attributing earliest articulation of constructivism to the writing of Vico Giambattista, an Italian eighteenth century philosopher, Glasersfeld (1996) remarked that Vico's treatise "which, as far as I know, is a first explicit formulation of constructivism"(pg.6). In time, this constructivist idea that knowledge is what an individual construct within themselves depending on their prior experiences spread from one generation of scholars and thinkers to another, and since the second half of the twentieth century has become a dominant theory. Apart from Vico, most prominent of mention is made of George Berkeley, and Jean Piaget as contributors to the modern idea of constructivism (For comprehensive review of the historical roots of constructivism, see (Glasersfeld, 2007; Llanas, 2018)

Despite widespread adoption of constructivism, it has not been without criticisms. Taylor(2015) articulated these three criticisms:

- Constructivism is tantamount to discovery learning.

  Empirical evidence suggests discovery learning is less efficient compared to direct instruction (Kirschner et al., 2006). The constructivists provide a rebuttal by saying critics have wrongly equated constructivism which is a theory of learning to a theory of pedagogy. In one of the most cited criticisms by Kirschner et al (2006), they admit that "the constructivist description of learning is accurate, but the instructional consequences suggested by constructivists do not necessarily follow."(pg. 78) Kirschner et al (2006) also concluded that "The advantage of guidance [i.e.. direct instruction over discovery learning] begins to recede only when learners have sufficiently high prior knowledge to provide "internal" guidance" (pg.75). That prior knowledge is one of core principles advocated by constructivism. That suggests that, the debates on the value of direct instruction or constructivist learning is probably needless after all. Each is suited to a particular purpose or context. In fact, following empirical evidence of the impact of both modes of instruction, some authors advocate for employing both approaches giving consideration to contexts (Kulasegaram et al., 2018; Margulieux et al., 2021; Stott, 2018)

- Majoring on minor by positing that student learning is based on prior knowledge. Critics say most people agree that prior knowledge is important for learners. The constructivists react by stressing the point that learners' prior knowledge or experience makes it essential for teachers to tailor their teaching to their backgrounds.

- Just a theory of learning, lacking theoretical explanations or prescription for pedagogy(Kirschner et al., 2006)

Ben-Ari(1998) identified these other criticisms:

- Undue emphasis of radical constructivism makes her supporters solipsistic, nursing the belief that nothing is real or can be known apart from self.
- Extreme social constructivist ideology can lead to belief that scientific knowledge can only be acquired by powerful privileged few.
- Emphasis on fallibility of knowledge means "truth" depends on an individual.

2.12.2 Constructionism

Constructionism is a constructivist philosophy of education propounded by the South African-born American mathematician and computer scientist, Seymour Papert (Ellison, 2021). Papert who had worked with Jean Piaget, extended Piaget's brand of constructivism by positing that students not only construct their own knowledge, but they do so as they develop artifacts of their interests in collaboration with their peers. That is why, it is known as theory of learning by doing. This theory is suitable to this study because it explains how providing students with a collaborative programming environment, and experiences for purposes of discovery and communication of programming knowledge, can be the key to their engagement and learning, which in the long run can enhance their programming performance. *Scratch* is a constructionist programming language. The inventors of this novice programming language captures some of the essence of the constructionist learning in the motto of *Scratch*: "Imagine, Program, Share". Rob and Rob (2018) re-echoed this claim of the constructionist educational philosophy stating "Thus at the heart of constructionism lies the belief that learning occurs in the process of creating a product that can be shared." [emphasis in the original] (pg. 5)

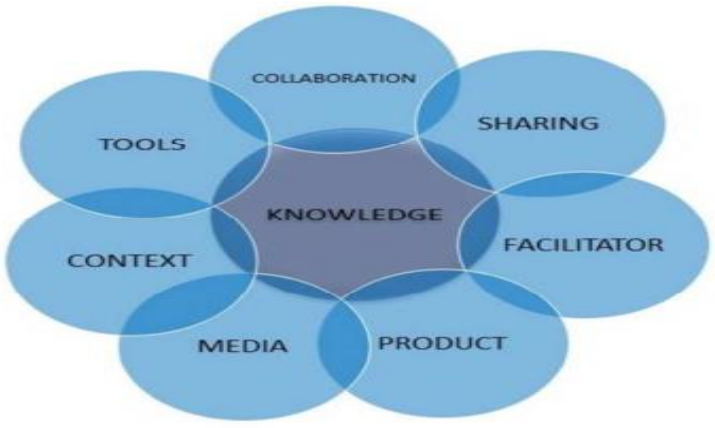


Figure 2-12 Learning dimensions in a constructionist programming class

Source: (Rob & Rob, 2018, p. 7)

To clarify what really constitute a constructionist program learning instruction, Rob and Rob (2018) provides an excellent characterisation.

The aim of constructionist learning for students is to develop from creative exercises concrete as well as abstract programming knowledge. Important components of such a constructionist novice programming class that contribute to these types of knowledge according to as indicated in Figure 2-12 includes:

Facilitator: the teacher acts as a facilitator or a coach for the students by providing authentic real-life challenging learning goals for them. Here, the teacher does not spoon-feed the students with knowledge, the best he or she does, is to provide the environment or impetus for learning through guided discovery.

Context: As stated earlier, learning happens in authentic context where students are encouraged to learn, and are motivated to find answers by developing artefacts or solution to real life problems that are of interest to them. Here, students are not just made to work on teacher-assigned exercises.

Collaboration: Often the constructionist class is characterised by students working in groups and sharing their solutions or artifacts with one another. Here due to the freedom for self-expression of individual student's creative ability, the tendency towards code plagiarism does not exist or is minimised.

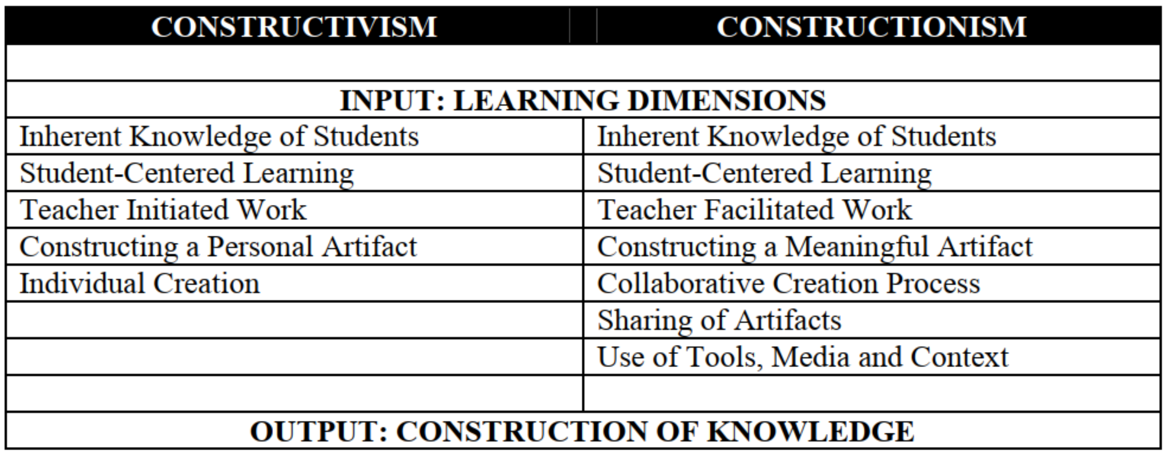
Tools: From its earliest beginning, constructionism had advocated the use of tools to engender students' creative expression. One of such constructionist projects was One Laptop per Child by Seymour Papert and his collaborators. Other tools include constructionist programming languages such as Logo, *Scratch*, and dozens of others.

Product: In a constructionist programming class, the products are programs developed by the students. These products encourage learning as students learn from each other's code as they explore them. In the *Scratch* community, this is called remixing. Each project submitted on the *Scratch* site are available for others to explore and extend.

Media: This refers to platforms for instruction. It could be the blackboard, electronic whiteboard, and online learning.

With the attendant confusions in the literature surrounding the two theories of constructivism and constructionism (Guzdial, 1997), Rob and Rob (2018) present the table to characterise the similarities and differences between the two theories.

Table 2-15 Compare and Contrast: Constructivism and Constructionism



CONSTRUCTIVISM	CONSTRUCTIONISM
INPUT: LEARNING DIMENSIONS	
Inherent Knowledge of Students	Inherent Knowledge of Students
Student-Centered Learning	Student-Centered Learning
Teacher Initiated Work	Teacher Facilitated Work
Constructing a Personal Artifact	Constructing a Meaningful Artifact
Individual Creation	Collaborative Creation Process
	Sharing of Artifacts
	Use of Tools, Media and Context
OUTPUT: CONSTRUCTION OF KNOWLEDGE	

Source: (Rob & Rob, 2018, p. 7)

Brennan (2015) who had been involved with research on *Scratch* since its launch in 2007 as an MIT graduate student, provides another excellent characterisation of the core features in a constructionist class. She identified these four activities:

- **Designing**: students are engaged using their faculties to create objects, develop solutions, unravel puzzles, and problem-solving.
- **Personalizing**: in contrast to the conventional class, the constructionist class devotes attention to the individual. Students experience an atmosphere of freedom or autonomy that encourages individual expressions. The theoretical framework for this activity comes from constructivism. The constructivist emphasizes that learning takes place in the individual students as they engage in the process of sense-making.
- **Sharing**: Students share ideas with teachers and fellow students. They share artefacts with colleagues with the opportunities of getting feedback or taking their ideas or creations to higher levels beyond their initial conception. Rather than struggling to contain plagiarism, remixing of programming codes is encouraged. The theoretical foundation for this activity includes social constructivism by Lev Vygotsky, situated learning and community of practice
- **Reflecting**:

2.13 Empirical Review

2.13.1 CS1 learning outcomes: Block-based Versus Textual programming

Two classes of programming platforms used by CS1 students are blocks-based (or visual) and text-based (or textual) programming environments. Common examples of blocks-based environments include *Scratch*, Alice, Blockly, Snap! etc., while popular textual programming environments are C++, Java and Python to name a few. While programming in a textual environment involves composing texts that are syntactically meaningful together to form logical instructions, programming in the blocks-based environment involves snapping graphic blocks together like jigsaw puzzles. In the former, students are often confronted with syntax errors in their codes, while in the latter syntax errors are eliminated, though semantic errors are still possible. In this section, we will consider empirical studies involving *Scratch* (or Blocks-based) programming and text-based programming environments and their impacts on CS1 students' learning outcomes.

Rizvi and Humphries (2012) presented a *Scratch* intervention course (CS0) taken before CS1 by at-risk first-year computer science students in a US university. The aim was to examine its effect on students' attitude to programming, performance in CS1 and their retention in computer science. It was a mixed-method design, involving quantitative and qualitative methods. They conducted the study for two sessions (2009 & 2010) and study sample include: 2009 - treatment group (n=35), control group (n=42); 2010 - treatment group (n=29), control group (n=15). The treatment group were the students with the weak mathematics background in CS0, while the control group were first-year students in CS1. Students in the CS1 class learnt C++. Anecdotal evidence from the focus group conducted in 2010 with the 2009 treatment group revealed that *Scratch* had a positive effect on their attitude and self-efficacy in CS1. Results from the CS1 performance data showed that the treatment groups in the two sessions had higher pass rates compared to the control groups. They also found an improvement in the retention of at-risk students in the two sessions compared to the session before introducing the intervention.

Erol and Kurt (2017) explored the impact of *Scratch* on the motivation and achievement of first-year Turkish university students in an introductory programming course. The study was a pretest-posttest quasi-experimental design involving a convenience sample of 52 students randomly assigned to the treatment and control groups. Data instruments used include an adapted MLSQ for measuring motivation and an achievement test for measuring programming knowledge. Students learnt *Scratch* programming and flowcharting in the treatment and control groups respectively for seven weeks, then they learnt C# programming in a combined class for another seven weeks. Students took a pre-test at the beginning of the first seven-week for both motivation and achievement in programming in both groups, and at the end, a post-test both groups. They took the second post-test for both outcomes at the end of the combined C# class. They analysed data using mean, Standard Deviation, paired sample t-test, independent samples t-test, ANOVA, and MANOVA. The result suggests that though both groups were not significantly different in motivation and programming knowledge at the beginning; they were significant differences in both outcome variables, at the middle and end of the course in favour of the treatment group. They found that while motivation increased in the middle and end of the course, in the treatment group; it plunges at the middle, and increased slightly at the end, in the control group. However, both groups' programming knowledge increased in the middle and at the end, though the mean achievement of the treatment was significantly higher compared to the control group on both occasions. One limitation of the study is failing to address the possibility of results being confounded by initial differences in covariates. MANCOVA will probably have been used to address this threat. Another problem with this study is, the control group were not given perhaps another form of programming, for instance, textual programming where students can run some codes like their *Scratch* counterparts and receive feedback. Perhaps this was the reason for the lower motivation and performance of those in the control group.

Topalli and Cagiltay (2018) in a longitudinal study of undergraduate computing engineering students, from a Turkish university, explores the effects of enriching traditional CS1 course with *Scratch* games developments. The study investigated the impact of this intervention on the performance of students in the CS1 course, their performance in the final year projects and their overall performance at the end of their studies. A quasi-experimental design was employed involving 322 students, who the researchers followed for 4 years from the freshman CS1 course to the senior year group project course. The sample was divided into the 48 students who took the enriched *Scratch* CS1 course (treatment) and 274 students who took the traditional CS1 (Control). The outcome variables were their CS1 grades, final year project grades and their overall CGPA. Study's data were analysed using mean, SD and independent samples t-test. Though no pretesting of students' prior knowledge was taken, the t-test analysis of the students CGPA at the beginning of the indicated that both groups were similar in their academic ability. The independent-sample t-test of the students’ course grades revealed that the treatment had significantly higher mean achievement compared to the control. They also found similar results in favour of the treatment in the senior year project and the overall CGPA. However, due to some threats to the validity of this study, these results are at best suggestive. For instance, there was no pretesting of participants to know if both groups had similar baseline programming knowledge at the beginning of the study. Organismic and environmental factors like differential learning contexts during their 4-year programme and socio-economic backgrounds may also confound the results of the study.

Chen, Haduong, Brennan, Sonnert and Sadler(2019), following a different approach to previous studies, conducted a retrospective study to explore the impacts of programming environments with which students were initiated into programming and the age of introduction on their attitude towards CS and achievement in CS1. The study employed a stratified random to realise a representative sample of students from 2- or 4-year colleges in the US. However, from the sample of 10,203 students (from 118 institutions) that provided complete research data, optimal matching was used to obtain 3 matched samples of those with graphic, textual and none programming backgrounds, resulting into 2 treatment groups (i.e., Graphic and Textual programming) and 1 control group (i.No programming). Regression analysis of the attitude data revealed that there were significant differences between graphic and control group, and between the textual and control group, in favour of each treatment group. However, there was no significant difference between the two treatment groups. This suggests that learning to write program before CS1 using any of the two modes is better for CS1 students' attitude than none. The regression analysis of the students' background and the CS1 achievement data also showed there were significant treatment effects of textual or graphic programming in comparison to control on CS1 achievement, though with small effects. This indicates that learning to program earlier may likely provide a little leverage for CS1 performance. Comparison of the two treatments initially indicated no significant effect on CS1 achievement, suggesting that any of the two types of programming environments can be equally effective. However, when age was introduced into the regression analysis, a significant main treatment and interaction effects were found, suggesting that the advantage derived from earlier introduction to programming depends on the type of programming environment and the age of introduction. Research results indicated that introducing students between age 6-10 with graphic programming is better than textual programming. They also found no significant difference in treatment effects at age 11-14 and age 15-18 for both programming types, suggesting younger students above age 10 can be introduced to programming using any of the two modes. One limitation of this study is the fact that, other confounding variables after students' initial introduction may also explain differences in their attitudes and achievements in CS1. With no pretesting of students' initial programming knowledge or including this as a covariate in the matching, valuable factor that may influence their CS1 performance has been left out.

Mladenović, Rosić, and Mladenović (2016) conducted an experiment involving Croatian elementary students to compare between those introduced to programming using *Scratch* and those that were first taught in a textual language called Logo. The dependent variables were attitudes towards programming and achievements. The experiment lasted 6 weeks. The study used a mixed-method design involving 23, 7th grade students with 13 students in *Scratch*-first group and 10 in Logo-first. They collected data using two achievement tests (in *Scratch* and Logo), the attitude survey and class observation. While students in *Scratch*-first group learnt to program games in *Scratch* during the first 3 weeks, the Logo-first group learnt to program in Logo environment. Both groups took a midterm post-test containing questions in the languages of instruction and continued learning to program by swapping programming environments in the last 3 weeks. Each group took final post-test with questions in the second programming language. The researchers adjudged both groups to be equivalent based on the results analysis of their performance in math and Croatian language. Also, Shapiro-Wilk's tests showed that the post-test results fulfilled the normality test. t-test analysis of both groups' overall post-test scores showed that there was no significant difference in their achievements. However, while those in *Scratch*-first group performed better in advanced concepts like nested loop compared to the Logo-first, there was no statistically significant difference in the performance of both groups in basic programming concepts. This suggests exposing elementary students, first, to programming using a visual language like *Scratch*, may ease the learning of concepts that are difficult to learn in a textual language like Logo. They also found from the attitude survey that the motivation of the *Scratch*-first group was higher than the Logo-first group. One limitation of this study is the fact that while students in the *Scratch*-first had used the Croatian version of *Scratch*, students in Logo-first had to program using English texts, which the study reported the students struggled with. This initial setback may be the reason for their lower motivation towards programming during the study.

To address a gap on the benefits and drawbacks of the growing use of block-based programming environments, Weintrop and Wilensky (2017) compared the impact of block-based programming over textual programming on the achievement and attitude of high school students in an elective introductory programming course. The study followed a quasi-experimental design in which 60 students from a public high school in a midwestern US city participated. The sample made up of students from the four different grades, were randomly assigned to two classes of 30 participants producing one treatment (block-based) group and one control (Text-based) group. Pencil.cc, an adaption of Pencil code (a free online dual-mode novice programming environment) served as treatment instrument. While the control made use of the text-based interface of Pencil.cc, the treatment used the block-based interface, thereby exposing students to programming using different modes. The study employed an attitudinal survey and an achievement test both taken online as pre-test and post-test. A semi-structured interview protocol was used to collect qualitative data. Several statistics used in the analysis include Mean, paired sample t-test, independent sample t-test, Mann-Whitney-Wilcoxon test, Wilcoxon Signed Rank test, and Wilcoxon Ranked sum. After 5 weeks of both groups learning to programs, the results of the analysis of the achievement test indicates, that both groups were similar in pre-test performance, and demonstrated significant learning in the two modes, the learning gains for the block-based group was significantly higher than the text-based group. The study also found that the block-based group outperformed the text-based in the three modes of questions presented to them as well as in all the concepts asked. However, the significant difference between the groups in the concepts asked, was found only in comprehension questions. The analysis of the data from the attitudinal survey revealed that for ease of learning or using four concepts in the two modes, namely conditionals, iterative logic, functions and variable, the block-based group demonstrated significantly higher positive perception in their learning or use of conditionals and iterative logic. This suggests some programming concepts are easier learned or used by a novice in one mode compared to the other. Unsurprisingly, the study found a significant correlation between this ease of use of concepts and the students' performance in answering questions on those concepts in the achievement test. While overall confidence in programming ability remained unchanged after five weeks of instruction, the post-test confidence for those in the block-based group was significantly higher than at the beginning, adding to a growing body of evidence that block-based programming increases the students' confidence in their ability to program. Similarly, the results of perceived enjoyment of programming indicated that no change between or within the two groups, even, surprisingly, for the block-based group. This seems to contradict the usual narrative around the block-based that students enjoy programming in it. This may suggest that the age of introduction, teacher, classroom environment or other factors other than the programming environments make learning to program enjoyable for students. Another interesting result of the study indicated both groups found programming to be hard, even the block-based that had better performance. This suggests what Seymour Papert calls, "hard fun", that though programming may be fun in a block-based environment, it does not remove the intellectual challenge to be encountered while developing meaningful programs. Unsurprisingly, the study found that while both groups interest for further computer science studies were similarly positive at the beginning, their views were divergent at the end, with the block-based group seeing a significant increase and the text-based group showing a significant decline. On the question of the authenticity of the programming environments (in comparison with professional programming environment like java), while quantitative data indicated that responses in both groups were equally positive and not significantly different, yet qualitative data show some of the students expressed some reservation.

Hu, Chen & Su (2021) conducted a meta-analysis to investigate the effectiveness of block-based programming, and some factors that may moderate its impact on students' achievements. The four factors included are block-based programming tool used (Alice, MIT App Inventor, *Scratch*, and others); nature of the block-based programming intervention (alternating with the traditional class or complementing); educational level ({elementary, middle, high} school, college/university) and geographical location (Asia, Europe, North America, Others). From a search net that involved publications from ACM Digital Library, ERIC, IEEE Xplore, ScienceDirect and Web of Science collection, 1485 initial relevant articles was uncovered, after which series of closer look and filtering led to a sample of 29 articles employed in the study. Effect sizes for each intervention, and their overall fixed and random effect sizes were computed. Other relevant analysis conducted include test of heterogeneity, moderator analysis and test of publication bias. The 29 studies yielded 34 effect sizes of which 22 (64.71%) effect sizes indicated statistically significant positive effects in favour of the block-based environment, 2 (5.88%) effect sizes showed statistically significant negative effect and 10 (29.41) effect sizes indicated no statistically significant effect. The overall effect and heterogeneity test reveal a mean overall fixed effect size of 0.37 (95% CI [0.30, 0.44]); and overall random mean effect size of 0.47 (95% CI [0.32, 0.62]). This suggests that learning to program using block-based programming environment is better compared to a text-based environment, though with small to medium effect on students' achievement. The results from moderator analysis indicated that the four factors have a significant effect on the impact of the block-based environment. This suggests that the impacts of block-based programming on students' achievement depend on some variables like the four selected factors amongst others. For the programming tools used, while Alice has a small to medium significant effect, *Scratch* showed a medium effect, App Inventor indicated no significant effect. This indicates that either *Scratch* or Alice used in CS1 class will impact on students' achievement, though *Scratch* may work better. The nature of the intervention also revealed that studies, where block-based was alternated with traditional pedagogy, had small to medium effect. On the other hand, when block-based programming was used to complement the traditional pedagogy, it had a medium effect. This suggests that enriching CS1 pedagogy with the block-based programming taken by students for a short while before continuing in the traditional text-based programming pedagogy appears to work better than students spending the whole course duration on block-based programming alone. Results for the educational level indicated that use of block-based programming intervention will have a medium to large effect in elementary or middle school, a small to medium effect in high school and small effect in college or university. This further indicates that the level of the impact made using block-based programming declines as students become more mature or knowledgeable. As per location of intervention, Asia had a nearly medium effect, Europe a medium to large effect, and a small effect in North America. This suggests that the impact of block-based programming intervention is not the same in all places. While it works well in some places, it is of little use in others. This study failed to capture the impact of block-based programming in Africa, though the use of such tools is growing on the continent.

2.13.2 CS1 learning outcomes: *Scratch* and gender

The low participation of women in computing in most parts of the world is a well-known phenomenon (Rubio et al., 2015). Interest in computer science appears to wane among girls from elementary schools. Computing without women's involvement in the creation of knowledge and technologies is increasingly recognized as unhealthy for the profession (Lehman et al., 2016). Therefore, broadening participation with interventions that provides engaging pedagogy for the female students in CS1 and beyond is a dominant theme in computing education research (Gunbata & Karalar, 2018). This section reviews empirical studies involving programming interventions and their impacts on male and female students of programming.

Rubio et al.(2015) employed a contextualized computing pedagogy involving MATLAB programming and Arduino, so that programming concepts become tangible to students using sound, movement, and light. This intervention was aimed at assessing, in comparison to the traditional instruction, gender differences in programming perceptions and learning outcomes of first-year Spanish university students who all had no prior experience in programming. Both experimental and control groups had 10-week instruction in introductory programming. The study was underpinned by the Technology Acceptance Model (TAM). A survey based on the three TAM constructs namely, perceived ease of programming, perceived usefulness of programming and intention to program, and a final achievement test were used to collect data. The programming perception survey was conducted thrice: at the beginning, middle and end of the course. Student t-test was used to analyse the programming perception data and a clustering technique was used to process the final exam scores. They found that while the three programming perceptions scores for males and females showed consistently diverging trends, closing with a significant difference in favour of male students in the control group, these scores, though exhibited mixed trends in course of intervention in the experimental group, gender differences remained consistently insignificant, suggesting that the contextualized instruction closed the gender gap in programming perception. On the other hand, there was no gender difference in the programming learning outcomes in both groups. However, the failure rate revealed a different picture. While female students' failure rate doubled those of male students in the control group, the rate was essentially the same in the experimental group. Unfortunately, with no pretesting of the students' programming knowledge, or use of a covariate, there is no guarantee that students are equivalent, and that initial difference would not have confounded the final exam outcome.

Likewise, Sabitzer, and Pasterk (2014) employed a neuro-educational pedagogy they called "brain-based programming" and examined its effects on students' achievement. The hypothesis of their study is: programming instruction that addresses each student's brain and memory functions by employing principles of neuro-didactics will improve their learning outcomes. Such principles include discovery-learning, active-learning, cooperative-learning and autonomous or individualized instruction. The study employed a quasi-experimental design involving first-year Austrian university computer science students consisting of 3 experimental groups (n=71) and 4 control groups(n=88). Three research instruments were used: a profile questionnaire, a midterm test and a final achievement test. Data collected were analysed using independent samples t-test. The results of the study revealed that the success rate was higher in the experimental groups (52%) compared to the control groups (40%) and the traditional CS1 sets from previous years (30-40%). While the results from the midterm test suggest that the intervention was more effective than the traditional CS1 pedagogy ($p = 0.008$, Cohen's $d = 0.42$), the average achievement in the experimental group though higher compared to the control in the final test, the difference was not significant. Another result also indicates that the intervention was effective at closing the gender gap, as the performance of both males and females was not significantly different in the experimental group. The males performed significantly better than the female in the control group. The drawback of this study like that of Rubio et al.(2015) was that: prior individual and group differences were not considered. These could have confounded the results of the experiment.

Papadakis and Kalogiannakis (2019) examined the cognitive and affective impact of a *Scratch* intervention on some female preservice kindergarten teachers who had no prior programming experience, in a Greek university. They employed one group pretest-posttest quasi-experimental design, with a convenience sample of 15 females that signed up for a 13-week elective course. Data were collected with an adapted Teacher Self-Efficacy in Computational Thinking (TSECT) questionnaire for pre-test and post-test; Dr *Scratch* (an online tool) for assessing students' *Scratch* projects to measure the seven computational thinking (CT) dimensions; a qualitative questionnaire and a structured interview protocol. The seven dimensions of CT are logical thinking (LT), data-information representation (IR), user-interactivity (IN), flow control (FC), abstraction (AB) and problem decomposition, parallelism (PA), and synchronization (SN). The result of the t-test of Self-Efficacy data indicated that students' self-efficacy increased significantly. Analysis results from Dr *Scratch* for the seven dimensions of computational thinking students employed in their project were: PA (M=1.88, SD= 0.38) LT (M=1.54, SD = 0.29), FC (M=2.16, SD=0.41), IN (M=1.81, SD=0.32), IR (M=1.68, SD=0.29), AB (M=0.72, SD=0.22), SN (M=1.84, SD=0.41), indicating, with a maximum score of 3 for each dimension, that while students performed poorly in Abstraction (a core skill in CT), their best performance was in the ability to write sequences of instructions in their programs. Overall, students' projects were categorised into Basic (15%), and Developing (85%), indicating none of them reached the Proficiency level in their *Scratch* programming skill. This suggests *Scratch* has a moderate effect on their cognitive achievement. However, qualitative results (together with earlier self-efficacy result) indicate that the intervention had a strong affective impact as all the students expressed satisfaction with learning *Scratch* and indicated intentions to learn more CT and teach CT in their classrooms after graduation. However, these results are only suggestive due to weaknesses in the study. A larger sample and a control group (even another equivalent or matched sample of female students) will probably produce a more definitive indication of *Scratch*'s impact.

Adleberg(2013) investigated the problem of gender differences in student engagement in programming by conducting a *Scratch* intervention. The study sample was 98 elementary school students from a private school in Virginia, USA. They were randomly assigned into three groups: 18 girl pairs, 17 boy pairs, and 14 mixed-gender pairs. Underpinning the study in theory of constructionism, participants were taken through four workshop sessions in *Scratch*, three offline and one online. Data collected included a pre-test of prior programming experience, a post-test survey of extrinsic and intrinsic motivation. Analysis of Variance results indicated that though boy pairs' prior programming was significantly higher than those of girl pairs or mixed pairs, yet there was no significant gender difference in engagement with *Scratch*. This suggests that *Scratch* was equally engaging for both genders, despite the initial difference in prior programming experience. While there was no significant gender difference in intrinsic motivation, yet, unsurprisingly, there was a significant gender difference in extrinsic, with girls having higher extrinsic motivation. However, the result is limited in its generalisability due to the sample being a homogenous group, not representative of elementary students, and no control group to compare with the treatment with.

Likewise, Hsu (2014), in the context of *Scratch* games design competition, explored gender differences among elementary students who were skilled in *Scratch* programming. Adopting a one-group post-test design, a purposive sample of 46 elementary school students in Taiwan were selected by their teachers for the study. They were divided into 23 teams with 13 boy pairs, 7 girl pairs and 3 mixed-gender pairs. Data were collected with a post-competition survey to measure computational thinking concepts; work analysis by Scrape (A computer program) to mark each team's *Scratch* game. From the independence t-test, the author found that while the overall results of students' performance in computational thinking concepts indicate no significant gender difference, the females scored significantly higher in the counting loop compared to the males. However, this study suffers from the same methodological weaknesses as the previous study by Adleberg (2013): Non-representative sample and lack of a control group.

Addressing some of the drawbacks of the above studies, Tekerek and Altan (2014) examined the impact of *Scratch* and gender on 6th-grade students' achievement in a Turkish school in an algorithm course. A pretest-posttest control group experimental design was used. The study involved 30 students (15 boys and 15 girls) in the control and similarly 30 students (15 boys and 15 girls) in the experimental group both from an elementary school in Turkey. Data were analysed using independent samples t-test, paired-sample t-test, and ANCOVA. While anecdotal evidence suggests *Scratch* was positively engaging for the students, empirical results indicate that the mean achievement of the control group (taught the traditional way) was higher compared to the experimental group, though the difference was not significant. Similarly, there is no significant effect of gender on student achievement. The result suggests both modes of instruction were equally effective for both groups and gender. However, the study's validity is threatened by the selection of students from a school. The possibility of the participants interacting can confound the outcome. Employing samples from two schools will probably mitigate this threat. The test of assumptions for ANCOVA was not stated and there’s no way to know if the covariate used, was appropriate in the ANCOVA model. Findings and interpreting results may be defective.

Some other studies present descriptive evidence of factors that predicts or differentiates male and female performance in introductory programming. For instance, Gunbatar and Karalar (2018) investigated the impact of mBlock (a *Scratch*-like) programming learning environment and gender on the self-efficacy perception and attitudes of Turkish 6th-grade students towards programming. The study which lasted 12 weeks, employed a one-group pretest-posttest design quasi-experimental design involving 82 public middle school students (39 girls and 43 boys) consisting of 3 classes. Results of statistical analysis of self-efficacy and attitudinal data indicated that the initial significant gender gap (with boys being higher) in self-efficacy disappeared at the end. While there was no initial gender difference in attitude towards programming, in the end, girls had a higher positive attitude towards programming, though the difference was not significant.

Quille, Culligan, and Bergin (2017) examined gender differences in factors that predict success in CS1 in three areas: background factors, psychological factor and programming performance. Employing a descriptive research design, study participants were CS1 students from 11 universities and colleges in Ireland and Denmark. Using an earlier developed machine learning web-based tool called PreSS#, complete students' data on these areas were collected from a sample of 693 students with a male to female ratio of 79:21. However, a cohort representing 36% of the sample (with male to female ratio of 73:27) was reported in their article. The online research instrument consisted of a Background and Student Data Survey; Psychological Questionnaire and Programming Test. Data were analysed using descriptive statistics (distribution in percentages) and inferential statistics (Welch t-test). Findings from this study revealed:

1. **For background**: while there were no gender differences in age, weekly part-time work hours, and the time to complete the survey, there were significant differences in prior mathematics ability, overall pass rates, daily social media hours (with females having higher value), and daily computer games playing hours and dropped out rate (with males having higher values).

2. **For the psychological factors**: All factors recorded significant gender differences, mostly in favour of males. Males recorded higher values in self-efficacy, self-rating of knowledge of concept, design, and completion of code, intrinsic motivation and expected course grade, and lower test anxiety compared to the females.

3. **Programming Test**: Significant differences appeared at different stages. In the early stage test males performed significantly better while at the end of the course module females outperformed the males. Suggesting that probably the deficit in self-efficacy at the beginning of the course influenced the females to put in more time and effort in their study.

However, with the reliability or validity of instruments not stated, the results of this study may only be taken as suggestive. A similar study by Pappas et al. (2016) addressed this important need for reliability and validity of research instruments. In the study involving a sample of 236 (180 males (76.3%) and 56 females (23.7%) Norwegian university computing students, with 21% of them in their first-year, they assessed gender differences in perceptions of factors that are believed to influence students towards enrolling and remaining in computer science, with a particular aim at identifying factors that are critical for women participation in CS. They employed a questionnaire containing 7-point Likert scale items divided into four parts: Demographics, Critical CS constructs, Barriers, and Future intentions. The reliability and validity of the instrument were established with acceptable Composite Reliability (CR) and Average Variance Extracted (AVE). Research data were analysed using the Welch t-test. Gender differences were found in five areas: cognitive gain, affective engagement, interest in schoolwork (in favour of males), and interest in CS, and satisfaction with learning effectiveness (in favour of females). No gender differences were found for non-cognitive gain, cognitive engagement, personal values, teaching quality, and retention. This implies that while both genders perceive later five factors equally, they differ in their perception of the former five. It is interesting to note that females who enrolled in computer science do that out of real interest, yet probably due to the environment of male-dominated classes and stereotypes, they may lose interest in school work later. However, the results suggest that females will value engaging pedagogy than their male counterparts. Klawe (2013) identified this as one of the reasons institutions like Harvey Mudd College in the USA have been more successful at enrolling and retaining women into CS.

Lishinski et al. (2016) explored the relationship between 4 Self-Regulated Learning (SRL) constructs namely, self-efficacy, metacognitive strategies, intrinsic and extrinsic motivation, and programming performance of CS1 students. The study also investigated if there were gender differences in the effects of these four constructs in their programming performance. The study adopted a one-group pretest-posttest design involving 346 students in a CS1 course at a large midwestern university in the USA. Of these students, 248 (73.1%) were male, 93 (26.9%) were female and 5 were unidentified. The study employed an adapted online Motivated Strategies for Learning Questionnaire (MSLQ) to measure the 4 SRL variables and two instruments developed for programming performance:

- multiple-choice exam
- programming project rubric.

Path analysis of the research data revealed that self-efficacy has the strongest direct relationship with programming performance, particularly the programming exams. Results also revealed a self-efficacy loop feedback in which, metacognitive strategy and intrinsic goal orientation influences programming performance, and improved programming performance leads to increase self-efficacy. While the overall results indicate a strong relationship between self-efficacy and programming performance, largely, there was no significant gender difference, except for female initial self-efficacy which was significantly lower than that of the males (t = 2.92, p-value = 0.004). This is a common research finding; male and females rates their self-efficacy differently at the beginning of CS1. However, in this study, there were also gender differences in their pattern of self-efficacy correction. While females were quicker in correcting their initial self-efficacy belief, males were slower in modifying theirs. That is, with positive performance feedback females receive at the beginning, their self-efficacy increased sharply while those of the male remained largely unchanged. But between the second and third efficacy measurements, males increased significantly in the correlation between self-efficacy and programming performance, while those of the females though increased, the increases were not significant.

Like the previous study, Aivaloglou and Hermans (2019) explored the relationship between gender, age, self-efficacy, prior programming, motivation, and stereotypical beliefs on programming performance and career goals of elementary students. The aim was to investigate whether factors that have been found to affect university computer science students, are also applicable at this level in an introductory programming class. The study followed a correlation design which involved 74 students from two public elementary schools in Netherland, with students in each school randomly divided into 2, making a total of 4 cohorts. In eight weeks, students were exposed to the *Scratch* lesson. Data were collected using a profile questionnaire, MLSQ, midterm and final achievement tests. Surprisingly, test performance was found to be uncorrelated with self-efficacy and motivation. Self-efficacy was found to be strongly correlated with intrinsic and extrinsic motivation. It was also found that, while career orientation was not correlated with test performance and stereotypical beliefs, it was strongly related to their midterm self-efficacy and intrinsic motivation. The effect of gender indicates that self-efficacy was strongly related to test performance for females than for males, though the difference was not significant. Another notable result indicates that intention for CS career measured in the middle of the course for females is strongly related to their self-efficacy measurements at the beginning and end of the course. Previous programming experience was found to be strongly correlated with self-efficacy, extrinsic motivation and career orientation.

While self-efficacy and motivation have often been found to be related to learning performance, the probable reason for the contrary result in this study may be that students reported a wrong level of their self-efficacy. The researchers gave difficulty of the tests as the probable reason. They also admitted that the self-efficacy measurement used may not be suited to the age of the students.

In summary, the following can be deduced from the above empirical studies in this section:

- Male and females rate their self-efficacy differently at the beginning of CS1. While male students generally rate their self-efficacy high, female counterparts often rate theirs low. However, research suggests this self-efficacy gap is often closed by effective interventions. (Gunbata & Karalar, 2018; Lehman et al., 2016; Lishinski et al., 2016; Quille et al., 2017)

- Male students often have higher intrinsic motivation while female students have higher extrinsic motivation for programming. (Adleberg, 2013; Quille et al., 2017)

- Self-efficacy is strongly related to programming performance. (Aivaloglou & Hermans, 2019; Lishinski et al., 2016)

- While *Scratch*'s impact on students' achievement is mixed, what is clear is that constructionist programming interventions employing *Scratch* can be equally engaging for both gender, it may be effective in closing gender gaps in self-efficacy and programming performance. (Adleberg, 2013; Rubio et al., 2015)

- CS1 interventions should target or leverage self-efficacy of the female students who often are in danger of dropping out from CS due to inherent low self-efficacy or their initial difficulty with traditional CS1 and the attendant low self-efficacy. Also, programming instruction should be tailored to satisfy their extrinsic motivation.

2.13.3 *Scratch* in CS1, Students' age, and learning outcomes

What is the relationship between the age of students exposed to programming (or *Scratch* in particular) and their performance in introductory programming classes? Generally, it is believed that there is a linear relation between programming knowledge and age (Rizvi & Humphries, 2012; Rountree et al., 2002; Silvast, 2015). In this section, we will consider some relevant studies to further explore this issue of age and programming achievement.

Hermans and Aivaloglou (2017) in a study involving Dutch elementary students sought to explore the hypothesis that young age is not a barrier to learning advanced CS concepts. Thus, the main aim of the intervention was to explore that idea, and confirm if age moderates the performance of students in their learning of programming and software engineering concepts. The study employed a descriptive research design with a convenience sample of 181 students that completed the study, out of the 3,179 and 2,270 that signed up and participated respectively, in a 6 weeks MOOC introductory *Scratch* Programming course on eDX. Students were exposed to programming by weekly video demos of *Scratch* programs. Data were collected using student profile questionnaires, formative quizzes, midterm term and final tests. They found from the comparison of students' overall mean scores in both programming and software engineering concepts, that there was no significant difference, indicating the students performed equally well in both. While the comparison of the 11-12 age group with 13-14 age group students indicated there were no differences in many concepts, nevertheless, significant differences (with at least a small effect size) were recorded in concepts of procedures and operators in favour of the older students, probably suggesting that there are programming or software engineering concepts that are too difficult to understand for younger students.   However, with one convenience sample, with no control group for comparison makes the result only suggestive. For a definitive finding, this phenomenon of age and programming achievement needs to be studied further using a true experimental design. While elementary students can learn advanced CS concepts, the result of this study suggests there are topics that the students will struggle to understand, so introducing such at that level will not be appropriate.

Like the previous study, Atmatzidou and Demetriadis (2016) employed Educational Robotics (ER) and guided instructional approach in an introductory class for secondary school students, intending to explore age and gender differences in the development of their Computational Thinking skills. This study was underpinned by Seymour Papert's theory of constructionism and Lev Vygotsky's social cognitive theory. The study followed a one group quasi-experimental design involving a convenient sample of 164 (89 Junior high students: aged 15, 48 boys and 41 girls and 75 High vocational students: aged 18, 64 boys and 11 girls) from Thessaloniki, Greece. Students were exposed to computational thinking education during an 8-week seminar, 4 weeks for the Junior and 4 weeks for High students. Students were taught with Lego Mindstorms NXT 2.0 robotics kits, and data instruments used were Student Profile Questionnaire, two intermediate questionnaires for CT skills, Students Opinion Questionnaire, Think Aloud rubric, Semi-structured interview protocol, and Structured observation protocol. From the analysis of the quantitative data, they found that, though none of the students has prior experience with robotics, there was no significant age difference in the overall CT skills of the participants, indicating that despite the age difference (15 years versus 18years) both groups exposed to the same educational robotics education instruction developed equally same computational thinking skills. The results also indicated that though girls performed initially lower than boys in CT skills, they were able to catch up, and in the end, there was no significant gender difference.

In an often-cited study on predictors of CS1 students' performance by Rountree, Rountree, and Robins (2002), the authors employed descriptive research design, to identify factors that determine students' success or failure in a CS1 course, for students at a university in New Zealand. The study involved a cohort of 472 students enrolled in two consecutive sessions 2000-2001. The course lasted 13 weeks of the second semester. Data were collected from an online survey and final course score. Percentage and Chi-square statistics were employed to analyze data. The study found that factors such as gender, studentship type or intention to continue CS did not matter in student success. However, while age did not matter among students with a pass rate of 70% and above, age difference reflected in the performance of students who attained at least a pass rate of 50%. Among these students the following pass rates were recorded 16–18, 80%, 19–21, 72%, 22–24, 54% and 25+, 74%, throwing up a surprise that more matured students (age 22-24) had lowest pass rate. The study found that the greatest predictor of success was student self-selected expectation of grade A, suggesting that student's self-efficacy belief at the early stage of a CS1 course is strongly related to their final performance. However, this study is fraught with several threats that make its findings only suggestive. Information on validity or reliability of research instruments were not indicated. Testing of assumptions for Chi-square analysis was not given. With a sample not representative of the CS1 population, generalization findings cannot be made.

Chen, Haduong, Brennan, Sonnert, & Sadler (2019) took a different approach to the above studies to find answers to the question of the relationship between age and CS1 achievement. They employed ex post facto quasi-experimental design, to explore retrospectively, the effects of the type of programming environment on student's age, their attitude at the beginning, of being introduced to programming and their achievement at the end of a college CS1 course experience. The study employed stratified random sample to realise a representative sample of students in the US. students from 118 colleges running 2- 0r 4-year CS programmes in the US, totalling 10,203 eventually completed the in-class survey at the beginning of CS1 and their instructors added their final grades at the end. This sample is made up 73% male (n=7,219) and 27% female (n= 2,660). However, for the analysis, optimal matching was used to generate a matched random sample size of 5764 students consisting of 306 students in graphical programming, 2995 in textual programming and 2463 had no prior programming language. t-test analysis of the matched samples revealed that samples consist of students with similar backgrounds. Data collected through an in-class survey and CS1 course assessments were analysed using regression analysis. The result indicated there were significant main effects of graphic programming and textual programming environments on the students' positive attitude towards CS (in comparison with those without prior experience). They found no significant difference in positive attitude between the students exposed to graphic and those with textual programming backgrounds. They also found significant interaction effect of age when students first started to program and the programming environment on students' positive attitude towards CS, suggesting that students' positive attitude towards CS depends on what age and how they were introduced to programming. For final CS1 grade, the study found significant effects of both graphic and textual programming (in comparison to control) on students' achievement, indicating it is better to learn to program using any of the two environments than none. The performance of students with prior programming in CS1 depended on the age they were introduced to programming. Those introduced to graphic programming between the age 6-10 or younger, are likely to score higher in CS1 than those introduced to textual at the same age. There was no significant difference in the achievements of students introduced to programming between age 11-14 and 15-18years in both treatment groups, suggesting that both modes work equally well in preparing students for college CS1.

To summarize findings in this section we make the following deductions:

- Students' age and mode of initiating them to programming before college matters in their later attitudes towards CS or programming, and their achievements in college or university first programming course (CS1). (Chen et al., 2019)

- Age in CS1 appears in most cases does not matter in students' achievement in programming, especially among students with no prior experience in programming (Hermans & Aivaloglou, 2017; Rountree et al., 2002)

- However, findings from the above studies suggest that there are few programming concepts where students' achievements are differentiated by their ages. (Hermans & Aivaloglou, 2017)

- Whatever the age, student's self-efficacy is strongly related to programming performance. (Aivaloglou & Hermans, 2019; Lishinski et al., 2016; Rountree et al., 2002).

- Constructionist programming interventions employing *Scratch* can be equally engaging for all ages in K-16 education (i.e. elementary or primary school, high or secondary school, and first-year college or university students) (Adleberg, 2013; Rubio et al., 2015)

2.13.4 *Scratch*, CS1, prior programming experience and learning outcomes

Previous computing experience is another major factor that may contribute to success in introductory programming. It is often argued that prior programming experience students bring into college or university computer science course (CS1) contributes to their CS1 performance. College or university students acquire such experience either by taking high school programming CS course or those without CS background are made to take a pre-CS1, usually called CS0. In this section, we want to examine the effect of students' prior programming experience and their CS1 learning outcomes by reviewing some empirical studies.

Wilson & Shrock (2001) conducted a study exploring the relationship between twelve predictive variables and students' achievement in CS1. Some of the factors included in their model are math background, comfort level, performance attribution to luck, previous programming experience to mention a few. The attribution theory served as the study's framework. The study followed a correlation design involving 105 students taking a CS1 course in a midwestern US university. A questionnaire and the Computer Programming Self-Efficacy Scale, served as instruments for collecting data. The results from multiple regression analysis of the full model including the twelve factors revealed that comfort level, math background, and performance attribution to luck contributed the most to success, with a negative influence of luck. This suggests that while comfort level and math background are expected to contribute positively to students' success, attributing performance to luck probably led student to study less, and thus negatively impact their achievement. Further analysis using multiple regression of the previous programming and non-programming experience variables, as predictors and midterm grade as the outcome, showed previous programming and game playing to be significant in predicting student's success in CS1. However, while the contribution of previous programming experience is positive, that of game playing is negative. This indicates that students who are given to playing computer games probably spend less time on their studies and invariably performs less in the course assessments.

Silva-Maceda et al. (2016) in an attempt at investigating the impacts of CS1 pedagogical approach and students' organismic factors on their CS1 achievements, conducted a longitudinal study involving seven cohorts (N = 1168) of students in a central Mexican university. The students were assigned to three pedagogical modes differentiated by variation in the length of learning time (2 or 3 semesters) or programming tools used in their first semester. The three groups consisted of those who did not take a CS0 course in the first semester, but took CS1 with C for 2 semesters, those who took a CS0 using C in their first semester and CS1 with C for 2 semesters, and those who had a CS0 using Raptor in their first semester and CS1 with C for 2 semesters. One of the questions they sought to answer was,"Does taking a CS0 course make a difference in the pass rates of each group in the overall CS1 score?". Logistics regression analysis of students' performance data indicated that students in the CS0 with Raptor group had highest pass rates, followed by students who had CS0 with C, and lastly those who took CS1 without a CS0 course. After including initial students' ability as control, the results of another logistic regressions analysis yet indicated a statistically significant difference in the pass rates between those without a prior experience and the other two groups that were exposed to a CS0 course before CS1. However, comparing the two groups with prior experience in a CS0 course, there was no statistically significant difference after including initial students' ability in the analysis. This suggests students first being exposed to programming, is better than taking CS1 without a prior programming course, probably evidence of the benefit of higher time-on-task. Also, it is better to have a prior programming experience using whatever pedagogical approach than having none. Though, the study controlled for the initial students' ability, nevertheless due to the longitudinal nature of the study, some other factors may have confounded the results. Some of these factors included differential teachings and assessments leading to the final CS1 grade. An experiment with students taking the same pre-test and post-test may address such limitation.

