



저작자표시-동일조건변경허락 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.
- 이차적 저작물을 작성할 수 있습니다.
- 이 저작물을 영리 목적으로 이용할 수 있습니다.

다음과 같은 조건을 따라야 합니다:



저작자표시. 귀하는 원저작자를 표시하여야 합니다.



동일조건변경허락. 귀하가 이 저작물을 개작, 변형 또는 가공했을 경우에는, 이 저작물과 동일한 이용허락조건하에서만 배포할 수 있습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 [이용허락규약\(Legal Code\)](#)을 이해하기 쉽게 요약한 것입니다.

[Disclaimer](#)

공학석사 학위논문

**From Music to Programming:  
Designing Method for Introductory  
Programming Using Musical  
Metaphor and Auditory Feedback**

초보자를 위한 프로그래밍 교육 방안 연구  
-음악 구조와 프로그래밍 코드의 연결고리를 중심으로-

2014년 2월

서울대학교 대학원

융합과학부 디지털정보융합전공

고 은 정



# Contents

LIST OF TABLES .....	iii
LIST OF FIGURES .....	iv
ABSTRACT.....	v
1 Introduction .....	1
1.1 Background .....	1
1.2 Research Question .....	3
1.3 Research Goal and Objectives .....	3
2 Literature Review .....	4
2.1 Computer Science Education .....	4
2.1.1 Educational Programming Languages .....	4
2.1.2 Introductory Programming .....	6
2.2 Music and Programming .....	10
2.3 Summary of Literature Review .....	11
3 Methodology .....	13
3.1 System .....	13
3.1.1 Programming Style .....	13
3.1.2 Key Structures.....	16
3.1.3 Programming Environment.....	20
3.2 User Survey.....	23

4 Experimental Evaluation .....	25
4.1 Introductory Course .....	26
4.2 Tutorials .....	29
4.3 Preliminary Case Study.....	31
4.4 Questionnaire .....	32
4.5 Participants.....	32
5 Results and Discussion .....	35
5.1 Data Analysis .....	35
5.2 Survey Feedback.....	41
5.3 In-depth Interview.....	46
6 Conclusion .....	48
6.1 Contributions.....	48
6.2 Limitation and Future work .....	50
ACKNOWLEDGEMENTS .....	51
REFERENCES .....	53
Appendix A: Online Survey Questionnaire .....	57
Appendix B: Offline Survey Questionnaire .....	59
Appendix C: Materials from SNU-IRB.....	65
Appendix D: Examples of tutorial.....	70

## List of Tables

Table 1	Lahtinen's (2005) check list in learning programming.....	6
Table 2	Data types and syntax .....	14
Table 3	Comparing between the two of function definitions .....	15
Table 4	Basic algorithms .....	16
Table 5	Musical settings as programming metaphors.....	20
Table 6	Background information of participants: Major (N=32) .....	33
Table 7	Inductive categorization of positive feedback from the student .	42
Table 8	Inductive categorization of negative feedback from the student	44

## List of Figures

Figure 1	The syntax of Java .....	14
Figure 2	The concept of proposed approach .....	15
Figure 3	Simplified programming syntax of conditional structure .....	17
Figure 4	Simplified programming syntax of loop structure .....	18
Figure 5	Simplified programming syntax of recursive example 1 .....	19
Figure 6	Simplified programming syntax of recursive example 2 .....	19
Figure 7	First prototype of code editor framework .....	20
Figure 8	Second prototype of code editor framework.....	21
Figure 9	Third prototype of code editor framework .....	21
Figure 10	An example of a Google Drive Excel sheet with the responses data from the participants .....	24
Figure 11	Schedule for introductory course .....	26
Figure 12	Introduction of Tutorial (Study version) .....	30
Figure 13	Background information of participants: Age (N=32).....	33
Figure 14	Comparison of control experimental group rating on self-reported level of assistance and enjoyment.....	36
Figure 15	Comparison of control and experimental group rating on self-reported level of understanding and satisfaction.....	37
Figure 16	Comparison of control and experimental group rating on self-reported level of content and understanding about the process of the course .	38
Figure 17	Comparison of control and experimental group rating on self-reported level of response about result form .....	40
Figure 18	Frequency of positive issues about the system .....	43
Figure 19	Frequency of negative issues about the system .....	45

## Abstract

Despite advances in programming language education, many beginner programmers face difficulties and give up in the early stages, just because they are not familiar with the programming syntax and semantics. In this research, we propose a method, for introductory programming using musical metaphor with an aim to entice beginners to program. This methodology is motivated by two concepts: first, music notation as an analogy to programming provides an enjoyable programming experience; second, on-line auditory feedback enables to notify the status of program for the users in a pleasant way. We described musical the settings as programming metaphors to help beginners learn them with ease and intuition. We built on this work through the system by incorporating with Java API for on-line auditory feedback. This system is to help beginners provide on-line auditory feedback as a communication medium to immediately notify the results in a pleasant way. We tested the methodology with 32 students as novice programmers and found that those in the experimental condition qualified significantly more inviting experience with this study. Participants in the control and experimental groups took a course for introductory programming, and most students felt that this methodology was a positive influence, particularly those with enjoyment. These findings suggest that some useful ideas how programming is taught, and it could be an impact on motivation to program and influence students' first-time experience.