Hagan and Markham (2000) in a study involving Australian first-year computer science students sought to confirm, whether students' prior experience in programming has an impact on their achievement in the university CS1 course. The study employed a longitudinal study design in which students' performance measured with a series of research instruments at a different stage of a semester course in introductory programming lasting for 13weeks. The study sample was made of 75 students which provided complete research data. While the t-test analysis of data indicated a significant statistical difference in the four formative assessments between students with none, and those with some experience in programming, there was no statistically significant difference in their summative assessment. However, t-test analysis of their final grade in the course showed a significant difference between the groups, in favour of those with some experience. Analysis of variance of the performance of the groups also revealed that the more languages a student earlier studied or had experience programming in, the better their performance in CS1. Furthermore, the analysis of attitudinal data collected at an early stage and just before the final exam indicated a statistically significant difference in the results from both surveys, measuring students' confidence in passing the course, in favour of those with some experience in programming before CS1. The study however did not provide information validity or reliability of the research instruments. The study was limited by a lack of pretesting to identify if students in both groups were comparable based on any covariate that can likely confound the result.

Mishra et al. (2014) conducted an intervention aimed at addressing the pedagogical needs of students with and without prior programming experience in an Indian university. *Scratch* was employed in the study for the first two weeks before all the students transitioned to C++ for the rest of the semester's CS1 course. The study involved 450 engineering students, however, only 332 of them that completed a programming background survey in the first week of the study were included in the analysis. This sample was made up of 217 and 115 students that were classified as novices and advanced learners respectively, based on their prior programming experience. Other research instruments used included semester quiz and midterm exam to assess students' programming knowledge, Scrape (an online tool) to assess *Scratch* projects, and a survey to assess students' perception about *Scratch* usefulness to their learning of C++ at the end of the semester. Independent samples t-test analysis of the midterm exam consisting of *Scratch* and C++ questions indicated that there was a statistically significant difference, between the students with no prior experience and those with some programming experience at the end of 6th week in some question types: "Predict the output" and "Debug a program". This suggests *Scratch* has helped the students with no prior experience to catch up with their counterparts with prior programming experience in some types of programming knowledge. Going by the assessment of student's projects, they inferred that the advanced learners were also highly engaged with programming using *Scratch*. The analysis of the perception data indicated that most of the students believed *Scratch* was beneficial to their learning of C++ programming language concepts. However, there was a significant difference between the two groups in "Write a program" questions, in favour of those with prior experience. This suggests that prior programming experience matters in students' performance in CS1. There was also a correlation between the midterm scores in *Scratch* and C++, suggesting student who performs well in *Scratch*, a block-based programming language, will likely perform well in C++, a text-based programming language. However, the results of this study are only suggestive due to some limitations. There was no control group and measuring of any covariate that can confound the results.

2.13.5 *Scratch*, CS1, visual artistic experience and learning outcomes

In this section we seek to explore the relationship between students' creative achievement in prior visual artistic exercises and their CS1 achievements. Does bringing a positive performance in prior creative activities makes any difference in the CS1 class? We will answer this question by reviewing some empirical works.

Following an earlier empirical study that suggests delaying or replacing a CS0 coding-first curriculum with "Story Programming" (a non-coding) pedagogy is promising for broadening participation in CS1 class, Parham-Mocello and Erwig (2020) sought to confirm the hypothesis that exposing students with little or no programming experience or interest in computing to only "Story Programming" will likely see better results for Drop-Fail-Withdrawal (DWF) rate and students' interest than those exposed to coding-first. The "Story Programming" approach is essentially using art (in form of stories) to develop creative or computational thinking abilities of the students. The study involving 147 to 191 CS1 students in a USA university employed a quasi-experimental design. Data collected were analysed using Kruskal-Wallis and Wilcoxon paired t-test. They found that while DWF rate was not significantly different between the class that had "Story Programming" and those that had coding, the DWF rate was higher in the traditional coding-first class. They also found that, there was no significant difference in the grade point average of students in both classes. This suggests that programming or computational thinking can be learnt by pedagogical means other than coding on a computer - usually called "CS unplugged". Another analysis sought for change in students’ interest in the class, coding and use of stories. They found that there was a significant drop in the class interest in both groups, with a higher drop in the coding-first class. This suggests that some students in both groups did not find the classes engaging, especially those in the coding-first group. This is further confirmed by the finding that while interest to learn more about programming or coding did not wane between pre and post-test in the treatment group, there was a significant decrease in the coding-first group. In answer to questions on the use of stories to learn programming, while the traditional group's opinions were more negative, those of the treatment were more positive. This study suggests students can learn programming using art and students exposed to such art-based learning environments, and by so doing arose their creative interest, may perform better in CS1 than those without such background.

Hermans and Aivaloglou (2017) explored the impact of introducing students to programming with or without the computer (usually referred to as plug and unplugged CS) on their learning outcomes. They employed a quasi-experimental design involving two random samples of 35 students (in total) from a Netherland elementary school. The study is underpinned by Bandura's theory of self-efficacy. Self-efficacy is the belief that one can perform steps necessary for realising a goal and has been found to correlate with students' programming achievement. In the first week, online accounts were opened for both groups on *Scratch*. Subsequently, for four weeks, while the "plugged" (the traditional) group (n=17) were introduced programming using *Scratch*, the treatment group (n=18) learnt programming concepts using CS unplugged activities without programming on a computer. Subsequently, both groups continued to learn to program using *Scratch* for two weeks. For another two weeks, both groups developed and presented programming projects of their interest in *Scratch*. The study employed MLSQ to collect pre-post data on self-efficacy at the end of week two and week eight respectively. Programming knowledge of both groups was assessed by an end-term test. Their projects were assessed using a rubric. The result of an independent samples t-test indicated no significant difference between the two groups in their programming knowledge. However, while the t-test analysis of self-efficacy data at the beginning showed no significant difference between the two groups, the result of their self-efficacy at the end revealed a significant difference in favour of the unplugged group. They also found from the projects that students in the unplugged groups used significantly more *Scratch* blocks than what both groups were taught. Probably suggesting that the unplugged pedagogy advanced the creative achievement of the students more than what *Scratch* lessons did during the four weeks of learn programming concepts. With such leverage of higher creative ability, the unplugged group outperformed their counterparts during the two weeks' *Scratch* projects. However, this study will have provided more convincing results if the students were pretested for prior computational or programming knowledge and creative achievement, as was done for the self-efficacy. Another control will be using representative sample from another school.

While previous studies in this section did not directly address the question of the impact of students' creative artistic ability in a CS1 class, Gestwicki and Ahmad (2010) sought to address this issue. They employed a correlational design involving 15 first-year students in an American university. Their hypothesis is that there is a statistically significant relationship between students' creative achievement and their CS1 performance. They employed Creative Achievement Questionnaire (CAQ) and other instruments to collect research data. Students went through a multimedia oriented CS1 which provided an environment for creative artistic expressions in a programming class. At the end, they found that creative achievement of the student is not significantly related to their CS1 achievements. Nevertheless, there was a significant relationship between Music aspect of CAQ and students' final exam. The study results raise some questions as to validity of results. The study sample of 15 students, four of which were CS majors, and 11 non-CS majors, is inadequate. The study aimed at exploring causation between creative achievement and CS1 performance. To realise that an appropriate design would be an experimental study involving adequate number of students in the treatment and control groups.

2.14 Conceptual Model for the Study

The conceptual framework for the study is shown in Figure 2-13. The conceptual model for this study is composed of the independent variables or the treatment namely: constructionist inquiry-based programming instruction and traditional lecture-based instruction. The researcher manipulated these variables to see their effect on the dependent variable (achievement of first-year computer science students in programming). As shown by the features of the two pedagogies in the figure, there is variation in the mode of programming instructions in the two treatment conditions. Those in the experimental group were exposed to *Scratch* programming in a constructionist way, while the control group was treated to programming in Visual Basic programming using direct instruction.

The intervening variables consist of organismic and environmental factors. The organismic factors are those factors which are resident within the individual such as level of academic skills, gender, self-efficacy, level of engagement, and age among others. However, in this study, gender, age, academic background, visual art background and prior programming are chosen as moderating variables. A moderating variable changes (or interacts with) the effects of the treatment on the dependent variable. That is, together with the treatment a moderating variable form an interaction term in the experimental model.

Mediating variables are intervening variables that stand between the independent variable and the dependent variable, having a positive impact on the relation between them. That is, the independent variable produces a positive effect on the mediating variable and then the mediating generates a positive effect on the dependent variable. Though several variables could mediate between programming instruction and students' achievement, in this study, student engagement has been chosen as a mediating variable (Ribeiro et al., 2019). Research suggests that when a programming instruction is engaging, it will positively impact students' achievements (Lei et al., 2018; Phuntsho & Dendup, 2021; Schnitzler et al., 2021).

The environmental factors are variables which are resident outside the individual and could affect the responses of the participants to the treatment. Examples of environmental factors are socioeconomic status differential learning context, amongst others. In this study, confounding variables are environmental factors though not measured yet, can positively or negatively affect both the independent (the treatment) and the dependent variables. To mitigate the threat to the validity of the experiment posed by a confounding variable, a researcher must put in place control measures to counteract its effect. Some of the control measures adopted in this study was employing comparable samples from public institutions enrolling mostly students from low- or middle-income families in the same north-central region of the country. Though this itself makes the samples less representative by leaving out students from private institutions who are likely from high-income homes. Another measure was to ensure that same lecturer or lecturers with similar experience provide the programming instructions in the two treatment groups.

The covariate is a variable that has a linear relationship with the dependent variable.

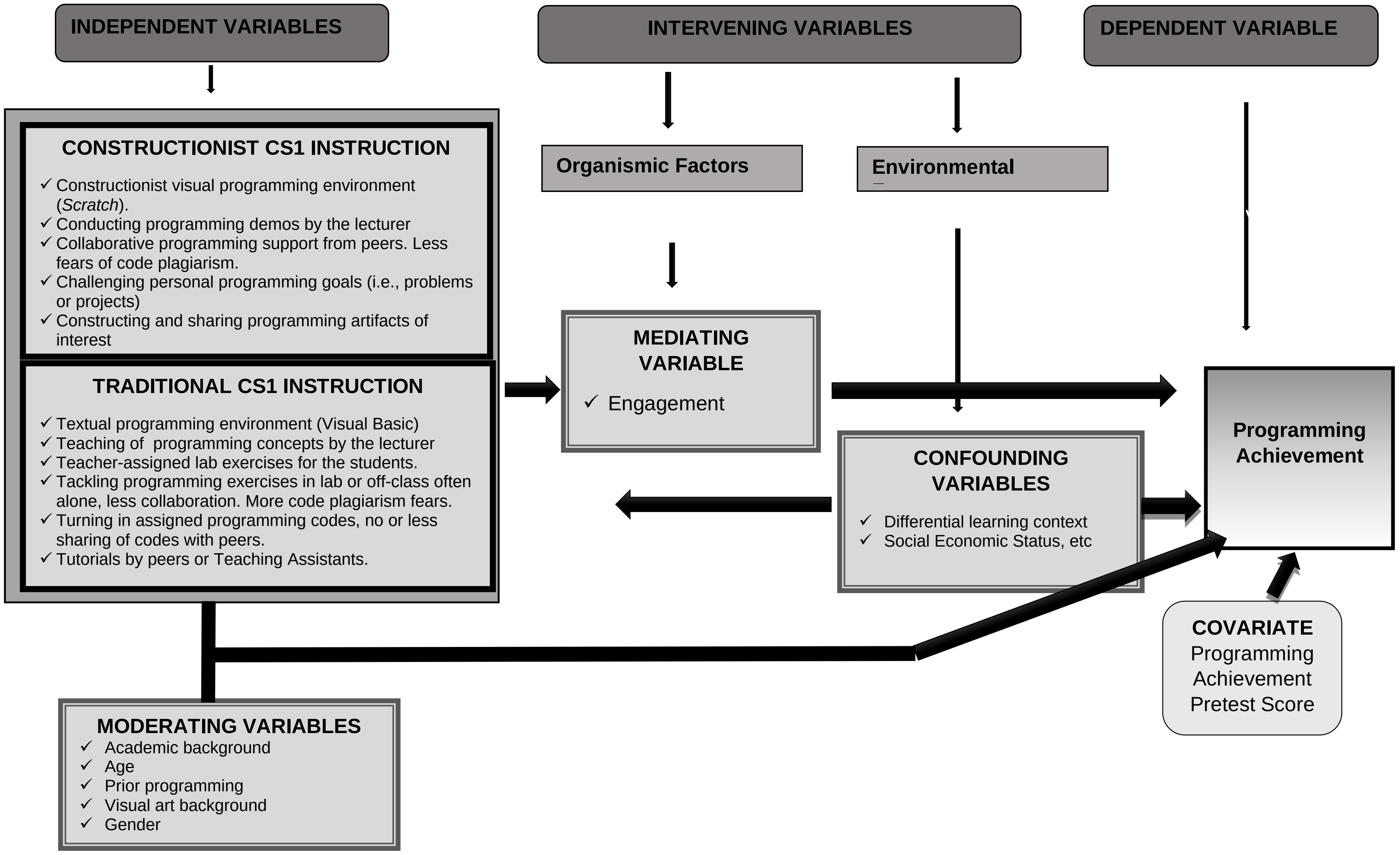

INDEPENDENT VARIABLES
INTERVENING VARIABLES
DEPENDENT VARIABLE
CONSTRUCTIONIST CS1 INSTRUCTION
✓ Constructionist visual programming environment (*Scratch*).
✓ Conducting programming demos by the lecturer
✓ Collaborative programming support from peers. Less fears of code plagiarism.
✓ Challenging personal programming goals (i.e., problems or projects)
✓ Constructing and sharing programming artifacts of interest
TRADITIONAL CS1 INSTRUCTION
✓ Textual programming environment (Visual Basic)
✓ Teaching of programming concepts by the lecturer
✓ Teacher-assigned lab exercises for the students.
✓ Tackling programming exercises in lab or off-class often alone, less collaboration. More code plagiarism fears.
✓ Turning in assigned programming codes, no or less sharing of codes with peers.
✓ Tutorials by peers or Teaching Assistants.
Organismic Factors
Environmental
MEDIATING VARIABLE
✓ Engagement
CONFOUNDING VARIABLES
✓ Differential learning context
✓ Social Economic Status, etc
Programming Achievement
COVARIATE
Programming Achievement Pretest Score
MODERATING VARIABLES
✓ Academic background
✓ Age
✓ Prior programming
✓ Visual art background
✓ Gender

Figure 2-13 The Study's Conceptual Model

Chapter 3

3 Research Methodology

This study explores the educational impact of a constructionist programming intervention on the programming ability of first-year computer science (CS1) students. Seymour Papert's theory of constructionism advances the idea that students learn (or construct knowledge) as they create and share artefacts of interest in collaboration with their peers. The main research question of this study is - what is the impact of a constructionist programming intervention using *Scratch* on the programming ability of CS1 students? Several corollary research questions following this main question are mentioned in chapter one. This chapter presents the approach adopted in finding answers to the research questions. The topics presented include the research paradigm, research design, the research population, the sample, and sampling technique used, the research instruments, validation and reliability of research instruments, data collection procedures, data analysis technique employed in the study, and ethical considerations that guided the research.

3.1 Research Paradigm

Science has always progressed as scientists conduct research (or specifically, experiments) by following a guiding principle called research paradigm (Bird, 2018; Fraenkel et al., 2018; Kuhn, 2012). J.S. Mills refers to this ancient idea as "Methods of Difference". This simply means we start our experiment with two instances resembling one another in every respect but differing in the presence or absence of the phenomenon we wish to study. At the end of the experiment, if we notice any difference in the two instances, we arrive with certainty that the phenomenon had caused the change.

Saint-Mont (2015) captures JS Mills' logic in a tabular form thus (Table 3-1):

Table 3-1 JS Mill's logic

Start of Experiment	T	=	C	T	≠	C
Intervention	Yes		No	Yes		No
End of Experiment (Observed Effect)	$\overline{X}$T	>	$\overline{X}$C	$\overline{X}$T	>	$\overline{X}$C
Conclusion	The intervention caused the effect			The intervention OR Prior difference between the groups caused the effect		

Source: Saint-Mont (2015). Note: T = Treatment group. C = Control group

Determining the impact of a constructionist programming instruction on CS1 student's programming achievement necessitates that we employ the scientific method, that gives empirical evidence of causation if there is any. That implies this study takes a postpositivist stance.

3.2 Research Design

The research design is the researcher's plan or blueprint for conducting a research study (Ary et al., 2014; Mouton, 2013). Elaborating further, Creswell (2018) defines it as a type "of inquiry within qualitative, quantitative, and mixed methods approaches that provide specific direction for procedures in a research study." (pg. 49)

There are several research designs but the main factor that determines a researcher's choice is the research question(s) addressed by the study.

Hence, by the research questions posed in chapter one, the study employed a quantitative research design for the collection and analysis of the research data. A quantitative research design is made of procedures for collecting, analysing and combining quantitative data in a single study (Creswell, 2014).

The main research question of this study is: What is the impact of a constructionist *Scratch* programming pedagogy on programming achievement of novice undergraduate computer science students? To address this main research question, the study employed a quasi-experimental design (with pretest-posttest non-equivalent experimental and control groups). The study was an in vivo study involving intact classes of newly admitted CS1 students. We employed between-group design in which these intact classes were randomly assigned to treatment conditions. An ecological control was employed, meaning those in the control group were taught the usual contents using lectures and labs as done in the past. However, the experimental class were exposed to a student-centred *Scratch* programming pedagogy.

This study followed a quasiexperimental design with quantitative approach as stated below

```
NR    O     X     O
_ _ _ _ _ _ _ _ _ _ _ _ _ _

NR    O           O
```

Selection differences in the two groups due to non-randomisation has been addressed by matching and use of ANCOVA.

The following factors informed the choice of this design:

- The subjects of the research were newly admitted first-year students who were not taking any other programming course apart from the introductory programming course. By controlling every factor that can affect students' programming ability, thus ensuring the internal validity of the study, the impact of constructionist *Scratch* instruction on the students could be empirically deduced.
- The study took place in the first six weeks of the 15-week teaching duration for the introductory programming course. In addition, the course being compulsory for all the students, the researcher is constrained to make use of intact CS1 classes of CS1 students after securing their informed consent to participate in the study.

This study is a 2 (Programming Pedagogy: Constructionist (*Scratch*), Conventional) by 2 (Gender: Male, Female) by 4 (Age: 16-18, 19-21, 22-24, above 24) by 3 (Prior Academic achievement: High, Average, Low) by 2 (Prior programming: Some, None) by 2 (Prior visual art: Some, None)

The experiment involved the use of two levels of one primary independent primary variable. The independent variables are a constructionist *Scratch* programming instruction and the conventional CS1 lecture while the dependent variable was the students' post-intervention learning post-test scores. The secondary independent variables include Gender, Age, Prior Academic Achievement, Prior program-writing and Prior visual art.

The layout of the design is given in Table 3-2 below.

Table 3-2: Experimental Design layout

Groups	Pre-test	Treatment	Post-test
Experimental Group	O1	Constructionist *Scratch* Programming	O2
Control Group	O1	Conventional CS1 Instruction	O2

The schematic representation of this research layout is as shown below:

O1 X1 O2

O1 X2 O2

Where,

O1 represents the Pre-testing of both control and Experimental Groups,

O2 represents the Post-testing of both control and Experimental Groups,

X1 represents the Treatment (Constructionist *Scratch* Programming Instruction) for the Experimental Group,

X2 represents the Treatment (Conventional CS1 Instruction) for the Control Group.

		CS1 Programming Instruction	
	Gender	*Scratch*	Conventional
Mean Posttest score	Males		
	Females		

Variables: CS1 Instruction – Independent, Categorical: *Scratch*/ Convention

Gender – Independent, Categorical: Males/females (1, 2)

Post-test – dependent, continuous: scores on IPAT, ranging from 1-100

		CS1 Programming Instruction	
	Age	*Scratch*	Conventional
Mean Posttest score	16-18		
	19-21		
	22-24		
	Above 24		

Variables: CS1 Instruction – Independent, Categorical: *Scratch*/ Convention
Age – Independent, Categorical: 16-18/19-21/22-24/Above 24
Posttest – dependent, continuous: scores on IPAT, ranging from 1-100

		CS1 Programming Instruction	
	Academic Background	*Scratch*	Conventional
Mean Posttest Score	High		
	Average		
	Low		

Variables: CS1 Instruction – Independent, Categorical: *Scratch*/ Conventional

Academic Background – Independent, Categorical: High, Average, Low (i.e., 3,2,1)

Post-test – dependent, continuous: scores on IPAT, ranging from 1-100

		CS1 Programming Instruction	
	Program Writing Background	*Scratch*	Conventional
Mean Posttest score	None		
	Some		

Variables: CS1 Instruction – Independent, Categorical: *Scratch*/ Conventional
Program Writing Background– Independent, Categorical: None/Some (1,2)
Posttest – dependent, continuous: scores on IPAT, ranging from 1-100

		CS1 Programming Instruction	
	Visual Art Background	*Scratch*	Conventional
Mean Posttest score	None		
	Some		

Variables: CS1 Instruction – Independent, Categorical: *Scratch*/ Conventional
Visual Art Background – Independent, Categorical: None/Some (1,2)
Post-test – dependent, continuous: scores on IPAT, ranging from 1-100

3.3 Sampling

The populations in this study were polytechnic students in Nigeria. The target population are first-year computer science National Diploma (ND), polytechnic students, in the north-central area of Nigeria (see Table 3.2). Polytechnic students were selected for this experimental study as polytechnics are the higher education environments, where, theory of constructionism will be most welcome due to the vocational and technical educational goals for their setup (Geschwind et al., 2020). I have experience teaching polytechnic students' introduction to programming course for over two decades. Therefore, the interest to study the phenomenon of novice programming learning among these set of college students informed their choice as research subjects.

Table 3-3 Polytechnics with Accredited Computer Science Programme in the Study Area

SN	NAME OF POLYTECHNIC	LOCATION	OWNERSHIP
1	Federal Polytechnic Bida	Bida. Niger State	Federal
2	Federal Polytechnic, Idah	Idah. Kogi State	Federal
3	Federal Polytechnic Nasarawa.	Nasarawa. State	Federal
4	Nasarawa State Polytechnic,	Lafia. Nasarawa State	State
5	Federal Polytechnic Offa	Offa. Kwara State.	Federal
6	Benue State Polytechnic	Ugbokolo. Benue State	State
7	Niger State Polytechnic	Zungeru. Niger State	State
8	Kogi State Polytechnic	Lokoja. Kogi State	State
9	Kwara State Polytechnic	Ilorin. Kwara State.	State
10	Plateau State Polytechnic	Barkin Ladi. Plateau State	State
11	Dorben Polytechnic	Bwari FCT- Abuja	Private
12	Fidei Polytechnic	Gboko. Benue State	Private
13	Nacab Polytechnic	Akwanga. Nasarawa State.	Private

SOURCE: National Board for Technical Education, www.nbte.gov.ng, 2015

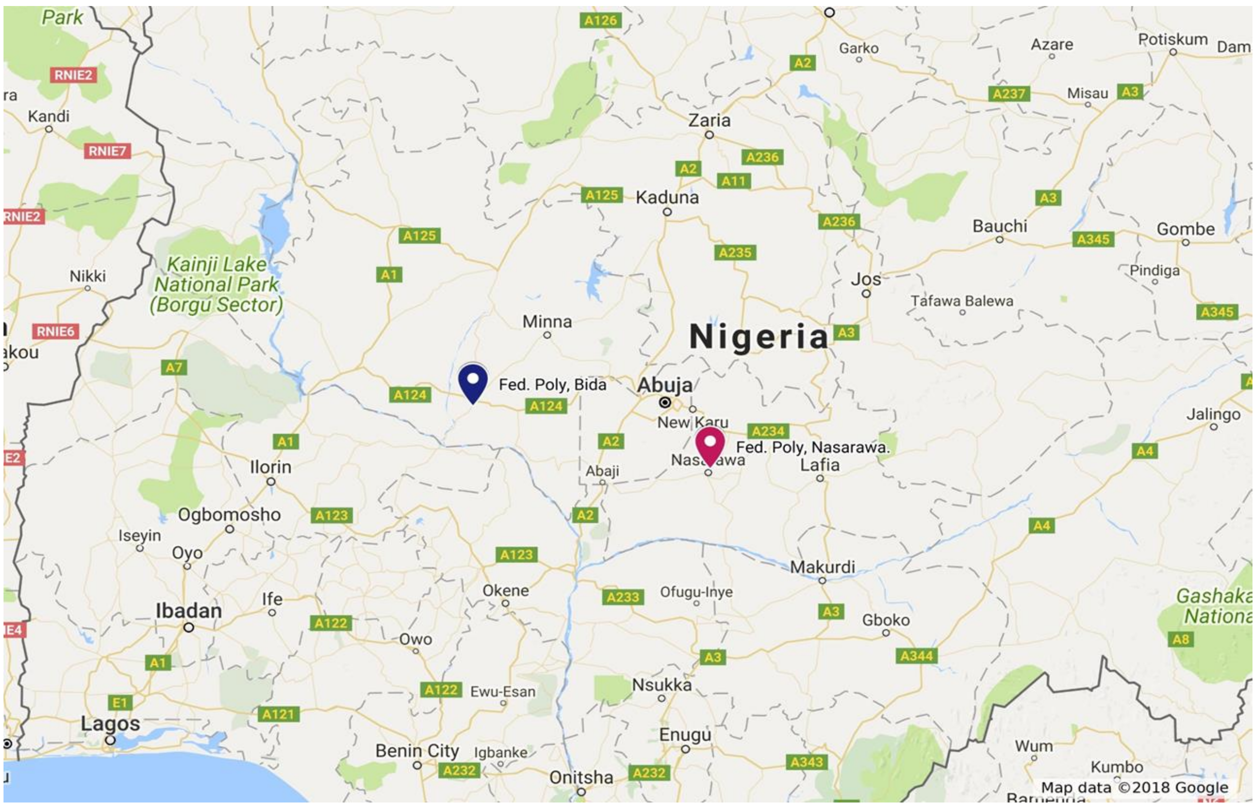
The researcher made contacts with colleagues in seven of the above schools intimating them of interest to conduct the study in their schools. Letters requesting permission to hold the study were sent to six of them.

A multi-stage sampling technique was used for selecting the sample for the quantitative study. Firstly, purposive sampling technique was used to select four polytechnics from the above sample frame of 13 polytechnics in the study area (see Table 3-3). The four polytechnics comprise clusters of two federal and two state government-owned polytechnics. Participating polytechnics were selected based on:

- Having accredited National Diploma in Computer Science
- School type (Public Polytechnics)
- Proximity to the researcher’s base

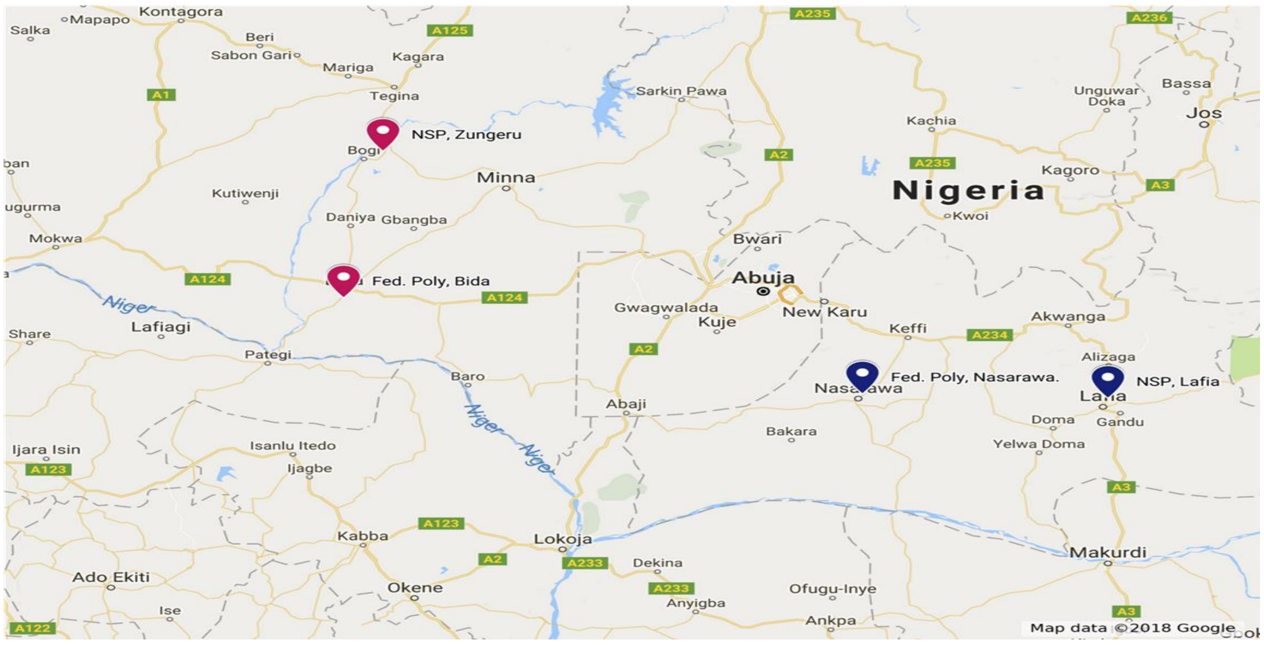
Selected polytechnics were:

- Federal Polytechnic, Bida, Niger State. (FPB)
- Federal Polytechnic, Nasarawa, Nasarawa State. (FPN)
- Niger State Polytechnic, Zungeru. Niger State. (NSPZ)
- Nasarawa State Polytechnic, Lafia, Narasawa State (NSPL)



Key: Red location = experimental site; Blue location = the control site.

Figure 3-1 A Map Showing the Pilot Study Sites



Key: Red location = experimental site; Blue location = the control site.

Figure 3-2 A Map Showing the Main Study Sites

Then selected polytechnics were randomly assigned to the experimental and control groups. Lastly, Coarsened Exact Matching (CEM) was used to randomly select equivalent samples for final analysis.

The study was done in two stages: pilot and main study. After securing permission from the management of selected schools, the pilot study took place in the 2014/2015 session with students from FPN and FPB serving as the experimental and control groups respectively. Permission for the main study was granted by four institutions. The intention was to have two experimental and two control groups. However, the study took place in 2015/2016 session with students from three of the schools. Thus, a newly admitted cohort of first-year computer science students from the NSPZ, formed the experimental group and students of NSPL and a new cohort from FPN formed the control groups. However, data from NSPL were dropped from the study analysis. The decision to drop data from NSPL was informed by detection of widespread collusion in students' responses to the open questions in their post-test, which happened to be the only test that was written in the absence of the researcher. On the set day, the test was rescheduled due to students' involvement in other school's activity. See Table 3-4 below for the breakdown of the participation in the two studies. The table contains the number of participants with complete data i.e., those that participated in answering the initial questionnaire, attempted the pre-test, participated in CS1 instruction, and completed the post-test questions.

Table 3-4 Participants in the study

Study Type	**Constructionist *Scratch* Class (Experimental Group)**			**Conventional CS1 Class (Control Group)**			**Study Total**
	Male	Female	Total	Male	Female	Total	
Pilot	80	36	116	78	42	120	236
Main	76	20	96	59	27	86	182

3.4 Instrumentation

We use instruments in scientific research to measure constructs of interest. In this study, we are measuring mainly CS1 students' basic programming learning or achievement and other associated variables such as prior academic and programming backgrounds, gender, and age. For this purpose, the study made use of the following research instruments to gather the needed data:

1. Treatment Instrument: *Scratch* 2 Offline Editor
2. Test instrument: (see appendix)
   i. Introductory Programming Achievement Test (IPAT)

3. Other Data Gathering Instruments: (see appendix)

i. CS1 Students Profile Questionnaire (CSPROQ)

ii. *Scratch* Class Observation Protocol (SCOP)

3.4.1 Development of Instruments

3.4.1.1 The *Scratch* 2 Offline Editor

This is a free offline version of *Scratch* - a novice educational programming language and an online visual multimedia programming environment developed by Massachuset Institute of Technology (MIT) Media Lab in the USA available at https://*Scratch*.mit.edu. We have chosen to use the offline edition (see Figure 3-3) during and outside classes making it possible for students to develop *Scratch* programs without having to connect to the internet. The issue of internet access is still a resource challenge in Nigeria that can possibly threaten the research outcome. However, students were introduced to the online version so that they can interact with the online community of *Scratch*ers to be able to upload, download and remix *Scratch* projects. This is a crucial aspect of their programming learning in *Scratch*.

Moreover, to make the *Scratch* program culturally relevant to the Nigerian context the additional *Scratch* objects such as audio and video clips, pictures, graphics and images were created and used during lab sessions. For instance, students working on program on the national anthem, use the picture of the Nigerian flag, and recorded audio clip of the Nigerian anthem. These local contents and several others were added to the *Scratch* environment.

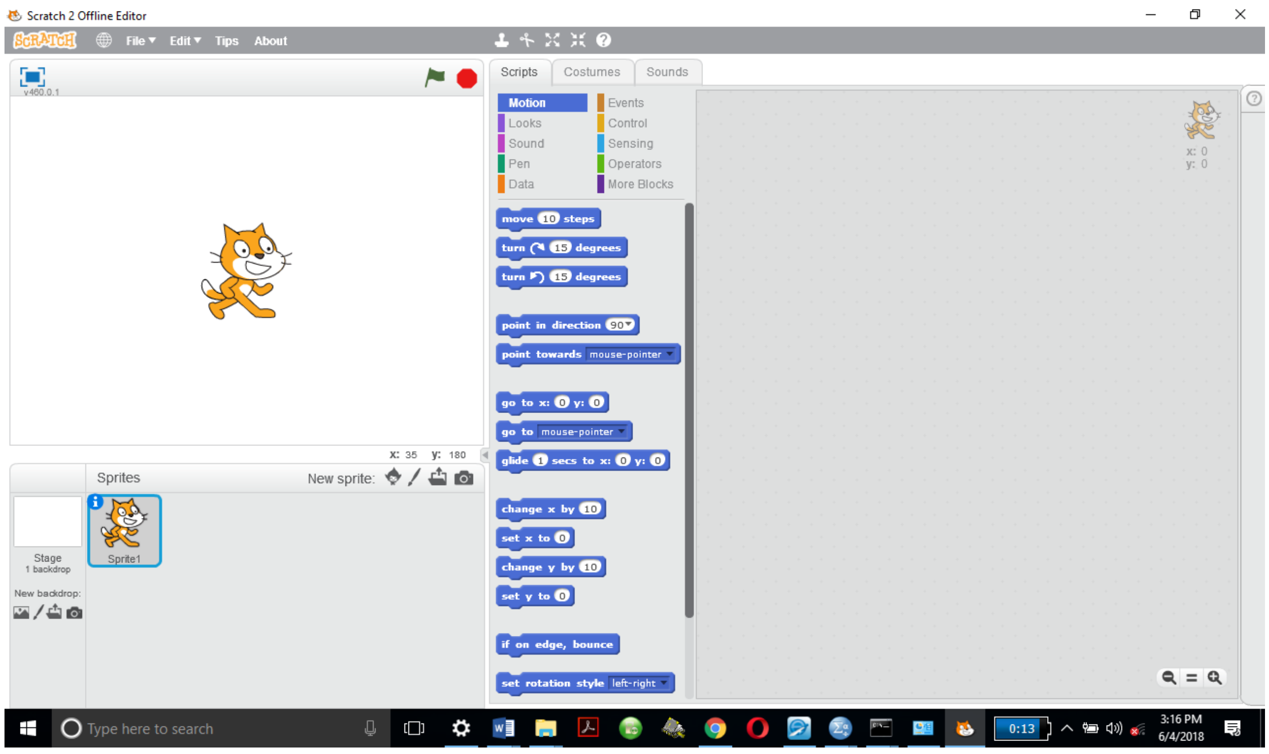


Figure 3-3 Scratch 2.0 programming environment

3.4.1.2 Introductory Programming Achievement Test (IPAT):

Participants’ programming achievement was assessed using IPAT. IPAT is an adapted pre-test instrument from a similar study, conducted in Israel, by Meerbaum-Salant et al.(Meerbaum-Salant et al., 2013). Some modifications were made to the original instrument to make it suitable for the Nigerian context. The changes include limiting concepts questions to those the students will be taught during the six weeks of teaching according to the NBTE syllabus. Also, Nigerian names were used in place of Israeli names in the open questions. In addition, we have moved some items requesting demographic or other information from the subjects in the original instrument into a separate instrument in as stated in the introduction to section 3.4. IPAT was used to measure the students’ baseline programming knowledge serving as the pre-test for the two groups. With some reordering of the questions, it was also used to measure the learning outcomes in both groups serving as post-test after six weeks of teaching. As earlier mentioned, the scope of questions was limited to concepts or topics introduced or were expected to have learnt during the six weeks of programming instruction.

The decision to use IPAT - a non-programming language achievement test –is based on these reasons:

The researcher is interested in measuring programming (or computational thinking) ability of the students not programming language knowledge. Once an algorithm or solution to a programming problem is formulated, coding it in a programming language is straightforward.

The researcher believes programming achievement of students include the conceptual understanding of programming as well as cognitive problem solving or algorithm design (in other word computational thinking) skills.

Therefore, the questions comprised:

- Programming concepts: These are qualitative open-ended questions eliciting students’ recall, knowledge and understanding of basic programming concepts they were taught or expected to have learnt during the six weeks. There are ten questions in this section having 2 marks each. The total mark awarded to this section is 20 marks.

- Programming skills / Problem-solving/Computational thinking questions: According to Özmen and Altun (2014), programming skills refers to students’ ability for, “designing solutions to problems in programming and to determining strategy to be followed while reviewing his/her programming knowledge.”(pg. 15) Similarly, Kothiyal et al. (2013) identified the following as goals of programming education: conceptual understanding, code tracing, designing program logic and writing program. Therefore, these are questions seeking to know the programming or computational thinking skills of the students. students are expected to apply specific domain and programming knowledge to solve the questions. There are three questions in this section with each awarded the total mark of 10. Each question has some sub-questions. the total mark for this section is 30 and the total for the test is 50 marks. Students were given 60 minutes to attempt all the questions. IPAT is available in the appendix.

3.4.1.3 CS1 Students Profile Questionnaire (CSPROQ):

CSPROQ is a questionnaire that was used to gather demographic information as well as academic, programming, and creative arts backgrounds of the students in both the experimental and control group, after registering their consent to participate in the study. CSPROQ was administered before pretesting and exposing the two groups to the independent variables. It consists of four parts: demographic, academic, prior programming, and creative arts experience. The first part contains nominal items such as age group and gender. The second part consists of academic items such as student's grade in the three (3) compulsory Ordinary Level (OL) (i.e., secondary, or high school) subjects for admission of undergraduate computer science students, and Unified Tertiary Matriculation Examination (UTME) score. From these grades, an index for student academic achievement level was automatically computed: namely, 1 for low, 2 for average, and 3 for high achieving. The third part consists of five (5) items asking for students' prior programming background. From data supplied by a participant, an index for experience was automatically derived: namely 0 for none, 1 for some. Lastly, the prior visual art experience part consists of five (5) Likert's scale-like items requesting for the level of students experience with creative arts. From data supplied by a participant, an index for visual art experience was automatically derived: namely, 0 for none, 1 for some.

3.4.2 Validity and Reliability

3.4.2.1 Validating Research Instruments

1. ***Scratch* Editor** was developed by a research group in MIT Media Lab in the US. Suitability of this novice educational environment for introducing students, to programming concepts in a visual multimedia way has been widely reported in the literature. Meerbaum-Salant et al.(Meerbaum-Salant et al., 2013) concluded from their study, introducing middle schools students to *Scratch*, that students learnt some programming or computer science concepts as a result of their exposure to *Scratch*. They, however, observed that students failed to learn core programming concepts such as variables.

Field Trial Validation of *Scratch*: The *Scratch* 2.0 offline editor was pilot tested with polytechnic CS1 students which were not part of the main study. At the end of six weeks of the field trial, the subjects were asked to express their perception of the suitability of *Scratch* to their programming learning experience. Everyone gave positive feedbacks; interesting opinions were made by some academically weak male and female students who had a phobia for programming before enrolling for computer science. These set of students discovered that their exposure to *Scratch* lessened their fears and awakened their self-belief in their ability to learn programming.

2. **Introductory Programming Achievement Test (IPAT)**: IPAT being an adapted instrument was sent to original instrument authors in Israel, and other computer science educators within and outside the country to examine it for the construct, face, and content validities. It was field-tested on some CS1 students who were not part of the main study. The instrument was amended following expert suggestions, criticisms, and remarks, as well as the result of pilot trial with some CS1 students.

3. **CSPROQ**: The CS1 Student Profile Questionnaire was given to science educators to examine it for construct, face, and content validities. Amendments were made following experts remarks. It was also pilot tested with some CS1 students. Subsequently, validation of the instrument was performed using exploratory factor analysis. Thereafter, amendments were made following the results of the analysis.

4. **SCOP**: The *Scratch* Class Observation Protocol was sent to computer or science educators to ascertain its validity. Amendments were made to SCOP following experts' assessments.

3.4.2.2 Reliability of Research Instruments

The reliability of the research instruments, IPAT and CSPROQ were established through a pilot trial of these instruments in single administration with a sample of CS1 students who were part of the population but were not part of the main study sample. For the CSPROQ, reliability was calculated using an R package for computing ordinal alpha (Table 3-5). The ordinal alpha was used rather than Cronbach alpha as the data here are ordinal data. Coefficient of alpha calculated for IPAT instrument is presented in the Table 3-6. Krippendorff Alpha (Inter-rater reliability) test was calculated for SCOP and the result is as shown in Table 3-7.

Table 3-5 CSPROQ's Reliability

Construct	**Items**	**Ordinal Alpha**
Academic background	3	.72
Programming Background	17	.85
Visual Art background	5	.75

Table 3-6 IPAT's Reliability

Cronbach's Alpha	**Cronbach's Alpha Based on Standardized Items**	**N of Items**
.844	.851	27

Table 3-7 SCOP's Reliability

Construct	Items	Krippendorf's Alpha
Affective, Cognitive, Psychomotor and Social impact	20	0.7037 95% CI [.5727, .8133]

3.5 Data Collection

3.5.1 The Pilot Study:

Two polytechnics in the study area gave permissions for the pilot study. At the beginning of 2014/2015 session, I visited the selected polytechnics. This gave me the opportunity to notify the schools of the intention to commence the pilot study, and also inspected the state of facilities needed for the research. Provisions for necessary materials for the study was made. I did seek for understanding and cooperation of staff and students in the departments of computer science in the selected schools, and to conduct training or orientation for research assistants. These activities lasted two weeks.

Before the commencement of teaching, the newly admitted National Diploma First Year (ND1) students in the selected polytechnics were administered the CSPROQ questionnaire as well as IPAT for the pre-test. Some lecturers and I, who teach CS1 in these schools conducted the pre-test. A computer science educator who was not part of the study marked the test using a rubric drawn for assessing students' answers. In addition, the test papers did not contain any personal or group information to enhance the objectivity of assessment.

Subsequently, the experimental and control groups of newly admitted ND1 students in the selected polytechnics, were introduced to CS1 course tagged COM113 in Nigerian polytechnics as indicated in Table 3-8 below. Problem-Based Learning was employed for the two groups. Hence, students were presented, in class and as assignments, real-life programming problems to engage their attention. This lasted six weeks in their first semester. The six weeks of teaching was guided, particularly in the control group, by UNESCO-NBTE curriculum and course outline currently being used in Nigerian polytechnics. On the other hand, the experimental groups were exposed to *Scratch* during the six weeks of teaching following the constructionist educational theory, a student-centred pedagogy where the instructor only facilitates their learning. The same instructor in the two groups led class sessions. I undertook this responsibility of handling weekly class sessions. *Scratch*, to the best of my knowledge, is unknown to fellow CS1 lecturers in Nigeria, and the need to guard against the threat to the validity of the experiment by lecturers having different qualifications and teaching experiences taking the two groups informed the decision to take the class. However, this raises the same potential threat to validity. This is mitigated by me not having a stake in the constructionism as an educational theory or *Scratch* programming language project.

At the end of the six weeks of teaching, the students in the two polytechnics took the post-test by answering questions in the IPAT2. Questions in IPAT2 were the same as those in IPAT1 but were reordered. This was done to make the achievement test appears different from the pre-test to take care of the threat to validity arising from the test instrument. The same computer science educator who was not part of the study marked the post-test using the same rubric as the pre-test.

Fellow CS1 educators in the two polytechnics observed the classes during the six weeks instruction in their respective schools.

Lastly, I interviewed five students from the experimental group to measure their experiences and opinions about programming learning in the *Scratch* environment. Random sampling was used to select the sample for the interview.

Table 3-8 Pedagogical Features of the CS1 intervention

	Pedagogical Characteristics	***Scratch* Class (Experimental group)**	**Traditional CS1 Class (Control group)**
1.	Interactive 2-hour weekly lecture?	No (only demos in *Scratch*)	Yes
2.	Weekly 2-hour lab sessions?	Yes	Yes
3.	Same or Similar lecturers?	Yes	Yes
4.	Weekly programming assignment?	Yes	Yes
5.	Same curricular focus/same topics?	Yes	Yes
6.	Worked examples in class?	Yes	Yes

At the end of the pilot study, a power analysis was conducted with the data to determine the required sample for the main study (see Figure 3-4)

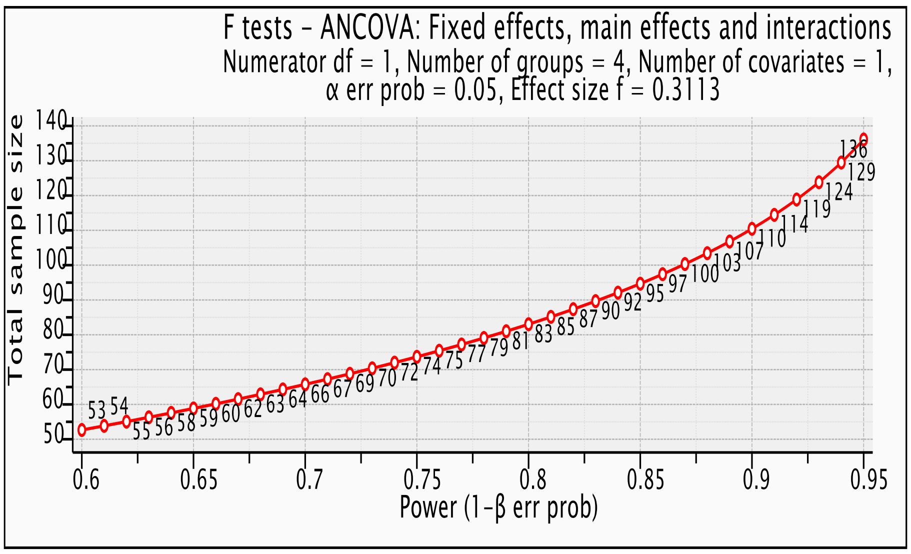


Figure 3-4 A graph of Power Analysis conducted to determine sample for the main study

3.5.2 The Main Study

The main study data collection was conducted as shown in

Figure 3-5 in the 2015/2016 academic session.

Data Collection Map (Main Study)

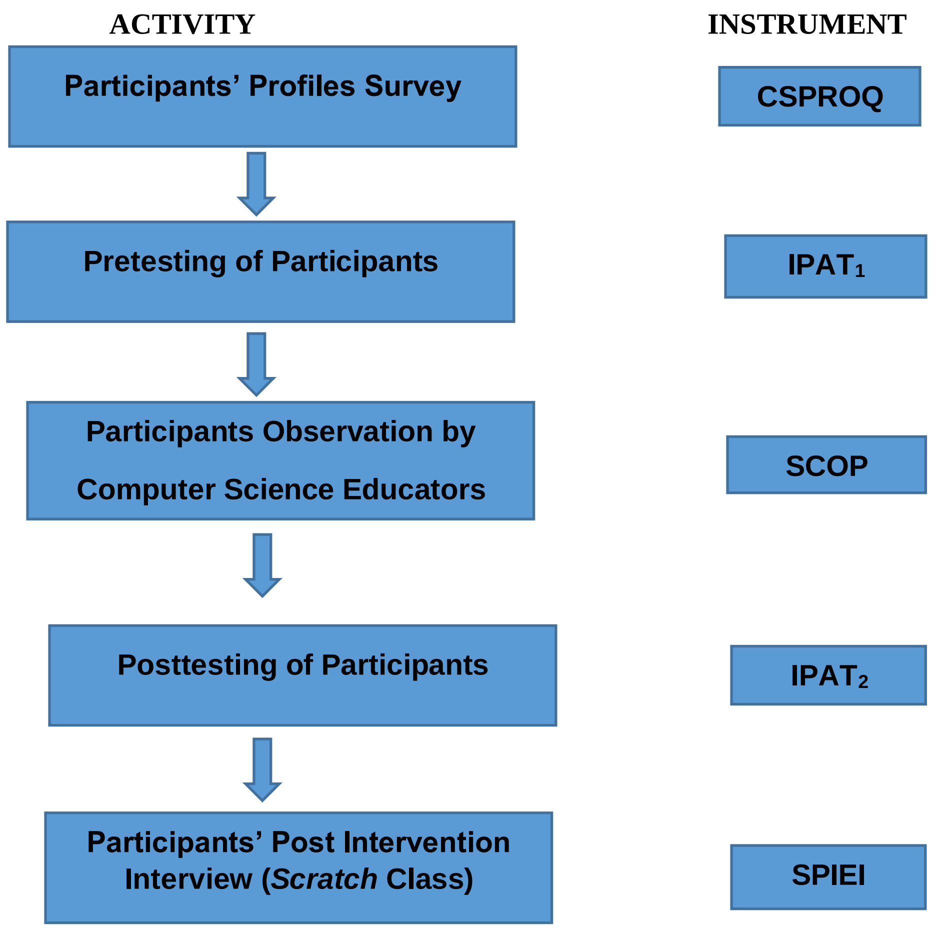

ACTIVITY
INSTRUMENT
Participants' Profiles Survey
CSPROQ
Pretesting of Participants
IPAT1
Participants Observation by Computer Science Educators
SCOP
Posttesting of Participants
IPAT2
Participants' Post Intervention Interview (Scratch Class)
SPIEI

Figure 3-5 The Main Study Data Collection Map

Permission to conduct the study was obtained from five polytechnics in the study area. Out of these five, four were selected for the study using purposive sampling. The four selected were all public institutions in north-central Nigeria with two belonging to the state governments and two federal government institutions. Random sampling was employed, in assigning intact classes of the research cohorts into two experimental and two control groups. However, the study took place in three polytechnics, as one of the earlier scheduled research sites, an experimental group, could not participate owing to disruption in the academic activities as a result of strike action by lecturers. Thus, the study cohorts had one experimental group and two control groups.

At the beginning of the academic session, I visited the participating institutions with the following aims

- notify the schools of the intention to commence the study,

- see the state of facilities needed for the research,
- make provisions for necessary study materials
- seek for understanding and the cooperation of staff and students in the departments of computer science in the selected schools,
- Conduct training or orientation for research assistants. These activities lasted for two weeks.

Before the commencement of teaching, the newly admitted National Diploma first-year (ND1) students in the three participating polytechnics were administered the CSPROQ questionnaire as well as IPAT for the pre-test. In these schools, some CS1 lecturers and I conducted the pre-test. However, a computer science educator who was not part of the study marked the test using a rubric drawn for assessing students' answers. In addition, the test papers did not contain any personal or group information to enhance the objectivity of assessment.

Subsequently, both the experimental and control groups were introduced to CS1 course tagged COM113 in Nigerian polytechnics as indicated in the table. Problem-Based Learning was employed in the two groups. Hence, students were presented real-life programming problems during class and assignments, to engage their attention. This lasted six weeks in first semester.

The six weeks of teaching was guided, particularly in the control group, by UNESCO-NBTE curriculum and course outline 2008 currently being used in Nigerian polytechnics. On the other hand, the experimental group were exposed to *Scratch* during the six weeks of teaching following the constructionist educational theory, a student-centred pedagogy where the instructor only facilitates their learning by demonstrating programming and students developed programs for projects of interest. Examples of students' projects include program that recites the Nigerian national anthem, games, animated stories etc.

Pedagogical features of class sessions were like what we had during the pilot study (see Table 3-8). The CS1 instructors in the two control groups led class sessions while the researcher handled class sessions in the experimental class. This is predicated on the fact that, to the best of the researcher's knowledge, *Scratch* is still unknown to fellow CS1 lecturers.

At the end of the six weeks of teaching, the students in the three polytechnics took the post-test (IPAT2). However, as earlier mentioned in section 3.2, the data from one of the state institutions, a control group, was dropped due to widespread evidence of collusion in the answers provided in the post-test achievement

The same computer science educator who earlier marked the pre-test marked the post-test using the same rubric as the pre-test.

Four computer science (CS) educators in the experimental polytechnic observed the *Scratch* classes during the six weeks of instruction. The validated observation protocol instrument (SCOP) was used to collect the data.

3.6 Data Analysis

Data collected from the administration of research instruments were analysed using Statistical Package for Social Sciences (SPSS) version 23. To answer the research questions and to test the eight hypotheses of the study, the data were analysed using descriptive and inferential statistics. The data from the research questions were analysed using descriptive statistics (mean and standard deviation) while the eight hypotheses were tested using paired samples t test and Analysis of Covariance (ANCOVA) at 0.05 level of significance.

3.7 Ethical Issues

Moral obligation to Christian vocation, professional body memberships, UNISA studentship, UNISA policy on copyright and plagiarism, and responsibility towards the consumers of the research findings made it personally compelling for the researcher to maintain the highest level of research integrity during the study. There was need to maintain a balance and checks on interests as a student or a researcher in search of knowledge and a higher degree, and the interests of other stakeholders in this research study. So, the researcher has made concerted efforts in the bids to ensure that the global and UNISA ethical standards were followed before, during and after the study.

3.7.1 Ethical Clearance

After the research proposal was accepted, the researcher applied for ethical clearance and both the Institute for Science and Technology Education as well as the College of Science, Engineering and Technology ethics committees granted ethical clearance to the study. See the ethics certificates in Appendix.

3.7.2 Informed Consent and Voluntary participation

At the institutional level, permissions for the pilot and main studies were obtained from selected schools (see appendix). In addition, the HODs, CS1 lecturers and lab assistants in departments of computer science of the schools, were fully informed of the details of the studies. Notwithstanding, the researcher gave an adequate briefing to the CS1 students who participated in the study to secure their informed consent without introducing any threat to research credibility due to the Hawthorne effect. Those who were willing to participate signed the informed consent form (see Appendix). In addition, research assistants were given orientation on the ethical requirement of their work during both studies.

3.7.3 Anonymity of research data

The anonymity of research subjects has been ensured in this thesis. The same rule will guide subsequent publications resulting from the study. This will be realized by removing or replacing Personally Identifiable Information (PII) with a pseudonym.

3.7.4 Honesty in handling research data and reporting

The researcher has sought to uphold the integrity required for ethical research by avoiding the use of deception during the research. Honest report of activities and challenges experienced during the study, has been brought to the notice of the supervisor to avoid any dishonest acts. The researcher has gone to the field and collected data rather than resorting to fabricating data. Evidence of this is provided within the thesis and in the appendix. Also, the original data contained in the research instruments used in the study has been kept.

In this thesis, researcher adhered to an honest report of the research, reporting only what was done, found, as wells as limitations, failures, or any information necessary to provide the true picture of this research work.

3.7.5 Plagiarism

In line with upholding the principles of ethical research, the researcher has avoided every act of plagiarism. Proper citations and references of all materials used have been provided in this thesis.

3.8 Limitations of the Study

Chronological: six weeks of teaching perhaps was not sufficient for learning of concepts, needed to develop programming knowledge and skill by novice CS1 students. permission from participating institutions allow their students to take part in the research for just six weeks with the intention of resuming the usual CS1 instruction for the rest of the semester. This limited instruction was taken into cognizance by limiting the test questions in the achievement test to such that the subjects are expected to have learnt during the first six weeks of CS1 instruction.

Environmental: The research took place in resource-constrained environments in which large classes of students are made to learn without adequate resources like seats in the lecture classes, computers in the lab and regular electric power to supply. However, this was a problem common to all the sites during the study.

Pedagogical: Subjects were taught by different instructors with the possibility of differences in the impact of instruction due to the difference in the quantity and quality of instruction. This was mitigated to some extent by involving instructors with the same level of proficiency in teaching CS1 students. Research data from a control group taught by a recent university graduate was eventually dropped due to the reason mentioned earlier.

Methodological: The study did not measure some variables like Social Economic Status and motivation of participants. Such unmeasured variables could affect the equivalence of treatment groups. However, the study tried to mitigate the effects of this limitation by using equivalent random samples generated by Coarsened Exact Matching.

Chapter 4

4 Research Results and Discussion of Findings

This study explored the impact of a constructionist programming intervention, on the achievement of novice computer science students in some Nigerian polytechnics. To realise these goals, the study employed a quasi-experimental design - strengthened, by employing random samples from intact classes derived using Coarsened Exact Matching (CEM) - to confirm study hypotheses. This chapter presents findings from the quantitative data collected during the study. The study was conducted in two stages: the pilot and the main studies.

4.1 The Pilot Study

Pilot study is a preliminary research project during which a researcher tries out a research design, instrument(s) and data collection procedure intended for a main study. This trial enables the researcher to see modifications to be made before embarking on the main study. Table 4-1 presents other definitions of pilot study in research literature.

Table 4-1 What is a Pilot study?

Definition	**Source**
"… a trial run of the major research or a pretest of a particular research instrument or procedure"	Salkind (2010, p. 1032)
"…..a research study that tests the feasibility of an approach that will later be used in a larger study"	(Frey, 2018)
… is the process whereby you try out the research techniques and methods which you have in mind, ."	(Blaxter et al., 2006, p. 137)
"the first defense against oversight (or stupidity) [in research design] and the bias it may invite" (emphasis mine)	(Sally Fincher & Petre, 2005)

Researchers have several motives for embarking upon pilot study. Some of these reasons mentioned in educational research literature include:

- "To establish content validity of scores on an instrument and to improve questions, format and scales" (Creswell, 2014, pg. 257)
- "To get the bugs out of the instrument so that respondents in your main study will experience no difficulty in completing it." (Bell, 2014)

From the foregoing, it is evident that pilot study may lead to better research study as grey areas in the research design discovered during the pilot are addressed in the main study.

In this research, the pilot study enlisted two cohorts of newly admitted computer science students from two Nigerian polytechnics located in two states in north central Nigeria during the 2014/2015 session.

4.1.1 Demographics of pilot study participants

The information about the participants is presented in Table 4-2 to Table 4-8

Table 4-2 presents two cohorts of first-year computer science students from the study area in north central Nigeria. The number shown represents participants that provided responses to all the data collection instruments used

Table 4-2 Pilot Study Participants by School (N= 236)

School	**Treatment Grouping**	**No of Participants**
Federal Polytechnic, Bida	Control	116
Federal Polytechnic, Nasarawa	Experimental	120
Total		236

Table 4-3 provides the gender mix in the two groups. Gender spread is similar in both groups; the ratio of male to female is about 2:1.

Table 4-3 Gender of Pilot Study Participants (N =236)

Treatment Grouping	**Gender**		**Total**
	Male n (%)	**Female n (%)**	
Control	78 (65.0)	42 (35.0)	120(100)
Experimental	80 (69.0)	36 (31.0)	116 (100)
Total	158(66.9)	78 (33.1)	236 (100)

Table 4-4 presents the age groups of participants in the two groups. In the control and experimental groups, only 8.3% and 13.8% respectively are between 16 and 18 years. In both groups, approximately half of the participants are between 19 and 21 years. College entry age in Nigeria is 16 years. Since fresh high school-leavers normally belong to the first age group (16-18), and the fact that in each group almost ninety percent of participants are above 18 years, this suggests that most participants have left secondary schools long before gaining admission into the CS programme.

Table 4-4 Age of Pilot Study Participants

Treatment Grouping	Age Group					Total
	16 - 18 n(%)	19-21 n(%)	22-24 n(%)	> 24 n(%)	Others n(%)	
Control	10 (8.3)	57 (47.5)	40 (33.3)	12(10.0)	1 (0.8)	120
Experimental	16 (13.8)	64 (55.2)	27 (23.3)	9 (7.7)	0 (0.0)	116
Total	26 (11.0)	121(51.3)	67(28.4)	21(8.9)	1(.4)	236

Academic achievement background presented in Table 4-5 is a computed index from two self-reported results by each participant: first, the final high school results in three compulsory subjects for admission into computer science, namely, English, Mathematics and Physics, and second, the Unified Tertiary Matriculation Exam (UTME) Score. The index goes from 1-3, for low, average, and high-achievement levels respectively. Participants in both groups have similar academic profiles. In both groups, about 68% are in the low-achieving category. Similarly, the average achieving students are approximately 30% of the participants in both groups. This suggests that both groups have roughly the same academic strength.

Table 4-5 Academic Background of Pilot Study Participants

Treatment Grouping	Academic Background Index			Total
	Low-Achieving n(%)	Average Achieving n(%)	High-Achieving n(%)	
Control	81(67.5)	33(27.5)	6 (5.0)	120
Experimental	79(68.1)	35 (30.2)	2 (1.7)	116
Total	160(67.8)	68(28.8)	8(3.4)	236

Two questions in the CSPROQ questionnaire ask if participants have learnt or written programs in any language. Results of responses to both questions are given in Tables 4-6 and 4-7 respectively. Information in Table 4-6 indicates that both groups lack background in prior programming learning.

Table 4-6 Prior Programming Learning of Pilot Study Participants

Treatment Grouping	**Prior Programming Learning**		**Total**
	No Background n(%)	**Some Background n(%)**	
Control	104 (86.7)	16 (13.3)	120
Experimental	100 (86.2)	16 (13.8)	116
Total	204(84.4)	32(13.6)	236

Responses by participants in both groups as shown in Table 4-7 also indicates that most of them had no experience with writing programs before they enrolled for a course in computer science. However, the control group had a higher number of participants (10.0%) who had prior program writing experience compared to the experimental group with 4.3% of the participants.

Table 4-7 Prior Program Writing of Pilot Study Participants

Treatment Grouping	**Prior Program Writing**		**Total**
	No Background n(%)	**Some Background n(%)**	
Control	108(90.0)	12 (10.0)	120
Experimental	111 (95.7)	5 (4.3)	116
Total	219(92.8)	17(7.2)	236

Visual art background is a computed measure from each participant's responses to questions, probing their earlier experiences with exercising their creativity in making, tinkering, or playing with artefacts either manually or on the computer before they enrolled into computer science. From their responses as indicated in Table 4-8, most participants in both groups have had some experiences or exposures with creative works of arts, though the experimental group had higher percentage (76.7) compared to that of the control (66.7).

Table 4-8 Visual Art background of Pilot Study Participants

Treatment Grouping	**Visual Art Background**		**Total**
	No Background n(%)	**Some Background n(%)**	
Control	40 (33.3)	80 (66.7)	120
Experimental	27 (23.3)	89 (76.7)	116
Total	67(28.4)	169(71.6)	236

Table 4-9 presents the t-test statistics for ascertaining if there is any difference in the mean baseline scores of both groups. The result indicates a significant mean difference between the groups.

4.1.2 Pilot Study: Pre-test Results

Table 4-9 Pilot Study Pre-test Achievement Performance

Group	N	Mean	S.D.	S.E.	t value	Significance
Control	116	21.20	12.88	1.18	-2.212	p = .035*
Experimental	120	24.78	13.01	1.21		

*mean difference is significant.

Table 4-10 provides a view into baseline performance of the two groups by gender. Clearly, male participants had better mean scores than their female counterparts in both groups. .

Table 4-10 Pre-test Scores of Pilot Study Participants based on Gender

Treatment Grouping	Gender	Mean	N	Std. Deviation	Std. Error of Mean
Control	Male	22.42	78	13.52	1.53
	Female	18.93	42	11.39	1.76
	Total	21.20	120	12.88	1.18
Experimental	Male	25.82	80	12.77	1.43
	Female	22.44	36	13.43	2.24
	Total	24.78	116	13.01	1.21
Total	Male	24.15	158	13.21	1.05
	Female	20.55	78	12.42	1.41
	Total	22.96	236	13.04	.85

Table 4-11 reveals that baseline performances of participants generally follow their prior academic achievement in both groups. Surprisingly, the high-achieving participants mean pre-test scores were lowest in both groups.

Table 4-11 Pretest Scores of Pilot Study Participants based on Academic Background

Treatment Grouping	Academic Background	Mean	N	Std. Dev.	Std. Error
Control	Low-Achieving	19.22	81	11.47	1.27
	Average Achieving	26.45	33	15.45	2.69
	High-Achieving	19.00	6	7.97	3.26
	Total	21.20	120	12.88	1.18
Experimental	Low-Achieving	25.95	79	13.50	1.52
	Average Achieving	22.29	35	11.97	2.02
	High-Achieving	22.00	2	5.66	4.00
	Total	24.78	116	13.01	1.21
Total	Low-Achieving	22.54	160	12.92	1.02
	Average Achieving	24.31	68	13.83	1.68
	High-Achieving	19.75	8	7.21	2.55
	Total	22.96	236	13.04	.85

Participants' performance in the pre-test presented in Table 4-12 shows similar patterns in both groups: participants with prior program writing had higher means scores than those without background.

Table 4-12 Pre-test Scores of Pilot Study Participants based on Prior Program Writing

Treatment Grouping	Prior Program Writing	Mean	N	Std. Deviation	Std. Error of Mean
Control	No Background	20.56	108	12.47	1.20
	Some Background	27.00	12	15.53	4.48
	Total	21.20	120	12.88	1.18
Experimental	No Background	24.22	111	12.57	1.19
	Some Background	37.20	5	17.92	8.01
	Total	24.78	116	13.01	1.21
Total	No Background	22.41	219	12.62	.85
	Some Background	30.00	17	16.40	3.98
	Total	22.96	236	13.04	.85

Participants' performance in the pre-test based on prior visual art, presented in Table 4-13, also shows similar patterns as previous performance based on prior program writing: those with prior visual art had higher mean scores than those with no background.

Table 4-13 Pretest Scores of Pilot Study Participants based on Visual Art Background

Treatment Grouping	Visual Art Background	Mean	N	Std. Dev.	Std. Error of Mean
Control	No Background	19.95	40	12.65	2.00
	Some Background	21.83	80	13.02	1.46
	Total	21.20	120	12.88	1.18
Experimental	No Background	20.59	27	11.75	2.26
	Some Background	26.04	89	13.17	1.40
	Total	24.78	116	13.01	1.21
Total	No Background	20.21	67	12.21	1.49
	Some Background	24.05	169	13.23	1.02
	Total	22.96	236	13.04	.85

4.1.3 Summary

From the foregoing, we observed that while there are similarities in the profiles of the two cohorts of CS1 students, some differences do exist. For instance, we can see that the experimental group had significantly higher mean baseline score compared to that of the control group. Such difference, threatens the validity of the causal inference we can make from this intervention. We would prefer our two cohorts to be same with no significant difference between them. Then, we may attribute the outcome of study to the experimental intervention we have applied.

4.1.4 Pilot Study: Post-test Results

After taking the pre-test, the participants went through six weeks of programming instruction in two ways: the control class had the conventional direct instruction-based pedagogy, using a textual programming environment (Visual Basic), while the experimental class had the constructionist discovery-learning instruction, using a block-based programming environment (*Scratch*). Subsequently, participants in both groups took the same questions they had in the pre-test with some reordering in the post-test to measure learning had taken place. The results of the post-test for the two groups are presented in this section

Table 4-14 presents the results from independence samples t-test analysis, to determine if there was significant difference in the post-test achievement scores between the two groups. The value of p < 0001 clearly suggests that significant difference exists in the mean post-test score of the experimental and control groups. Negative value of the t statistic reveals that the mean post-test score for the control group was significantly lesser than the mean score of the experimental group.

Table 4-14 Pilot Study Participants Post-test Achievement Performance

Grouping	**N**	**Mean**	**Std. Deviation**	**Std. Error Mean**	**t value**	**Significance**
Control	120	37.37	18.06	1.65	-4.76	*p <.0001
Experimental	116	48.59	18.13	1.68		

* mean difference is significant.

To see the nature of learning gains by participants in the two groups, another independent samples t analysis was conducted with the gain score as the dependent variable. The result presented on Table 4-15 reveals that the experimental group made significant learning gains compared to the control group.

Table 4-15 Pilot Study Learning Gain Score Achievement Performance

Group	**N**	**Mean**	**S.D.**	**S.E.**	**t value**	**Significance**
Control	120	16.17	13.33	1.22	-3.927	*p <.0001
Experimental	116	23.81	16.45	1.53		

* mean difference is significant.

Table 4-16 provides a descriptive statistic on the post-test achievements of participants in the two groups along gender lines. While the mean post-test scores of male and female participants, are almost the same in the control groups, the female participants in the experimental class had slightly higher mean post-test score compared to male participants.

Table 4-16 Post-test Scores of Pilot Study Participants based on Gender

Treatment Grouping	Gender	Mean	N	Std. Dev.	Std. Error of Mean
Control	Male	37.79	78	18.64	2.11
	Female	36.57	42	17.12	2.64
	Total	37.37	120	18.06	1.65
Experimental	Male	47.88	80	18.71	2.09
	Female	50.17	36	16.91	2.82
	Total	48.59	116	18.13	1.68
Total	Male	42.90	158	19.29	1.53
	Female	42.85	78	18.24	2.07
	Total	42.88	236	18.91	1.23

Table 4-17 reveals that the participants belonging to the three academic achievement levels in the experimental group had higher mean post-test scores than participants in related levels in the control group. Strangely, the mean post-test score for high-achieving students records lowest in the control group. Looking at both the mean and standard deviation for the high achieving students in the control group, it is evident this data will contribute to problems of outliers in our research data.

Table 4-17 Post-test Scores of Study Participants based on Academic Background

Treatment Grouping	Academic Background	Mean	N	Std. Dev.	Std. Err. of Mean
Control	Low-achieving	36.79	81	18.97	2.11
	Average-achieving	40.61	33	16.68	2.90
	High-achieving	27.33	6	5.16	2.11
	Total	37.37	120	18.06	1.65
Experimental	Low-achieving	48.96	79	18.91	2.13
	Average-achieving	47.60	35	16.64	2.81
	High-achieving	51.00	2	21.21	15.00
	Total	48.59	116	18.13	1.68
Total	Low-achieving	42.80	160	19.84	1.57
	Average-achieving	44.21	68	16.91	2.05
	High-achieving	33.25	8	14.26	5.04
	Total	42.88	236	18.91	1.23

Table 4-18 presents the comparative information about the performance of the two groups based on their prior programming experience. As expected, participants with prior programming experience in both groups scored higher in the post-test than their colleagues with no prior experience in writing programming.

Table 4-18 Post-test Scores of Pilot Study Participants based on Prior Program Writing

Treatment Grouping	*Prior Program Writing*	*Mean*	*N*	*Std. Deviation*	*Std. Err. of Mean*
Control	No Background	36.89	108	18.14	1.75
	Some Background	41.67	12	17.43	5.03
	Total	37.37	120	18.06	1.65
Experimental	No Background	48.52	111	18.02	1.71
	Some Background	50.00	5	22.85	10.22
	Total	48.59	116	18.13	1.68
Total	No Background	42.79	219	18.96	1.28
	Some Background	44.12	17	18.83	4.57
	Total	42.88	236	18.91	1.23

Visual art background measures student's experience with creative art works with or without the computer. Prominent computer science figures like Donald Knuth, Edsger Dijkstra, and Grace Hopper have hinted that programming involves some art (Booch, 2019). The guess here is that those with some background in artistic creations or compositions, are likely to score higher in programming achievement. Information in Table 4-19 seems to support such assumption. However, while there appears to be no significant difference in performance between those with or no background in visual art in the control group, the performance of participants with background in visual art is significantly higher than those without background in the experimental group.

Table 4-19 Post-test Scores of Pilot Study Participants based on Visual Art Background

Treatment Grouping	**Visual Art Background**	**Mean**	**N**	**Std. Dev.**	**Std. Error of Mean**
Control	No Background	36.60	40	16.38	2.59
	Some Background	37.75	80	18.93	2.12
	Total	37.37	120	18.06	1.65
Experimental	No Background	40.89	27	17.52	3.37
	Some Background	50.92	89	17.75	1.88
	Total	48.59	116	18.13	1.68
Total	No Background	38.33	67	16.85	2.06
	Some Background	44.69	169	19.42	1.49
	Total	42.88	236	18.91	1.23

4.1.5 Matching for Valid Causal Inference

To address the problem of significant differences (as revealed by prior descriptive statistics) in the two treatment groups, data from the pilot study was pre-processed using Coarsened Exact Matching (CEM). CEM is a free SPSS add-in available at https://projects.iq.harvard.edu/cem-spss/pages/installation. After installation, CEM resides in the *Analyze* menu in an SPSS program.

Another reason for matching, is to mitigate the weakness of ANCOVA analysis when intact groups are used and there was non-random assignment to groups, making ANCOVA results and its interpretation prone to errors (Miller & Chapman, 2001).

The pilot study data were processed using CEM and the results presented by subsequent tables and figures show the effect of this pre-processing on the research data.

Figure 4-1 presents the box plots showing the pre-test scores for both groups before CEM. You will observe more outliers in the control group than the experimental class. Another problem is the mean score for the experimental class, which is significantly higher than the mean score for the control. This apparently revealed both classes are not comparable. Without equivalence of the two groups, we cannot conclude that the intervention, rather than this prior difference, is responsible for their final performance in the post-test.

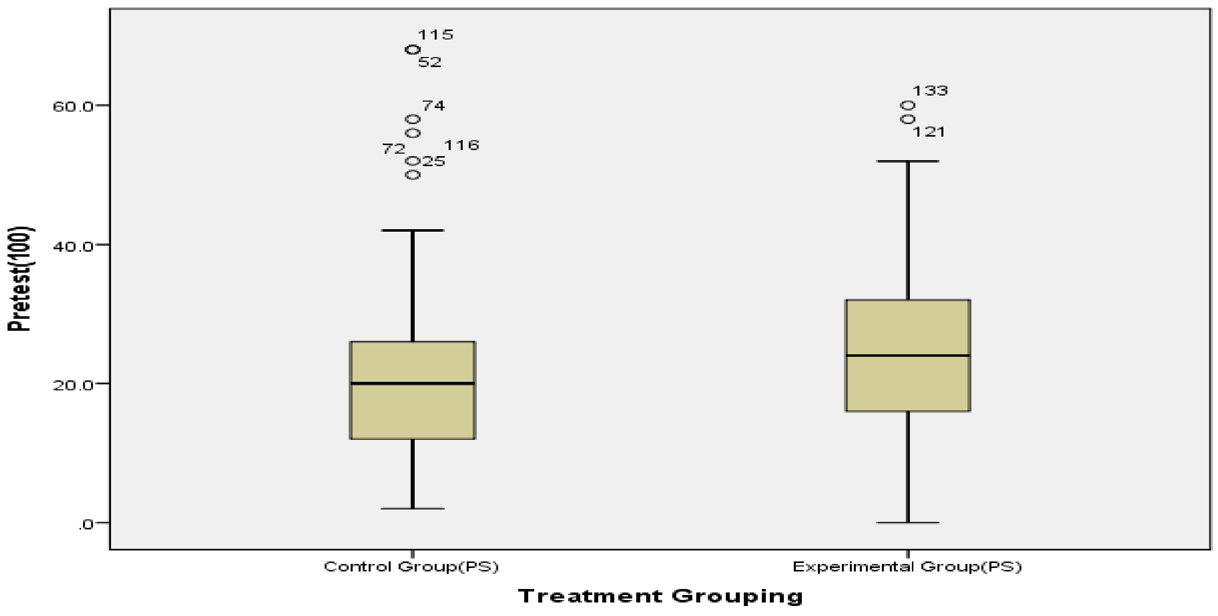


Figure 4-1 Boxplot showing the problems of outliers in pre-test data before matching

Figure 4-2 presents the box plot, for the matched data samples for the two treatment groups. A visual inspection of the figure suggests similarity in the two sample. The outliers' problem in previous data sets has disappeared. Interestingly, there is a reversal in the mean pre-test score: the mean pre-test score for the control is now slightly higher than that of the experimental class. However, as shown by the t-test analysis result in Table 4-20, this present difference in the pre-test score of both groups is not significant. You will recall that before matching, the mean difference in their pre-test score was significant. Thus, CEM matching seems to have produced comparable samples or data sets from which we may make valid causal inference at the end.

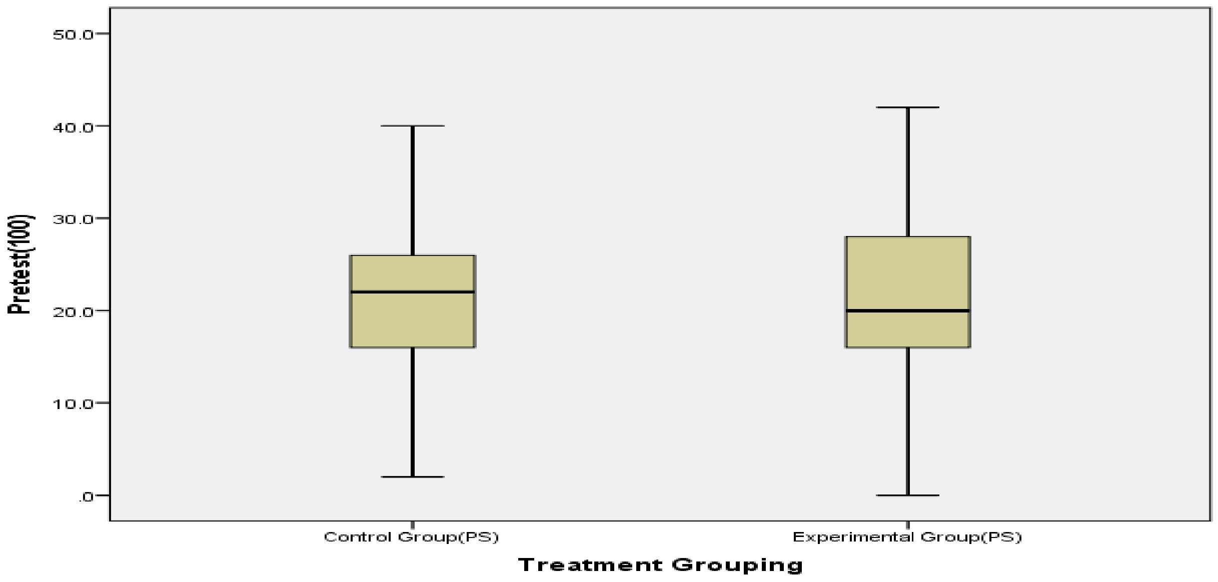


Figure 4-2 Boxplot showing disappearance of outliers in pre-test data after matching

4.1.6 Pilot Study: Descriptive Statistics of Matched Sample

Table 4-20 Pilot Study Pretest Achievement Performance of Matched Sample

Group	N	Mean	S.D.	S.E.	t value	Significance
Control	41	22.05	8.84	1.38	.144	*p = .886
Experimental	41	21.76	9.60	1.50		

*Note: mean difference is no longer significant.

As shown in Table 4-21, the gender mix in the two group is similar. With this, we hope to mitigate against threat to validity of the research result, as one factor that may contribute to mean difference between the two groups is kept constant.

Table 4-21 Gender of Matched Sample Pilot Study Participants (n = 82)

Treatment Grouping	Gender		Total
	Male n (%)	Female n(%)	
Control	28 (68.3)	13 (31.7)	41
Experimental	28 (68.3)	13 (31.7)	41
Total	56(68.3)	26(31.7)	82

Like what we observe in the previous table for gender, information in Table 4-22 shows that we have a 100% balance in the matching of the two samples based on the age groups of participants in the study. We have the same results with the matching based on other covariates as presented in Tables 4-23, 4-24, and 4-25 for academic background, prior program writing, and visual art background respectively. Note, in Table 4-23, both groups have only low, average, and no high-achieving participants. Similarly, Table 4-24 shows samples with no participants with prior program writing. The original unmatched samples as shown in Table 4-5 and Table 4-7 reveal that most of the participants were low or average-achieving in their academic level with no prior program writing experience.

Table 4-22 Age of Matched Sample Pilot Study Participants

Treatment Grouping	**Age Group**				**Total**
	16 – 18 n (%)	**19-21 n(%)**	**22-24 n(%)**	**> 24 n (%)**	
Control	4 (9.8)	24 (58.5)	12 (29.3)	1(2.4)	41
Experimental	4 (9.8)	24 (58.5)	12 (29.3)	1 (2.4)	41
Total	8(9.8)	48(58.5)	24(29.3)	2(2.4)	82

Table 4-23 Academic Background of Matched Sample Pilot Study Participants

Treatment Grouping	**Academic Background Index**		**Total**
	Low-Achieving n(%)	**Average Achieving n(%)**	
Control	32 (78.0)	9 (22.0)	41
Experimental	32 (78.0)	9 (22.0)	41
Total	64(78.0)	18(22.0)	82

Table 4-24 Prior Program Writing of Matched Sample Pilot Study Participants

Treatment Grouping	**Prior Program Writing**	**Total**
	No Background n(%)	
Control	41 (100.0)	41
Experimental	41 (100.0)	41
Total	82(100.0)	82

Table 4-25 Visual Art background of Matched Sample Pilot Study Participants

Treatment Grouping	Visual Art Background		Total
	No Background n(%)	Some Background n(%)	
Control	7 (17.1)	34 (82.9)	41
Experimental	7 (17.1)	34 (82.9)	41
Total	14(17.1)	68(82.9)	82

4.1.7 Pre-test Results after Matching

Pre-test scores of both groups in Tables 4-26, 4-27, 4-28, and 4-29 present further insights into the baseline equality of the groups, based on the various covariates chosen in the CEM analysis.

Table 4-26 Pretest Scores of Matched Sample based on Gender

Treatment Grouping	Gender	Mean	N	Std. Deviation	Std. Err. of Mean
Control Group	Male	22.286	28	8.3038	1.5693
	Female	21.538	13	10.2357	2.8389
	Total	22.049	41	8.8373	1.3802
Experimental Group	Male	22.214	28	9.4960	1.7946
	Female	20.769	13	10.1502	2.8151
	Total	21.756	41	9.6041	1.4999
Total	Male	22.250	56	8.8384	1.1811
	Female	21.154	26	9.9948	1.9601
	Total	21.902	82	9.1727	1.0130

Table 4-27 Pre-test Scores of Matched Sample based on Academic Background

Treatment Grouping	Academic Background	Mean	N	Std. Deviation	Std. Err. of Mean
Control Group	Low-achieving	21.750	32	9.1546	1.6183
	Average-achieving	23.111	9	8.0069	2.6690
	Total	22.049	41	8.8373	1.3802
Experimental Group	Low-achieving	21.813	32	9.7631	1.7259
	Average-achieving	21.556	9	9.5801	3.1934
	Total	21.756	41	9.6041	1.4999
Total	Low-achieving	21.781	64	9.3884	1.1735
	Average-achieving	22.333	18	8.6023	2.0276
	Total	21.902	82	9.1727	1.0130

Table 4-28 Pre-test Scores of Matched Sample based on Prior Program Writing

Treatment Grouping	Prior Program Writing	Mean	N	Std. Deviation	Std. Error
Control Group	No Background	22.049	41	8.8373	1.3802
	Total	22.049	41	8.8373	1.3802
Experimental Group	No Background	21.756	41	9.6041	1.4999
	Total	21.756	41	9.6041	1.4999
Total	No Background	21.902	82	9.1727	1.0130
	Total	21.902	82	9.1727	1.0130

Table 4-29 Pre-test Scores of Matched Sample Based on Prior Visual Art

Treatment Grouping	**Prior Visual Art**	**Mean**	**N**	**Std. Deviation**	**Std. Err. of Mean**
Control Group	No Background	21.143	7	9.1548	3.4602
	Some Background	22.235	34	8.9003	1.5264
	Total	22.049	41	8.8373	1.3802
Experimental Group	No Background	21.714	7	9.6904	3.6626
	Some Background	21.765	34	9.7330	1.6692
	Total	21.756	41	9.6041	1.4999
Total	No Background	21.429	14	9.0615	2.4218
	Some Background	22.000	68	9.2591	1.1228
	Total	21.902	82	9.1727	1.0130

4.1.8 Summary of Pre-test Results of the Matched Sample

From the data and statistics presented above, we can assume that the two treatment groups are similar. This implies, except for bias or errors, that may be due to omissions of relevant covariates and other methodological flaws in the study, the stage is set to make valid inference from the study.

4.1.9 Post-test Results after Matching

As Table 4-30 shows, there is significant mean post-test difference (t = -2.49, p = 0.015) between the two groups. Compared with the information on Table 4-20, we observe a reversal in the post-test achievement: the experimental group has significantly higher mean post-test score. Recall that analysis of data from unmatched samples, also gave the same result (see Table 4-14)

Table 4-30 Pilot Study Matched Sample Posttest Achievement Performance

Grouping	**N**	**Mean**	**Std. Deviation**	**Std. Error Mean**	**t value**	**Significance**
Control	41	39.61	17.60	2.75	-2.49	*p =0.015
Experimental	41	48.88	16.07	2.51		

* mean difference is significant.

Table 4-31 reveals that female students in each group, had slightly higher post-test scores compared to their male counterparts.

Table 4-31 Post-test Scores of Matched Sample based on Gender

Treatment Grouping	Gender	Mean	N	Std. Deviation	Std. Error of Mean
Control Group	Male	38.857	28	19.3864	3.6637
	Female	41.231	13	13.5287	3.7522
	Total	39.610	41	17.6024	2.7490
Experimental Group	Male	48.714	28	16.1127	3.0450
	Female	49.231	13	16.6240	4.6107
	Total	48.878	41	16.0689	2.5095
Total	Male	43.786	56	18.3489	2.4520
	Female	45.231	26	15.3995	3.0201
	Total	44.244	82	17.3856	1.9199

Unsurprisingly, while average achieving students had slightly higher post-test performance, in comparison with the low-achieving students in the *Scratch* class, the performance of the two strata were the same in the control class (see Table 4-32).

Table 4-32 Post-test Scores of Matched Sample based on Academic Background

Treatment Grouping	**Academic Background**	**Mean**	**N**	**Std. Deviation**	**Std. Err. of Mean**
Control Group	Low-achieving	39.688	32	17.9469	3.1726
	Average achieving	39.333	9	17.3494	5.7831
	Total	39.610	41	17.6024	2.7490
Experimental Group	Low achieving	48.000	32	16.1764	2.8596
	Average achieving	52.000	9	16.2173	5.4058
	Total	48.878	41	16.0689	2.5095
Total	Low achieving	43.844	64	17.4585	2.1823
	Average achieving	45.667	18	17.5466	4.1358
	Total	44.244	82	17.3856	1.9199

Since both groups have one stratum, Table 4-33 presents what we already knew about the post-test performance of both classes.

Table 4-33 Post-test Scores of Matched Sample based on Prior Program Writing

Treatment Grouping	Prior Program Writing	Mean	N	Std. Dev.	Std. Err. of Mean
Control Group	No Background	39.610	41	17.6024	2.7490
	Total	39.610	41	17.6024	2.7490
Experimental Group	No Background	48.878	41	16.0689	2.5095
	Total	48.878	41	16.0689	2.5095
Total	No Background	44.244	82	17.3856	1.9199
	Total	44.244	82	17.3856	1.9199

Table 4-34 shows similar mix of students in both groups. While those with prior experience had significantly higher post-test compare to those with no prior experience in the *Scratch* class, students with some prior experience in visual art had slightly lower post-test performance compared to those with no prior experience in the control group.

Table 4-34 Post-test Scores of Matched Sample based on Prior Visual Art

Treatment Grouping	Prior Visual Art	Mean	N	Std. Deviation	Std. Err. of Mean
Control Group	No Background	41.143	7	12.8508	4.8571
	Some Background	39.294	34	18.5726	3.1852
	Total	39.610	41	17.6024	2.7490
Experimental Group	No Background	40.857	7	19.1784	7.2487
	Some Background	50.529	34	15.1542	2.5989
	Total	48.878	41	16.0689	2.5095
Total	No Background	41.000	14	15.6844	4.1918
	Some Background	44.912	68	17.7493	2.1524
	Total	44.244	82	17.3856	1.9199

4.1.10 Testing Assumptions for Inferential Statistics

To test research hypotheses, the researcher employed two inferential statistical analysis namely: paired sample t-test and Analysis of Covariance (ANCOVA). Both statistical analyses require that research data meet some conditions for proper conduct of the analysis and interpretation of the results. We present the results of tests conducted for conditions required for both analysis in this section.

4.1.10.1 Assumptions for Paired Sample t-Test

Some general and specific assumptions that research data must fulfil before conducting a paired t-test according to (Field, 2018; Pallant, 2016) include:

- Homogeneity of variance

- Normality of pre-test, post-test, and difference scores
- Homogeneity of variance

Information in Table 4-35 indicates that both pre-test and post-test scores of participants, have not violated the assumption of homogeneity of variances (p >0.05).

Table 4-35 Test of Homogeneity of Variance

	Levene Statistic	**df1**	**df2**	**Sig.**
Pre-test	.480	1	80	.490
Post-test	.610	1	80	.437

You can visualize this same information for both classes in the study, by looking at figures Figure 4-3 and Figure 4-4.

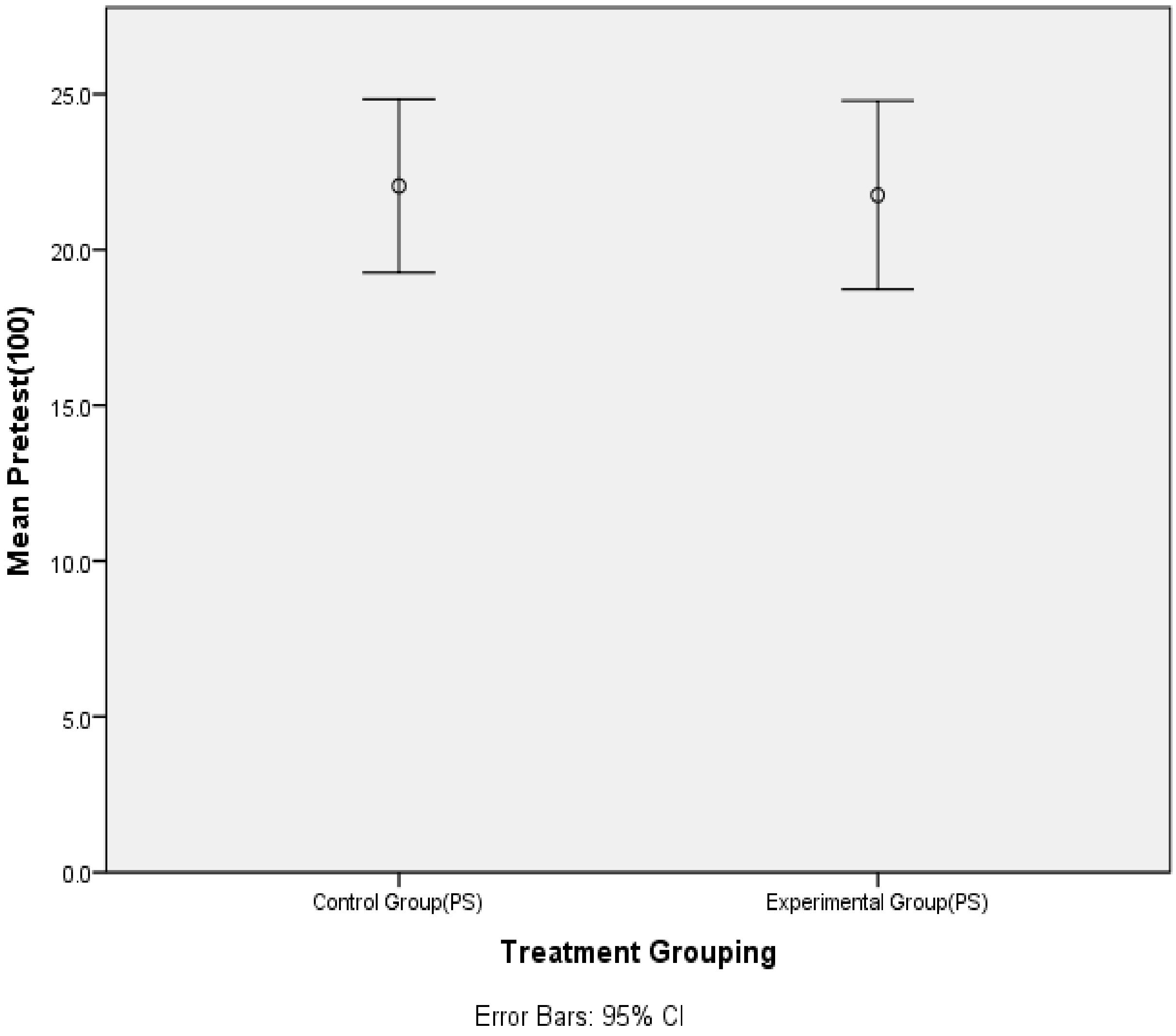
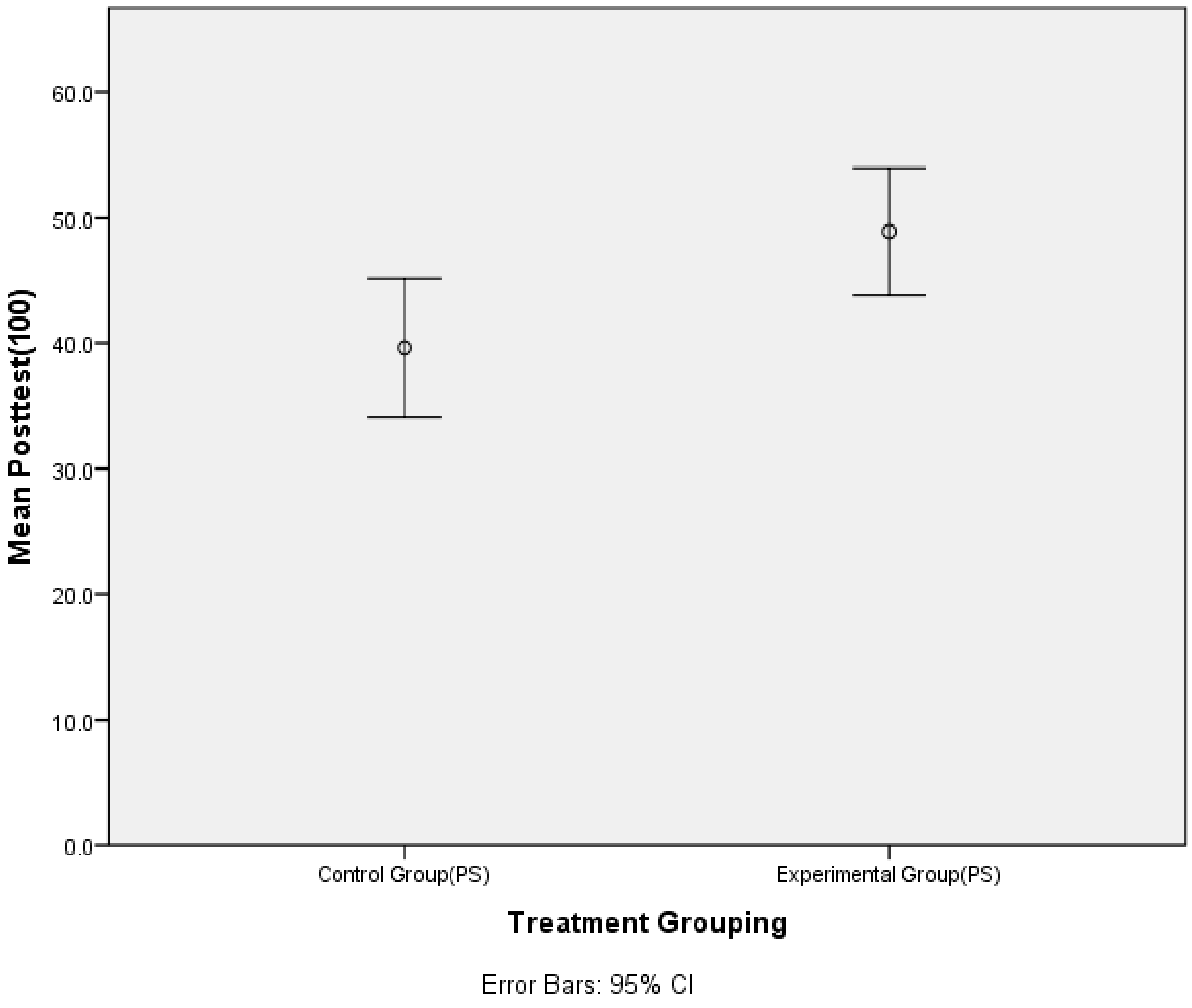


Figure 4-3 Visualizing the equality of variance (pre-test)



;

Figure 4-4 Visualizing the equality of variance (post-test)

Normality of Outcome data

The outcome data for pre-test, post-test, and gain scores for both groups are normal (p > 0.05) as indicated by the information in table 4.36. The same result is presented in figures 4-5, 4-6, 4-7, 4-8 and 4-9.

Table 4-36 Test of Normality Pilot Study Data

		Kolmogorov-Smirnov			**Shapiro-Wilk**		
	Treatment Grouping	**Statistic**	**df**	**Sig.**	**Statistic**	**df**	**Sig.**
Pre-test(100)	Control Group(PS)	.083	41	.200*	.983	41	.776
	Experimental Group (PS)	.134	41	.064	.976	41	.528
Post-test (100)	Control Group (PS)	.111	41	.200*	.954	41	.099
	Experimental Group (PS)	.093	41	.200*	.955	41	.103
Gain Score (100)	Control Group (PS)	.092	41	.200*	.974	41	.454
	Experimental Group (PS)	.110	41	.200*	.973	41	.419

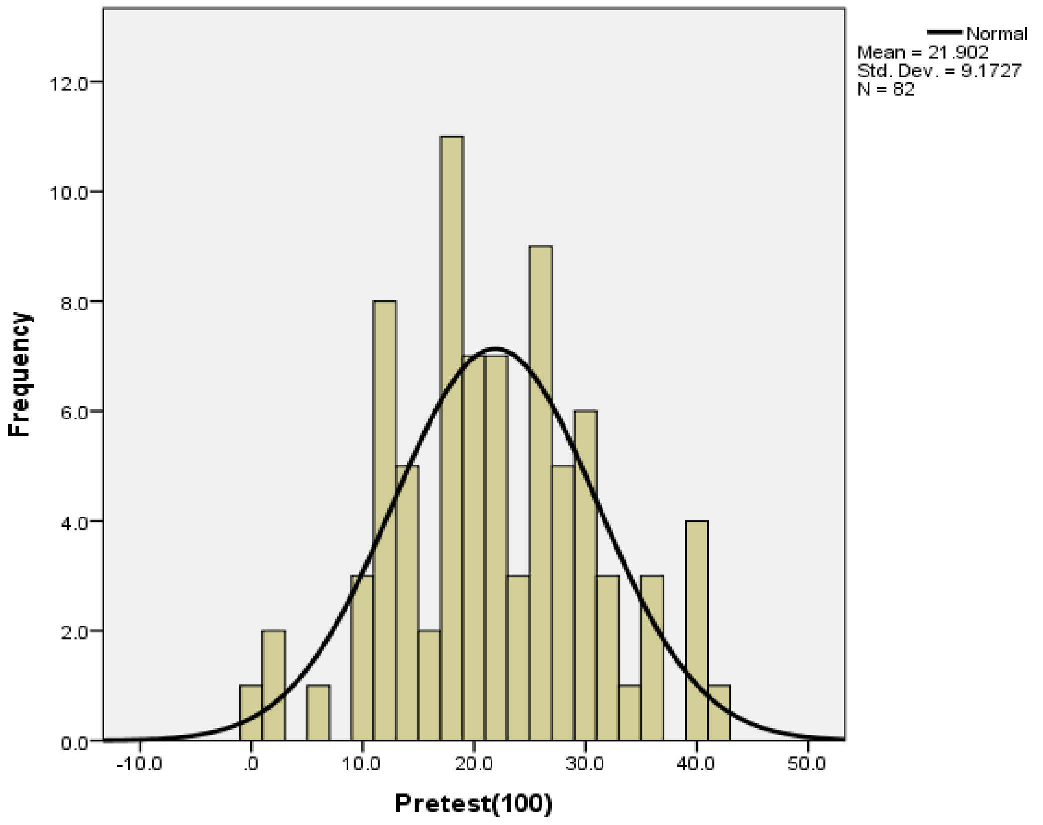


Figure 4-5 Normality plot for pre-test

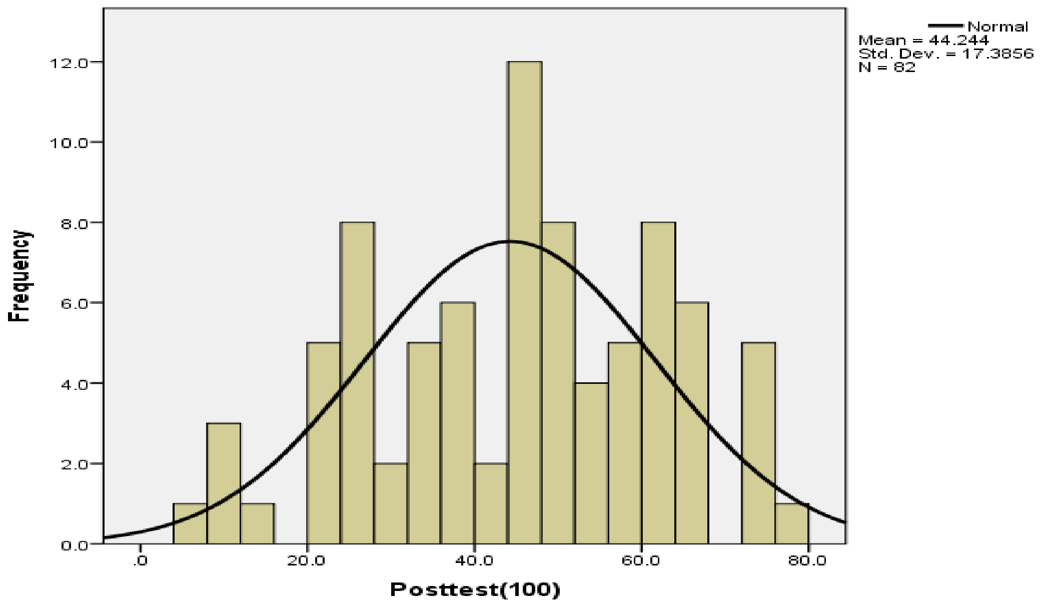


Figure 4-6 Normality plot for post-test

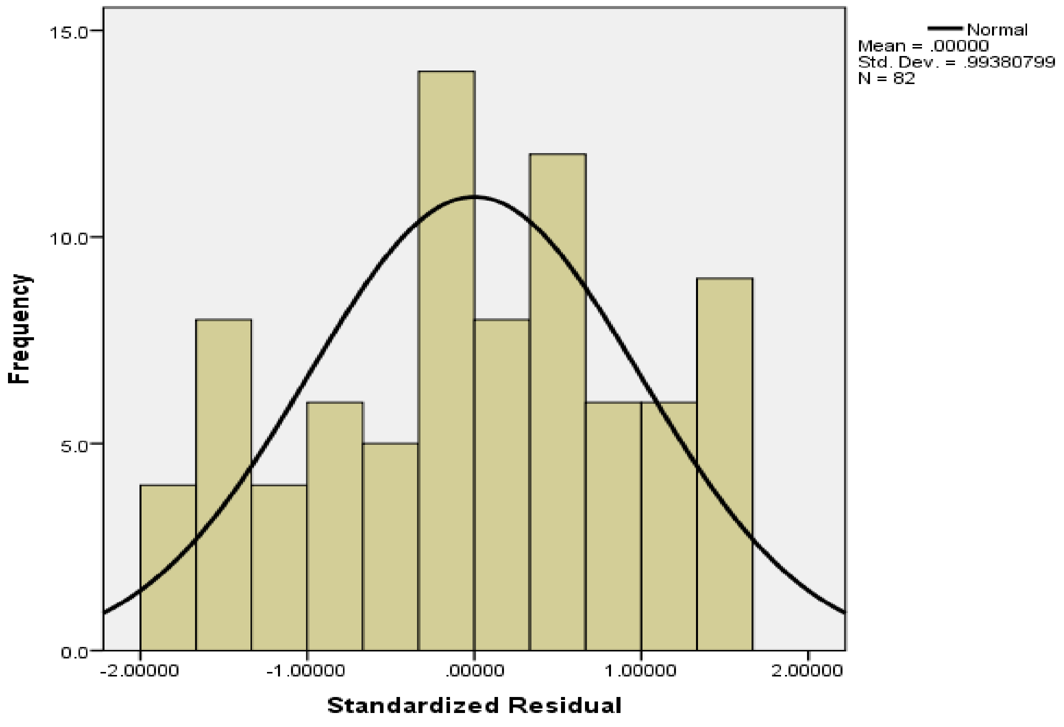


Figure 4-7 Normality plot for Standardized Residual

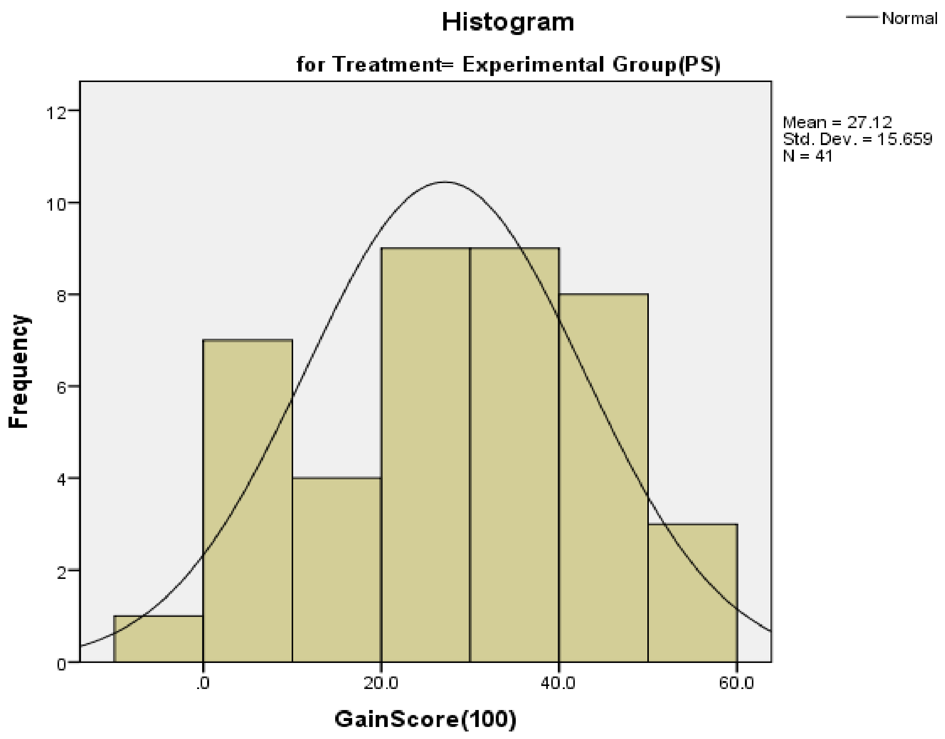


Figure 4-8 Normality plot for Difference (gain) scores

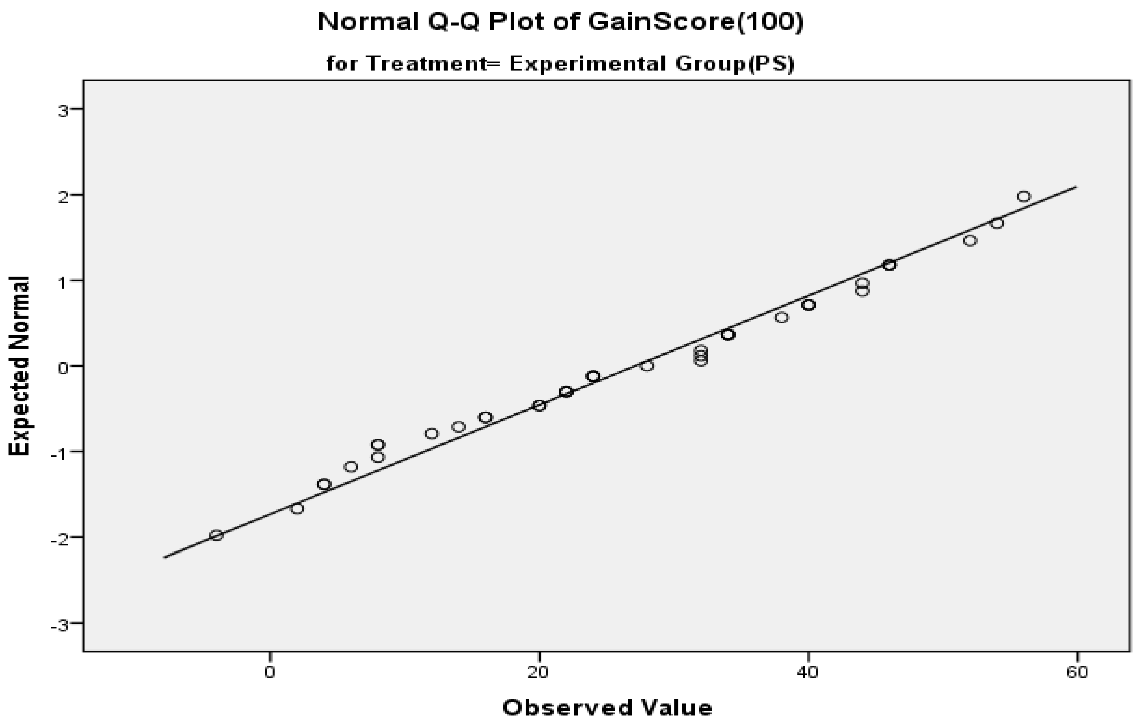


Figure 4-9 QQ Plot showing normality of Difference scores

Additional Assumptions for ANCOVA:

Measurement of covariate before treatment:

The researcher collected pre-test scores from participants in both groups before introducing them to programming.

Reliability of the Covariate:

After collecting the achievement data from CS1 students using IPAT, reliability analysis conducted produced a Cronbach alpha value of 0.844. This value suggests that the covariate was reliable.

Independence of Treatment and the Covariate:

In this study, independence of treatment and the covariate means the pre-test scores of both conventional and the *Scratch* class are not significantly different. To confirm this, Field (2018) recommends performing an ANOVA or t-test using the treatment groups as independent variable and the covariate as outcome. The result from ANOVA in Table 4-37, shows that the main effect of the pre-test scores is not significant, F (1, 80) = .021, *p* = .886. In other words, the pre-test mean in both groups are not significantly different. Hence, the independence of treatment and covariate, a requirement for performing ANCOVA, has been satisfied.

Table 4-37 Test of independence Treatment and covariate

Dependent Variable: Pre-test

	Sum of Squares	Df	Mean Square	F	Sig.
Between Groups	**1.756**	**1**	**1.756**	**.021**	**.886**
Within Groups	**6813.463**	**80**	**85.168**		
Total	**6815.220**	**81**			

Linearity of relationship between dependent variable and the covariate for the two groups:

A plot of the dependent variable (post-test) and the covariate (pre-test) for *Scratch* and control groups in Figure 4-10 reveals that linear relationships exist between the dependent and the covariate for both groups.

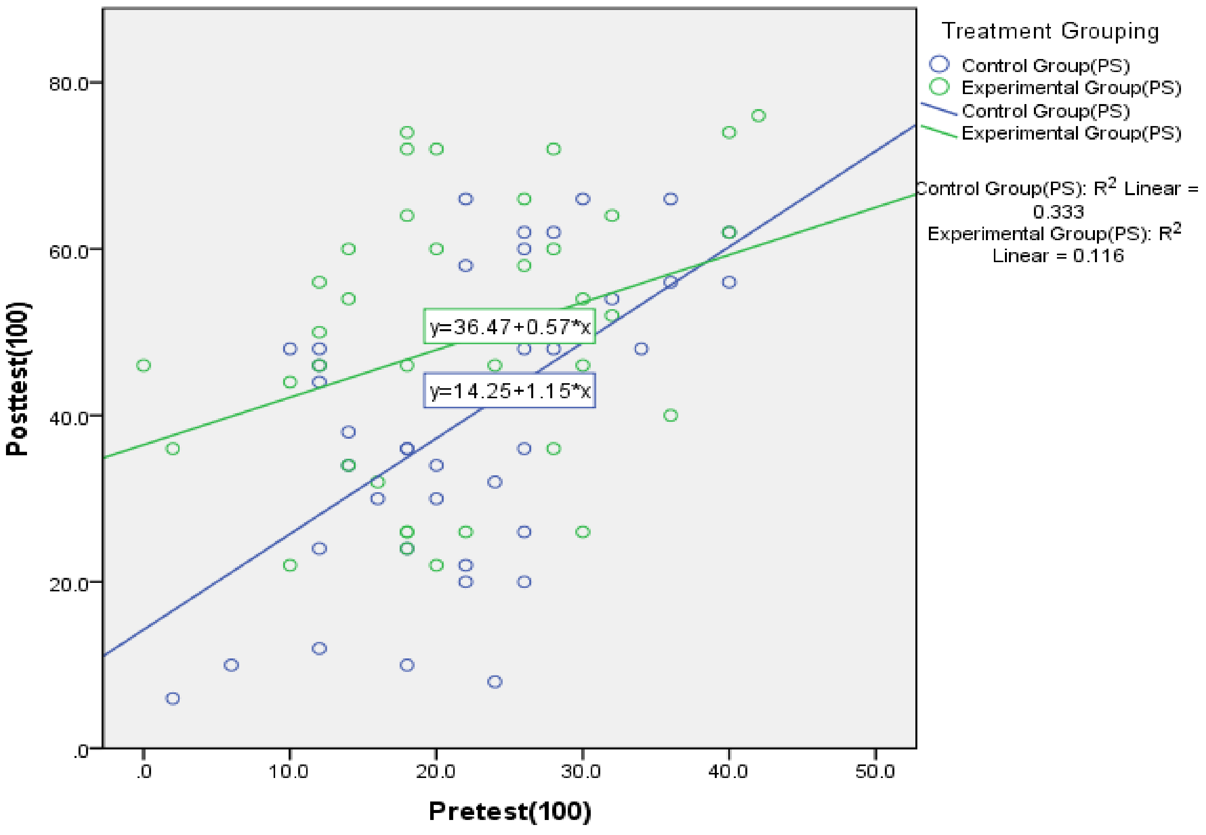


Figure 4-10 Scatter plot showing linearity

The result of univariate analysis of covariance in Table 4-38, suggests that there is no interaction between treatment and covariate (p =.144). As shown in the table, the significance value of the interaction term (Treatment*Pre-test) is greater than 0.10 which shows, it is not important in the fitted model. In addition, its partial eta squared is 0.032, which is near zero, showing that its contribution to the model is negligible. Hence, we can assume that the homogeneity of covariate coefficient has been satisfied.

Table 4-38 Homogeneity of Regression Slope

Tests of Between-Subjects Effects
Dependent Variable: Post-test

Source	Type III Sum of Squares	df	Mean Square	F	Sig.	Partial Eta Squared
Corrected Model	7093.722b	3	2364.574	10.606	.000	.290
Intercept	7731.984	1	7731.984	34.682	.000	.308
Treatment	1483.413	1	1483.413	6.654	.012	.079
Pretest	5007.780	1	5007.780	22.462	.000	.224
Treatment * Pretest	568.326	1	568.326	2.549	.114	.032
Error	17389.400	78	222.941			
Total	185000.000	82				
Corrected Total	24483.122	81				

4.1.11 Why Use ANCOVA After Matching?

To address problems of confounders, data collected during the study have been subjected to pre-processing using Coarsened Exact Matching (CEM) algorithm. The need to ensure equivalence of the two experimental groups informed the use of CEM. This provides some ground for validity, for causal inference on the impact of the treatments on the achievement of the students in the study. It has been suggested that the resultant data after matching, can be subjected to same statistical analysis we would have employed on the original unmatched data (Iacus et al., 2009, 2012). Possible statistical analysis for our matched data includes weighted least square linear regression and Analysis of Covariance (ANCOVA). I decided to use ANCOVA. The reasons informing this decision include relevance to the study's research hypotheses and power of ANCOVA for detecting treatment effect. More so, in ANCOVA we have benefits of regression and ANOVA combined.

4.1.12 Pilot Study: Research Hypotheses Testing

The Impact of a Constructionist Scratch instruction on achievement

The study tested following hypotheses at the 0.05 level of significance:

$H_0$1: There is no significant difference between the mean scores of the pre- and post- Introductory Programming Achievement Test (IPAT) of first-year polytechnic CS students, after a six-week *Scratch* programming instructionH_a1: **There is significant difference between the mean scores of the pre- and post- Introductory Programming Achievement Test (IPAT) of first-year polytechnic CS students, after a six-week *Scratch* programming instruction.** In other words,

$H_0$1: $\mu_{diff} = 0$

H_a1: $\mu_{diff} \neq 0$

Where $\mu_{diff} = \mu_{posttest} - \mu_{pretest}$

Paired sample t-test was employed to determine whether exposure to a constructionist *Scratch* programming pedagogy made any significant improvement on novice computer science students' programming ability. The results of the test are presented in Tables Table 4-39, Table 4-40 and

Table 4-41

Table 4-39 Paired Samples Statistics

		Mean	N	Std. Deviation	Std. Error Mean
Pair 1	Post-test	48.878	41	16.0689	2.5095
	Pre-test	21.756	41	9.6041	1.4999

Table 4-40 Paired Samples Correlations

		N	Correlation	Sig.
Pair 1	Post-test & Pre-test	41	.341	.029

Table 4-41 Paired samples t -test

		Paired Differences							
					95% Confidence Interval of the Difference				
		Mean	Std. Deviation	Std. Error Mean	Lower	Upper	t	df	Sig. (2-tailed)
Pair 1	Posttest - Pretest	27.1220	15.6592	2.4456	22.1793	32.0646	11.090	40	.000

Comparative impacts of the two CS1 programming instructions on achievement

The study tested following hypotheses at the 0.05 level of significance:

H_o2: There is no significant difference in the mean scores of the post Introductory Programming Achievement Test (IPAT) of first-year polytechnic CS students in a constructionist *Scratch* class (experimental group) and those in the conventional class (control group), while controlling for their pre-test scores.

H_o2:µCS = µCL

CS – Constructionist *Scratch*

CL – Conventional Lecture

H_a2: There is significant difference in the mean scores of the post Introductory Programming Achievement Test (IPAT) of first-year polytechnic CS students in a constructionist *Scratch* class (experimental group) and those in the conventional class (control group), while controlling for their pre-test scores.

H_a2: $\mu CS \neq \mu CL$

To determine if there was any statistically significant difference, between the post-test means of the students in both classes while controlling for the difference in their pre-test scores, the researcher performed a One-way analysis of covariance. The result of this analysis is presented in Table 4-42

A look at the third row (highlighted) in Table 4-42 containing the result for the main effect of the Treatment, shows there is statistically significant difference between the post-test scores of students in the *Scratch* and control classes, F(1,79) = 8.159, p = 0.005, partial η^2= .094. The partial eta squared value, according to Cohen (1988), indicates that treatment has a moderate effect on students' programming achievement.

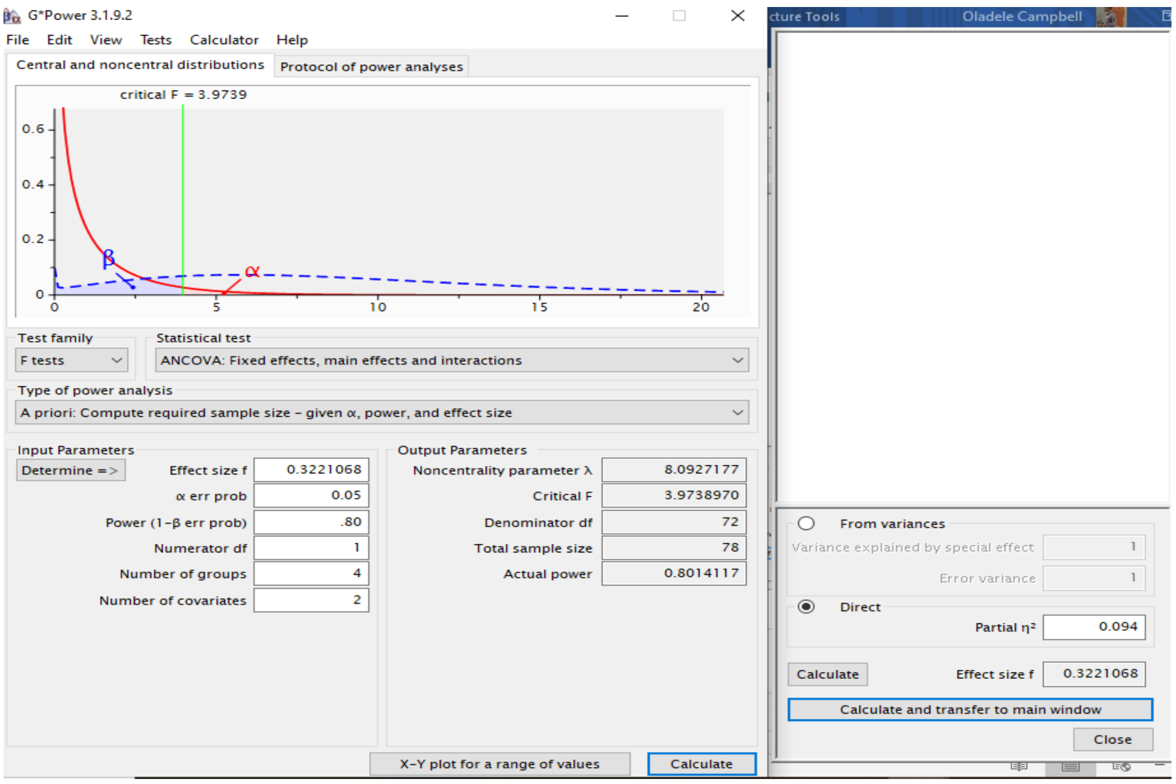
Table 4-42 Pilot Study, One-way ANCOVA Results: Tests of Between-Subjects Effects

Dependent Variable: Post-test

Source	Type III Sum of Squares	df	Mean Square	F	Sig.	Partial Eta Squared	Noncent. Parameter	Observed Power
Corrected Model	6525.396	2	3262.698	14.353	.000	.267	28.707	.998
Intercept	8138.919	1	8138.919	35.805	.000	.312	35.805	1.000
Treatment	1854.730	1	1854.730	8.159	.005	.094	8.159	.806
Pretest	4764.420	1	4764.420	20.960	.000	.210	20.960	.995
Error	17957.726	79	227.313					
Total	185000.000	82						
Corrected Total	24483.122	81						

Deciding Main Study Sample Size

Figures Figure 4-11 and Figure 4-12 present the outputs of the analysis conducted in the power analysis software (G*power version 3.1.9.2) with input from the pilot study in order to determine the sample size for the main study. The effect size (i.e., partial eta squared) used in this analysis was from Table 4-42. This value (0.094) suggests the size of effect likely to be observed, in a study comparing learning in *Scratch* and the conventional CS1 programming classes. You will observe in Figure 4-11 that with statistical power of 0.8—a value widely recommended (Aberson, 2019)— and alpha value of 0.05, we will need a sample size of about 80 in all the four groups, to detect any effect of intervention from an ANCOVA test.


G*Power 3.1.9.2
File Edit View Tests Calculator Help
Central and noncentral distributions
Protocol of power analyses
critical F = 3.9739
β
α
cture Tools
Oladele Campbell
Calculate
Calculate and transfer to main window
Close
X-Y plot for a range of values
Calculate
Determine =>

Test family: F tests

Statistical test: ANCOVA: Fixed effects, main effects and interactions

Type of power analysis: A priori: Compute required sample size – given α, power, and effect size

Input Parameters

Effect size f	0.3221068
α err prob	0.05
Power (1-β err prob)	.80
Numerator df	1
Number of groups	4
Number of covariates	2

Output Parameters

Noncentrality parameter λ	8.0927177
Critical F	3.9738970
Denominator df	72
Total sample size	78
Actual power	0.8014117

From variances
Variance explained by special effect 1
Error variance 1

Direct
Partial η² 0.094

Effect size f 0.3221068

Figure 4-11Computing the sample size in G*Power

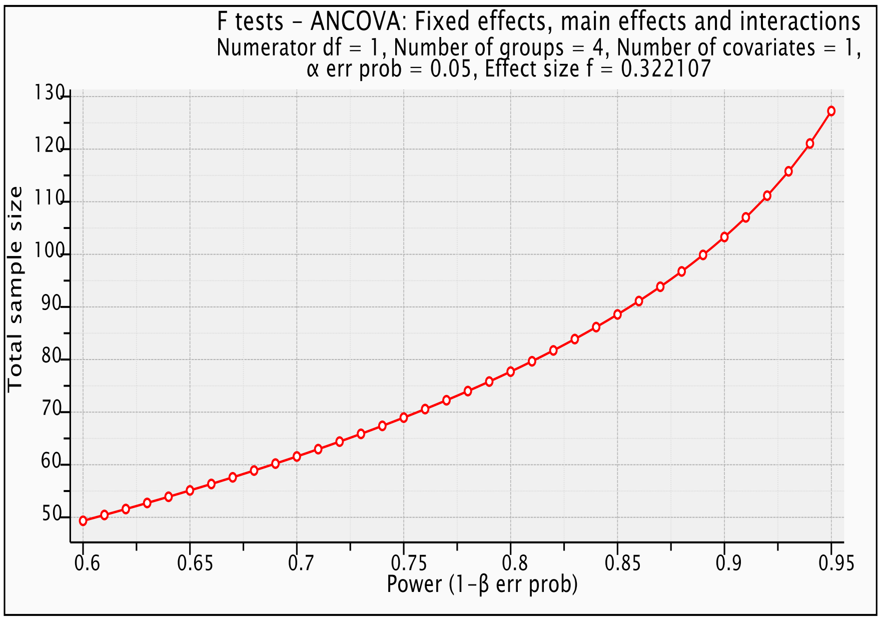


Figure 4-12 Graph of sample size versus power

4.2 The Main Study

4.2.1 The Quasi-Experimental Research

In this section, we present the results of the statistical analysis of the data collected from the quasi-experiment research. Due to reasons earlier mentioned, data presented here are from two out of four polytechnics earmarked for the main study. Though the main study requires about 80 students as indicated in section 4.4.3, first, we will present the results of data from the intact classes of CS1 students that were involved in the main study. Ethical reasons informed this decision. From this unmatched sample, a random sample of 84 students was realised using Coarsened Exact Matching. Hypothesis testing was conducted using this matched sample. lastly, we will present the results from the matched sample in section 4.6.

4.2.2 Demographics of the main study participants

While Table 4-43 reveals what is characteristic of most CS1 classes - more males enrolling than females - the control group has more females with 31% compared to the experimental class which has about 21%.

Table 4-43 Gender of Main Study Participants (N =182)

Treatment Grouping	**Gender**		**Total**
	Male n(%)	**Female n(%)**	
Control	59 (68.6)	27(31.4)	86
Experimental	76 (79.2)	20 (20.8)	96
Total	135	47	182

Table 4-44 indicates almost similar distributions in the ages of participants in both experimental and control groups.

Table 4-44 Age of Main Study Participants (n=182)

Treatment Grouping	**Age Group**					**Total**
	16 - 18 n(%)	**19-21 n(%)**	**22-24 n(%)**	**> 24 n(%)**	**Others n(%)**	
Control	15 (17.9)	37 (44.0)	27 (32.1)	5(6.0.0)	0 (0.0)	84
Experimental	12 (12.5)	44 (45.8)	34 (35.4)	5 (5.2)	1(1.0)	96
Total	27(15.0)	61(33.9)	81(45.0)	10(5.6)	1(0.6)	180

Note: Two participants in the Control group did not indicate their age in the questionnaire.

While the participants in the control group seem to be academically higher compare to the experimental group, the experimental group had students with high-achieving academic while the control has none (Table 4-45). These two students' data may be outliers in the research data.

Table 4-45 Academic Background of Main Study Participants (n=182)

Treatment Grouping	**Academic Background Index**			**Total**
	Low-Achieving n(%)	**Average Achieving n(%)**	**High-Achieving n(%)**	
Control	53(61.6)	33(38.4)	0 (0.0)	86
Experimental	80(83.3)	14 (14.6)	2 (2.1)	96
Total	133(73.1)	47(25.8)	2(1.1)	182

In their response to question in the survey to know if participants have known about programming (i.e., theoretically) in earlier education levels, the data in Table 4-46 reveals similar distributions in the backgrounds of students in both groups.

Table 4-46 Prior Programming Learning of Main Study Participants

Treatment Grouping	**Prior Programming Learning**		**Total**
	No Background n(%)	**Some Background n(%)**	
Control	68 (79.1)	18 (20.9)	86
Experimental	77 (80.2)	19 (19.8)	96
Total	145(79.7)	37(20.3)	182

In their response to the question - if the participants have learnt to write (practically) programs in earlier education levels, the data in Table 4-47 reveals almost similar distributions in the backgrounds in both groups.

Table 4-47 Prior Programming Writing of Main Study Participants

Treatment Grouping	**Prior Programming Learning**		**Total**
	No Background n(%)	**Some Background n(%)**	
Control	74 (86.0)	12 (14.0)	86
Experimental	86 (89.6)	10 (10.4)	96
Total	160	22	182

Table 4-48 provides information revealing the prior experience of the participant in creative or artistic activities in their earlier education. More students reported they have experience in the experimental group compared to the control group.

Table 4-48 Visual Art background of Study Participants

Treatment Grouping	Visual Art Background		Total
	No Background n(%)	Some Background n(%)	
Control	44 (51.2)	42 (48.8)	86
Experimental	36 (37.5)	60 (62.5)	96
Total	80(44.0)	102(56.0)	182

4.2.3 Main Study: Pre-test Performance

Table 4-49 indicates a higher statistically significant pre-test performance by the control group (M=18.65, SD =10.95) compared to the experimental group (M= 13.44, SD = 8.53).

Table 4-49 Main Study Pre-test Achievement Performance

Group	N	Mean	S.D.	S.E.	t value
Control	86	18.65	10.95	1.18	3.603
Experimental	96	13.44	8.53	0.87	

* Mean difference is significant.

Table 4-50 provides the pre-test performance along gender lines in both groups. The performance of both genders was similar in both groups. However, while the males performed higher in the control group, the females performed higher in the experimental group.

Table 4-50 Pre-test Scores of Main Study Participants based on Gender

Treatment Grouping	Gender	Mean	N	Std. Deviation	Std. Error of Mean
Control	Male	19.29	59	10.75	1.40
	Female	17.26	27	11.45	2.20
	Total	18.65	86	10.95	1.18
Experimental	Male	13.13	76	8.47	.97
	Female	14.60	20	8.85	1.98
	Total	13.44	96	8.53	.87
Total	Male	15.82	135	9.98	.86
	Female	16.13	47	10.41	1.52
	Total	15.90	182	10.06	.75

Table 4-51 shows pre-test performance aligning with student's prior academic abilities. Strangely, the mean of the two high achieving students in the experimental group, raises question or doubts about their self-reported prior academic record or error in the data. Nevertheless, the data as presented confirm that the experimental class is academically weaker compared to the control group.

Table 4-51 Pre-test Scores of Main Study Participants based on Academic Background

Treatment Grouping	Academic Background	Mean	N	Std. Deviation	Std. Error of Mean
Control Group	Low-Achieving	18.38	53	10.80	1.48
	Average Achieving	19.09	33	11.35	1.97
	Total	18.65	86	10.95	1.18
Experimental Group	Low-Achieving	12.98	80	7.80	.87
	Average Achieving	17.57	14	11.18	2.99
	High-Achieving	3.00	2	1.41	1.00
	Total	13.44	96	8.53	.87
Total	Low-Achieving	15.13	133	9.46	.82
	Average Achieving	18.64	47	11.20	1.63
	High-Achieving	3.00	2	1.41	1.00
	Total	15.90	182	10.06	.75

Performance of the two groups are similar as shown in Table 4-52. Those with background in writing programs performed better compared to those without.

Table 4-52 Pre-test Scores of Main Study Participants based on Prior Program Writing

Treatment Grouping	**Prior Program Writing**	**Mean**	**N**	**Std. Deviation**	**Std. Error of Mean**
Control Group	No Background	18.36	74	11.03	1.28
	Some Background	20.42	12	10.76	3.11
	Total	18.65	86	10.95	1.18
Experimental Group	No Background	13.21	86	7.89	.85
	Some Background	15.40	10	13.23	4.19
	Total	13.44	96	8.53	.87
Total	No Background	15.59	160	9.78	.77
	Some Background	18.14	22	11.93	2.54
	Total	15.90	182	10.06	.75

Performance of the two groups as shown in Table 4-53 presents similar information as in previous table for prior program writing. Those with background in prior visual arts performed better compare to those without.

Table 4-53 Pre-test Scores of Main Study Participants based on Visual Art Background

Treatment Grouping	Visual Art Background	Mean	N	Std. Deviation	Std. Error of Mean
Control Group	No Background	17.82	44	9.86	1.49
	Some Background	19.52	42	12.05	1.86
	Total	18.65	86	10.95	1.18
Experimental Group	No Background	10.17	36	5.50	.92
	Some Background	15.40	60	9.42	1.22
	Total	13.44	96	8.53	.87
Total	No Background	14.38	80	9.00	1.01
	Some Background	17.10	102	10.72	1.06
	Total	15.90	182	10.06	.75

4.2.4 Main Study: Post-test Results

While Table 4-49 showed that the control had higher statistically significant mean pre-test score compared to the experimental, Table 4-54 indicates that performance of both groups in the post-test is not significantly different. The experimental class seems to have had a catch-up in their performance.

Table 4-54 Main Study Participants Post-test Achievement Performance

Grouping	N	Mean	S.D.	S.E.	t value	Significance
Control	86	27.63	13.09	1.41	0.40	*p= .969
Experimental	96	27.56	8.44	.86		

* Mean difference is not significant.

Table 4-55 further confirms what we now know: The learning gain of the experimental class is significantly higher than the control group.

Table 4-55 Main Study Learning Gain Score Achievement Performance

Grouping	N	Mean	S.D.	S.E.	t value	Significance
Control	86	8.98	10.91	1.18	-3.455	*p=.001
Experimental	96	14.13	9.18	0.94		

* Mean difference is significant.

Looking at the post-test performance along gender lines, Table 4-56 gives an interesting information. The performance in the post-test compared to the pre-test has reversed along gender lines in both groups. In the control group, the males performed higher, while in the pre-test their performance is lower. In the experimental where the female performed higher, in the pre-test, their performance is lower.

Table 4-56 Main Study Post-test Achievement Performance based on Gender

Treatment Grouping	**Gender**	**Mean**	**N**	**Std. Deviation**	**Std. Error of Mean**
Control Group	Male	26.81	59	12.94	1.68
	Female	29.41	27	13.49	2.60
	Total	27.63	86	13.09	1.41
Experimental Group	Male	28.53	76	8.24	.94
	Female	23.90	20	8.40	1.88
	Total	27.56	96	8.44	.86
Total	Male	27.78	135	10.54	.91
	Female	27.06	47	11.82	1.72
	Total	27.59	182	10.86	.80

To probe which gender has gained more in which group, Table 4-57 reveals that while the females gained more in the lecture class (Control), the males gained more in the *Scratch* class (Experimental

Table 4-57 Main Study Learning Gain Score Achievement Performance based on Gender

Treatment Grouping	**Gender**	**Mean**	**N**	**Std. Deviation**	**Std. Error of Mean**
Control Group(MS)	Male	7.53	59	10.00	1.30
	Female	12.15	27	12.27	2.36
	Total	8.98	86	10.91	1.18
Experimental Group(MS)	Male	15.39	76	9.15	1.05
	Female	9.30	20	7.77	1.74
	Total	14.13	96	9.18	.94
Total	Male	11.96	135	10.27	.88
	Female	10.94	47	10.59	1.54
	Total	11.69	182	10.33	.77

Compare to the pre-test, Table 4-58 presents similar information about performance of the students in both groups. Fairly, their post-test performance was aligned with their prior academic background.

Table 4-58 Main Study Post-test Achievement Performance based on Academic Background

Treatment Grouping	Academic Background	Mean	N	Std. Deviation	Std. Error of Mean
Control Group(MS)	Low-Achieving	27.13	53	13.56	1.86
	Average Achieving	28.42	33	12.47	2.17
	Total	27.63	86	13.09	1.41
Experimental Group(MS)	Low-Achieving	27.35	80	8.63	.96
	Average Achieving	29.43	14	7.77	2.08
	High-Achieving	23.00	2	2.83	2.00
	Total	27.56	96	8.44	.86
Total	Low-Achieving	27.26	133	10.82	.94
	Average Achieving	28.72	47	11.20	1.63
	High-Achieving	23.00	2	2.83	2.00
	Total	27.59	182	10.86	.80

Interestingly, Table 4-59 showing learning gains, reveals that low-achieving (including those who self-reported they were "high-achieving") learnt more from *Scratch* class compared to the averaged-achieving students in that class. Recall that the high-achieving students in that class scored terribly low in their pre-test.

Table 4-59 Main Study Learning Gain Performance based on Academic background

Treatment Grouping	Academic Background	Mean	N	Std. Deviation	Std. Error of Mean
Control Group(MS)	Low-Achieving	8.75	53	11.73	1.61
	Average Achieving	9.33	33	9.60	1.67
	Total	8.98	86	10.91	1.18
Experimental Group(MS)	Low-Achieving	14.38	80	9.36	1.05
	Average Achieving	11.86	14	8.47	2.27
	High-Achieving	20.00	2	4.24	3.00
	Total	14.13	96	9.18	.94
Total	Low-Achieving	12.14	133	10.69	.93
	Average Achieving	10.09	47	9.26	1.35
	High-Achieving	20.00	2	4.24	3.00
	Total	11.69	182	10.33	.77

Table 4-60 reveals that while students with some background in program writing performed higher than those without background in the control group, the reverse is the case in the experimental class, though with probably insignificant difference.

Table 4-60 Main Study Post-test Score based on Prior Program Writing Background

Treatment Grouping	**Prior Program Writing**	**Mean**	**N**	**Std. Deviation**	**Std. Error of Mean**
Control Group(MS)	No Background	26.92	74	13.31	1.55
	Some Background	32.00	12	11.18	3.23
	Total	27.63	86	13.09	1.41
Experimental Group(MS)	No Background	27.67	86	8.22	.89
	Some Background	26.60	10	10.62	3.36
	Total	27.56	96	8.44	.86
Total	No Background	27.33	160	10.84	.86
	Some Background	29.55	22	11.02	2.35
	Total	27.59	182	10.86	.80

Table 4-61 **.**reveals the learning gains in the two classes. It suggests that while those with some background in program writing learnt more in the control group, those with no background learnt more in the experimental.

Table 4-61 Main Study Learning Gain based on Prior Program Writing background

Treatment Grouping	Prior Program Writing	Mean	N	Std. Deviation	Std. Error of Mean
Control Group(MS)	No Background	8.55	74	10.54	1.23
	Some Background	11.58	12	13.16	3.80
	Total	8.98	86	10.91	1.18
Experimental Group(MS)	No Background	14.47	86	9.39	1.01
	Some Background	11.20	10	6.89	2.18
	Total	14.13	96	9.18	.94
Total	No Background	11.73	160	10.34	.82
	Some Background	11.41	22	10.54	2.25
	Total	11.69	182	10.33	.77

Table 4-62 indicates that while background in visual arts seems not to make much difference in the post-test performance in the control group, those with some background performed higher in the experimental group.

Table 4-62 Main Study Post-test Performance based on Visual Art background

Treatment Grouping	Visual Art Background	Mean	N	Std. Deviation	Std. Error of Mean
Control Group(MS)	No Background	28.05	44	12.50	1.89
	Some Background	27.19	42	13.82	2.13
	Total	27.63	86	13.09	1.41
Experimental Group(MS)	No Background	24.50	36	6.11	1.02
	Some Background	29.40	60	9.13	1.18
	Total	27.56	96	8.44	.86
Total	No Background	26.45	80	10.24	1.14
	Some Background	28.49	102	11.29	1.12
	Total	27.59	182	10.86	.80

4.2.5 Matching for Valid Causal Inference

To address the problem of significant differences (as revealed by prior descriptive statistics) in the treatment groups, data from intact classes during the main study were pre-processed using Coarsened Exact Matching (CEM). CEM is a free SPSS add-in available at https://projects.iq.harvard.edu/cem-spss/pages/installation. After installation, CEM resides in the Analyze menu in an SPSS program.

Another reason for matching, is to mitigate the weakness of ANCOVA analysis when intact groups are used and there is non-random assignment to groups, making ANCOVA results and its interpretation prone to errors (Miller & Chapman, 2001).

The main study data were processed using CEM and the results presented by subsequent tables and figures show the effect of this pre-processing on the research data.

4.2.6 Main Study: Matched Sample

The result of the matching performed on the samples from the two treatment groups are presented in Tables 4-63 to 4-68. You will observe that while the two samples are perfectly matched by academic background, prior program writing and visual art background, they are fairly or almost matched by gender and age. They are perfectly matched on what prior research have found to influence achievement in CS1.

4.2.7 Descriptive Statistics of the Matched Sample

Table 4-63 Gender of Matched Sample (n =84)

Treatment Grouping	**Gender**		**Total**
	Male n(%)	**Female n(%)**	
Control	27 (64.3)	15 (35.7)	42
Experimental	33 (78.6)	9 (21.4)	42
Total	60(71.4)	24(28.6)	84

Table 4-64 Academic Background of Matched Sample (n =84)

Treatment Grouping	**Academic Background Index**		**Total**
	Low-Achieving n(%)	**Average Achieving n(%)**	
Control	32 (76.2)	10 (23.8)	42
Experimental	32 (76.2)	10(23.8)	42
Total	64(76.2)	20(23.8)	84

Table 4-65 Age of Matched Sample from Main Study (n =84)

Treatment Grouping	**Age Group**				**Total**
	16 – 18 n(%)	**19-21 n(%)**	**22-24 n(%)**	**> 24 n(%)**	
Control	6 (14.3)	17 (40.5)	17 (40.5)	2(4.7)	42
Experimental	6 (14.3)	17 (40.5)	16 (38.1)	3 (7.1)	42
Total	12	33	32	5	84

Table 4-66 Academic Background of Matched Sample from Main Study (n =84)

Treatment Grouping	**Academic Background**		**Total**
	Low-Achieving n(%)	**Average-Achieving n(%)**	
Control	32(76.2)	10(23.8)	42
Experimental	32(76.2)	10(23.8)	42
Total	64(76.2)	20(23.8)	84

Table 4-67 Prior Programming Writing of Matched Sample from Main Study (n =84)

Treatment Grouping	**Prior Programming Writing**		**Total**
	No Background n(%)	**Some Background n(%)**	
Control	41 (97.6)	1 (2.4)	42
Experimental	41(97.6)	1 (2.4)	42
Total	82(97.6)	2(2.4)	84

Table 4-68 Visual Art background of Matched Sample from Main Study (n =84)

Treatment Grouping	**Visual Art Background**		**Total**
	No Background n(%)	**Some Background n(%)**	
Control	22 (52.4)	20 (47.6)	42
Experimental	22 (52.4)	20 (47.6)	42
Total	44(52.4)	40(47.6)	84

4.2.8 Main Study: Pre-test Performance of Matched Sample

Another view of the result of matching is presented in Table 4-69, showing the baseline performance of the treatment groups. Looking at the unmatched pre-test performance in Table 4-49 where the control group had a significant higher mean pre-test score compared to the experimental group, you will observe that with matched samples, the significant difference has disappeared.

Table 4-69 Matched Sample from Main Study Participants Pre-test Achievement

Group	N	Mean	S.D.	S.E.	t value	Level of Significance
Control	42	14.29	8.592	1.326	.217	0.828 (not significant)
Experimental	42	13.88	8.469	1.307		

4.2.9 Main Study: Posttest Performance of Matched Sample

Information in Table 4-54 indicated that the treatment and the control have insignificant difference in their mean post-test score, though Table 4-55 revealed that the experimental group had significantly higher gain score compared to the control. But with matched samples the difference in the mean post-test score is clear as shown in Table 4-70. The experimental group has a significant higher score compared to the control.

Table 4-70 Matched Sample Post-test Achievement Performance

Treatment	N	Mean	Std. Deviation	Std. Error Mean	t value	Significance
Control	42	23.43	11.648	1.797	-2.255	*p = .027
Experimental	42	28.47	8.600	1.327		

* mean difference is significant.

Table 4-71 Matched Sample Learning Gain Achievement Performance

Grouping	N	Mean	Std. Deviation	Std. Error Mean	t value	Significance
Control	42	9.14	10.363	1.599	-2.660	* p = .009
Experimental	42	14.60	8.314	1.283		

* Mean difference is significant.

After adjusting an extreme outlier in the experimental group, we have the post-test performance as shown in Table 4-72

Table 4-72 Matched Sample Post-test Achievement Performance (Adjusted)

	Treatment Grouping	N	Mean	Std. Deviation	Std. Error Mean
Post-test	Control Group	42	23.43	11.648	1.797
	Experimental Group	42	28.52	8.755	1.351

4.2.10 ANCOVA after Matching

To mitigate problems of confounders, data collected during the study have been subjected to pre-processing using Coarsened Exact Matching (CEM) algorithm. The need to ensure equivalence of the two experimental groups informed the use of CEM. This provides some ground for causal inference validity on the impact of the treatments, on students' achievement in the study. It has been suggested that, the resultant data after matching can still be subjected to same statistical analysis we would have employed on the original data (Iacus et al., 2009, 2012). Possible statistical analysis for our matched data includes, weighted least square linear regression and Analysis of Covariance (ANCOVA). I decided to use ANCOVA. The reasons informing this decision include relevance to the study's research questions and power for detecting treatment effect. More so, in ANCOVA we have benefits of regression and ANOVA combined.

4.2.11 Testing Assumptions Before Conducting Data Analysis

Running a statistical test without satisfying the assumptions the test requires, leads to flawed computation, interpretation, and conclusion (Field, 2018; Mayers, 2013). In this section, we subject the matched data to series of test, to know if the data satisfy or violate the conditions for ANCOVA to work in a study. Several assumptions are mentioned in the literature that are desirable or that must be satisfied to run ANCOVA (Field, 2018; Mayers, 2013; Pallant, 2016). They include:

- Normality of data
- Covariates normally distributed
- Dependent variable is normally distributed (across the groups)
- Measurement of covariate before treatment:
- Reliability of the covariate
- Reasonable correlation between the covariate and the dependent variable
- Independence of treatment and the covariate
- Equality of variance
- Linearity of relationship between dependent variable and the covariate for the levels of independent variable
- Sample sizes sufficient to ensure enough power to detect the hypothesis.

Testing for Normality:

Covariates normally distributed: Information in Table 4-73 shows a mixed result. Pre-test Scores for the control group appears to be normally distributed, W(42) = .952, *p* = .074, but may not be for the experimental group, W(42) = .926, *p* = .009.

To address this problem, Mayers(2013, p. 51) suggests "to additionally test for z-scores of the skew and kurtosis" of the scores. z-score is computed by dividing the value of the skew or kurtosis with their corresponding standard error. The result of z-scores for the pre-test scores of both groups are given in Table 4-74. How do we know whether our data are still within reasonable bound of normality? Mayers(2013) provides an answer: "Statisticians have calculated that we reach the limits of normal distribution when z-scores are greater than ±1.96 (plus or minus 1.96)" (pg. 52). This cut-off points of ±1.96 for the determining normality is applicable to sample size < 50 (H Kim, 2013; Mayers, 2013). So, with our sample sizes being less 50, we deduce that our pre-test scores for both groups are within reasonable limit of normality.

Table 4-73 Shapiro-Wilk Normality Test for Pre-test Scores Across Treatment Groups

Tests of Normality

		Kolmogorov-Smirnova			**Shapiro-Wilk**		
	Treatment Grouping	Statistic	Df	Sig.	Statistic	df	Sig.
Pretest	Control Group	.118	42	.151	.952	42	.074
	Experimental Group	.161	42	.008	.926	42	.009

Table 4-74 z-scores Test for Pre-test scores across Treatment groups

DV: Pretest

Treatment Grouping	**Mean**	**N**	**SD**	**Skewness**	**Std. Error of Skewness**	**z-score of Skewness**	**Kurtosis**	**Std. Error of Kurtosis**	**z-score**
Control Group	14.29	42	8.592	0.454	0.365	1.243836	-0.372	0.717	-0.51883
Experimental Group	13.88	42	8.469	0.707	0.365	1.936986	-0.252	0.717	-0.35146

Dependent variable is normally distributed (across the groups): Information in Table 4-75 indicates that our dependent variable (the post-test) appears to satisfy normality test.

Using the z-score test for the postscore, the same evidence is provided that suggests, as shown in Table 4-76, that both scores are normal across the two groups.

Table 4-75 Shapiro-Wilk Normality Test for Post-test Scores Across Treatment Groups

	Kolmogorov-Smirnov			Shapiro-Wilk		
Treatment Grouping	Statistic	df	Sig.	Statistic	df	Sig.
Control Group	.127	42	.087	.949	42	.058
Experimental Group	.138	42	.043	.948	42	.057

Table 4-76 z-scores Test for Normality of Post-test scores across Treatment groups

Dependent Variable: Post-test Score

Treatment Grouping	Mean	N	SD	Skewness	SESkewness	Zskewness	Kurtosis	SEKurtosis	ZKurtosis
Control	23.43	42	11.648	0.286	0.365	0.783562	-0.804	0.717	-1.12134
Experimental	28.43	42	8.523	0.623	0.365	1.706849	0.088	0.717	0.122734

Measurement of covariate before treatment: The researcher collected pre-test scores from participants in both groups before introducing them to programming.

Reliability of the Covariate: Data was collected from CS1 students using IPAT, and reliability analysis conducted produced a Cronbach alpha value of 0.844. This value suggests that the covariate was reliable.

Reasonable correlation between the covariate and the dependent variable: **Error! Reference source not found.** suggests that correlation between the covariate and the dependent variable (r = .488, *p*<.05) is acceptable. Reasonable correlation is: between r = .30 and r = .90 (Mayers, 2013). The requirement is satisfied.

Table 4-77 Correlation between the covariate and the dependent variable

		Pre-test	Post-test
Pre-test Score	Pearson Correlation	1	.488**
	Sig. (2-tailed)		.000
	N	84	84
Post-test Score	Pearson Correlation	.488**	1
	Sig. (2-tailed)	.000	
	N	84	84

**. Correlation is significant at the 0.01 level (2-tailed).

Independence of Treatment and the Covariate: In this study, independence of treatment and the covariate means, the pre-test scores of both conventional and the *Scratch* class are not significantly different. To confirm this, Field (2018) recommends performing an ANOVA or t-test using the treatment groups as independent variable and the covariate as outcome. The result from ANOVA in Table 4-78 shows that the main effect of the pre-test scores is not significant, F (1, 82) = .047, *p* = .828. In other words, the pre-test mean in both groups are not significantly different. Hence, the independence of treatment and covariate, a requirement for performing ANCOVA, has been satisfied.

Table 4-78 Test of Independence of treatment and the covariate

DV:Pre-test					
	Sum of Squares	df	Mean Square	F	Sig.
Between Groups	3.440	1	3.440	.047	.828
Within Groups	5966.976	82	72.768		
Total	5970.417	83			

Equality of variance

Results in Table 4-79 suggests that the variances for the pre-test score is equal for both groups. Figure 4-13 presents visual evidence of the same information.

Table 4-79 Test of Homogeneity of variances

Dependable Variable: Pre-test			
Levene Statistic	df1	df2	Sig.
.118	1	82	.732

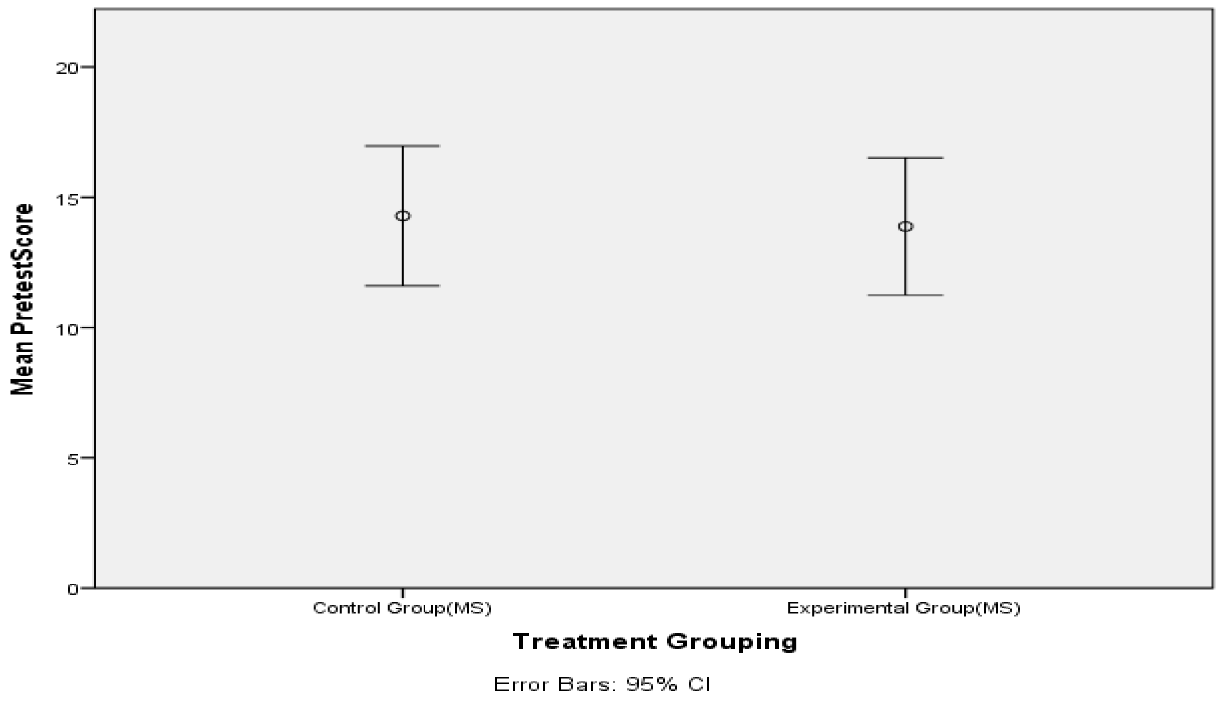


Figure 4-13 Visualizing the equality of variances (pre-test)

Similar to what we observe for the pre-test, Table 4-80 and Figure 4-14 present information suggesting that our post-test data, satisfies the condition for equality of variances

Table 4-80 Test of Homogeneity of variances (Post-test)

Dependent Variable: Post-test			
F	df1	df2	Sig.
3.460	1	82	.066

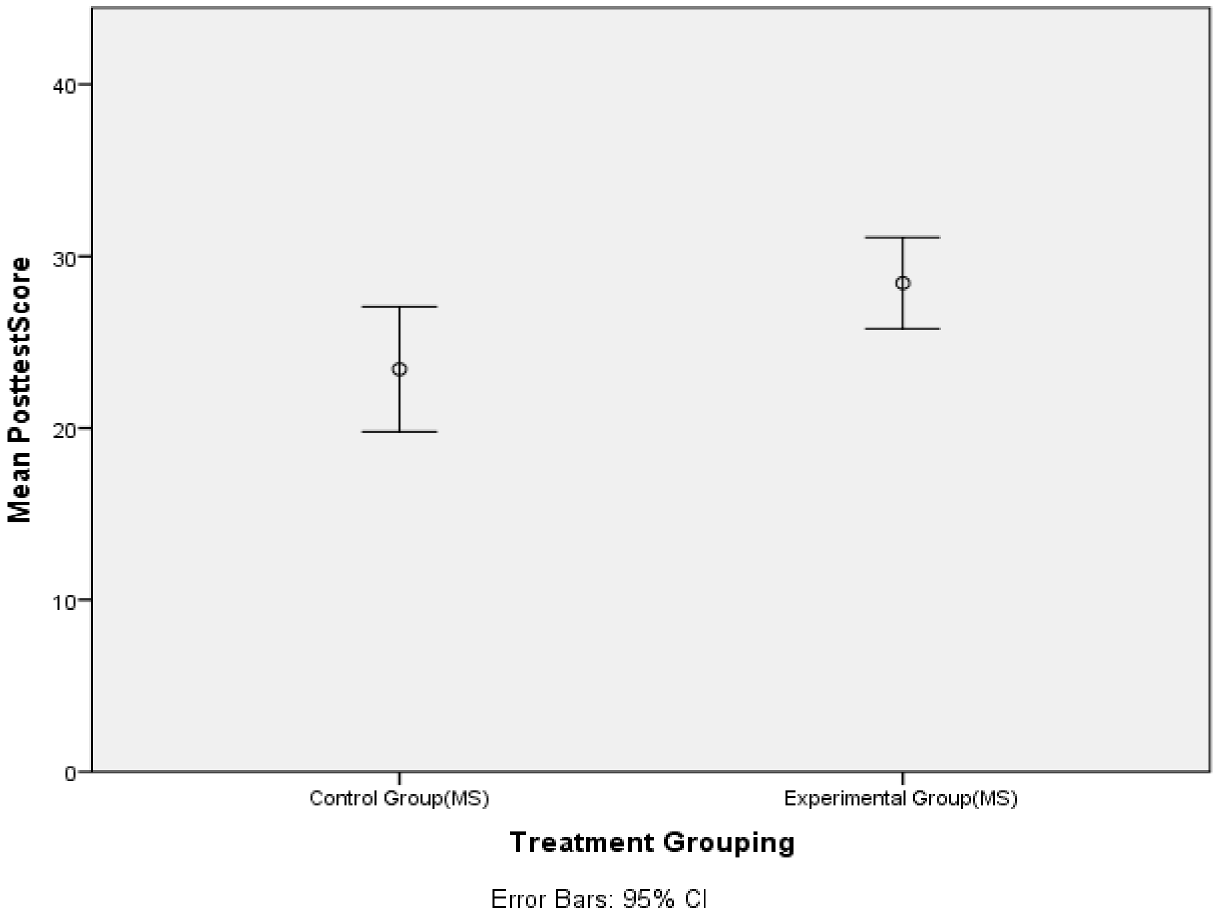


Figure 4-14 Visualizing the equality of variances (post-test)

Linearity of relationship between Dependent variable and the covariate for the two groups: Figure 4-15 presents information suggesting that there is a linear relationship between pre-test and post-test across the two groups

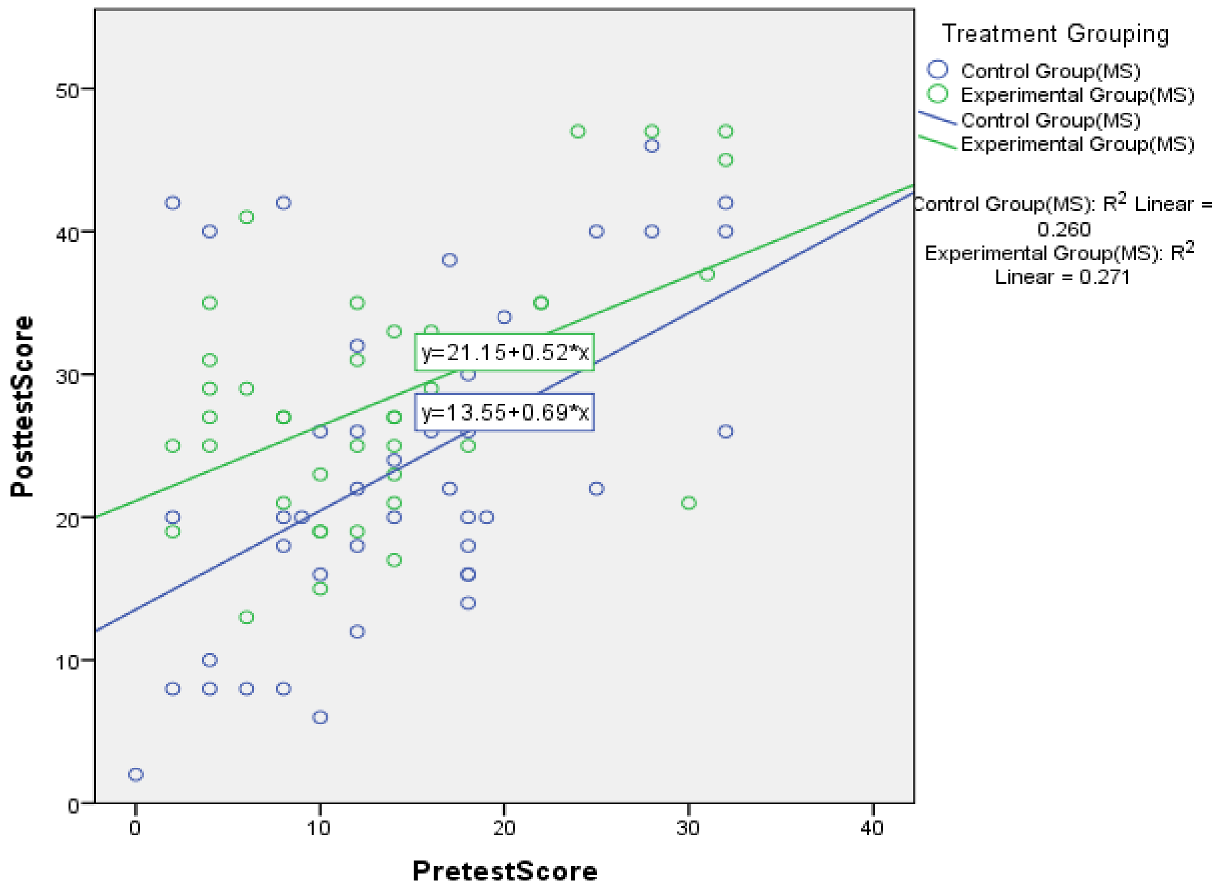


Figure 4-15 Linear relationship between covariate and dependent variable

Homogeneity of covariate coefficient (also called Homogeneity of regression slopes): Homogeneity of regression slopes implies, the lines for both groups should be parallel. Figure 4-15 indicates linearity, you will observe that the lines are not parallel. So, we want to confirm how far or near to being parallel are these lines. A way of testing this, is by conducting One-way ANCOVA to see if the interaction of treatment and covariate is not significant. The result of univariate analysis of covariance in Table 4-81 suggests that there is no interaction between treatment and pre-test, F(1,80), p =.468. As shown in table, the significance value of the interaction term (Treatment*Pre-test) is greater than 0.10 which shows it is not important in the fitted model. In addition, its partial eta squared is 0.007, which is near zero, showing that its contribution to the model is negligible. Hence, we can assume that the homogeneity of covariate coefficient has been satisfied. Another implication of this result is that, the pre-test score as a covariate can be used to explain some of the variance in the experimental outcome (the post-test).

Table 4-81 Testing for Homogeneity of Regression Slopes

Dependent Variable: Post-test Score								
Source	**Type III Sum of Squares**	**df**	**Mean Square**	**F**	**Sig.**	**Partial Eta Squared**	**Noncent. Parameter**	**Observed Powerc**
Corrected Model	2779.952b	3	926.651	11.794	.000	.307	35.382	.999
Intercept	6669.250	1	6669.250	84.883	.000	.515	84.883	1.000
Treatment	320.125	1	320.125	4.074	.047	.048	4.074	.514
Pretest	2203.940	1	2203.940	28.051	.000	.260	28.051	.999
Treatment * Pretest	41.778	1	41.778	.532	.468	.007	.532	.111
Error	6285.619	80	78.570					
Total	65538.000	84						
Corrected Total	9065.571	83						

Sample sizes sufficient to ensure enough power to detect the hypothesis: This appears to be satisfied by the information in Figure 4-16 which indicates, with alpha value of 0.05 and sufficiently high power of 0.85 we need a sample of 84 subjects for ANCOVA to detect any effect of the intervention.

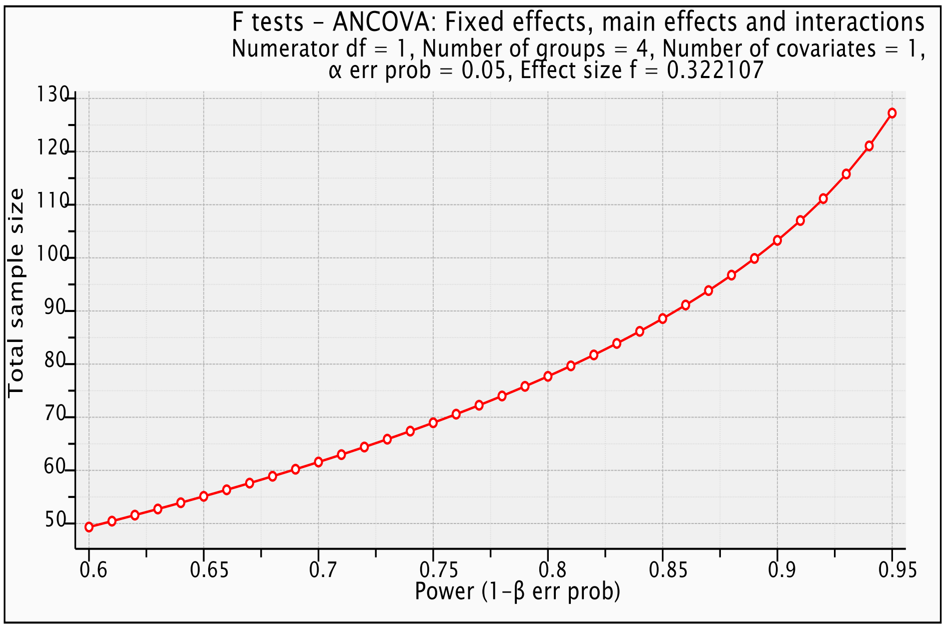


Figure 4-16 Power analysis graph showing required sample size

Assumption for paired sample t-test: To determine if exposure to a six-week constructionist *Scratch* programming pedagogy made any significant improvement on novice computer science students' programming ability, paired sample t-test will be employed. Before conducting the test, assumption of normality of difference scores must be satisfied (Field, 2018). To achieve this, a descriptive statistical analysis was conducted using the Explore feature in SPSS. The result is shown in Table 4-82

Table 4-82 Tests of Normality of Learning Gains in Scratch Class

Tests of Normality

		Kolmogorov-Smirnov			Shapiro-Wilk		
	Treatment Grouping	Statistic	Df	Sig.	Statistic	df	Sig.
Gain Score	Experimental Group(MS)	.145	42	.026	.973	42	.416

Results of the normality test in table provides conflicting information. While the first test (Kolmogorov-Smirnov) says the difference scores (i.e., Gain Score) of participants in the *Scratch* class is not a normal distribution (p = 0.26), the second test (Shapiro-Wilk) says that the data follows normal distribution (p = 0.416).

To further ascertain normality of the difference scores, we compute z-scores for the skewness and kurtosis of mean difference. Setting a cut-off point of z-score ±1.96 as specified by Mayers(2013), we determine if our data fulfil normality or not. The information on Table 4-83 shows that the z-score for both the skewness and kurtosis are within the limit for normality.

In the light of these facts, we can conclude that our mean difference score for the *Scratch* class has satisfied the normality assessment.

Table 4-83 z-score Test of Normality of Difference Score in Scratch Class

DV: Gain Score

Treatment Grouping	Mean	N	SD	Skewness	SE of Skewness	z-score of Skewness	Kurtosis	SE of Kurtosis	z-score of Kurtosis
Experimental	14.55	42	8.317	-0.012	0.365	-0.033	0.822	0.717	1.15

4.3 Hypothesis Testing

The following are the summary of findings on the eight hypotheses tested at 0.05 level of significance.

H_o1: There is no significant difference between the mean scores of the pre- and post- Introductory Programming Achievement Test (IPAT) of first-year polytechnic CS students, after a six-week *Scratch* programming instruction.

H_a1: There is significant difference between the mean scores of the pre- and post- Introductory Programming Achievement Test (IPAT) of first-year polytechnic CS students, after a six-week *Scratch* programming instruction.

In other word,

Table 4-84 Summary of Paired Sample t-Test (Scratch Class)

Group	**N**	**$\bar{X}$**	**SD**	**df**	**t**	**P**
Pre-test	42	13.88	8.47	41	-11.34	<.000
Post-test	42	28.43	8.52			

The result from Table 4-84 shows that there is significant mean difference in the achievement of experimental participants between their pre-test and the post-test (t (41) = -11.335, *p* = 0.001 (two-tailed)). A further observation of means shows that the achievement of the constructionist *Scratch* programming participants in their post-test ($\bar{X}$= 28.43, SD= 8.52) is significantly higher than in their pre-test ($\bar{X}$ = 13.88, SD = 8.47). This infers that the constructionist *Scratch* programming intervention improved the programming ability of the participants. The stated hypothesis is therefore rejected.

Table 4-85 Summary of Paired Sample t-Test (Control Class)

Group	N	$\bar{X}$	SD	Df	t	P
Pre-test	42	14.29	8.59	41	-5.718	<.000
Post-test	42	23.43	11.64			

Result from Table 4-85 reveals that there is significant mean difference in the achievement of participants in the lecture-based programming instruction (control) in the pre-test and post-test (t(41) = -5.718, *p* =.001). A further observation of means however shows that the achievement of the lecture-based programming participants in the post- test ($\bar{X}$ = 23.43, SD= 11.64) is higher than in the pre-test ($\bar{X}$ = 14.29, SD = 8.59). The observed difference is equally significant. This suggests that lecture-based instruction was also effective. However, the mean difference of the *Scratch* class is higher compared to that of the lecture-based class, suggesting that *Scratch* intervention was more effective at imparting first-year computer science students with programming knowledge.

H_o2: There is no significant difference in the mean scores of the post Introductory Programming Achievement Test (IPAT) of first-year polytechnic CS students in a constructionist *Scratch* class (experimental group) and those in the conventional class (control group), while controlling for their pretest scores.

Table 4-86 Summary of ANCOVA table showing the main effect of treatment

Source	Type III Sum of Squares	Df	Mean Square	F	Sig.	Partial Eta Squared
Corrected Model	2738.174a	2	1369.087	17.526	.000	.302
Intercept	6669.186	1	6669.186	85.375	.000	.513
Pretest	2213.174	1	2213.174	28.332	.000	.259
Treatment	**577.710**	**1**	**577.710**	**7.396**	**.008**	**.084**
Error	6327.398	81	78.116			
Total	65538.000	84				
Corrected Total	9065.571	83				

a. R Squared = .302 (Adjusted R Squared = .285)

Testing the effect of treatment while controlling for the students pre-test scores, an ANCOVA result in Table 4-86 revealed that that there was a significant effect of covariate, F(1, 81) = 28.332, p < .001, ῆ2 = 0.259, which indicates pre-test scores of students predicted their post-test scores. Result from Table 4-86 also shows that there was a significant main effect of treatment on achievement in programming, F (1, 81) = 7.396, p = 0.008, ῆ2 = 0.084. This means that there was a significant effect of the constructionist *Scratch* programming on achievements of participant in programming. The partial eta squared value (an effect size measure) of 0.084 indicates a medium effect, since according to Cohen (1988) the value of 0.01, 0.06 and 0.14 represent small, medium, and large effect respectively. Therefore, we reject the null hypothesis.

H$_o$3: Gender has no effect on the mean scores of the post Introductory Programming Achievement Test (IPAT) of first-year polytechnic CS students in a constructionist *Scratch* class (experimental group) and those in the conventional class (control group), while controlling for their pre-test scores.

Table 4-87 Summary of ANCOVA table showing the main effect of gender

Source	Type III Sum of Squares	Df	Mean Square	F	Sig.	Partial Eta Squared
Corrected Model	2743.584a	3	914.528	11.573	.000	.303
Intercept	6602.759	1	6602.759	83.553	.000	.511
Pretest	2202.715	1	2202.715	27.874	.000	.258
Treatment	580.518	1	580.518	7.346	.008	.084
Gender	**5.410**	**1**	**5.410**	**.068**	**.794**	**.001**
Error	6321.988	80	79.025			
Total	65538.000	84				
Corrected Total	9065.571	83				

a. R Squared = .303 (Adjusted R Squared = .276)

Table 4-87 shows there was no significant main effect of gender on achievement in programming of participants (F (1, 80) = .068, *p* = .794, ῃ2 = 0.001). This means that gender did not have effect on the participants' achievement in programming. Therefore, we fail to reject null hypothesis.

Table 4-88 Showing the mean values of gender on achievement

Group	Gender	Mean	Std. Deviation	N
Experimental	Male	29.42	8.303	33
	Female	24.78	8.800	9
	Total	28.43	8.523	42
Control	Male	22.81	11.823	27
	Female	24.53	11.649	15
	Total	23.43	11.648	42
Total	Male	26.4500	10.48716	60
	Female	24.6250	10.46656	24
	Total	25.9286	10.45102	84

Table 4-88 reveals that both gender in the treatment group underwent the constructionist *Scratch* programming instruction and it has a positive effect on their achievement, in programming. A further observation of means shows that the achievement of males in programming ($\bar{X}$ = 29.42, SD= 8.303) improved more than that of females ($\bar{X}$= 24.78, SD= 8.800). In contrast, the achievement of females in the control group ($\bar{X}$= 24.53, S.D.= 11.649) improved than that of the males ($\bar{X}$ = 22.81, SD= 11.823). However, the difference in the observed improvements in both groups was not statistically significant.

H$_o$4: Age has no effect on the mean score of the post Introductory Programming Achievement Test (IPAT) of first-year polytechnic CS students in a constructionist *Scratch* class (experimental group) and those in the conventional class (control group), while controlling for their pre-test scores.

Table 4-89 Summary of ANCOVA table showing the main effect of age

Source	**Type III Sum of Squares**	**Df**	**Mean Square**	**F**	**Sig.**	**Partial Eta Squared**
Corrected Model	2939.900a	5	587.980	7.487	.000	.324
Intercept	5567.317	1	5567.317	70.890	.000	.476
Pretest	1935.933	1	1935.933	24.651	.000	.240
Treatment	549.816	1	549.816	7.001	.010	.082
Age	201.726	3	67.242	.856	.468	.032
Error	6125.671	78	78.534			
Total	65538.000	84				
Corrected Total	9065.571	83				

a. R Squared = .324 (Adjusted R Squared = .281)

Table 4-89 showed that there was no significant main effect of age on achievement in programming of participants (F (1, 78) = .856, *p* =.468, ῆ2 = 0.032). This means that age did not have effect on achievement in programming of the participants. Therefore, we fail to reject the null hypothesis.

Table 4-90 Showing mean values of age on achievement

Treatment Grouping	Age Group	Mean	Std. Deviation	N
	16 - 18 years	30.00	6.542	6
	19 - 21 years	29.35	10.253	17
Experimental	22 - 24 years	25.50	6.044	16
	Above 24 years	35.67	10.263	3
	Total	28.43	8.523	42
	16 - 18 years	25.67	11.272	6
	19 - 21 years	24.00	12.042	17
Control Group	22 - 24 years	21.41	11.462	17
	Above 24 years	29.00	18.385	2
	Total	23.43	11.648	42
	16 - 18 years	27.83	9.074	12
	19 - 21 years	26.68	11.342	34
Total	22 - 24 years	23.39	9.334	33
	Above 24 years	33.00	12.268	5
	Total	25.93	10.451	84

Result from Table 4-90 reveals that the three age groups in the treatment group, underwent constructionist *Scratch* programming and it has a positive effect on their achievement in programming. A further observation of means show that, the achievement in programming of participants between the ages of 24 years and above in the experimental group ($\bar{X}$= 35.67, S.D= 10.236) improved than that of those participants between the ages of 16-18 years ($\bar{X}$ = 30.00, S.D= 6.542), between the ages of 19-21 years ($\bar{X}$ = 29.35, S.D= 10.253) and those between the ages of 22-24 years ($\bar{X}$ = 25.50, S.D= 6.044). Also, the achievement of those participants between the ages of 24 years and above in the control group who were given lecture based instruction ($\bar{X}$ = 29.00, S.D=

18.385) improved than that of participants between the ages of 16-18 years ($\bar{X}$= 25.67, S.D= 11.272), between the ages of 19-21 years ($\bar{X}$ = 24.00, S.D= 12.042) and those between the ages of 22-24 years ($\bar{X}$ = 21.41, S.D= 11.462). However, as indicated earlier the differences between observed improvements were not statistically significant.

H$_o$5: Academic background has no effect on the mean score of the post Introductory Programming Achievement Test (IPAT) of first-year polytechnic CS students in a constructionist *Scratch* class (experimental group) and those in the conventional class (control group), while controlling for their pre-test scores.

Table 4-91 Summary of ANCOVA table showing the main effect of academic background

Source	**Type III Sum of Squares**	**df**	**Mean Square**	**F**	**Sig.**	**Partial Eta Squared**
Corrected Model	2747.885a	3	915.962	11.599	.000	.303
Intercept	5316.225	1	5316.225	67.319	.000	.457
Pre-test	2202.951	1	2202.951	27.896	.000	.259
Treatment	578.285	1	578.285	7.323	.008	.084
Academic background	9.711	1	9.711	.123	.727	.002
Error	6317.686	80	78.971			
Total	65538.000	84				
Corrected Total	9065.571	83				

a. R Squared = .303 (Adjusted R Squared = .277)

Table 4-91 showed that there was no significant main effect of academic background on achievement in programming of participants (F (1, 80) = .123, *p* =.727, ῆ2 = 0.002). This means that academic background did not have effect on achievement in programming of the participants. Therefore, we fail to reject the null hypothesis.

Table 4-92 Showing mean values of academic background on achievement

Group	**Academic background**	**Mean**	**Std. Deviation**	**N**
Treatment	Low	28.25	9.027	32
	Average	29.00	7.055	10
	Total	28.43	8.523	42
Control	Low	23.06	11.584	32
	Average	24.60	12.403	10
	Total	23.43	11.648	42
Total	Low	25.66	10.628	64
	Average	26.80	10.077	20
	Total	25.93	10.451	84

Table 4-92 reveals that participants with both academic backgrounds in the treatment group underwent the constructionist *Scratch* programming and it has a positive effect on their achievement in programming. A further observation of means shows that the achievement in programming of those participants with average academic background ($\bar{X}$ = 29.00, SD= 7.055) improved more than those with low academic background ($\bar{X}$ = 28.25, SD= 9.027). Also, the achievement programming instruction of those with average academic background in the control group, who were exposed to lecture based ($\bar{X}$ = 24.60, SD= 12.403) improved than that those with low academic background ($\bar{X}$ = 24.06, SD= 11.584). However, the difference between the two groups was not statistically significant.

H$_o$6: Prior programming experience has no effect on the mean score of the post Introductory Programming Achievement Test (IPAT) of first-year polytechnic CS students in a constructionist *Scratch* class (experimental group) and those in the conventional class (control group), while controlling for their pre-test scores.

Table 4-93 Summary of ANCOVA table showing the main effect of prior program writing

Source	Type III Sum of Squares	df	Mean Square	F	Sig.	Partial Eta Squared
Corrected Model	2840.301a	3	946.767	12.167	.000	.313
Intercept	2072.755	1	2072.755	26.637	.000	.250
Pre-test	1682.729	1	1682.729	21.624	.000	.213
Treatment	573.572	1	573.572	7.371	.008	.084
Prior program writing	**102.127**	**1**	**102.127**	**1.312**	**.255**	**.016**
Error	6225.271	80	77.816			
Total	65538.000	84				
Corrected Total	9065.571	83				

a. R Squared = .313 (Adjusted R Squared = .288)

Table 4-93 showed that there was no significant main effect of prior programming writing on achievement in programming of participants (F (1, 80) = 1.312, p =.255, ῆ2 = 0.016). This means that prior programming writing had no effect on achievement in programming of the participants. Therefore, we fail to reject the null hypothesis.

Table 4-94 Showing mean values of prior program writing on achievement

Group	Prior programming	Mean	Std. Deviation	N
	No background	28.02	8.211	41
Experimental	Some background	45.00	.	1
	Total	28.43	8.523	42
	No background	22.98	11.412	41
Control	Some background	42.00	.	1
	Total	23.43	11.648	42
	No background	25.50	10.201	82
Total	Some background	43.50	2.121	2
	Total	25.93	10.451	84

Table 4-94 reveals that participants with prior programming writing, in the treatment group underwent the constructionist *Scratch* programming and it has a positive effect on their achievement in programming. A further observation of means shows that the achievement in programming of those participants with some background ($\bar{X}$ = 45.00, S.D = -) improved than those with no background ($\bar{X}$ = 28.02, SD= 8.211). Also, the achievement of those with some background in the control group ($\bar{X}$ = 42.00, SD = -) improved than those with none ($\bar{X}$ = 22.98, SD= 11.412). Though, the difference in the observed improvements in both groups was not statistically significant.

H_o7: Prior visual art experience has no effect on the mean score of the post Introductory Programming Achievement Test (IPAT) of first-year polytechnic CS students in a constructionist *Scratch* class (experimental group) and those in the conventional class (control group), while controlling for their pre-test scores.

Table 4-95 Summary of ANCOVA table showing the main effect of prior visual art

Source	Type III Sum of Squares	df	Mean Square	F	Sig.	Partial Eta Squared
Corrected Model	2866.009a	3	955.336	12.328	.000	.316
Intercept	5483.182	1	5483.182	70.756	.000	.469
Pre-test	2316.074	1	2316.074	29.887	.000	.272
Treatment	582.281	1	582.281	7.514	.008	.086
Prior visual art	127.836	1	127.836	1.650	.203	.020
Error	6199.562	80	77.495			
Total	65538.000	84				
Corrected Total	9065.571	83				

a. R Squared = .316 (Adjusted R Squared = .290)

Table 4-95 showed that there was no significant main effect of prior visual art on achievement in programming of participants (F (1, 80) = 1.650, *p* =.203, ῃ2 = 0.020). This means prior visual art had no effect on achievement of participants in programming. Therefore, we fail to reject the null hypothesis.

Table 4-96 Showing mean values of prior visual knowledge on achievement

Group	Prior visual art	Mean	Std. Deviation	N
Experimental	No background	27.09	6.156	22
	Some background	29.90	10.513	20
	Total	28.43	8.523	42
Control Group	No background	23.73	10.977	22
	Some background	23.10	12.624	20
	Total	23.43	11.648	42
Total	No background	25.41	8.958	44
	Some background	26.50	11.972	40
	Total	25.93	10.451	84

Table 4-96 reveals that participants with prior visual art in the treatment group underwent the constructionist *Scratch* programming and it has a positive effect on their achievement in programming. A further observation of means shows that the achievement in programming of those participants with some background ($\bar{X}$= 29.90, SD=10.513) improved than those with none ($\bar{X}$ = 27.09, SD= 6.156). In contrast, the achievement of those with some background in the control group ($\bar{X}$ = 23.10, SD= 12.624) was a bit less than that those with none ($\bar{X}$ = 23.73, SD= 10.977). However, the difference in the observed improvements in both groups was not statistically significant here.

H$_o$8: Treatment, Gender, academic background, prior programming experience and prior visual art have no interaction on the mean score of the post Introductory Programming Achievement Test (IPAT) of first-year polytechnic CS students in a constructionist *Scratch* class (experimental group) and those in the conventional class (control group), while controlling for their pre-test scores.

Table 4-97 Summary of ANCOVA table showing the significant interaction effect of treatment, gender, age, academic background, prior programming writing and prior visual on achievement

Source	**Type III Sum of Squares**	**Df**	**Mean Square**	**F**	**Sig.**	**Partial Eta Squared**
Corrected Model	4461.366a	35	127.468	1.590	.070	.547
Intercept	2410.008	1	2410.008	30.053	.000	.395
Pre-test	1072.895	1	1072.895	13.379	.001	.225
Treatment	577.710	1	577.710	7.375	.008	.099
Treatment * gender	290.609	2	145.304	1.812	.175	.073
Treatment * age	223.231	6	37.205	.464	.831	.057
Treatment * academic background	9.024	2	4.512	.056	.945	.002
Treatment * prior program writing	272.406	2	136.203	1.698	.194	.069
Treatment * prior visual art	2.528	2	1.264	.016	.984	.001
Treatment * gender * age * academic background * prior program writing * prior visual art	1379.204	19	72.590	.905	.580	.272
Error	3688.829	46	80.192			
Total	65470.000	84				
Corrected Total	8150.195	81				

a. R Squared = .547 (Adjusted R Squared = .203)

The result of the findings in Table 4-97, revealed that there was a significant main effect of treatment on achievement in programming of the participants (F (1,46) = 7.375; p < 0.05, ῆ2 = 0.099). However, there is no significant interaction effect of treatment on gender, age, academic background, prior programming writing and prior visual on achievement in programming of participant (F (1, 46) = .905, *p* =.580, ῆ2 = 0.272). This implies that the participants in the experimental groups benefitted from the treatment, as they were able to get higher achievement score in programming compare to their counterparts in the control group.

Table 4-98 Estimated marginal means for the treatment and control group

Treatment Grouping	**Mean**	**Std. Error**	**95% Confidence Interval**	
			Lower Bound	Upper Bound
Control Group(MS)	24.094	1.727	20.621	27.566
Experimental Group(MS)	28.158	1.750	24.639	31.676

Table 4-98 shows that the experimental group has the highest mean score ($\bar{X}$ = 28.158) compared to the control group ($\bar{X}$ = 24.094). This implies that students who were exposed to treatment (constructionist *Scratch* programming) improved more in their programming ability, than those in the control group who were exposed to the conventional programming instruction.

Table 4-99 Estimated marginal means for the treatment and gender

Gender	Mean	Std. Error	95% Confidence Interval	
			Lower Bound	Upper Bound
Male	26.120	1.326	23.480	28.759
Female	24.656	2.155	20.367	28.944

Table 4-99 shows that males had the higher mean score ($\bar{X}$ = 26.120) compared to females ($\bar{X}$ = 24.656). This implies that the treatment was more effective on the male participants than females. That is, males benefitted more in the constructionist *Scratch* programming than the females.

Table 4-100 Estimated marginal means for the treatment and age

Age Group	Mean	Std. Error	95% Confidence Interval	
			Lower Bound	Upper Bound
16 - 18 years	27.833	2.882	22.090	33.577
19 - 21 years	27.176	1.739	23.712	30.641
22 - 24 years	24.063	1.765	20.545	27.580
Above 24 years	32.333	4.557	23.252	41.414

Table 4-100 shows that students whose ages falls between 24 years and above, has the highest mean score ($\bar{X}$ = 32.333) than those between 16-18 years ($\bar{X}$ = 27.833), 19-21 years ($\bar{X}$ = 27.176) as well as those between the ages of 22-24 years ($\bar{X}$ = 24.063). This implies that those participants between the ages of 24 years and above improved in programming learning than their counterparts.

Table 4-101 Estimated marginal means for the treatment and Academic background

Academic Background Index	**Mean**	**Std. Error**	**95% Confidence Interval**	
			Lower Bound	Upper Bound
Low-Achieving	25.607	1.270	23.076	28.138
Average Achieving	30.983	2.861	25.282	36.685

Table 4-101 shows that average-achieving participants had the higher mean score ($\bar{X}$ = 30.983) than low-achieving participants ($\bar{X}$ = 25.607). This implies that the treatment was more effective on the average-achieving participants, compare to the low-achieving participants. That is, first-year computer science students with average academic background benefitted more in the constructionist *Scratch* programming than those with low academic background.

Table 4-102 Estimated marginal means for the treatment and prior program writing

Prior Program Writing	**Mean**	**Std. Error**	**95% Confidence Interval**	
			Lower Bound	Upper Bound
No Background	25.166	1.354	22.469	27.863
Some Background	43.500a	7.129	29.296	57.704

Table 4-102 shows that students with some prior program writing background had the highest mean score ($\bar{X}$ = 43.500) than those with no prior program writing ($\bar{X}$ = 25.166). This implies that the treatment was more effective on the participants with some prior program writing background than those with no background.

Table 4-103 Estimated marginal means for the treatment and visual art background

Visual Art Background	**Mean**	**Std. Error**	**95% Confidence Interval**	
			Lower Bound	Upper Bound
No Background	24.375	1.970	20.449	28.301
Some Background	31.805	2.679	26.466	37.144

Table 4-103 shows that students with some visual art background had the highest mean score ($\bar{X}$ = 31.805) than those with no visual art background ($\bar{X}$ = 24.375). This implies that, the treatment was more effective on the participants with some visual art background than those with none.

4.4 Discussion of Findings

This study examined the impact of *Scratch*, a visual programming environment on the achievement of first year computer science students in programming in some Nigerian Polytechnics. Paired sample t-test and ANCOVA as statistical tools were used, to analyse the data collected. The findings are discussed below:

H_o1: There is no significant difference between the mean scores of the pre- and post- Introductory Programming Achievement Test (IPAT) of first-year polytechnic CS students, after a six-week *Scratch* programming instruction.

The above hypothesis was rejected because the result in Table 4-84 clearly shows there is statistically significant difference between the post-test and pre-test scores of those CS1 students exposed to a constructionist *Scratch* intervention. This implies that even when those in the *Scratch* class were not specifically given lectures on programming, the six-week intervention led to significant students' learning gains in conceptual and procedural programming knowledge. This finding is consistent with earlier studies (Hijón-Neira et al., 2021; Meerbaum-Salant et al., 2013; Tijani et al., 2020) who found programming learning gains after students went through a period of *Scratch* instruction. In a meta-analysis by Scherer et al. (2020) it was found that employing Scratch even for a short while in novice programming classes led to positive overall effect. The result of this study aligns with those of Cárdenas-Cobo et al. (2021) who found Scratch led to remarkable improvement in the achievement of university students in a CS1 known for high failure rates. The students were from one of low SES parts of Ecuador. In contrast, this outcome negates those of Kalelioğlu & Gülbahar (2014) who found no significant difference between the pre-test and post-test of 5th grade Turkish primary school pupils after five weeks exposure to *Scratch.*

H_o2: There is no significant difference in the mean scores of the post Introductory Programming Achievement Test (IPAT) of first-year polytechnic CS students in a constructionist *Scratch* class (experimental group) and those in the conventional class (control group), while controlling for their pre-test scores.

Or,

H_o2: There is no significant main effect of treatment on programming achievement of first year computer science students in Nigerian polytechnics

The hypothesis stated above was rejected because the result in Table 4-86 clearly shows that there is a significant main effect of treatment on the mean post Introductory Programming Achievement Test (IPAT) scores, between first-year polytechnic CS students in a constructionist *Scratch* class (experimental group) and those in the conventional class (control group). This finding is in congruence with the findings of Kim et al (2012) who discovered that unlike the traditional computer programming languages, *Scratch* helped Korean university pre-service education students, to focus on what they could do with programming languages, rather than struggling with learning the language syntax. This resulted into implicit rather than explicit learning of programming concepts, development of programming skills, creativity, confidence, and collaborations among students. In a similar study, Erol and Kurt (2017) explored the impact of *Scratch* on the motivation and achievement of first-year Turkish university students, in an introductory programming course. The authors reported that though both groups were not significantly different in motivation and programming knowledge at the beginning, they were significant differences in both outcome variables, at the middle and end of the course in favour of the treatment group. They found that while motivation increased in the middle and end of the course in the treatment group; it plunges at the middle, and increased slightly at the end, in the control group. However, both groups' programming knowledge increased in the middle and at the end, though, the mean achievement of the treatment was significantly higher compared to the control group on both occasions. The result of this study is supported by similar research by Cetin (2016) in which higher education students were exposed to *Scratch* (in the experimental class) and C (in the control class). After six weeks of instruction, the experimental group performed significantly better in programming achievement compared to the control. The finding also lends credence with that of Topalli and Cagiltay (2018), who conducted a longitudinal study of undergraduate computing engineering students from a Turkish university exploring the effects of enriching traditional CS1 course with *Scratch* games developments. The study investigated the impact of this intervention on the performance of students in the CS1 course, their performance in the final year projects and overall performance at the end of their studies in the university. The authors reported that the treatment had significantly higher mean achievement compared to the control. They also found similar results in favour of the treatment in the senior year project and the overall CGPA. In a meta-analysis comparing the impacts of block-based and text-based programming environments on students' programming cognition, Xu et al.(2019) found the former to have greater, albeit moderate effect.

By implication *Scratch* visual programming was effective at engaging and in improving the programming skills of polytechnic students. Level of students' engagement in first year, has been found to have significant effect on their achievement, even for those disadvantaged by their prior educational performance or parental status (Kuh et al., 2008). Another probable reason for *Scratch*'s impact on student's achievement comes from improved self-efficacy. Research suggests that increased self-efficacy leads to increase performance in programming. In the study by Cetin (2016), students' self-efficacy which was lower before *Scratch* particularly for the females, was found to have increased after *Scratch* programming instruction. In summary, the finding confirms the effectiveness of the treatment tool on novice students in programming. Although, the intervention was effective, Table 4-98 clearly showed the marginal difference between the intervention and the control group.

H_o3: Gender has no effect on the mean scores of the post Introductory Programming Achievement Test (IPAT) of first-year polytechnic CS students in a constructionist *Scratch* class (experimental group) and those in the conventional class (control group), while controlling for their pre-test scores.

Or

H_o3: There is no significant main effect of gender on achievement in programming among first year students of computer science in some Nigerian Polytechnics

The result in Table 4-87 indicates a value of $p > .05$ for gender, therefore we fail to reject the above hypothesis. The result reveals that in this study, gender had no significant main effect on achievement in programming among computer science students in some Nigerian Polytechnics. By implication, gender differences of first year computer science students, have no significant impact on their achievements in programming. In essence, gender did not predict CS1 students' achievements in programming. The report of this study aligns with the findings of some other studies, that discovered the insignificance of gender to the achievement in programming of first year students of computer science(Gjelsten et al., 2021; Lishinski & Rosenberg, 2021; Veerasamy et al., 2019). For instance, Ayalew et al.(2018) found no significant correlation between gender and achievement in introductory programming among first-year university students in Botswana. The finding also goes in line with that of Rubio et al.(2015) who employed a contextualized computing pedagogy involving MATLAB programming and Arduino so that programming concepts become tangible to students using sound, movement, and light. This intervention was aimed at assessing, in comparison to the traditional instruction, gender differences in programming perceptions and learning outcomes of first-year Spanish university students, who all had no prior experience in programming. The authors reported that while the three programming perceptions scores for males and females showed consistently diverging trends, closing with a significant difference in favour of male students in the control group, these scores, though exhibited mixed trends in course of intervention in the experimental group, gender differences remained consistently insignificant, suggesting that the contextualized instruction closed the gender gap in programming perception. Within each group, there was no significant gender differences in the programming learning outcomes. However, the failure rate revealed a different picture. While female students' failure rate doubled those of male students in the control group, the rate was essentially the same in the experimental group. The finding also goes in line with that of Sabitzer and Pasterk (2014) who employed a neuroeducation pedagogy called "brain-based programming" and examined its effects on students' achievement. They found that the intervention was effective at closing the gender gap, as the performance of both males and females was not significantly different in the experimental group. In the control group, the males performed significantly better than the female.

From a social viewpoint**,** it was thought that the higher rate of programming anxiety among females might be due to differences in gender roles, and that greater equality between the sexes regarding their status, self-efficacy or opportunities for prior experience with programming should balance the rates and prevalence of programming anxiety in male and female. For instance, Gjelsten et al.(2021) identified the reason for disappearance of gendered difference in CS1 achievement: prior programming experience in high school of female CS1 students. That, suggests prior programming experience is a mediating variable between gender and CS1 achievement. Another mediating variable is, first year class is engagement(Kuh et al., 2008). Providing engaging learning contents that interest females may also level the playing field with males who have come into CS1 with higher self-efficacy and prior programming experience. For instance, CS1 assignments focused on people rather than things were found more engaging by female CS1 students (Marcher et al., 2021). That likely leads increase in self-efficacy and eventual increase in performance in CS1.

H_o4: Age has no effect on the mean score of the post Introductory Programming Achievement Test (IPAT) of first-year polytechnic CS students in a constructionist *Scratch* class (experimental group) and those in the conventional class (control group), while controlling for their pre-test scores.

or

H_o4: There is no significant main effect of age on achievement in programming among first year students of computer science in some Nigerian Polytechnics

The result in Table 4-89 indicates a value of $p > .05$ for age, therefore we fail to reject the above hypothesis.

shows that age in this study had no significant main effect, on achievement in programming among computer science students in some Nigerian Polytechnics. By implication, age difference of first year students of computer science has no significant impact on their achievement in programming. In essence, achievement in programming is not a function of age. The report of this study aligns with the findings of some other studies, that discovered the insignificance of age to the achievement in programming of first year students of computer science. For instance, Hermans and Aivaloglou (2017) in a study involving Dutch elementary students explored the hypothesis that young age is not a barrier to learning advanced CS concepts. The authors found from the comparison of students' overall mean scores, in both programming and software engineering concepts, that there was no significant difference, indicating the students performed equally well in both. While the comparison of the 11-12 age group with 13-14 age group students indicated there were no differences in many concepts, (though significant differences with insignificant effect sizes were found in few topics), nevertheless, significant differences (with at least a small effect size) were recorded in concepts of procedures and operators in favour of the older students, probably suggesting that there are programming or software engineering concepts that are too difficult to understand for younger students. However, with one convenience sample with no control group for comparison makes the result only suggestive.

The finding corresponds with Atmatzidou and Demetriadis (2016) who employed Educational Robotics (ER) and guided instructional approach, in an introductory class for secondary school students to explore age and gender differences in the development of their Computational Thinking skills. They reported that though girls performed initially lower than boys in CT skills, they were able to catch up, and in the end, there was no significant gender or age difference. The finding however negates some findings by Chen et al.(2019) who took a different approach to the above studies, to find answers to the question of the relationship between age and CS1 achievement. They employed ex post facto quasi-experimental design to explore retrospectively, the effects of the type of programming environment and student's age at the time of being introduced to programming on their attitude at the beginning, and their achievement at the end of a college CS1 course experience. They reported that significant interaction effect of age when students first started to learn to program, and the programming environment on students' positive attitude towards CS, suggesting that students' positive attitude towards CS depends on what age and how they were introduced to programming. For final CS1 grade, the study found significant effects of both graphic and textual programming (in comparison to control) on students' achievement, indicating it is better to learn to program using any of the two environments than none. The performance of students with prior programming in CS1 depends on the age they were introduced to programming. Those introduced to graphic programming between the age 6-10 or younger are likely to score higher in CS1 than those introduced to textual at the same age. There was no significant difference in the achievements of students introduced to programming between age 11-14 and 15-18years in both treatment groups, suggesting that both modes work equally well in preparing students for college CS1.

H_o5: Academic background has no effect on the mean score of the post Introductory Programming Achievement Test (IPAT) of first-year polytechnic CS students in a constructionist *Scratch* class (experimental group) and those in the conventional class (control group), while controlling for their pre-test scores.

Or

H_o5: There is no significant main effect of academic background on achievement in programming among first year students of computer science in some Nigerian Polytechnics

The result in Table 4-91 indicates a value of $p > .05$ for academic background, therefore we fail to reject the above hypothesis. That shows that academic background had no significant main effect on achievement, in programming among computer science students in some Nigerian Polytechnics. By implication, academic backgrounds of first year computer science students had no significant influence on their programming achievement. In essence, achievement in programming is not necessarily a function of academic background. The report of this study aligns with the findings of some other studies, that discovered the insignificance of academic background to the achievement in programming of first year students of computer science(Rizvi & Humphries, 2012). The finding is not consonance with that of Ramos(2018) who reported that academic background played a role in predicting Filipino CS1 students' programming performance. The author found that the higher the student's academic level in the university entrance exam, the higher their final grades in CS1. A similar study by Mindetbay et al.(2019) also found that academic achievement of 8th grade Kazakhstani pupils predicted their computational performance. This suggests that academic background promotes students' capacity for learning and their achievements in programming. However, in this study the effect of that background did not reach a level that makes it a significant factor. An explanation for this may be that *Scratch* was effective in engaging and increasing the self-efficacy of the low achieving students, leading to learning gains that made them to catch up with the average achieving students. Evidence from their pre-test scores in Table 4-51, post-test scores in Table 4-58 and learning gains in Table 4-59 suggests this. The study by Rizvi and Humphries (2012) also substantiates that earlier assertion. They employed a *Scratch* intervention course (CS0) taken before CS1, by at-risk first-year computer science students in a US university. They found no significant difference in the CS1 post-test scores between those with weak mathematical background in the at-risk group exposed to *Scratch* and those in the conventional programming class.

H_o6: Prior programming experience has no effect on the mean score of the post Introductory Programming Achievement Test (IPAT) of first-year polytechnic CS students in a constructionist *Scratch* class (experimental group) and those in the conventional class (control group), while controlling for their pre-test scores.

Or,

H_o6: There is no significant main effect of prior programming writing on achievement in programming among first year students of computer science in some Nigerian Polytechnics

The result in Table 4-93 indicates a value of $p > .05$ for prior programming writing, therefore we fail to reject the above hypothesis. Table 4-93 shows that prior programming writing had no significant main effect on achievement in programming among computer science students in some Nigerian Polytechnics. By implication, prior programming writing of first year students of computer science has no significant impact on the programming achievement. In essence, achievement in programming is not necessarily a function of prior programming writing. The finding did not lend credence with that of Wilson and Shrock (2001), who conducted a study exploring the relationship between twelve predictive variables and students' achievement in CS1. Some of the factors included in their model are math background, comfort level, performance attribution to luck, previous programming experience, etc. The finding revealed that comfort level, math background, and performance attribution to luck contributed the most to success, with a negative influence of luck. This suggests while comfort level and math background as expected contributed positively to students’ success, attributing performance to luck probably led student to put in less effort into their study, and thus negatively impacting on their achievement.

However, the finding corroborates with that of Silva-Maceda, David Arjona-Villicana, & Edgar Castillo-Barrera (2016) who - to explore the impacts of CS1 pedagogical approach and students’ organismic factors on their CS1 achievements, - conducted a longitudinal study involving seven cohorts (N = 1168) of students in a central Mexican university. The authors reported that there was no statistically significant difference, after including initial students’ ability in the analysis. This suggests that students spending more time by first being exposed to programming, are better than taking CS1 without a prior programming course, probably evidence of the benefit of higher time-on-task. Also, it is better to have a prior programming experience using whatever pedagogical approach than having none. The finding of this study is also inconsistent with that of Hagan & Markham (2000) in a study involving first-year computer science students in an Australian university, which sought to confirm whether students' prior experience in programming has an impact on their achievement in the university CS1 course. The finding revealed that while a significant statistical difference in the four formative assessments was found between students with none and those with some experience in programming, there was no statistically significant difference in their summative assessment. The finding also negates that of Wilson and Shrock (2001) who reported that the grades of students were strongly associated with their prior knowledge of the programming course.

In contrast, the report of this study aligns with the findings of some other studies, that discovered the insignificance of prior programming writing, to the achievement in programming of first year students of computer science. For instance, Chen et al. (2021) found that taking AP CS (a programming course) in high school by first-year American higher ed CS students had no significant effect on their CS1 grades. This finding is also consistent Ayalew et al(2018) who found no correlation between prior programming on the achievement in introductory programming among first-year university students in Botswana.

An explanation for this result may be understood by fact that, *Scratch* enhanced the engagement and self-efficacies of the students with no prior programming writing experiences. Such mediating variables have been found to promote students' achievements. However, the limitation of this result is the fact that only 1 out of 42 students in each group, had prior program writing experience. That seriously unbalanced within-group effect may be responsible for the lack of effect in the between-group outcome. Nevertheless, considering the information in Table 4-94, the *Scratch* intervention enhanced the achievement of students with no prior program writing than those in the conventional instruction.

H_o7: Prior visual art experience has no effect on the mean score of the post Introductory Programming Achievement Test (IPAT) of first-year polytechnic CS students in a constructionist *Scratch* class (experimental group) and those in the conventional class (control group), while controlling for their pre-test scores.

Or,

H_o7: There is no significant main effect of prior visual art on achievement in programming among first year students of computer science in some Nigerian Polytechnics.

Result in Table 4-95 indicates a value of p > .05 for prior visual arts, therefore we fail to reject the above hypothesis. In other word, prior visual arts had no significant main effect, on achievement in programming among computer science students in some Nigerian Polytechnics. By implication, prior visual artistic abilities of first year students of computer science have no significant impact on their programming achievements. In essence, achievement in programming is not necessarily a function of prior visual art. Although, **Table *4-103*** shows those with prior visual art had higher mean post-test achievement, compared to those with none**,** probably suggestion a correlation. But our hypothesis test is for causation and the result reveals no such. The report of this study aligns with the findings of some other studies, that discovered the insignificance of prior visual art ability to the achievement in programming of first year students of computer science. The finding supports that of Parham-Mocello, Erwig and Dominguez (2020) which sought to confirm the hypothesis that, exposing students with little or no programming experience or interest in computing to only "Story Programming" will likely see better results for Drop-Fail-Withdrawal (DWF) rate and students' interest than those exposed to coding-first. The finding revealed that there was no significant difference in the grade point average of students in both classes. This suggests that programming or computational thinking can be learnt by pedagogical means other than coding on a computer - usually called "CS unplugged". The finding also goes in line with that of Hermans and Aivaloglou (2017) who explored the impact of introducing students to programming with or without the computer (usually referred to as plug and unplugged CS) on their learning outcomes. The finding revealed no significant difference between the two groups in their programming knowledge. This study's result is consistent with that of Gestwicki & Ahmad (2010) who found no correlation between university students' creative achievement and their academic achievement, after doing a media computation oriented CS1 course. The reason for this result may be that, there is no direct link between prior visual art and CS1 achievement, or some other variables mediate or moderate the relationship between the two. In the same vein, the finding also negates that of Reid (2005) which shows that students that are familiar with visual art skills when compared with students from non-visual arts are better adjusted and have better skills in computer and programming.

H_o8: Treatment, Gender, academic background, prior programming experience and prior visual art have no interaction on the mean score of the post Introductory Programming Achievement Test (IPAT) of first-year polytechnic CS students in a constructionist *Scratch* class (experimental group) and those in the conventional class (control group), while controlling for their pre-test scores. Or,

H_o8: There was no significant interaction effect of treatment, gender, age, academic background, prior programming writing and prior visual art on programming achievement of first year computer science students in Nigerian polytechnics

The result in Table 4-97 shows that there was no significant interaction effect of treatment, gender, age, academic background, prior programming writing and prior visual programming on students' achievement. Hence, we fail to reject the above hypothesis. Consistent with some current research on programming achievement of first year computer, this study did not find evidence for the significant interaction effect of treatment, gender, age, academic background, prior programming writing and prior visual programming on the programming achievement of first year computer (Guo, 2020; Hyeonjin Kim et al., 2012). In a study of first-year CS students at an American University, Guo(2020) operationalised age as the time the student started to program before enrolling. The study found no interaction between student's age and gender on their CS1 or CS2 achievement. Surprisingly, treatment, gender, age, academic background, prior programming writing and prior visual programming did not predict changes in programming achievement of first year computer in this study. The finding negates that of Chen et al.(2019) who conducted a retrospective study to explore the impacts of type of programming environments, with which students were initiated into programming and the age of introduction on their attitude towards CS and achievement in CS1. The participants comprised those with graphic, textual and none programming backgrounds, resulting into 2 treatment groups (i.e., Graphic and Textual programming) and 1 control group (i.e., No prior programming). The authors reported that there were significant differences between graphic and control group, and between the textual and control group, in favour of each treatment group. Including age as a covariate in the linear regression analysis, they found significant interaction effect of prior programming experience and age on the achievement of students in the CS1. That implied students' achievement in CS1 depended on the type of programming environment they were exposed to and the age before their enrolment into higher education.

It is interesting to point out that this study did not find interaction of treatment and usual variables on CS1 achievement. A possible explanation for the above finding may be due to some mediating variables, such as engagement and self-efficacies. Results from this study suggests students found the constructionist *Scratch* programming to be engaging. Students who are so engaged, are likely to improve in their self-efficacies and increase in self-efficacy is likely to enhance their achievements, regardless of differences in their background variables. Level of students' engagement in first year has been found to have significant effect on their achievement, even for those disadvantaged by their prior educational performance or parental status (Kuh et al., 2008). Corroborating this assertion, Ribeiro (2019) conducted a study with Portuguese first-year students, to investigate the mediatory role of student engagement in the relationship between background variables and their academic achievement. They employed Structural Equation Modelling (SEM) and a model including variables such as engagement, learning approach, background variables and a course grade was tested at the beginning and end of the first semester. They found engagement significantly fitted the model at those two occasions as a mediator between students' background variables (such as language skill, High school GPA, SES, etc) and their performance in a first-year course in their respective departments. They concluded that mediatory role played by engagement, is responsible for studies that found background variables (like SES) not having effect on the student's achievement.

Chapter 5

5 Summary, Conclusion and Recommendations

This chapter presents the summary of the study and its findings. Conclusions were drawn and recommendations on how to address the identified problems were suggested.

5.1 Summary

The study investigated the impact of a constructionist *Scratch* programming pedagogy, on programming achievement of first-year computer science students in programming in some Nigerian Polytechnics. The study adopted a quasi-experimental design (with pre-test post-test non-equivalent experimental and control groups). This is a type of quantitative research design that seeks to establish the impact of *Scratch* on the students, in which the researcher has control over some variables of interest and therefore can manipulate them. The study was an in vivo study, involving intact classes of newly admitted students. We employed between-group design in which these intact classes were randomly assigned to treatment conditions to achieve the purpose of the study. The population of the study comprised all first-year computer science students in polytechnics in North central Nigeria. Four polytechnics namely Federal Polytechnic, Bida, Niger State (FPB), Federal Polytechnic, Nasarawa, Nasarawa State (FPN), Niger State Polytechnic, Zungeru. Niger State (NSPZ), Nasarawa State Polytechnic, Lafia, Nasarawa State (NSPL) were selected using purposive sampling technique. Then selected institutions were randomly assigned to treatment groups. From the data collected in the main study, eighty-four students were selected randomly using *Coarsened Exact Matching* – a form of simple random sampling. This generated two matched samples of which 42 students were in the experimental group and the remaining 42 in the control group. This comprised 60 males and 24 females, with their ages ranging from 16 > = years.

The findings revealed there was a significant main effect of the constructionist *Scratch* intervention, on the achievements of first year computer science students, in programming in the Nigerian polytechnics studied. The result also indicates students' achievements was not moderated by their gender, age, academic background, prior programming writing and prior visual art. The outcome also reveals that there was no significant interaction effect of treatment and gender, age, academic background, prior programming writing, prior visual arts on achievement in programming of first year computer science students in Nigeria polytechnics. The study recommends that, to enhance the programming skill of the students with no prior programming, programming instructors can employ *Scratch* in a collaborative atmosphere – the hallmark of a constructionist pedagogy.

5.2 Conclusion

In this section we make conclusion on each hypothesis and proceed to give an overall conclusion on the outcome of the study.

H_o1: There is no significant difference between the mean scores of the pre- and post- Introductory Programming Achievement Test (IPAT) of first-year polytechnic CS students, after a six-week *Scratch* programming instruction.

This study revealed that there was a significant difference between the pre-test and post-test scores of the students in the *Scratch* class. This outcome provides evidence that, notwithstanding the constructionist nature of the pedagogy employed in this class, compared to their counterparts who were specifically given programming lectures, the students indeed learnt some programming. This indicates the value of a constructionist *Scratch* programming in higher ed CS1 class, in view of a prior study by Kalelioğlu & Gülbahar (2014) that found no effect of learning programming after five weeks of exposing primary school students to *Scratch*. Using an inquiry-based constructionist pedagogy in a novice *Scratch* programming class can lead to meaningful programming knowledge for students so engaged.

H_o2: There is no significant difference in the mean scores of the post Introductory Programming Achievement Test (IPAT) of first-year polytechnic CS students in a constructionist *Scratch* class (experimental group) and those in the conventional class (control group), while controlling for their pre-test scores.

The main finding of this study showed that the constructionist *Scratch* intervention, led to a significant students' programming learning, compared to the conventional instruction. Though results from the pilot and main studies consistently reveal moderate effect of treatment on first year CS students, this finding adds to the growing body of evidence that block-based programming environments, may be more engaging, leading to students with no prior programming knowledge learning more compared to those in the text-based programming language classes (Weintrop & Wilensky, 2017; Xu et al., 2019). Evidence in literature suggests that "little head start" may make the difference between novice higher education CS classrooms with more engaged or more disengaged students, between leaving many students behind or levelling the playing fields for many, between retaining more in CS or raising CS1 attrition, between speeding up more diverse professionals or strengthening CS stereotypes in computing industries.

H_o3: Gender has no effect on the mean scores of the post Introductory Programming Achievement Test (IPAT) of first-year polytechnic CS students in a constructionist *Scratch* class (experimental group) and those in the conventional class (control group), while controlling for their pre-test scores.

This study indicates that gender has no significant effect on the programming achievements of the participants. While there may be gender differences in attitudes and self-efficacies of novice CS students, an engaging programming pedagogy can mediate between gender and meaningful learning of programming.

H_o4: Age has no effect on the mean score of the post Introductory Programming Achievement Test (IPAT) of first-year polytechnic CS students in a constructionist *Scratch* class (experimental group) and those in the conventional class (control group), while controlling for their pre-test scores.

This study shows that age has no significant main effect on novice students' achievement in programming. That implies, notwithstanding the diverse age of participants, their performances in programming were not significantly different. This shows that age is no barrier to engaging and developing programming abilities in a constructionist *Scratch* class, being that *Scratch* was originally developed for the younger age 8-16 (i.e., students in K-12 education).

H_o5: Academic background has no effect on the mean score of the post Introductory Programming Achievement Test (IPAT) of first-year polytechnic CS students in a constructionist *Scratch* class (experimental group) and those in the conventional class (control group), while controlling for their pre-test scores.

The study shows that academic backgrounds of CS1 students, had no significant effect on their achievements in programming. The implication of this finding is that an engaging *Scratch* programming instruction, can level the playing fields for novice CS students with varying academic achievement levels. Such programming instruction - while doing no harm to students with higher achievement as this study and literature show they learn more - motivates and engages those with lower achievements to do catch-up in their achievements in programming.

H_o6: Prior programming experience has no effect on the mean score of the post Introductory Programming Achievement Test (IPAT) of first-year polytechnic CS students in a constructionist *Scratch* class (experimental group) and those in the conventional class (control group), while controlling for their pre-test scores.

This study reveals prior program writing experiences had no significant effects on students' achievements in CS1. While the grossly unbalanced sizes within each group, between those with or without programming experiences raises question on the strength of this outcome, yet the outcome shows that *Scratch* enhanced the performance of students without prior programming, compared to the conventional instruction. Employing an engaging *Scratch* programming with students without prior programming writing is likely to promote students' achievements in CS1.

H_o7: Prior visual art experience has no effect on the mean score of the post Introductory Programming Achievement Test (IPAT) of first-year polytechnic CS students in a constructionist *Scratch* class (experimental group) and those in the conventional class (control group), while controlling for their pre-test scores.

This study indicates that prior visual artistic experiences of CS1 students have no significant effect on their achievements in programming. Yet, this study reveals that those with such experiences performed better than those without it, regardless of whatever form of instructions. Comparing those with such backgrounds in the two treatment groups, those exposed to *Scratch* performed better. This implies *Scratch* amplifies the effect of prior visual art experience or mediates between prior visual art and achievement in programming.

H_o8: Treatment, Gender, academic background, prior programming experience and prior visual art have no interaction on the mean score of the post Introductory Programming Achievement Test (IPAT) of first-year polytechnic CS students in a constructionist *Scratch* class (experimental group) and those in the conventional class (control group), while controlling for their pre-test scores. In view of past research, it appears surprising that in this study treatment, gender, age, academic background, prior programming experience, and prior visual artistic ability had no significant interaction effect on the students' achievement in programming. This suggests with an engaging novice programming class, usual predictors of programming performance may not make significant difference in students' outcomes. Such an engaging programming instruction mediates between students' baseline differences and their achievements – levelling the playfield.

Overall Conclusion

The overriding research goal in this study was to search for empirical evidence of the effect of a constructionist *Scratch* intervention in a higher education, taking CS students in some Nigerian polytechnics as case studies. The results show that CS students exposed to this inquiry-based pedagogy were motivated, engaged and learnt programming during the six weeks, going by their mean IPAT scores. Even though their counterparts in the lecture class equally learnt programming, however the learning gains in the *Scratch* was significantly higher. The result of this study reveals that a constructionist *Scratch* programming in higher education can be engaging for students, especially those without programming experience, regardless of some other background differences like gender, academic background, prior visual art.

Based on the findings of this study, I conclude that the *Scratch* programming intervention had significant main effect on the achievement of first year computer science students in introductory programming in Nigeria polytechnics. This outcome suggests that, the treatment improved the programming ability as the students without prior programming experience were able to develop concrete and abstract knowledge of programming.

5.3 Recommendations

The following recommendations were made based on the finding of the study:

- Programming instructors should introduce programming writing and/or language for students from simple to complex. Therefore, introducing them to programming using *Scratch* - a low ceiling and high floor visual programming environment – works better than other textual programming languages, especially for those with no prior experience in programming in their

K-12 education. Furthermore, it makes the programming class more interesting to students and aids quick retention.

- Institution administrators should introduce Scratch programming as a general course for all the students as this can motivate and get them hooked to programming. Thus, students can become vast in programming and be prepared for the competitive labour market. This can in turn, give them an upper hand in the ever-changing job market as they would have the necessary skills to handle computing or IT related works.
- To enhance the programming skill of the students, programming instructors should ensure that there is a collaborative atmosphere like the one employed in this study while the programming class is ongoing. This would increase their self-efficacies, motivate them, and reduce the time instructor would spend training the students on programming.
- Policy makers and educational stakeholders should ensure that *Scratch* programming as a subject is taught from primary to secondary school level. This will help the pupils and students to be familiar with some concepts. Hence, when programming is introduced to them in higher institution it would not be strange. This can reduce the stress on lecturers or instructors in introducing programming to the students, using probably the more difficult textual programming environment.
- Lecturers in computer science should ensure that while introducing programming to the students, there should be opportunities for experimentation, tinkering and bricolage as this tends to raise motivation, engagement, and retention on the part of the students. The value of such environment in a CS1 class is emphasised by Simonton(2018) who remarked "After all, creative solutions can emerge through either internal thinking or external tinkering"(p. 82)

5.4 Implications of the study

The findings of the present study, have several implications for computer science educators and policymakers. Computer science educators could use findings from the present study on achievement in programming to develop new or improve on the programming courses for students, which will focus on developing their programming abilities.

Policy makers can also play a major role in the development, implementation, and evaluation of information and communication technology (ICT) aimed at improving the programming skills for novice programmers. Policy makers could add *Scratch* programming to the curricular being used in higher institutions training preservice education students. By so doing, teachers in primary and secondary schools would have developed some knowledge and skills needed to teach their learners programming. In addition, computer scientist could develop empowerment training workshops, using *Scratch* for first year computer science students to empower and help them learn strategies and skills for programming. Based upon the findings of the present study, the conventional mode for introducing novice CS students to programming appears not to work for many students. Effective interventions for improving the achievement in programming of first year computer science students, in Nigerian polytechnics, like the one employed in this study, are urgently needed.

5.5 Contributions to Knowledge

The findings of this study have contributed to knowledge in the following ways:

The findings from this study, provides valuable insight into the programming achievement in the context of computer science students, in their first year in Nigerian polytechnics. Also, the study has added to the existing literature on the effectiveness of *Scratch* visual programming, in the achievement of first year computer science students studying programming in higher education. Since research reports affirm that achievement in programming, is one of the co-morbidities that are often overlooked in the effective development of programming skills of students, with no idea about programming, there is dire need to further explore the programming achievement amongst year one computer science students in Nigerian polytechnics with an aim to providing a needed yet overlooked area of need. The literature reviewed in this study as well as the training sessions used in executing this study, has given a better understanding and knowledge of *Scratch* visual programming environment. The study has further proven that *Scratch* visual programming was effective in building the programming skills of students with no idea of programming. In general, the study has filled a research gap, researchers or CS educators seeking to adopt *Scratch* visual programming to improve the programming achievement and skills among first year computer science students in programming, having discovered *Scratch* could improve their programming skills.

5.6 Limitations of the study

A number of obvious limitations existed in the study, however, the notable one is the large number of personal variables not included in the study. Given the demographic differences (e.g., ethnicity, socio-economic status (SES), motivation, geographical location, self-esteem), future research should include the above variables. These would further strengthen the profile base of the participants. Thus, these self-selection biases, could limit the generalizability of the present findings, to those who do not have the idea of programming, as well as other participants with prior programming background. As such, the present study should be replicated with a larger and randomly selected sample size and with a greater representation of students with or without programming background. Also, the lack of data on a comparative normal population constitutes a limitation to this study. It is hoped that future studies in this area will address these challenges.

Another limitation identified in this study is that, it may be limited in depicting the programming skills of the participants. The list of potential demographic information of programming achievement among the participants, that were explored in this study is by no means exhaustive. Hence, further research in this regard would be needful. In the study, it is estimated that it took participants approximately 45 minutes or less to complete all the instruments in sections A and B, based upon feedback from volunteers who assisted on the field. However, the number of instruments and the length of completion time, may have facilitated a fatigue factor among participants, given the fact that they were undergoing some distress which learning programming may bring.

In addition, it was not possible due to financial restrictions in the present study's budget, to translate the questionnaires and the treatment packages into other languages other than English. The option of not having other versions of the survey instruments may have attracted a less culturally diverse sample of research participants.

Future studies would also benefit from gathering data from multiple sources (for example, prior knowledge of computer) and employing qualitative research methods to condition mono-method bias.

Furthermore, while carrying out the research work, there were certain constraints experienced by the researcher. Some of these limitations included poor conducive environment, insufficient time frame and cost in extending the scope of the research. Another constraint was lack of adequate support from the management of the institution under study, because, the management was not in support of their students being used for research. This research was limited to only polytechnic institutions in the North Central of Nigeria. Moreover, the issue of internet access is still a resource challenge in Nigeria and this threatens the research outcome. However, students were introduced to the online version so that they can interact with online community of *Scratch*ers to be able to upload, download and remix *Scratch* projects. This is a crucial aspect of their programming learning in *Scratch*. Despite all the challenges in this study, the researcher was able to scale through.

5.7 Suggestions for further research

The study on the effectiveness of *Scratch* visual programming, on the programming achievement of first year computer science students in Nigerian polytechnics, should also be carried out in other geo-political zones such as South-West, South-East etc. Also, in terms of future research, it would be beneficial for researchers to obtain a larger sample size and to include not only participants who are in their first year, but also those in their second and third year. Other psychological and environmental variables other than gender, age, prior programming writing and prior visual arts, can be examined as moderating variables to identify other variables that could possibly influence the effectiveness of *Scratch*, on the programming achievement of first year computer science students. *Scratch* programming can also be introduced to undergraduates in various universities across the six geo-political zones, in Nigeria to explore its effect in such contexts. This study revealed that a constructionist *Scratch* programming pedagogy promotes novice students' learning of programming, better than the conventional instruction. However, a question can be raised: What led to better learning, *Scratch* or the constructionist approach? A way to explore this is to design an experiment that involves two equivalent sample of novice CS students, both taught *Scratch* - one the constructionist way and the other, another approach.

As of date (November 2021), students can program in *Scratch* in more than 70 world languages other than English. While there are some African languages in the list, such as Afrikaans, Kiswahili, isiZulu to mention few, no major languages in Nigeria (like Hausa, Igbo and Yoruba) are yet to make the list. Hausa which is a language of communication among other African nations does not make the list. Incorporating such local languages into *Scratch* environment online and offline can encourage participation and learning of programming by more kids. A study on the impact of such contextualisation of the programming language in *Scratch* on the attitude and achievements of K-12 students would make an interesting investigation.

References

Aberson, C. L. (2019). Applied power analysis for the behavioral sciences. In *Applied Power Analysis for the Behavioral Sciences* (2nd ed.). https://doi.org/10.4324/9780203860274

Adleberg, B. M. (2013). Scratch programming and remix culture: Gender differences in interaction and motivation for pre-adolescents. *ProQuest Dissertations and Theses*, 71. https://search.proquest.com/docview/1355174689?accountid=15928%0Ahttp://onesearch.unifi.it/openurl/39UFI/39UFI_Services?tabs=viewOnlineTab&?url_ver=Z39.88-2004&rft_val_fmt=info:ofi/fmt:kev:mtx:dissertation&genre=dissertations+%26+theses&sid=ProQ:ProQuest+

Agapito, J. L., & Rodrigo, M. M. T. (2018). *Investigating the Impact of a Meaningful Gamification-Based Intervention on Novice Programmers' Achievement* (pp. 3–16). https://doi.org/10.1007/978-3-319-93843-1_1

Aho, A. V. (2012). Computation and computational thinking. *Computer Journal*, *55*(7), 833–835. https://doi.org/10.1093/COMJNL/BXS074

Aivaloglou, E., & Hermans, F. (2019). Early programming education and career orientation: The effects of gender, self-efficacy, motivation and stereotypes. *SIGCSE 2019 - Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, 679–685. https://doi.org/10.1145/3287324.3287358

Al-Sheeb, B. A., Hamouda, A. M., & Abdella, G. M. (2019). Modeling of student academic achievement in engineering education using cognitive and non-cognitive factors. *Journal of Applied Research in Higher Education*, *11*(2), 178–198. https://doi.org/10.1108/JARHE-10-2017-0120

Alvarado, C., Dodds, Z., & Libeskind-Hadas, R. (2012). Increasing women's participation in computing at Harvey Mudd College. *Acm Inroads*, *3*(4), 55–64.

Alvarado, C., Umbelino, G., & Minnes, M. (2018). The Persistent Effect of Pre-College Computing Experience on College CS Course Grades. *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, 876–881. https://doi.org/10.1145/3159450.3159508

Anyanwu, J. A. (1978). Computer science education. *ACM SIGCSE Bulletin*, *10*(1), 37–40. https://doi.org/10.1145/990654.990573

Armoni, M., Meerbaum-Salant, O., & Ben-Ari, M. (2015). From Scratch to "Real" Programming. *ACM Transactions on Computing Education*, *14*(4), 1–15. https://doi.org/10.1145/2677087

Arpaci, I., Durdu, P. O., & Mutlu, A. (2019). The Role of Self-Efficacy and Perceived Enjoyment in Predicting Computer Engineering Students' Continuous Use Intention of Scratch. *International Journal of E-Adoption*, *11*(2), 1–12. https://doi.org/10.4018/IJEA.2019070101

Ary, D., Jacobs, L. C., Sorensen, C. K., & Walker, D. A. (2014). *Introduction to Research in Education* (9th ed.). Wadsworth, Cengage Learning. https://books.google.com/books?id=FqF7n0zGJm0C&pgis=1

Asarta, C. J., & Schmidt, J. R. (2017). Comparing student performance in blended and traditional courses: Does prior academic achievement matter? *The Internet and Higher Education*, *32*, 29–38. https://doi.org/10.1016/j.iheduc.2016.08.002

Atmatzidou, S., & Demetriadis, S. (2016). Advancing students' computational thinking skills through educational robotics: A study on age and gender relevant differences. *Robotics and Autonomous Systems*, *75*, 661–670. https://doi.org/10.1016/j.robot.2015.10.008

Ayalew, Y., Tshukudu, E., & Lefoane, M. (2018). Factors Affecting Programming Performance of First Year Students at a University in Botswana. *African Journal of Research in Mathematics, Science and Technology Education*, *22*(3), 363–373. https://doi.org/10.1080/18117295.2018.1540169

Becker, B. A. (2019). A Survey of Introductory Programming Courses in Ireland. *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, 58–64. https://doi.org/10.1145/3304221.3319752

Beecher, K. (2017). *Computational Thinking: A Beginner's Guide to Problem-Solving and Programming*. BCS Learning & Development Limited.

Bell, J. (2014). *Doing Your Research Project: A Guide For First-Time Researchers* (6th ed.). Open International Publishing Limited.

Ben-Ari, M. (1998). Constructivism in computer science education. *Proceedings of the Twenty-Ninth SIGCSE Technical Symposium on Computer Science Education - SIGCSE '98*, 257–261. https://doi.org/10.1145/273133.274308

Bennedsen, J., & Caspersen, M. E. (2019). Failure rates in introductory programming. *ACM Inroads*, *10*(2), 30–35. https://doi.org/10.1145/3324888

Beyer, S. (2014). Why are women underrepresented in Computer Science? Gender differences in stereotypes, self-efficacy, values, and interests and predictors of future CS course-taking and grades. *Computer Science Education*, *24*(2–3), 153–192. https://doi.org/10.1080/08993408.2014.963363

Bird, A. (2018). Thomas Kuhn. In E. N. Zalta (Ed.), *The {Stanford} Encyclopedia of Philosophy* ({W}inter 2). Metaphysics Research Lab, Stanford University.

Blaxter, L., Hughes, C., & Tight, M. (2006). *How to Research* (3rd ed.). Open University Press.

Blikstein, P., & Moghadam, S. H. (2019). Computing EducationLiterature Review and Voices from the Field. In S. A. Fincher & A. V. E. Robins (Eds.), *The Cambridge Handbook of Computing Education Research* (pp. 56–78). Cambridge University Press. https://doi.org/10.1017/9781108654555.004

Boljat, I., Mladenović, M., & Mustapić Jogun, N. (2019). Students' Attitudes Towards Programming After the First Year of Implementing a New Informatics Curriculum in the Elementary Schools. *ICERI2019 Proceedings*, *1*, 9486–9495. https://doi.org/10.21125/iceri.2019.2303

Bond, M., Buntins, K., Bedenlier, S., Zawacki-Richter, O., & Kerres, M. (2020). Mapping research in student engagement and educational technology in higher education: a systematic evidence map. *International Journal of Educational Technology in Higher Education*, *17*(1), 2. https://doi.org/10.1186/s41239-019-0176-8

Brennan, K. (2015). Beyond technocentrism: Supporting constructionism in the classroom. *Constructivist Foundations*, *10*(3), 289–296.

Britannica, T. E. of E. (2020). Art. In *Britannica*. Encyclopedia Britannica. https://www.britannica.com/art/visual-arts

Bruce, K. B. (2018). Five big open questions in computing education. *ACM Inroads*, *9*(4), 77–80. https://doi.org/10.1145/3230697

Bughin, J., Hazan, E., Lund, S., & Dahlstrom, P. (2018). Skill Shift: Automation and the Future of the Workforce. *McKinsey & Company*, *May*, 3–84. https://www.mckinsey.com/featured-insights/future-of-work/skill-shift-automation-and-the-future-of-the-workforce

Butler, M., Sinclair, J., Morgan, M., & Kalvala, S. (2016). Comparing international indicators of student engagement for computer science. *Proceedings of the Australasian Computer Science Week Multiconference*, 1–10. https://doi.org/10.1145/2843043.2843065

Campbell, O. O., & Atagana, H. I. (2022). Impact of a Scratch programming intervention on student engagement in a Nigerian polytechnic first-year class: verdict from the observers. *Heliyon*, *8*(3), e09191. https://doi.org/10.1016/j.heliyon.2022.e09191

Cárdenas-Cobo, J., Puris, A., Novoa-Hernández, P., Parra-Jiménez, Á., Moreno-León, J., & Benavides, D. (2021). Using scratch to improve learning programming in college students: A positive experience from a non-weird country. *Electronics (Switzerland)*, *10*(10), 1180. https://doi.org/10.3390/electronics10101180

Carlos Begosso, L., Ricardo Begosso, L., & Aragao Christ, N. (2020). An analysis of block-based programming environments for CS1. *2020 IEEE Frontiers in Education Conference (FIE)*, 1–5. https://doi.org/10.1109/FIE44824.2020.9273982

CC2020 Task Force. (2020). *Computing Curricula 2020*. ACM. https://doi.org/10.1145/3467967

Ceruzzi, P. E. (1991). When computers were human. *Annals of the History of Computing*, *13*(3), 237–244.

Cetin, I. (2016). Preservice Teachers' Introduction to Computing. *Journal of Educational Computing Research*, *54*(7), 997–1021. https://doi.org/10.1177/0735633116642774

Chen, C., Haduong, P., Brennan, K., Sonnert, G., & Sadler, P. (2019). The effects of first programming language on college students' computing attitude and achievement: a comparison of graphical and textual languages. *Computer Science Education*, *29*(1), 23–48. https://doi.org/10.1080/08993408.2018.1547564

Chen, C., Kang, J. M., Sonnert, G., & Sadler, P. M. (2021). High School Calculus and Computer Science Course Taking as Predictors of Success in Introductory College Computer Science. *ACM Transactions on Computing Education*, *21*(1), 1–21. https://doi.org/10.1145/3433169

Chetty, J., & Barlow-Jones, G. (2014). Novice students and computer programming: Toward constructivist pedagogy. *Mediterranean Journal of Social Sciences*, *5*(14), 240–251. https://doi.org/10.5901/MJSS.2014.V5N14P240

Copeland, B. J. (2017). The Modern History of Computing. In E. N. Zalta (Ed.), *The {Stanford} Encyclopedia of Philosophy* ({W}inter 2). Metaphysics Research Lab, Stanford University.

Creswell, J. W. (2014). *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches* (4th ed.). Sage Publications, Inc.

Creswell, J. W., & Crewell, J. D. (2018). *Research design: quantitative, qualitative and mixed methods approaches* (5th ed.). SAGE Publications, Inc.

Curzon, P., Bell, T., Waite, J., Dorling, M., Denning, P. J., & Tedre, M. (2019). Computational Thinking. In Sally Fincher & A. Robins (Eds.), *The Cambridge handbook of computing education research* (pp. 513–546). Cambridge University Press.

Denning, P. J. (2017). Remaining trouble spots with computational thinking. *Communications of the ACM*, *60*(6), 33–39.

Denning, P. J., & Martell, C. H. (2015). *Great principles of computing*. MIT Press.

Denning, P. J., & Tedre, M. (2019). *Computational thinking*. Mit Press.

DeVito, M. (2016). *Factors influencing student engagement* [Sacred Heart University, Fairfield]. http://digitalcommons.sacredheart.edu/cgi/viewcontent.cgi?article=1010&context=edl

Eccles, J., & Wang, M.-T. (2012). Part I Commentary: So What Is Student Engagement Anyway? In S. Christenson, A. Reschly, & C. Wylie (Eds.), *Handbook of Research on Student Engagement* (pp. 133–145). Springer US. https://doi.org/10.1007/978-1-4614-2018-7_6

Ellison, N. (2021). Seymour Papert. In *Encyclopedia Britannica*. https://www.britannica.com/biography/Seymour-Papert

Erol, O., & Kurt, A. A. (2017). The effects of teaching programming with scratch on pre-service information technology teachers' motivation and achievement. *Computers in Human Behavior*, *77*, 11–18. https://doi.org/10.1016/j.chb.2017.08.017

Erwig, M. (2017). *Once upon an algorithm: how stories explain computing*. MIT Press.

Esaak, S. (2019). *What are the visual arts?* https://www.thoughtco.com/what-are-the-visual-arts-182706

Falkner, K., & Sheard, J. (2019). Pedagogic Approaches. In S. A. Fincher & A. V. Robins (Eds.), *The Cambridge Handbook of Computing Education Research* (pp. 445–480). Cambridge University Press. https://doi.org/10.1017/9781108654555.016

Fee, S. B., Holland-Minkley, A. M., & Lombardi, T. E. (2017). New directions for computing education: Embedding computing across disciplines. In *New Directions for Computing Education: Embedding Computing Across Disciplines*. Springer Publishing. https://doi.org/10.1007/978-3-319-54226-3

Ferrer, J., Ringer, A., Saville, K., A Parris, M., & Kashi, K. (2020). Students' motivation and engagement in higher education: the importance of attitude to online learning. *Higher Education*. https://doi.org/10.1007/s10734-020-00657-5

Field, A. (2018). *Discovering Statistics Using IBM SPSS Statistics*. SAGE Publications. https://books.google.com.ng/books?id=JIrutAEACAAJ

Fields, D. A., Kafai, Y. B., & Giang, M. T. (2017). Youth computational participation in the wild: Understanding experience and equity in participating and programming in the online Scratch community. *ACM Transactions on Computing Education*, *17*(3), 1–22. https://doi.org/10.1145/3123815

Fincher, Sally, & Petre, M. (Eds.). (2005). *Computer Science Education Education*. Taylor and Francis.

Fincher, Tenenberg, J., Dorn, B., Hundhausen, C., McCartney, R., & Murphy, L. (2019). Computing Education Research Today. In S. Fincher & A. Robins (Eds.), *The Cambridge Handbook of Computing Education Research* (pp. 40–55). Cambridge University Press. https://doi.org/10.1017/9781108654555.003

Finn, J. D., & Zimmer, K. S. (2012). Student engagement: What is it? Why does it matter? In S. Christenson, A. Reschly, & C. Wylie (Eds.), *Handbook of research on student engagement* (pp. 97–131). Springer.

Fraenkel, J. R., Hyun, H. H., & Wallen, N. E. (2018). *How to Design and Evaluate Research in Education*. McGraw-Hill Education. https://books.google.com.ng/books?id=lKjluAEACAAJ

Fredricks, J. A., Blumenfeld, P. C., & Paris, A. H. (2004). School engagement: Potential of the concept, state of the evidence. *Review of Educational Research*, *74*(1), 59–109.

Frey, B. B. (2018). *The SAGE encyclopedia of educational research, measurement, and evaluation*. Sage Publications.

Geschwind, L., Broström, A., & Larsen, K. (Eds.). (2020). *Technical Universities* (Vol. 56). Springer International Publishing. https://doi.org/10.1007/978-3-030-50555-4

Gestwicki, P., & Ahmad, K. (2010). A pilot study on the impact of creative achievement on academic achievement in media-oriented CS1. *The Journal of Computing Sciences in Colleges*, 85.

Gjelsten, B. K., Bergersen, G. R., Sjøberg, D. I. K., & Cutts, Q. (2021). No Gender Difference in CS1 Grade for Students with Programming from High School: An Exploratory Study. *21st Koli Calling International Conference on Computing Education Research*, 1–5. https://doi.org/10.1145/3488042.3488071

Glasersfeld, E. von. (1996). *Radical constructivism: a way of knowing and learning*. 210. http://books.google.com/books?id=XgXgRaG50xoC&pgis=1

Glasersfeld, E. von. (2007). Key Works in Radical Constructivism. In M. Larochelle (Ed.), *Key Works in Radical Constructivism*. Sense Publishers. https://doi.org/10.1163/9789087903480

Goldie, T. (2014). The Man Who Invented Gender: Engaging the Ideas of John Money by Terry Goldie. In *University of Toronto Quarterly* (Vol. 85, Issue 3). University of British Columbia Press.

Gunbata, S. M., & Karalar, H. (2018). Gender Differences in Middle School Students' Attitudes and Self-Efficacy Perceptions towards mBlock Programming. *European Journal of Educational Research*, *7*(4), 925–933. https://doi.org/10.12973/eu-jer.7.4.925

Guo, A. (2020). *Analysis of Factors and Interventions Relating to Student Performance in CS1 and CS2*. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2020/EECS-2020-22.html

Guzdial, M. (2015). Learner-Centered Design of Computing Education: Research on Computing for Everyone. *Synthesis Lectures on Human-Centered Informatics*, *8*(6), 1–165. https://doi.org/10.2200/s00684ed1v01y201511hci033

Guzdial, M. (2013). Exploring hypotheses about media comptation. *ICER 2013 - Proceedings of the 2013 ACM Conference on International Computing Education Research*, 19–26. https://doi.org/10.1145/2493394.2493397

Guzdial, M., & du Boulay, B. (2019). The History of Computing Education Research. In *The Cambridge handbook of computing education research* (pp. 11–39). Cambridge University Press.

Guzdial, M., & Soloway, E. (2002). Teaching the Nintendo Generation to Program. *Communications of the ACM*, *45*(4), 17–21. https://doi.org/10.1145/505248.505261

Hagan, D., & Markham, S. (2000). Does it help to have some programming experience before beginning a computing degree program? *Proceedings of the 5th Annual SIGCSE/SIGCUE ITiCSEconference on Innovation and Technology in Computer Science Education - ITiCSE '00*, 25–28. https://doi.org/10.1145/343048.343063

Hanks, B., Fitzgerald, S., McCauley, R., Murphy, L., & Zander, C. (2011). Pair programming in education: A literature review. *Computer Science Education*, *21*(2), 135–173. https://doi.org/10.1080/08993408.2011.579808

Haseski, H. I., Ilic, U., & Tugtekin, U. (2018). Defining a New 21st Century Skill-Computational Thinking: Concepts and Trends. *International Education Studies*, *11*(4), 29–42.

Hemmendinger, D. (2010). A plea for modesty. *ACM Inroads*, *1*(2), 4–7.

Hermans, F., & Aivaloglou, E. (2017). Teaching Software Engineering Principles to K-12 Students: A MOOC on Scratch. *2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering Education and Training Track (ICSE-SEET)*, 13–22. https://doi.org/10.1109/ICSE-SEET.2017.13

Hijón-Neira, R., Connolly, C., Palacios-Alonso, D., & Borrás-Gené, O. (2021). A Guided Scratch Visual Execution Environment to Introduce Programming Concepts to CS1 Students. *Information*, *12*(9), 378. https://doi.org/10.3390/info12090378

Holden, E., & Weeden, E. (2004). The experience factor in early programming education. *SIGITE 2004 Conference*, 211–218. https://doi.org/10.1145/1029533.1029585

Hsu, H. J. (2014). *Gender Differences in Scratch Game Design* □. *Icibet*, 100–103.

Hu, Y., Chen, C.-H., & Su, C.-Y. (2021). Exploring the Effectiveness and Moderators of Block-Based Visual Programming on Student Learning: A Meta-Analysis. *Journal of Educational Computing Research*, *58*(8), 1467–1493. https://doi.org/10.1177/0735633120945935

Iacus, S. M., King, G., & Porro, G. (2009). cem : Software for Coarsened Exact Matching . *Journal of Statistical Software*, *30*(9). https://doi.org/10.18637/jss.v030.i09

Iacus, S. M., King, G., & Porro, G. (2012). Causal Inference without Balance Checking: Coarsened Exact Matching. *Political Analysis*, *20*(1), 1–24. https://doi.org/10.1093/pan/mpr013

International Labour Office, & International Labour Organisation. (2017). World Employment and Social Outlook: Trends 2017. In *International Labour Organization*. https://www.ilo.org/wcmsp5/groups/public/---dgreports/---dcomm/---publ/documents/publication/wcms_734455.pdf

Jazwinski, S. M., & Kim, S. (2019). Examination of the Dimensions of Biological Age. *Frontiers in Genetics*, *10*. https://doi.org/10.3389/fgene.2019.00263

Kafai, Y. B., & Fields, D. A. (2018). *Some Reflections on Designing Constructionist Activities for Classrooms*.

Kafai, Y. B., Fields, D. A., Lui, D. A., Walker, J. T., Shaw, M. S., Jayathirtha, G., Nakajima, T. M., Goode, J., & Giang, M. T. (2019). Stitching the Loop with Electronic Textiles. *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, 1176–1182. https://doi.org/10.1145/3287324.3287426

Kalelioğlu, F., & Gülbahar, Y. (2014). The effects of teaching programming via Scratch on problem solving skills: A discussion from learners' perspective. *Informatics in Education*, *13*(1), 33–50.

Kim, H. (2013). Statistical notes for clinical researchers: assessing normal distribution (2) using skewness and kurtosis. *Restorative Dentistry & Endodontics*, *38*(1), 52. https://doi.org/10.5395/rde.2013.38.1.52

Kim, Hyeonjin, Choi, H., Han, J., & So, H. J. (2012). Enhancing teachers' ICT capacity for the 21st century learning environment: Three cases of teacher education in korea. *Australasian Journal of Educational Technology*, *28*(6), 965–982. https://doi.org/10.14742/ajet.805

Kirschner, P. A., Sweller, J., & Clark, R. E. (2006). Why minimal guidance during instruction does not work: An analysis of the failure of constructivist, discovery, problem-based, experiential, and inquiry-based teaching. *Educational Psychologist*, *41*(2), 75–86. https://doi.org/10.1207/s15326985ep4102_1

Klem, A. M., & Connell, J. P. (2004). Relationships matter: Linking teacher support to student engagement and achievement. *Journal of School Health*, *74*(7), 262–273.

Knuth, D. E. (2007). Computer Programming as an Art. In *ACM Turing Award Lectures* (p. 1974). Association for Computing Machinery. https://doi.org/10.1145/1283920.1283929

Kock, N., Moqbel, M., Jung, Y., & Syn, T. (2018). Do older programmers perform as well as young ones? Exploring the intermediate effects of stress and programming experience. *Cognition, Technology & Work*, *20*(3), 489–504. https://doi.org/10.1007/s10111-018-0479-x

Kong, S.-C., Chiu, M. M., & Lai, M. (2018). A study of primary school students' interest, collaboration attitude, and programming empowerment in computational thinking education. *Computers & Education*, *127*, 178–189. https://doi.org/10.1016/j.compedu.2018.08.026

Kothiyal, A., Majumdar, R., Murthy, S., & Iyer, S. (2013). Effect of think-pair-share in a large CS1 class. *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research*, 137–144. https://doi.org/10.1145/2493394.2493408

Kuh, G. D., Cruce, T. M., Shoup, R., Kinzie, J., & Gonyea, R. M. (2008). Unmasking the Effects of Student Engagement on First-Year College Grades and Persistence. *The Journal of Higher Education*, *79*(5), 540–563. https://doi.org/10.1080/00221546.2008.11772116

Kuhn, T. S. (2012). *The Structure of Scientific Revolutions: 50th Anniversary Edition*. https://books.google.pt/books/about/The_Structure_of_Scientific_Revolutions.html?id=3eP5Y_OOuzwC&source=kp_cover&redir_esc=y%5Cnhttps://books.google.pt/books?id=3eP5Y_OOuzwC&printsec=frontcover&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false

Kulasegaram, K., Axelrod, D., Ringsted, C., & Brydges, R. (2018). Do One Then See One. *Academic Medicine*, *93*(11S), S37–S44. https://doi.org/10.1097/ACM.0000000000002378

Lam, S. F., Wong, B. P. H., Yang, H., & Liu, Y. (2012). Understanding student engagement with a contextual model. In S. Christenson, A. Reschly, & C. Wylie (Eds.), *Handbook of Research on Student*

Engagement (pp. 403–419). Springer US. https://doi.org/10.1007/978-1-4614-2018-7_19

Lambić, D., Đorić, B., & Ivakić, S. (2020). Investigating the effect of the use of code.org on younger elementary school students' attitudes towards programming. *Behaviour & Information Technology*, 1–12. https://doi.org/10.1080/0144929X.2020.1781931

Lehman, K. J., Sax, L. J., & Zimmerman, H. B. (2016). Women planning to major in computer science: Who are they and what makes them unique? *Computer Science Education*, *26*(4), 277–298. https://doi.org/10.1080/08993408.2016.1271536

Lei, H., Cui, Y., & Zhou, W. (2018). Relationships between student engagement and academic achievement: A meta-analysis. *Social Behavior and Personality: An International Journal*, *46*(3), 517–528. https://doi.org/10.2224/sbp.7054

Liao, S. N., Shah, K., Griswold, W. G., & Porter, L. (2021). A Quantitative Analysis of Study Habits Among Lower- and Higher-Performing Students in CS1. *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1*, 366–372. https://doi.org/10.1145/3430665.3456350

Liénardy, S., Leduc, L., & Donnet, B. (2021). Promoting Engagement in a CS1 Course with Assessment for Learning. *Student Success*, *12*(1), 102–111. https://doi.org/10.5204/ssj.1668

Lindqvist, A., Sendén, M. G., & Renström, E. A. (2021). What is gender, anyway: a review of the options for operationalising gender. *Psychology & Sexuality*, *12*(4), 332–344. https://doi.org/10.1080/19419899.2020.1729844

Lishinski, A., & Rosenberg, J. (2021). All the Pieces Matter: The Relationship of Momentary Self-efficacy and Affective Experiences with CS1 Achievement and Interest in Computing. *Proceedings of the 17th ACM Conference on International Computing Education Research*, 252–265. https://doi.org/10.1145/3446871.3469740

Lishinski, A., Yadav, A., Good, J., & Enbody, R. (2016). Learning to program: Gender differences and interactive effects of students' motivation, goals, and self-efficacy on performance. *ICER 2016 - Proceedings of the 2016 ACM Conference on International Computing Education Research*, 211–220. https://doi.org/10.1145/2960310.2960329

Llanas, I. S. (2018). Brevísimo análisis doxográfico sobre el Constructivismo: de los presocráticos a la cibernética de segundo orden. *Bajo Palabra*, *2018*, 61–76. https://doi.org/10.15366/bp2018.18.003

Malan, D. J., & Leitner, H. H. (2007). Scratch for budding computer scientists. *SIGCSE 2007: 38th SIGCSE Technical Symposium on Computer Science Education*, 223–227. https://doi.org/10.1145/1227310.1227388

Malik, S. I., & Coldwell-Neilson, J. (2018). Gender differences in an introductory programming course: New teaching approach, students' learning outcomes, and perceptions. *Education and Information*

Technologies, *23*(6), 2453–2475. https://doi.org/10.1007/s10639-018-9725-3

Maloney, J., Peppier, K., Kafai, Y. B., Resnick, M., & Rusk, N. (2008). Programming by choice: Urban youth learning programming with scratch. *SIGCSE'08 - Proceedings of the 39th ACM Technical Symposium on Computer Science Education*, 367–371. https://doi.org/10.1145/1352135.1352260

Maloney, J., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The scratch programming language and environment. *ACM Transactions on Computing Education*, *10*(4), 1–15. https://doi.org/10.1145/1868358.1868363

Marcher, M. H., Christensen, I. M., Grabarczyk, P., Graversen, T., & Brabrand, C. (2021). Computing Educational Activities Involving People Rather Than Things Appeal More to Women (CS1 Appeal Perspective). *Proceedings of the 17th ACM Conference on International Computing Education Research*, 145–156. https://doi.org/10.1145/3446871.3469761

Margulieux, L., Denny, P., Cunningham, K., Deutsch, M., & Shapiro, B. R. (2021). When Wrong is Right: The Instructional Power of Multiple Conceptions. *Proceedings of the 17th ACM Conference on International Computing Education Research*, 184–197. https://doi.org/10.1145/3446871.3469750

Marling, C., & Juedes, D. (2016). CS0 for Computer Science Majors at Ohio University. *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, 138–143. https://doi.org/10.1145/2839509.2844624

Mayers, A. (2013). *Introduction to Statistics and SPSS in Psychology*. Pearson. https://books.google.com.ng/books?id=SDHtMgEACAAJ

McCormick, A. C., & Kinzie, J. (2014). Refocusing the quality discourse: The United States National Survey of Student Engagement. In H. Coates & A. C. McCormick (Eds.), *Engaging University Students: International Insights from System-Wide Studies* (pp. 13–29). Springer Singapore. https://doi.org/10.1007/978-981-4585-63-7_2

McGettrick, A., Boyle, R., Ibbett, R., Lloyd, J., Lovegrove, G., & Mander, K. (2005). Grand challenges in computing: Education—a summary. *The Computer Journal*, *48*(1), 42–48.

McKlin, T., Lee, T., Wanzer, D., Magerko, B., Edwards, D., Grossman, S., Bryans, E., & Freeman, J. (2019). Exploring the Correlation Between Teacher Pedagogical Content Knowledge and Content Knowledge in Computer Science Classrooms. *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, 315–315. https://doi.org/10.1145/3304221.3325556

Medeiros, R. P., Ramalho, G. L., & Falcão, T. P. (2019). A Systematic Literature Review on Teaching and Learning Introductory Programming in Higher Education. *IEEE Transactions on Education*, *62*(2), 77–90. https://doi.org/10.1109/TE.2018.2864133

Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (Moti). (2013). Learning computer science concepts with Scratch. *Computer Science Education*, *23*(3), 239–264. https://doi.org/10.1080/08993408.2013.832022

Mehmet Tekerek, & Tuğba Altan. (2014). The effect of Scratch environment on student's achievement in teaching algorithm. *World Journal on Educational Technology*, *6*(2), 132–138. www.awer-center/wjet

Merino, M. V., Vinju, J., & Brand, M. van den. (2021). DRAFT-What you always wanted to know but could not find about block-based environments. *ArXiv Preprint ArXiv:2110.03073*.

Miller, G. A., & Chapman, J. P. (2001). Misunderstanding analysis of covariance. *Journal of Abnormal Psychology*, *110*(1), 40–48. https://doi.org/10.1037/0021-843X.110.1.40

Mindetbay, Y., Bokhove, C., & Woollard, J. (2019). What is the Relationship between Students' Computational Thinking Performance and School Achievement? *International Journal of Computer Science Education in Schools*, 3–19. https://doi.org/10.21585/ijcses.v0i0.45

Mishra, S., Iyer, S., Balan, S., & Murthy, S. (2014). *Effect of a 2-week Scratch Intervention in CS1 on Learners with Varying Prior Knowledge*. https://doi.org/10.1145/2591708.2591733

Mladenović, M., Rosić, M., & Mladenović, S. (2016). Comparing Elementary Students' Programming Success based on Programming Environment. *International Journal of Modern Education and Computer Science*, *8*(8), 1–10. https://doi.org/10.5815/ijmecs.2016.08.01

Mohamed, A. (2021). Teaching highly mixed-ability CS1 classes: A proposed approach. *Education and Information Technologies*. https://doi.org/10.1007/s10639-021-10546-8

Morgan, M., Sinclair, J. E., Butler, M., Thota, N., Fraser, J., Cross, G. W., & Jacková, J. (2017). Understanding International Benchmarks on Student Engagement: Awareness and Research Alignment from a Computer Science Perspective. In J. Sheard & A. Korhonen (Eds.), *Proceedings of the 2017 ITiCSE Working Group Reports, ITiCSE-WGR 2017, Bologna, Italy, July 3-5, 2017* (pp. 1–24). ACM. https://doi.org/10.1145/3174781.3174782

Morrison, P., & Murphy-Hill, E. (2013). Is programming knowledge related to age?: An exploration of stack overflow. *IEEE International Working Conference on Mining Software Repositories*, 69–72. https://doi.org/10.1109/MSR.2013.6624008

Mouton, J. (2013). *How to Succeed in Your Master's and Doctoral Studies: A South African Guide and Resource Book*. https://books.google.co.za/books/about/How_to_Succeed_in_Your_Master_s_and_Doct.html?id=uX4lAQAAIAAJ&pgis=1

National Academies of Sciences, Engineering, and M. (2018). *Assessing and Responding to the Growth of Computer Science Undergraduate Enrollments*. The National Academies Press. https://doi.org/10.17226/24926

Nwachukwu, M. (1994). Development of information technology in Nigeria. In E. P. Drew & F. G. Foster (Eds.), *Information Technology in Selected Countries: Reports from Ireland, Ethiopia, Nigeria, and Tanzania*. United Nations University Press.

http://archive.unu.edu/unupress/unupbooks/uu19ie/uu19ie00.htm#Contents

OECD Glossary of Statistical Terms. (2008). OECD Glossary of Statistical Terms. https://doi.org/10.1787/9789264055087-en

Orbey, E. (2020, July). How Harvard's Star Computer-Science Professor Built a Distance-Learning Empire. *The New Yorker*. https://www.newyorker.com

Organisation, W. H. (2018). *Gender and health*. Handbook of Health Psychology. https://doi.org/10.4324/9781315167534-22

Özmen, B., & Altun, A. (2014). Undergraduate Students' Experiences in Programming: Difficulties and Obstacles. *Turkish Online Journal of Qualitative Inquiry*, *5*(3). https://doi.org/10.17569/tojqi.20328

Pallant, J. (2016). *SPSS Survival Manual: A Step By Step Guide to Data Analysis Using SPSS Program* (6th ed.). McGraw-Hill Education.

Papadakis, S., & Kalogiannakis, M. (2019). Evaluating a course for teaching introductory programming with Scratch to pre-service kindergarten teachers. *International Journal of Technology Enhanced Learning*, *11*(3), 231–246. https://doi.org/10.1504/IJTEL.2019.100478

Pappas, I. O., Aalberg, T., Giannakos, M. N., Jaccheri, L., Mikalef, P., & Sindre, G. (2016). Gender Differences in Computer Science Education: Lessons Learnt from an Empirical Study at NTNU. *Nik*.

Parham-Mocello, J., & Erwig, M. (2020). Does story programming prepare for coding? *Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE*, 100–106. https://doi.org/10.1145/3328778.3366861

Parker, M. C., Solomon, A., Pritchett, B., Illingworth, D. A., Marguilieux, L. E., & Guzdial, M. (2018). Socioeconomic Status and Computer Science Achievement. *Proceedings of the 2018 ACM Conference on International Computing Education Research*, 97–105. https://doi.org/10.1145/3230977.3230987

Parsons, J., & Taylor, L. (2011). Improving student engagement. *Current Issues in Education*, *14*(1).

Pérez-Marín, D., Hijón-Neira, R., Bacelo, A., & Pizarro, C. (2020). Can computational thinking be improved by using a methodology based on metaphors and scratch to teach computer programming to children? *Computers in Human Behavior*, *105*, 105849. https://doi.org/10.1016/j.chb.2018.12.027

Phuntsho, U., & Dendup, R. (2021). The relationship between school climate, student engagement and academic achievement in higher secondary school. *Bhutan Journal of Research and Development*, *9*(2). https://bjrd.rub.edu.bt/index.php/bjrd/article/view/81

Piaget, J. (2008). Intellectual Evolution from Adolescence to Adulthood. *Human Development*, *51*(1), 40–47. https://doi.org/10.1159/000112531

Piggot, V., & Frawley, D. (2019). *An Analysis of Completion in Irish Higher Education : 2007 / 08 Entrants*

An Analysis of Completion in Irish Higher (Issue February). Higher Education Authority.

Pino-James, N. (2018). Evaluation of a pedagogical model for student engagement in learning activities. *Educational Action Research*, *26*(3), 456–479. https://doi.org/10.1080/09650792.2017.1354771

Pino-James, N., Shernoff, D. J., Bressler, D. M., Larson, S. C., & Sinha, S. (2019). Instructional interventions that support student engagement: An international perspective. In A. L. R. and S. L. C. Jennifer A. Fredricks (Ed.), *Handbook of Student Engagement Interventions: Working with Disengaged Students* (pp. 103–119). Elsevier. https://doi.org/10.1016/B978-0-12-813413-9.00008-5

Porter, L., Bouvier, D., Cutts, Q., Grissom, S., Lee, C., McCartney, R., Zingaro, D., & Simon, B. (2016). A multi-institutional study of peer instruction in introductory computing. *ACM Inroads*, *7*(2), 76–81. https://doi.org/10.1145/2938142

Powell, K. C., & Kalina, C. J. (2009). Cognitive and social constructivism: developing tools for an effective classroom. *Education*, *130*(2), 241–250. http://content.ebscohost.com.ezp.waldenulibrary.org/ContentServer.asp?T=P&P=AN&K=47349084&S=R&D=ehh&EbscoContent=dGJyMNXb4kSeqa84zdnyOLCmr0qep7VSrqm4S66WxWXS&ContentCustomer=dGJyMPGss0q1qK5IuePfgeyx44Dt6fIA%5Cnhttp://ezp.waldenulibrary.org/login?url=http://

Prayitno, B. A., Corebima, D., Susilo, H., Zubaidah, S., & Ramli, M. (2017). Closing the science process skills gap between students with high and low level academic achievement. *Journal of Baltic Science Education*, *16*(2), 266–277. https://doi.org/10.33225/jbse/17.16.266

Qian, Y., & Lehman, J. (2017). Students' Misconceptions and Other Difficulties in Introductory Programming. *ACM Transactions on Computing Education*, *18*(1), 1–24. https://doi.org/10.1145/3077618

Quille, K., & Bergin, S. (2019). CS1: how will they do? How can we help? A decade of research and practice. *Computer Science Education*, *29*(2–3), 254–282. https://doi.org/10.1080/08993408.2019.1612679

Quille, K., Culligan, N., & Bergin, S. (2017). Insights on Gender Differences in CS1. *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*, 263–268. https://doi.org/10.1145/3059009.3059048

Rahman, F. (2020). Understanding Barriers and Motivations of Non-Traditional Students Learning Programming in an Online CS1 Course. *Proceedings of the 21st Annual Conference on Information Technology Education*, 38–41. https://doi.org/10.1145/3368308.3415455

Ramos, M. C. M. (2018). Correlation between Entrance Exam Scores (Stanine) and Academic Performance. *Proceedings of the 2018 2nd International Conference on Algorithms, Computing and Systems - ICACS '18*, 110–114. https://doi.org/10.1145/3242840.3242866

Randell, B. (2018). Fifty years of software engineering-or-the view from garmisch. *ArXiv Preprint*

ArXiv:1805.02742.

Raskin, J. D. (2002). Constructivism in psychology: Personal construct psychology, radical constructivism, and social constructionism. *American Communication Journal, 5*(3).

Reeve, J. (2012). A Self-determination Theory Perspective on Student Engagement. In S. Christenson, A. Reschly, & C. Wylie (Eds.), *Handbook of Research on Student Engagement* (pp. 149–172). Springer US. https://doi.org/10.1007/978-1-4614-2018-7_7

Reschly, A. L., & Christenson, S. L. (2012). Jingle, Jangle, and Conceptual Haziness: Evolution and Future Directions of the Engagement Construct. In S. Christenson, A. Reschly, & C. Wylie (Eds.), *Handbook of Research on Student Engagement* (pp. 3–19). Springer US. https://doi.org/10.1007/978-1-4614-2018-7_1

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., & others. (2009). Scratch: programming for all. *Communications of the ACM*, *52*(11), 60–67.

Ribeiro, L., Rosário, P., Núñez, J. C., Gaeta, M., & Fuentes, S. (2019). First-Year Students Background and Academic Achievement: The Mediating Role of Student Engagement. *Frontiers in Psychology*, *10*. https://doi.org/10.3389/fpsyg.2019.02669

Rich, P. J., Browning, S. F., Perkins, M. K., Shoop, T., Yoshikawa, E., & Belikov, O. M. (2019). Coding in K-8: International Trends in Teaching Elementary/Primary Computing. *TechTrends*, *63*(3), 311–329. https://doi.org/10.1007/S11528-018-0295-4

Rizvi, M., & Humphries, T. (2012). A Scratch-based CS0 course for at-risk computer science majors. *Proceedings - Frontiers in Education Conference, FIE*, 1–5. https://doi.org/10.1109/FIE.2012.6462491

Rob, M., & Rob, F. (2018). Dilemma between Constructivism and Constructionism: Leading to the Development of a Teaching-Learning Framework for Student Engagement & Learning. *Journal of International Education in Business*, *11*, 0. https://doi.org/10.1108/JIEB-01-2018-0002

Robins, A. V. (2019). Novice Programmers and Introductory Programming. *The Cambridge Handbook of Computing Education Research*, 327–376. https://doi.org/10.1017/9781108654555.013

Rodríguez, S., Núñez, J. C., Valle, A., Freire, C., Ferradás, M. del M., & Rodríguez-Llorente, C. (2019). Relationship Between Students' Prior Academic Achievement and Homework Behavioral Engagement: The Mediating/Moderating Role of Learning Motivation. *Frontiers in Psychology*, *10*. https://doi.org/10.3389/fpsyg.2019.01047

Rountree, N., Rountree, J., & Robins, A. (2002). Predictors of success and failure in a CS1 course. *ACM SIGCSE Bulletin*, *34*(4), 121–124. https://doi.org/10.1145/820127.820182

Rubio, M. A., Romero-Zaliz, R., Mañoso, C., & de Madrid, A. P. (2015). Closing the gender gap in an introductory programming course. *Computers & Education*, *82*, 409–420. https://doi.org/10.1016/J.COMPEDU.2014.12.003

Runco, M. A., & Jaeger, G. J. (2012). The Standard Definition of Creativity. *Creativity Research Journal*, *24*(1), 92–96. https://doi.org/10.1080/10400419.2012.650092

Rushton, A., Gray, L., Canty, J., & Blanchard, K. (2019). Beyond Binary: (Re)Defining "Gender" for 21st Century Disaster Risk Reduction Research, Policy, and Practice. *International Journal of Environmental Research and Public Health*, *16*(20), 3984. https://doi.org/10.3390/ijerph16203984

Russo, M. R., & Bryan, V. C. (2012). Technology, the 21 st century workforce, and the construct of social justice. In V. Wang (Ed.), *Handbook of Research on Technologies for Improving the 21st Century Workforce: Tools for Lifelong Learning* (Vol. 1, pp. 56–75). IGI Global. https://doi.org/10.4018/978-1-4666-2181-7.ch005

Ryoo, J. J. (2019). Pedagogy that Supports Computer Science for All. *ACM Transactions on Computing Education*, *19*(4), 1–23. https://doi.org/10.1145/3322210

Sabitzer, B., & Pasterk, S. (2014). Brain-based programming continued: Effective teaching in programming courses. *2014 IEEE Frontiers in Education Conference (FIE) Proceedings*, 1–6. https://doi.org/10.1109/FIE.2014.7044233

Said-Metwaly, S., Noortgate, W. Van den, & Kyndt, E. (2017). Approaches to Measuring Creativity: A Systematic Literature Review. *Creativity. Theories – Research - Applications*, *4*(2), 238–275. https://doi.org/10.1515/ctra-2017-0013

Saint-Mont, U. (2015). Randomization Does Not Help Much, Comparability Does. *PLOS ONE*, *10*(7), e0132102. https://doi.org/10.1371/journal.pone.0132102

Salkind, N. (2010). Encyclopedia of Research Design. In *Encyclopedia of Research Design*. https://doi.org/10.4135/9781412961288

Sands, P. (2019). Addressing cognitive load in the computer science classroom. *ACM Inroads*, *10*(1), 44–51. https://doi.org/10.1145/3210577

Sawyer, S. M., Azzopardi, P. S., Wickremarathne, D., & Patton, G. C. (2018). The age of adolescence. *The Lancet Child and Adolescent Health*, *2*(3), 223–228. https://doi.org/10.1016/S2352-4642(18)30022-1

Scherer, R., Siddiq, F., & Sánchez Viveros, B. (2020). A meta-analysis of teaching and learning computer programming: Effective instructional approaches and conditions. *Computers in Human Behavior, 109*, 106349. https://doi.org/10.1016/j.chb.2020.106349

Schnitzler, K., Holzberger, D., & Seidel, T. (2021). All better than being disengaged: Student engagement patterns and their relations to academic self-concept and achievement. *European Journal of Psychology of Education*, *36*(3), 627–652. https://doi.org/10.1007/s10212-020-00500-6

Schöber, C., Schütte, K., Köller, O., McElvany, N., & Gebauer, M. M. (2018). Reciprocal effects between self-efficacy and achievement in mathematics and reading. *Learning and Individual Differences, 63,* 1–11. https://doi.org/10.1016/j.lindif.2018.01.008

Schoeman, M. A. (2015). *Enhancing Comprehension in Open Distance Learning Computer Programming Education With Visualization* (Issue October) [University of South Africa]. https://uir.unisa.ac.za/bitstream/handle/10500/20715/thesis_schoeman_ma.pdf?sequence=1&isAllowed=y

Scott, M. J., & Ghinea, G. (2014). Measuring enrichment: the assembly and validation of an instrument to assess student self-beliefs in CS1. *Proceedings of the Tenth Annual Conference on International Computing Education Research*, 123–130.

Scratch - Imagine, Program, Share. (2022). https://scratch.mit.edu/statistics/

Selby, C., & Woollard, J. (2013). *Computational thinking: the developing definition*.

Sharmin, S. (2021). Creativity in CS1: A Literature Review. *ACM Transactions on Computing Education, 22*(2), 1–26. https://doi.org/10.1145/3459995

Shute, V. J., Sun, C., & Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational Research Review, 22*, 142–158.

Silva-Maceda, G., David Arjona-Villicana, P., & Edgar Castillo-Barrera, F. (2016). More Time or Better Tools? A Large-Scale Retrospective Comparison of Pedagogical Approaches to Teach Programming. *IEEE Transactions on Education*, *59*(4), 274–281. https://doi.org/10.1109/TE.2016.2535207

Silvast, A. (2015). An Oral History of Programming Practices: Gender and Age Dynamics and. *Human Technology: An Interdisciplinary Journal on Humans in ICT Environments*, *11*(1), 4–29. https://doi.org/10.17011/ht/urn.201505061738

Simonton, D. K. (2017). Domain-general Creativity: On generating original, useful, and surprising combinations. *The Cambridge Handbook of Creativity across Domains*, 41–60. https://doi.org/10.1017/9781316274385.004

Simonton, D. K. (2018). Defining Creativity: Don't We Also Need to Define What Is Not Creative? *Journal of Creative Behavior*, *52*(1), 80–90. https://doi.org/10.1002/jocb.137

Skinner, E. A., & Pitzer, J. R. (2012). Developmental Dynamics of Student Engagement, Coping, and Everyday Resilience. In S. Christenson, A. Reschly, & C. Wylie (Eds.), *Handbook of Research on Student Engagement* (pp. 21–44). Springer US. https://doi.org/10.1007/978-1-4614-2018-7_2

Statistics South Africa. (2019). Education Series Volume V: Higher Education and Skills in South Africa, 2017. In *Stats SA Library Cataloguing-in-Publication (CIP) Data: Vol. V*. Statistics South Africa.

Stott, A. E. (2018). Are instructivist pedagogies more appropriate for learning the sciences in South African low-quintile schools than western literature suggests? *Journal of Education*, *71*. https://doi.org/10.17159/2520-9868/i71a03

Szabo, C., Sheard, J., Luxton-Reilly, A., Simon, Becker, B. A., & Ott, L. (2019). Fifteen years of introductory programming in schools: A global overview of K-12 initiatives. *PervasiveHealth: Pervasive Computing Technologies for Healthcare*. https://doi.org/10.1145/3364510.3364513

Szentgyorgyi, Z. (1999). A short history of computing in Hungary. *IEEE Annals of the History of Computing*, *21*(3), 49–57.

Task Group on Information Technology Curriculum. (2017). Information Technology Curricula 2017: Curriculum Guidelines for Baccalaureate Degree Programs in Information Technology. In *Information Technology Curricula 2017: Curriculum Guidelines for Baccalaureate Degree Programs in Information Technology*. Association for Computing Machinery. https://doi.org/10.1145/3173161

Taylor, P. C. (2015). Constructivism. In R. Gunstone (Ed.), *Encyclopedia of Science Education* (pp. 218–224). Springer Netherlands. https://doi.org/10.1007/978-94-007-2150-0_102

Tedre, M., Simon, & Malmi, L. (2018). Changing aims of computing education: a historical survey. *Computer Science Education*, *28*(2), 158–186.

Tijani, F., Callaghan, R., & de Villers, R. (2020). An Investigation into Pre-service Teachers’ Experiences While Transitioning from Scratch Programming to Procedural Programming. *African Journal of Research in Mathematics, Science and Technology Education*. https://doi.org/10.1080/18117295.2020.1820798

Tikhonenko, & Pereira, C. (2019). *Informatics Education in Europe: Institutions, degrees, students, positions, salaries. Key Data 2013-2018* (S. Tikhonenko & C. Pereira (Eds.)). Informatics Europe. t.ly/VBvJ

Topalli, D., & Cagiltay, N. E. (2018). Improving programming skills in engineering education through problem-based game projects with Scratch. *Computers & Education*, *120*, 64–74. https://doi.org/10.1016/j.compedu.2018.01.011

Trowler, V. (2010). Student engagement literature review. *Higher Education*, *11*(November), 1–15. http://americandemocracy.illinoisstate.edu/documents/democratic-engagement-white-paper-2_13_09.pdf

UNESCO Institute for Statistics. (2015). *International Standard Classification of Education. Fields of Education and Training 2013 (ISCED-F 2013) Detailed Field Descriptions, 1–96.* https://doi.org/https://doi.org/10.15220/978-92-9189-179-5-en

Vahrenhold, J., Caspersen, M., Berry, G., Gal-Ezer, J., Kölling, M., McGettrick, A., Nardelli, E., Pereira, C., & Westermeier, M. (2017). *Informatics Education in Europe: Are We All In The Same Boat?*

Van den Broeck, L., De Laet, T., Lacante, M., Pinxten, M., Van Soom, C., & Langie, G. (2019). Predicting the academic achievement of students bridging to engineering: the role of academic background variables and diagnostic testing. *Journal of Further and Higher Education*, *43*(7), 989–1007. https://doi.org/10.1080/0309877X.2018.1431209

Veerasamy, A. K., D'Souza, D., Lindén, R., & Laakso, M.-J. (2018). The Impact of Prior Programming Knowledge on Lecture Attendance and Final Exam. *Journal of Educational Computing Research*, *56*(2), 226–253. https://doi.org/10.1177/0735633117707695

Veerasamy, A. K., D'Souza, D., Lindén, R., & Laakso, M. J. (2019). Relationship between perceived problem-solving skills and academic performance of novice learners in introductory programming courses. *Journal of Computer Assisted Learning*, *35*(2), 246–255. https://doi.org/10.1111/jcal.12326

Ventura, P. R. (2005). Identifying predictors of success for an objects-first CS1. *Computer Science Education*, *15*(3), 223–243. https://doi.org/10.1080/08993400500224419

Vygotsky, L. S. (2004). Imagination and Creativity in Childhood. *Journal of Russian & East European Psychology*, *42*(1), 7–97. https://doi.org/10.1080/10610405.2004.11059210

Weintrop, D., & Wilensky, U. (2017). Comparing Block-Based and Text-Based Programming in High School Computer Science Classrooms. *ACM Transactions on Computing Education*, *18*(1), 1–25. https://doi.org/10.1145/3089799

Wilcox, C., & Lionelle, A. (2018). Quantifying the Benefits of Prior Programming Experience in an Introductory Computer Science Course. *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, 80–85. https://doi.org/10.1145/3159450.3159480

Wilson, B. C., & Shrock, S. (2001). Contributing to success in an introductory computer science course. *ACM SIGCSE Bulletin*, *33*(1), 184–188. https://doi.org/10.1145/366413.364581

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33–35. https://doi.org/10.1145/1118178.1118215

Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, *366*(1881), 3717–3725. https://doi.org/10.1098/rsta.2008.0118

Wing, J. M. (2010). *Computational Thinking: What and Why?*

Wing, J. M. (2017). Computational thinking 's influence on research and education for all. In *Italian Journal of Educational Technology* (Vol. 25, Issue 2). Edizioni Menabò-Menabò srl. https://www.learntechlib.org/p/183466/

World Economic Forum. (2018). The Future of Jobs Report 2018. In *Economic Development Quarterly* (Vol. 31, Issue 2). World Economic Forum. http://www.weforum.org/

Xu, Z., Ritzhaupt, A. D., Tian, F., & Umapathy, K. (2019). Block-based versus text-based programming environments on novice student learning outcomes: a meta-analysis study. *Computer Science Education*, *29*(2–3), 177–204. https://doi.org/10.1080/08993408.2019.1565233

Zhang, Y., & Yan, H. (2010). Reform and practice of programming teaching based on applied ability building. *2010 10th IEEE International Conference on Computer and Information Technology*, 2205–2208.

Appendix A

Research Instruments

CS1 Students Profile Questionnaire (CSPROQ)

A - STUDENTS' DEMOGRAPHIC QUESTIONNAIRE

The purpose of this questionnaire is to provide information showing the backgrounds of first-year computer science students taking part in the above PhD research study. The study is aimed at improving the students' performance in introductory programming course (COM 113). I want to assure you that information you provide shall be kept confidential. So, please feel free to supply information as honest and accurate as possible. Thank you for your willingness to participate in the study.

By

Oladele Campbell
Institute for Science and Technology Education
University of South Africa (UNISA), Pretoria. South Africa.

<u>INTRODUCTORY PROGRAMMING ACHIEVEMENT TEST (IPAT)</u>

PRETEST

The purpose of these pretest questions is to measure your current knowledge or ideas about programming. This test is not counted as part of your continuous assessments for the semester. So feel free and be yourself as you answer the questions. You do not need to copy your colleagues' answers. Your answer(s) provide data in a study aimed at understanding students' problems with COM 113 (Introduction to Programming) and exploring an alternative approach to teaching the course so as to improve students' performance. So please be sincere and serious in providing answers where you can. I want to assure you that the information you provide shall be kept confidential and used only for research purpose. Thanks for your willingness to participate in this study.

INSTRUCTION:

Kindly complete all items/questions (where you can) by **writing** your answer or **mark (X**) if you do not have an idea about an item in the space provided.

Name of Polytechnic:____

Identification Number:

Concepts

Here is a list of programming concepts. Please kindly write a short explanation of each one. You are expected to write two or more sentences showing your understanding of each concept. Please write clearly and neatly so that the researcher can understand your answer. If a concept is not familiar, write an "X" in the space provided indicating you do not have an idea.

Concept	**Explanation (2 marks each)**	For office use Only
Program		CMU1

Algorithm		CMU2
Assignment		CMU3
Output		CMU4
Variable		CMU5
Input		CMU6

Looping structure		CMU7
Selection structure		CMU8
Sequence structure		CMU9
Arithmetic operators		CMU10

Question 1

There are three playing cards laid out in a row on a table; each card is labelled with a number. You are given the following sequence of instructions:

```
1. compare the number on the left-hand card with the number on the center card
2. if the number on the left-hand card is greater than the number on the center card
        2.1 exchange the two cards
3. compare the number on the center card with the number on the right-hand card
4. if the number on the center card is greater than the number on the right-hand card
        4.1 exchange the two cards
```

On the table are the following cards:

(a)What will be the numbers on the cards after you carry out the above instructions? (**5 marks**) Q1MA1

(b) What is the purpose of the above sequence of instructions? **(5 marks)**

Q1RU2

Question 2

Here is a sequence of instructions:

```
1. Stand at the origin
2. Turn left
3. Carry out step 3.1 10 times:
      3.1 Move 5 steps
4. Turn right
5. Carry out step 5.1 10 times:
      5.1 Move 5 steps
6. Turn right
7. Carry out step 7.1 10 times:
      7.1 Move 5 steps
```

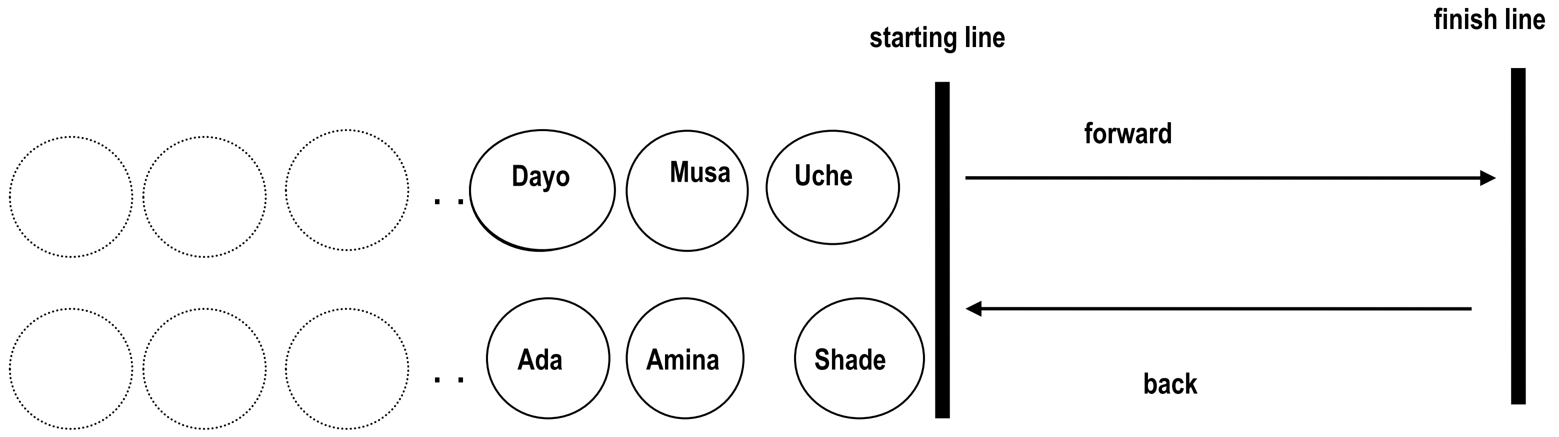
(a) If you carry out these instructions, you will follow a path that is the form of some letter in English. Q2MA2

What is it? (You can also draw the path here.) **(1+.5+1+.5+1+.5+1=5.5) marks for drawing the path. 1 mark for identifying the letter)**

(b) Add more instructions at the end of the list of instructions above so that the path obtained will be a square**.** **(3.5 marks)** Q2MC1

Question 3

Two groups of kids are competing in a relay race. There are 10 kids in each group but here you are given the names of first three kids in each group. Each one runs to the other side of the yard and back, and hands over the baton to the next kid. It takes 5 minutes for each kid in the first group to run back and forth, and 7 minutes for each kid in the second group to run back and forth. The first group consists of Uche followed by Musa followed by Dayo followed by . . . . . . The second group consists of Shade followed by Amina followed by Ada followed by . . . . . See the following diagram:



.

(a) Whose turn is it to run after Musa's?____________________________ **(1 marks)** Q3UU1

(b) Amina's turn comes before whose turn?_________________________ **(1marks)** Q3UU2

(c) How much time will pass until it is Dayo’s turn? Explain briefly_____________________ **(1.5 marks)** Q3MA3

(d) How much time will pass until all members of the first team finish? Explain briefly _________________ **(1.5 marks)**

Q3MA4

(e) How much time will pass until all members of the second team finish? Explain briefly ___________________ **(1.5 marks)**

Q3MA5

(f) How much time will the whole race take? Explain briefly ____________________________________ **(2 marks)**

Q3RA1

(g) What will happen if Musa loses the baton while he is running? **(1.5marks)**

Q3RA2

*** This work, IPAT, is an adaptation of a pretest instrument employed in a study “Learning Computer Science Concepts in *Scratch*” by Orni Meerbaum-Salant, Michal Armoni and Moti Ben-Ari, available at** http://stwww.weizmann.ac.il/g-cs/*Scratch*/tests-cs-concepts-in-*Scratch*.zip **used under CC BY..**

By

Oladele Campbell
Institute for Science and Technology Education
University of South Africa (UNISA), Pretoria. South Africa.

INTRODUCTORY PROGRAMMING ACHIEVEMENT TEST (IPAT)

POSTTEST

The purpose of these posttest questions is to measure your current knowledge or ideas about programming. This test is not counted as part of your continuous assessments for the semester. So feel free and be yourself as you answer the questions. You do not need to copy your colleagues' answers. Your answer(s) provide data in a study aimed at understanding students' problems with COM 113 (Introduction to Programming) and exploring an alternative approach to teaching the course so as to improve students' performance. So please be sincere and serious as you provide answers where you can. I want to assure you that the information you provide shall be kept confidential and used only for research purpose. Thanks for your willingness to participate in this study.

INSTRUCTION:

Kindly complete all items/questions (where you can) by **writing** your answer or **mark (X**) if you do not have an idea about an item in the space provided.

Name of Polytechnic:____

Identification Number:____

Concepts

Here is a list of programming concepts. Please kindly write a short explanation of each one. You are expected to write two or more sentences showing your understanding of each concept. Please write clearly and neatly so that the researcher can understand your answer. If a concept is not familiar, write an "X" in the space provided indicating you do not have an idea.

Concept	**Explanation (2 marks each)**
Program	

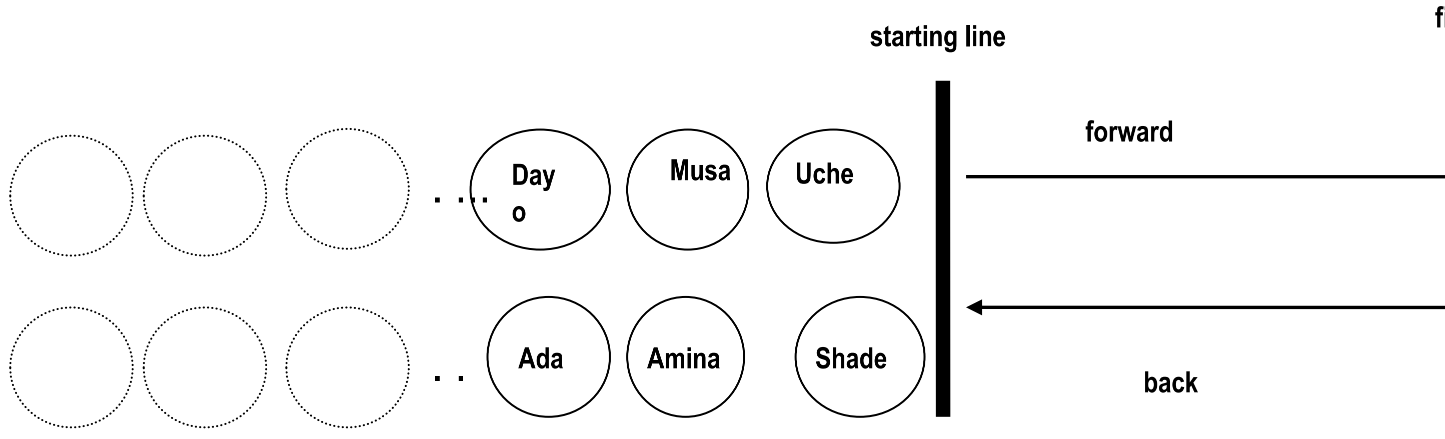
Algorithm	
Assignment	
Output	
Variable	
Input	
Looping structure	
Selection structure	

Sequence structure	
Arithmetic operators	

Question 1

Two groups of kids are competing in a relay race. There are 10 kids in each group but here you are given the names of first three kids in each group. Each one runs to the other side of the yard and back, and hands over the baton to the next kid. It takes 5 minutes for each kid in the first group to run back and forth, and 7 minutes for each kid in the second group to run back and forth. The first group consists of Uche followed by Musa followed by Dayo followed by . . . . . The second group consists of Shade followed by Amina followed by Ada followed by . . . . . See the following diagram:

.



(a) Whose turn is it to run after Musa's?_____________________________ **(1 marks)**

Q3UU1

(b) Amina's turn comes before whose turn?__________________________ **(1marks)**

Q3UU2

(c) How much time will pass until it is Dayo's turn?______________________ **(1.5 marks)**

Q3MA3

(d) How much time will pass until all members of the first team finish? __________________ **(1.5 marks)**

Q3MA4

(e) How much time will pass until all members of the second team finish?____________________ **(1.5 marks)**

Q3MA5

(f) How much time will the whole race take?_______________________________________ **(2 marks)**

Q3RA6

(g) What will happen if Musa loses the baton while he is running?

(1.5marks) Q3RA7

____

____

____

Question 2

Here is a sequence of instructions:

1. Stand at the origin
2. Turn left
3. Carry out step 3.1 10 times:
   3.1 Move 5 steps
4. Turn right
5. Carry out step 5.1 10 times:
   5.1 Move 5 steps
6. Turn right
7. Carry out step 7.1 10 times:
   7.1 Move 5 steps

(a) If you carry out these instructions, you will follow a path that is the form of some letter in English. Q2MA1

What is it? (You can also draw the path here.) **(1+.5+1+.5+1+.5+1=5.5) marks for drawing the path. 1 mark for identifying the letter)**

(b) Add more instructions at the end of the list of instructions above so that the path obtained will be a square**. (3.5 marks)** Q2MC2

Question 3

There are three playing cards laid out in a row on a table; each card is labelled with a number. You are given the following sequence of instructions:

1. compare the number on the left-hand card with the number on the center card
2. if the number on the left-hand card is greater than the number on the center card

    2.1 exchange the two cards

3. compare the number on the center card with the number on the right-hand card
4. if the number on the center card is greater than the number on the right-hand card

    4.1 exchange the two cards

On the table are the following cards: $\boxed{8}$ $\boxed{2}$ $\boxed{6}$

(a)What will be the numbers on the cards after you carry out the above instructions? $\boxed{}$ $\boxed{}$ $\boxed{}$

(**5 marks**) Q1MA1

(b) What is the purpose of the above sequence of instructions? **(5 marks)**

Q1RU2

Introduction to Programming Achievement Test (IPAT) Rubric

INFORMATION FOR TEST MARKERS:

The taxonomy used in this rubric has three categories: unistructural, multistructural and relational cognitive classes. In addition, each category has three cognitive levels: understanding, applying and creating.

Unistructural cognition means that the student has a local perspective mainly knowing only one item or aspect of body of concepts. The other points or ideas are missed neither can the student make connections between related ideas.

Multi-structural cognition means the student knows or makes use of several ideas or concepts in his or her answer. However, the student fails to make connections between these related ideas.

Relational cognitive category means that the student has knowledge of all the related ideas or concepts and is able to make the appropriate connection among them.

PART 1: TESTING CONCEPTUAL PROGRAMMING KNOWLEDGE

(MU = Multi-structural Understanding). So CMU stands for Conceptual Multicultural Understanding.

Here 10 programming concepts are given to the students to measure their learning at the level of multi-structural understanding of these concepts.

For CMU1 – 10, if the students shows:

- Complete and correct understanding – **2 marks**
- Incomplete but correct understanding – **1 marks**
- Incorrect answers – **0 mark**

SUBTOTAL = 20 marks

Concept	**Explanation (2 marks each)**	For office use Only
Program	A program is a set of instructions that are executed by a computing device in order to perform a task or solve a problem. Programs are codes or routines or applications written in a particular programming language that can be understood or translated, and then executed by a computing device.	CMU1
Algorithm	An algorithm is a finite ordered list of steps for solving a computational problem or performing a task.	CMU2
Assignment	Assignment is an operation or a statement in a program that assigns the result (or value) of an expression to a variable.	CMU3
Output	This is the result of a program that may be displayed or written on the monitor as a soft copy or on paper by a printer as a hard copy.	CMU4
	Information resulting from processing input to a computing device.	
Variable	An identifier in a program that can assume different values. A placeholder whose values can change during the execution of a program.	CMU5

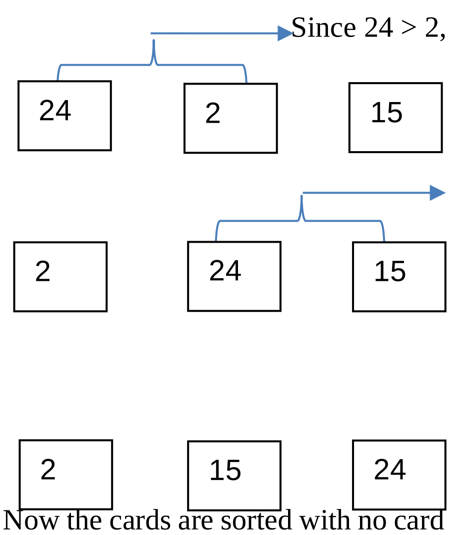
Input	Data entered into a computing device.	CMU6
Looping structure	A set of commands or statements in a program to be repeatedly executed by a computing device.	CMU7
Selection structure	A block of statements in a program that makes a computing device to take alternative execution path depending on specific condition.	CMU8
Sequence structure	A block of statements that are to be executed in a serial manner. That is, the execution of statements are performed one after another strictly in the order they are placed in a sequence.	CMU9
	Arithmetic operators are symbols in an expression in a program that indicate arithmetic operations to be performed during the execution of the program.	

PART 2: TESTING COMPUTATIONAL/PROGRAMMING KNOWLEDGE

Question 1:

- **Q1MA1** (MA = Multi-structural Applying). What is tested here is almost like code-tracing where the student is expected to reason following steps in a piece of code in order to arrive at a particular result.

Following the given algorithm, we have the changes in the positions of the cards as shown below:



Since 24 > 2, then we need to swap the positions of the cards

Since 24 > 15, then we need to swap the positions of the cards

2½

2½

Now the cards are sorted with no card on the left having a number greater than the number on the card at the right.

- **Q1RU1** – (RU = Relational Understanding)> What it is tested here is code explaining ability of the student.

  Answer:

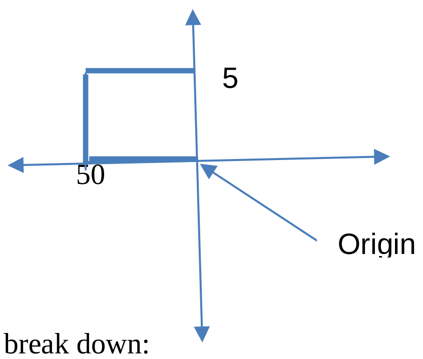
  The purpose of the instructions (or the given algorithm) is to rearrange (or sort) the cards (or numbers) in ascending order.

5

SUBTOTAL = 10 marks

Question 2:

Q2MA2 (MA = Multi-structural Applying). What is tested here is code-tracing skill where the student is expected to reason following steps in a piece of code in order to arrive at a particular result.



Marks break down:

1 mark – for identifying the origin and starting at the right position.

½ mark for turning (tracing the path) in the right direction (i.e. to the left)

1 mark for carrying out the loop and arriving at the right point (50,0) i.e.50 units on the x axis.

½ **mark** for turning (tracing the path) in the right direction (i.e. to the right)

1 mark for carrying out the loop and arriving at the right point (50,50) i.e. 50 units and 50 units on the x axis and y axis respectively

½ **mark** for turning (tracing the path) in the right direction (i.e. to the right)

1 mark for carrying out the loop and arriving at the right point (0, 50) i.e. 50 units on the y axis.

1 mark for identifying the path as the third letter of the alphabet (Letter C)

Q2MC2 (MC = Multi-structural Creating) What is tested here is similar to code tracing, but actually it is code-writing skill we want to measure here.

The three instructions to be added to make the path traced look like a square are:

8. Turn right	**- 1 mark**
9. Carryout <u>step 9.1</u> <u>10 times</u>	**-1½ marks**
9.1 Move 5 steps	**- 1 mark**

SUBTOTAL = 10 marks

Question 3:

Q3UU1 (UU = Unistructural Understanding)

Answer: Dayo **- 1 mark**

Q3UU2

Answer: Ada **- 1 mark**

Q3MA3. (Code tracing skill)

Answer: 10 minutes. Since Dayo's turn comes after those of Uche and Musa, and each of these two forerunners will run for five minutes.

(1½ marks)

Q3MA4. (Code tracing skill)

Answer: 50 minutes. There are 10 members in this group with each running for 5 minutes. **(1½ marks)**

Q3MA5 (Code tracing skill)

Answer: 70 minutes. There are 10 members in this group with each running for 7 minutes. **(1½ marks)**

Q3RA1. (The way the race runs is like the way a code works. So we want to test whether the student understands this way. So it is a code-explaining skill being measured here)

Answer: 70 minutes. The race starts at the same time for the two competing groups (i.e. concurrently). When the first team has finished the race, the eight member of the second team is just starting to run. So the whole race ends when the last member of the second team gets back to the starting line.**(2 marks)**

Q3RA2 (The way the race runs is like the way a code works. So we want to test whether the student understands this way. Hence it is a code-explaining skill being measured here)

Answer: The first group will take longer than 50 minutes to complete the race and the second group may win the race. **(1½ marks)**

SUBTOTAL = 10 marks

TOTAL FOR THE TEST = 50 marks

The Impact of *Scratch*, a Visual Programming Environment, on the Achievement of First-year Computer Science Students in Programming in some Nigerian Polytechnics

***Scratch* Class Observation Protocol (SCOP)**

Polytechnic:______________**Date of Observation:**__________**Number of Students:**____________

Lecturer_________________**Topic**:________________________**Duration**:__________________

To what extent are the following items present in the class during your observation?
Rating Scale (1= Never, 2 = Sometimes, 3 = Often, 4 = Generally, 5 = Almost always)

Serial No.	**ITEM**	**Tick where applicable**				
		1	**2**	**3**	**4**	**5**
AFFECTIVE IMPACT (Programming attitude)						
1.	Students show sign of **boredom** during class					
2	Students are **confused** with the programming task.					
3	Students are **confused** with the programming topic					
4	Students are **delighted** during the class					
5	Students show sign of **surprise** during class.					
6	Students show signs of **frustration** during class.					
7	Students are **immersed** with topics or tasks in class.					
8	Students are **highly motivated** wanting to learn programming.					
9.	Students show sign of increasing **self-confidence**					
10.	Students show sign of **resentment** to programming.					
COGNITIVE IMPACT (Programming Knowledge)						
1	Students works shows **creativity**					
2	Students are '**gaming the system'**, guessing and arbitrarily performing programming exercises.					
3	Students develop **correct algorithms**.					
4	Students develop **correct programs** in *Scratch*					
PSYCHOMOTOR IMPACT (Programming Skills)						
1	Students **navigate *Scratch*** environment quickly.					
2	Students **turn in** programming solutions **quickly**.					
3	Students are **inactive** during class.					
SOCIAL IMPACT (Programming Collaborations)						
1	Students are **interacting** in class to solve programming problems.					
2	Students **ask their colleagues** when they have questions.					
3	Students **ask the teacher** when they have problems.					

Other comments or observations:
____
____
____
____

____

____

____

Observer:__________________________________

The Impact of *Scratch* on the Achievement of First-year Computer Science Students in Programming in some Nigerian Polytechnics

***SCRATCH* POST INTERVENTION EVALUATION INTERVIEW (SPIEI)**

The purpose of these interview questions is to gather information about your experiences, knowledge, ideas, and opinions about programming and *Scratch* after the six weeks of teaching. So please feel free and be yourself as you answer the questions as best as you can. Your answer(s) provide data in a study aimed at understanding students' problems with COM 113 (introductory to programming) in Nigerian polytechnics and exploring an alternative approach to teaching the course so as to improve students' performance. I hope you would not mind if I take a recording of our conversation. I want to assure you that the information you provide shall be kept confidential and used only for research purpose. However, you are free to object to answering any question or withdraw from the interview without any consequence. Thanks for your willingness to participate in this study.

A. *Scratch* and Programming Learning

1. Tell me about your programming experience in *Scratch*, what it looks like from the beginning of the *Scratch* class to this time.
2. How will you describe your class' overall programming learning experience in the *Scratch* environment?

   Probe: You have just described your class' experience with *Scratch* as ____________, how did you arrive at this answer?

3. Has your programming ability improved due to your participation in the *Scratch* class?

   Probe: if yes, can you mention some of the programming works or projects you created with *Scratch*. If no, what hindered you?

B. *Scratch*, Prior Programming Background and Programming Ability

1. Did you have knowledge of computer programming before your admission to the Polytechnic?
2. Do you see any advantage of this your programming background and programming learning in *Scratch*?

   Probe: In what ways has your background helped (…or makes no difference to or hindered) programming ability in *Scratch*?

C. *Scratch*, Gender and Programming Ability

1. To what extent do you think your gender affected your programming ability during your working in *Scratch*?

   Probe: Why do you think gender has this kind of effect on a student's programming ability in *Scratch*?

2. Did you notice any difference along gender lines in the performance of students during your *Scratch* Classes?

   Probe: Can you mention specific roles being played by the different genders during classes?

D. *Scratch*, Student Academic Achievement Level and Programming Ability

1. How do you rate your academic achievement level before exposure to *Scratch*? High, average or low.

   Probe: Did you see this academic background playing a significant role in your programming ability in the *Scratch* environment?

E. *Scratch*, Visual Art Background and Programming Ability

1. Do you have skills in drawing or making things before your exposure to *Scratch*?

   Probe: What influence do you think your (or lack of) background in arts had in your learning programming in *Scratch*?

ATTACHMENT B

The Impact of *Scratch* on the Achievements of First-year Computer Science Students in Programming in some Nigerian Polytechnics

INFORMED CONSENT FORM

Purpose of research

The proposed research with the above title is undertaken in order to investigate what impact *Scratch*, a visual programming environment, can have on the programming ability of first-year computer science students. Possible outcomes are significantly positive, none or significantly negative effects on students ‘programming ability and attitude.

Risks and benefits

No any other risk is involved in this research other normal risks of participating in classes as a student that you are used to. Potential benefits of the study include providing scientific knowledge about the use of *Scratch* in introductory programming class. If the result of this research provides a positive evidence in favour of introducing first-year undergraduate students to programming using *Scratch*, this can lead to improved teaching and learning for students.

This research require your answering voluntarily questions contained in questionnaire, achievement test and possibly interview at the end of the six weeks of classes. Data to be collected from you include demographic information about yourself, educational and programming background, as well as knowledge, experiences and opinions about programming in the class you will be participating in.

Methods of study and participants’ actual role in research

The research will make use of the following research methods: questionnaire, achievement test, observation of class sessions and semi-structured interview. The questionnaires which will administered once in class at the beginning of the study will take 20 minutes while the achievements tests (which will be taken twice, i.e. before the programming class and after six weeks of instruction) will take about 60 minutes. The interview to be taken by selected participants will last for about 45 minutes.

Identity of the researchers

In case you have any question about the research you are free to contact the following:
Oladele Campbell (the researcher) – ISTE, University of South Africa. +2348059062424
Prof H.I. Atagana (Supervisor) – ISTE, University of South Africa. +27822009855

Why you were selected

The method used for selecting the participants in this study is multistage. First, we have used purposive sampling to select four federal polytechnics among the thirteen accredited polytechnics running National Diploma programme in computer science in the north central region. Second, the same earlier sampling technique was used to assign your class into one of the two study groups. You have been selected for this study (after due permission from your school authority) because the study involves students in their first-year computer science programme in selected Nigerian polytechnics.

Privacy, anonymity and confidentiality

You are assured that your right to privacy will be respected in during and after this research. Data to be collected will be used only for the purpose of research. Information that can jeopardize your privacy, anonymity and confidentiality will be removed or replaced with pseudonyms in research reports.

Future use of information.

Information obtained from this research will be published in my PhD thesis, research paper(s) in conferences and journal, and online repositories for educational and research purposes.

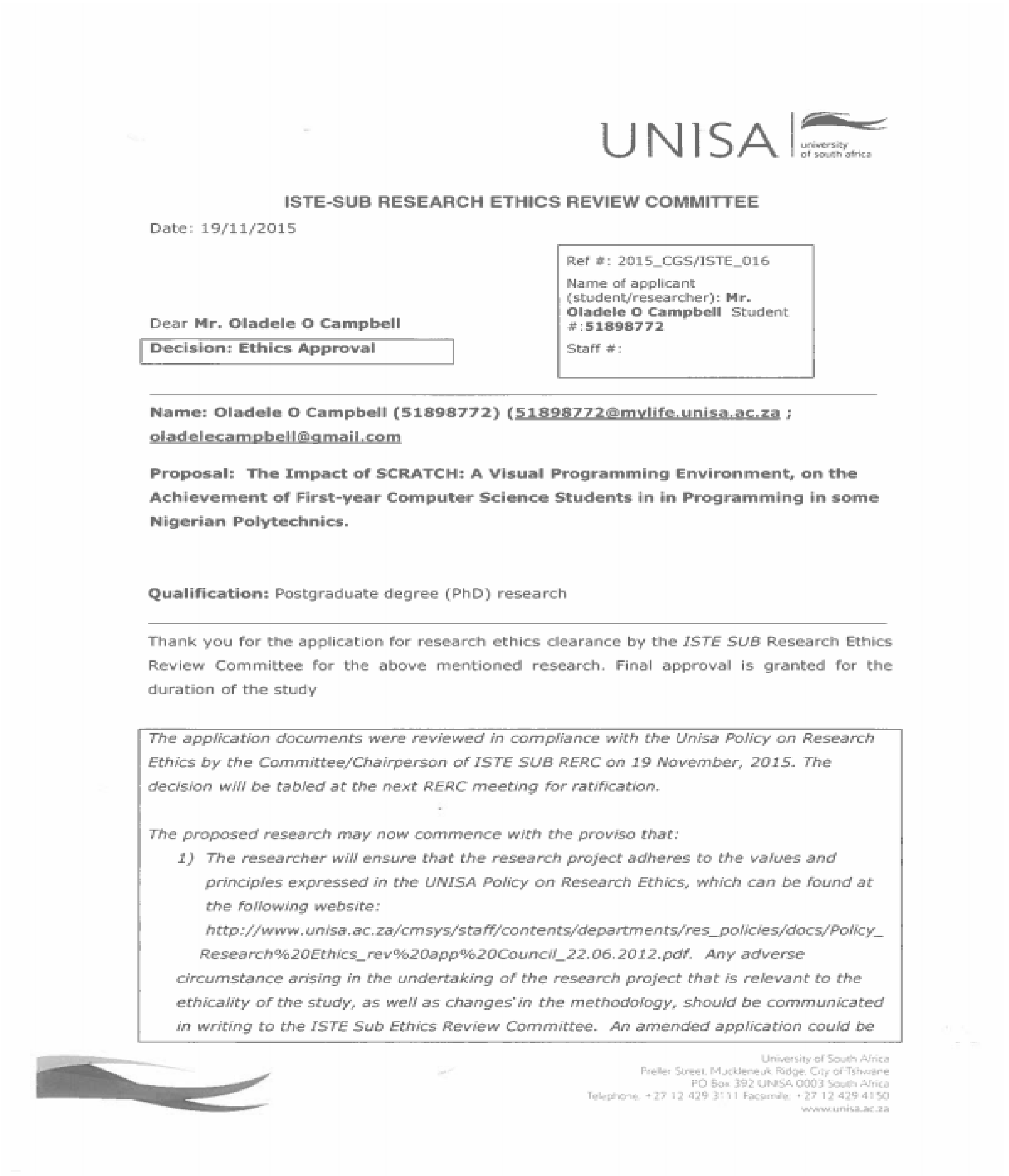
Right not to participate and to withdraw
Please be informed that you have the right to decline from participating in this study or to withdraw from your earlier given consent at any time without fear of any penalty. You are also free to answer or decline from answering certain questions in the questionnaire, achievement test, or the interview. In addition if you are selected and consent to participate in the interview you are free to object the use of data gathering devices such camera, tape recorder etc.

PARTICIPANT'S CONSENT
I have read the information presented above about the study. I have had the opportunity to ask any questions related to this study and I have received satisfactory answers to my questions. I am aware that, if I am selected for interview at the end of the study, I have the option of allowing my interview to be audio recorded to ensure an accurate recording of my responses. I am also aware that information to be collected in the study may be included in publications to come from this research, with the understanding that the Personally Identifiable Information (PII) about me will be made anonymous. I was informed that I may withdraw my consent at any time without penalty. With full knowledge of all foregoing, I agree, of my own free will, to participate in this study.

Participant's Name:____
Participant's Signature:____
Researcher's Name:____
Researcher's Signature:____
Date:____

APPENDIX B

Ethical Clearance, Study Request and Permission Letters



UNISA university of south africa

ISTE-SUB RESEARCH ETHICS REVIEW COMMITTEE

Date: 19/11/2015

Ref #: 2015_CGS/ISTE_016

Name of applicant (student/researcher): **Mr. Oladele O Campbell** Student #:**51898772**

Staff #:

Dear **Mr. Oladele O Campbell**

Decision: Ethics Approval

Name: Oladele O Campbell (51898772) (51898772@mylife.unisa.ac.za ; oladelecampbell@gmail.com

Proposal: The Impact of SCRATCH: A Visual Programming Environment, on the Achievement of First-year Computer Science Students in in Programming in some Nigerian Polytechnics.

Qualification: Postgraduate degree (PhD) research

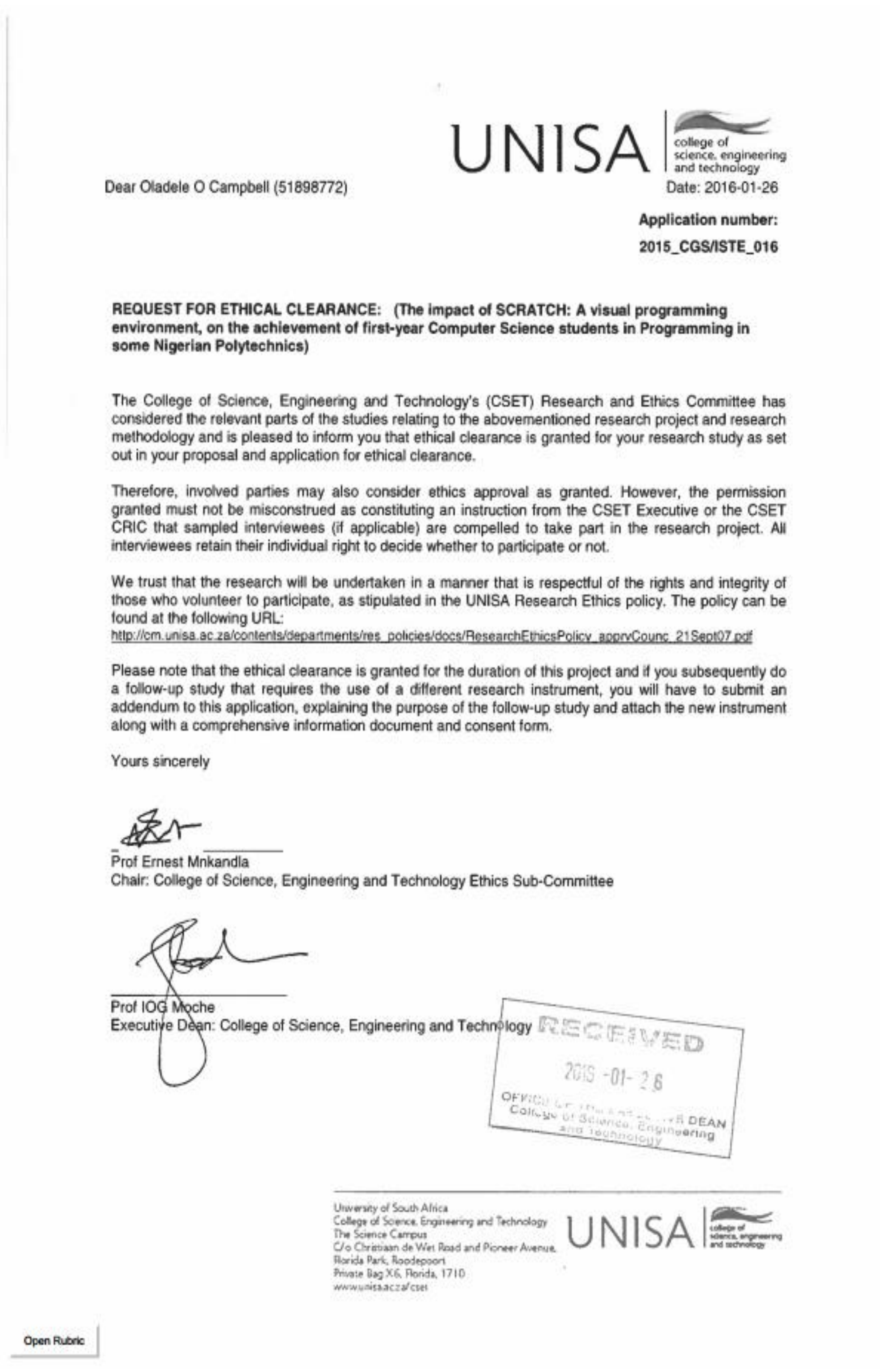
Thank you for the application for research ethics clearance by the *ISTE SUB* Research Ethics Review Committee for the above mentioned research. Final approval is granted for the duration of the study

The application documents were reviewed in compliance with the Unisa Policy on Research Ethics by the Committee/Chairperson of ISTE SUB RERC on 19 November, 2015. The decision will be tabled at the next RERC meeting for ratification.

The proposed research may now commence with the proviso that:

1) *The researcher will ensure that the research project adheres to the values and principles expressed in the UNISA Policy on Research Ethics, which can be found at the following website: http://www.unisa.ac.za/cmsys/staff/contents/departments/res_policies/docs/Policy_Research%20Ethics_rev%20app%20Council_22.06.2012.pdf. Any adverse circumstance arising in the undertaking of the research project that is relevant to the ethicality of the study, as well as changes in the methodology, should be communicated in writing to the ISTE Sub Ethics Review Committee. An amended application could be*

University of South Africa
Preller Street, Muckleneuk Ridge, City of Tshwane
PO Box 392 UNISA 0003 South Africa
Telephone: +27 12 429 3111 Facsimile: +27 12 429 4150
www.unisa.ac.za



UNISA college of science, engineering and technology

Dear Oladele O Campbell (51898772)

Date: 2016-01-26

Application number:
2015_CGS/ISTE_016

REQUEST FOR ETHICAL CLEARANCE: (The impact of SCRATCH: A visual programming environment, on the achievement of first-year Computer Science students in Programming in some Nigerian Polytechnics)

The College of Science, Engineering and Technology's (CSET) Research and Ethics Committee has considered the relevant parts of the studies relating to the abovementioned research project and research methodology and is pleased to inform you that ethical clearance is granted for your research study as set out in your proposal and application for ethical clearance.

Therefore, involved parties may also consider ethics approval as granted. However, the permission granted must not be misconstrued as constituting an instruction from the CSET Executive or the CSET CRIC that sampled interviewees (if applicable) are compelled to take part in the research project. All interviewees retain their individual right to decide whether to participate or not.

We trust that the research will be undertaken in a manner that is respectful of the rights and integrity of those who volunteer to participate, as stipulated in the UNISA Research Ethics policy. The policy can be found at the following URL:
http://cm.unisa.ac.za/contents/departments/res_policies/docs/ResearchEthicsPolicy_apprvCounc_21Sept07.pdf

Please note that the ethical clearance is granted for the duration of this project and if you subsequently do a follow-up study that requires the use of a different research instrument, you will have to submit an addendum to this application, explaining the purpose of the follow-up study and attach the new instrument along with a comprehensive information document and consent form.

Yours sincerely

[illegible]

Prof Ernest Mnkandla
Chair: College of Science, Engineering and Technology Ethics Sub-Committee

[illegible]

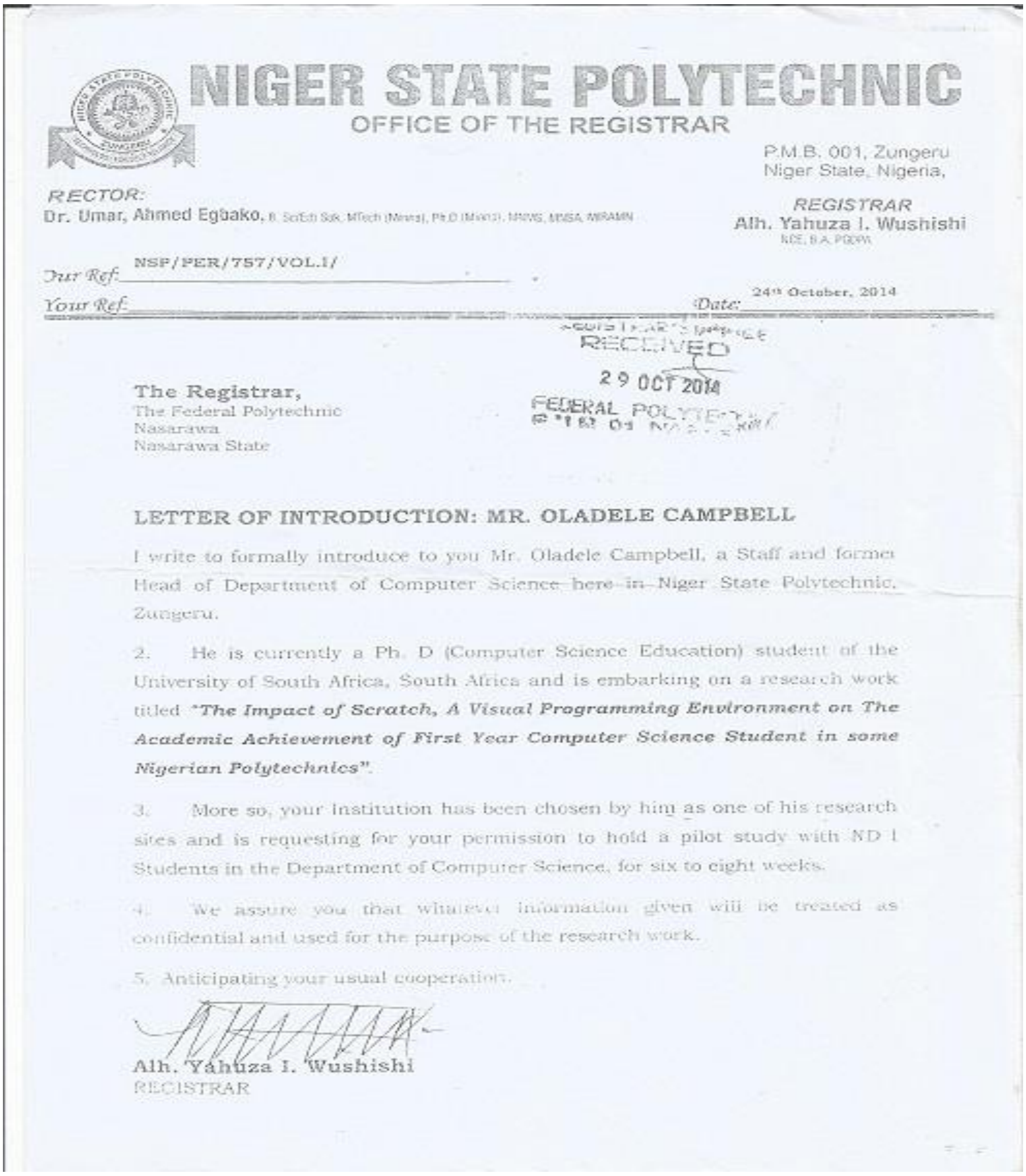
Prof IOG Moche
Executive Dean: College of Science, Engineering and Technology

RECEIVED
2015 -01- 26
OFFICE OF THE EXECUTIVE DEAN
College of Science, Engineering and Technology

University of South Africa
College of Science, Engineering and Technology
The Science Campus
C/o Christiaan de Wet Road and Pioneer Avenue, Florida Park, Roodepoort
Private Bag X6, Florida, 1710
www.unisa.ac.za/cset

UNISA college of science, engineering and technology

Open Rubric



NIGER STATE POLYTECHNIC
OFFICE OF THE REGISTRAR

P.M.B. 001, Zungeru
Niger State, Nigeria,

RECTOR:
Dr. Umar, Ahmed Egbako, B. Sc(Ed) Sok, MTech (Mines), Ph.D (Mines), MNIVS, MNSA, MIRAMN

REGISTRAR
Alh. Yahuza I. Wushishi
NCE, B.A, PGDPA

Our Ref: NSP/PER/757/VOL.I/
Your Ref:
Date: 24th October, 2014

REGISTRAR'S OFFICE
RECEIVED
29 OCT 2014
FEDERAL POLYTECHNIC
P.M.B. 01 NASARAWA

The Registrar,
The Federal Polytechnic
Nasarawa
Nasarawa State

LETTER OF INTRODUCTION: MR. OLADELE CAMPBELL

I write to formally introduce to you Mr. Oladele Campbell, a Staff and former Head of Department of Computer Science here in Niger State Polytechnic, Zungeru.

2. He is currently a Ph. D (Computer Science Education) student of the University of South Africa, South Africa and is embarking on a research work titled ***"The Impact of Scratch, A Visual Programming Environment on The Academic Achievement of First Year Computer Science Student in some Nigerian Polytechnics"***.

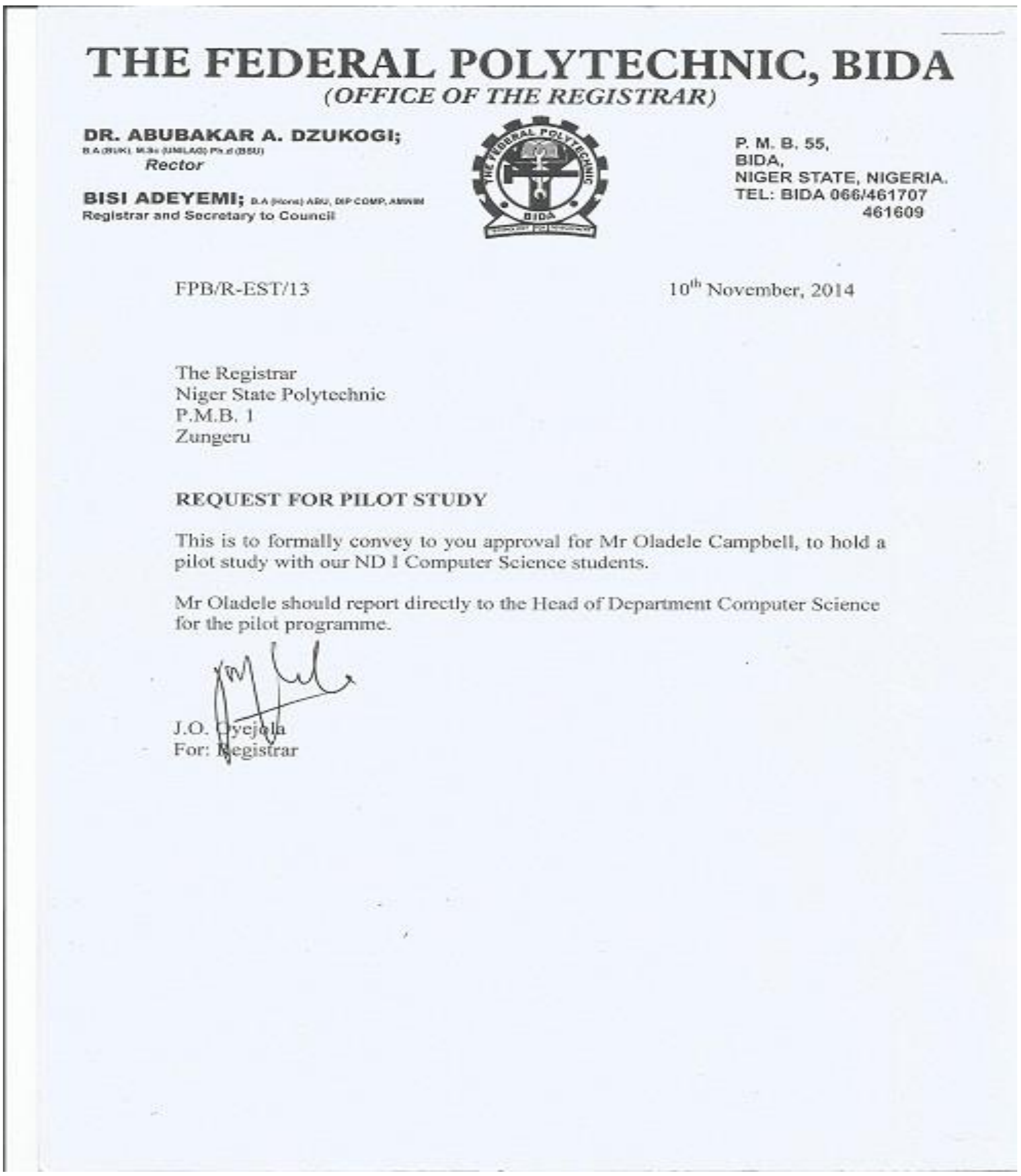
3. More so, your institution has been chosen by him as one of his research sites and is requesting for your permission to hold a pilot study with ND I Students in the Department of Computer Science, for six to eight weeks.

4. We assure you that whatever information given will be treated as confidential and used for the purpose of the research work.

5. Anticipating your usual cooperation.

[illegible]

Alh. Yahuza I. Wushishi
REGISTRAR



THE FEDERAL POLYTECHNIC, BIDA
(OFFICE OF THE REGISTRAR)

DR. ABUBAKAR A. DZUKOGI; B.A (BUK), M.Sc (UNILAG) Ph.d (BSU)
Rector

BISI ADEYEMI; B.A (Hons) ABU, DIP COMP, AMNIM
Registrar and Secretary to Council

P. M. B. 55,
BIDA,
NIGER STATE, NIGERIA.
TEL: BIDA 066/461707
461609

FPB/R-EST/13

10th November, 2014

The Registrar
Niger State Polytechnic
P.M.B. 1
Zungeru

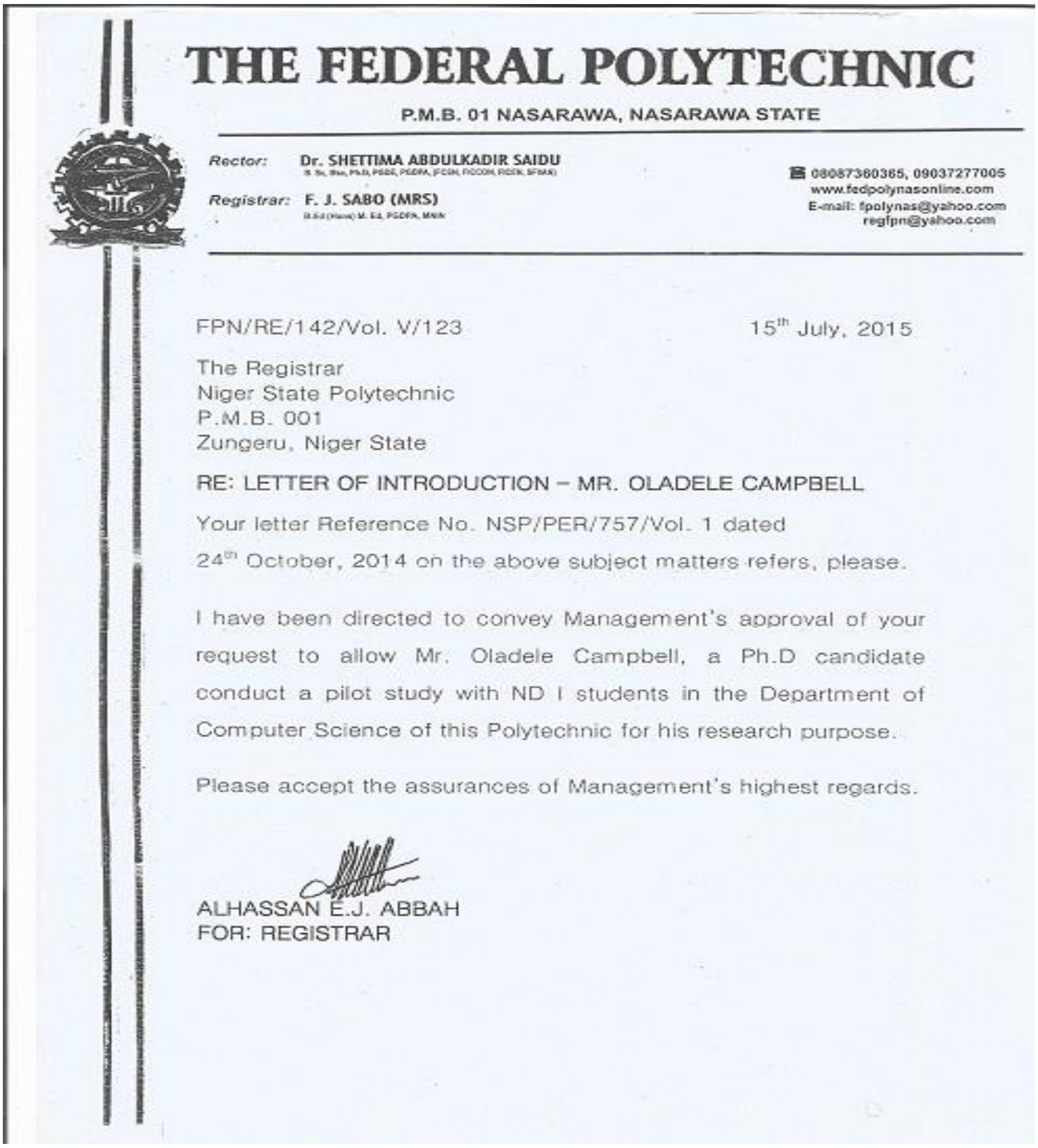
REQUEST FOR PILOT STUDY

This is to formally convey to you approval for Mr Oladele Campbell, to hold a pilot study with our ND I Computer Science students.

Mr Oladele should report directly to the Head of Department Computer Science for the pilot programme.

[illegible]

J.O. Oyejola
For: Registrar



THE FEDERAL POLYTECHNIC

P.M.B. 01 NASARAWA, NASARAWA STATE

Rector: Dr. SHETTIMA ABDULKADIR SAIDU
B. Sc, Msc, PhD, PGDE, PGDPA, (FCSN, FICCON, FGEN, SFMAN)

Registrar: F. J. SABO (MRS)
B.Ed (Hons) M. Ed, PGDPA, MNIM

☎ 08087360365, 09037277005
www.fedpolynasonline.com
E-mail: fpolynas@yahoo.com
regfpn@yahoo.com

FPN/RE/142/Vol. V/123 15th July, 2015

The Registrar
Niger State Polytechnic
P.M.B. 001
Zungeru, Niger State

RE: LETTER OF INTRODUCTION – MR. OLADELE CAMPBELL

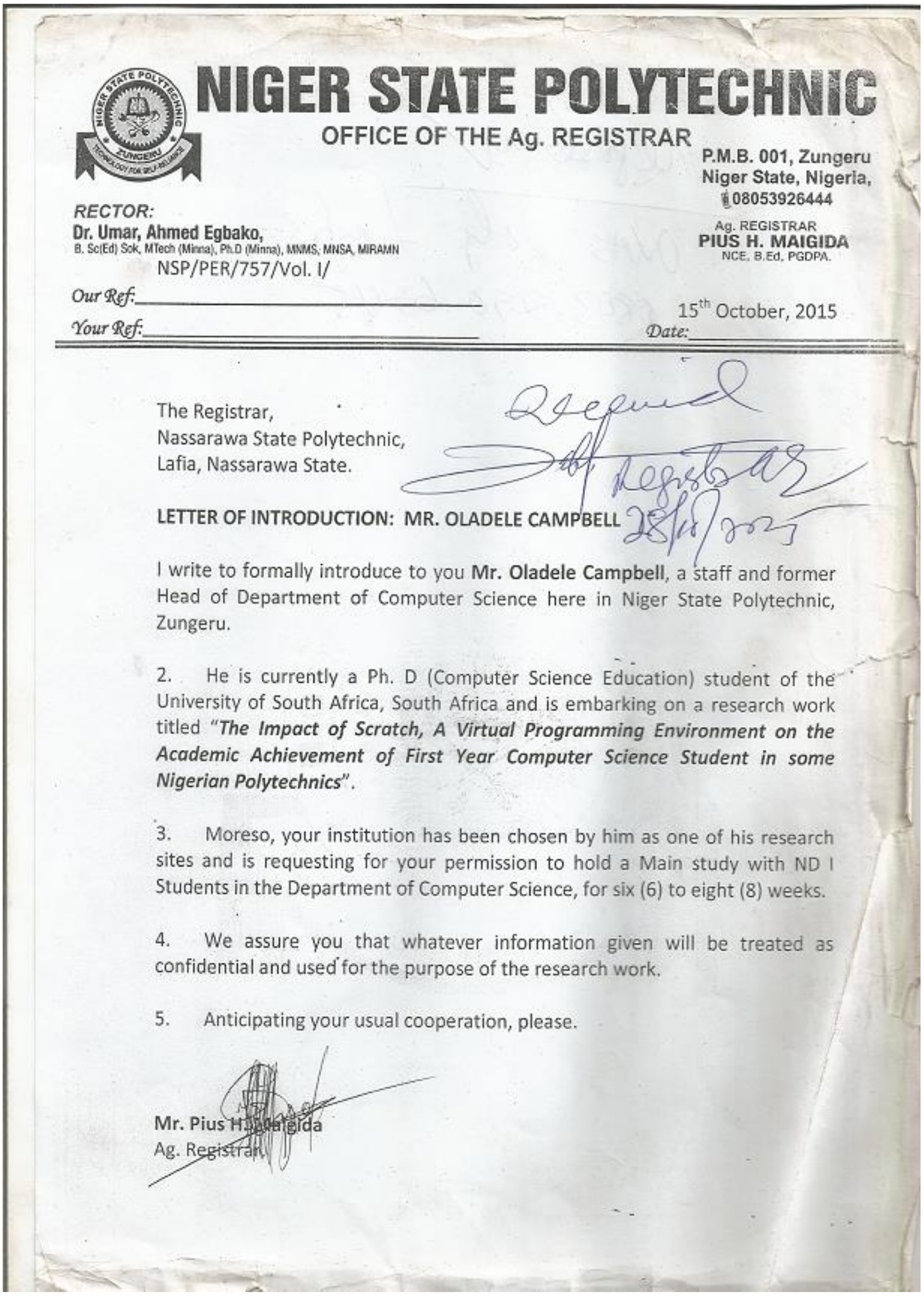
Your letter Reference No. NSP/PER/757/Vol. 1 dated 24th October, 2014 on the above subject matters refers, please.

I have been directed to convey Management's approval of your request to allow Mr. Oladele Campbell, a Ph.D candidate conduct a pilot study with ND I students in the Department of Computer Science of this Polytechnic for his research purpose.

Please accept the assurances of Management's highest regards.

[illegible]

ALHASSAN E.J. ABBAH
FOR: REGISTRAR



NIGER STATE POLYTECHNIC

OFFICE OF THE Ag. REGISTRAR

P.M.B. 001, Zungeru
Niger State, Nigeria,
08053926444

RECTOR:
Dr. Umar, Ahmed Egbako,
B. Sc(Ed) Sok, MTech (Minna), Ph.D (Minna), MNMS, MNSA, MIRAMN

Ag. REGISTRAR
PIUS H. MAIGIDA
NCE, B.Ed, PGDPA.

Our Ref: NSP/PER/757/Vol. I/

Your Ref:

Date: 15th October, 2015

The Registrar,
Nassarawa State Polytechnic,
Lafia, Nassarawa State.

Received
[illegible] Registrar
28/10/2015

LETTER OF INTRODUCTION: MR. OLADELE CAMPBELL

I write to formally introduce to you **Mr. Oladele Campbell**, a staff and former Head of Department of Computer Science here in Niger State Polytechnic, Zungeru.

2. He is currently a Ph. D (Computer Science Education) student of the University of South Africa, South Africa and is embarking on a research work titled "***The Impact of Scratch, A Virtual Programming Environment on the Academic Achievement of First Year Computer Science Student in some Nigerian Polytechnics***".

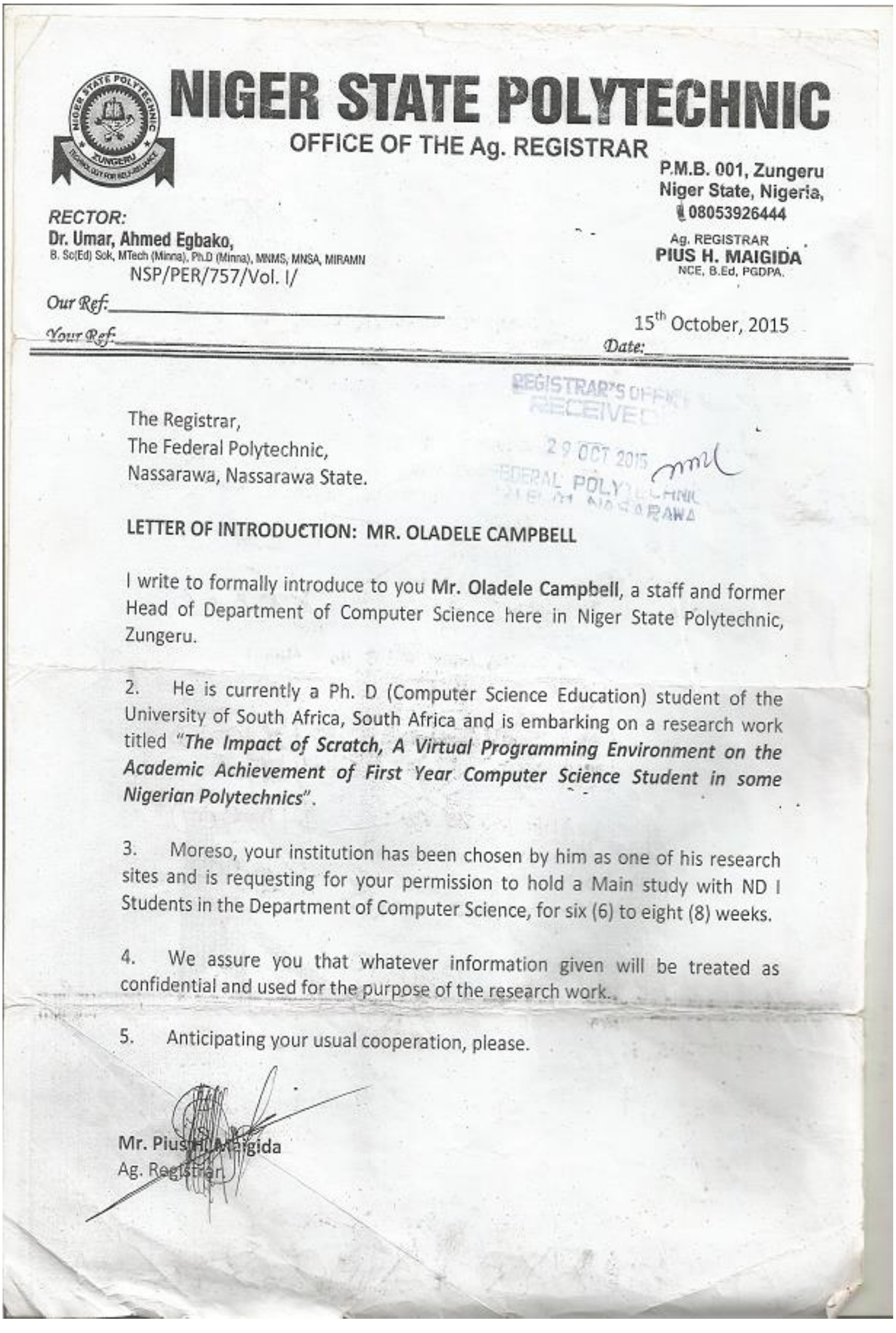
3. Moreso, your institution has been chosen by him as one of his research sites and is requesting for your permission to hold a Main study with ND I Students in the Department of Computer Science, for six (6) to eight (8) weeks.

4. We assure you that whatever information given will be treated as confidential and used for the purpose of the research work.

5. Anticipating your usual cooperation, please.

[illegible]

Mr. Pius H. Maigida
Ag. Registrar



NIGER STATE POLYTECHNIC

OFFICE OF THE Ag. REGISTRAR

P.M.B. 001, Zungeru
Niger State, Nigeria,
08053926444

RECTOR:
Dr. Umar, Ahmed Egbako,
B. Sc(Ed) Sok, MTech (Minna), Ph.D (Minna), MNMS, MNSA, MIRAMN

Ag. REGISTRAR
PIUS H. MAIGIDA
NCE, B.Ed, PGDPA.

Our Ref: NSP/PER/757/Vol. I/

Your Ref:

Date: 15th October, 2015

REGISTRAR'S OFFICE RECEIVED 29 OCT 2015 FEDERAL POLYTECHNIC NASARAWA

The Registrar,
The Federal Polytechnic,
Nassarawa, Nassarawa State.

LETTER OF INTRODUCTION: MR. OLADELE CAMPBELL

I write to formally introduce to you **Mr. Oladele Campbell**, a staff and former Head of Department of Computer Science here in Niger State Polytechnic, Zungeru.

2. He is currently a Ph. D (Computer Science Education) student of the University of South Africa, South Africa and is embarking on a research work titled "***The Impact of Scratch, A Virtual Programming Environment on the Academic Achievement of First Year Computer Science Student in some Nigerian Polytechnics***".

3. Moreso, your institution has been chosen by him as one of his research sites and is requesting for your permission to hold a Main study with ND I Students in the Department of Computer Science, for six (6) to eight (8) weeks.

4. We assure you that whatever information given will be treated as confidential and used for the purpose of the research work.

5. Anticipating your usual cooperation, please.

[illegible]

Mr. Pius H. Maigida
Ag. Registrar