**Keywords: Introductory Programming; Computer Science Education; Musical Metaphor; Auditory Feedback; Interactive Environments;**  
**Student Number: 2012-22457**



# 1. Introduction

## 1.1 Background

In order to gain and improve their knowledge in programming, novice programmers conflict goals that occur in complex programming syntax and semantics (Carter, 2006). People who do not write programs regularly face many barriers in the process of learning a programming language (Fitzgerald, 2008). While there are numerous ways to teach people through educational programming languages, it is inevitable that novice programmers often feel difficulties in understanding abstract concepts without any similar phenomenon in the daily life for comparison (Lahtinen, Ala-Mutka & Jarvinen, 2005).

There are not enough resources and programming metaphors to support an understand-able programming learning experience (Tarkan et al, 2010). Furthermore, there is a rich history of studies on programming environments for beginners, though; many of these methods have focused on visual programming, because graphic aids for thinking allow amplifying cognition in learning situations (Lee & Ko, 2011). These technologies do not provide immediate feedback, which is essential in helping novice programmers understand the status of their programs. There are rare cases that address the form of result with auditory factors for interactive communication in educational settings (Ko, Myers & Aung, 2004).

If inability to connect programs with metaphor on the environment affects

people's performance on traditional programming then, does useful metaphor affect novice programmer's motivation and learning success? To examine this question, we created the methodology to provide a positive programming experience for students and introduce concepts of programming. Our decision to use musical metaphor was motivated by the desire to make relationship between music and programming that encourages effective interaction with the system (Bramwell-Dicks et al, 2013; Huron, 2006). We designed on this work through the system by incorporating with Java API for on-line auditory feedback. The system is to help beginners provide on-line auditory feedback as a communication medium to immediately notify the results in a pleasant way.

To evaluate this design decision, we conducted a study comparing the conventional method for introductory programming with Python and involving observation and interviews of 32 students as novice programmers. Our result shows that the process of introductory course in each condition was easy for students to understand. However, this research offers several significant advantages from learning perspective of introductory programming. Among these, this research is more inviting and provides better support for motivation to program. We hope that this study will provide concrete evidence that using musical metaphor and on-line auditory feedback for introductory programming can be an effective way to promote intrinsically supported educational activities for students.

## **1.2 Research Question**

There are two research questions being investigated in this study.

RQ1. Can this research assist students develop positive experience towards programming?

RQ2. Can on-line auditory feedback proposed in this research effectively aid students notify the status of program?

## **1.3 Research Goal and Objectives**

The main goal of this research is:

*To help students develop positive attitude towards programming and lower the barriers to programming with the assistance of musical metaphor and on-line auditory feedback.*

The research goal leads the implementation of a set of research objectives supporting the purpose of this study and its formulation. The research objectives are:

- a) To gain empirical evidences of the realities on the students' learning experience associated with this research.
- b) To validate the musical features proposed in this research effectively attract students to programming.

## **2. Literature Review**

### **2.1 Computer Science Education**

#### **2.1.1 Educational Programming Languages**

With respect to education, Wing stated that every educated person in the 21<sup>st</sup> century will know core computer science concepts, known as computational thinking (Wing, 2006). The key point of computational thinking is that information and tasks would be processed more systematically and efficiently on the premise that one has knowledge in programming. This knowledge assists students comprehend how system works. There are numerous ways to equip students with dealing information systematically, teaching programming is appropriate educational approach for thinking about computational thinking and enticing students to design and modify program to adapt their needs (Swan, 1991). We discuss our research on computer programming education as an expanded concept of computational thinking.

There is a rich history of research on educational programming language since Storytelling Alice. It was one of the first applications that demonstrated the potential for the storytelling-style of programming and studies for novice programmers have motivated learners to acquire programming skills (Kelleher, Pausch & Kiesler, 2007). Alice helps students learn programming by constructing 3D virtual stories. To encourage young people, the study focused on inviting beginners to create and share stories. As computers play an

important role in our everyday lives, familiarity with computer and programming becomes essential and many researches have explored methods to introduce programming. The research with Alice had result that performance and interest in programming depended on previous programming experience (Kelleher, Pausch & Kiesler, 2007). Scratch, the most famous language in computer science education, focused on a graphical programming environment where young people could build interactive characters on game, and art by creating blocks (Resnick, 2009). A lot of experiments have proven the effectiveness and appealing points of Scratch, and many recent languages have also adopted a puzzle piece metaphor incorporating with the concept of connection interlocking visual elements. Today, this methodology of providing a syntax-free programming interface that involves drag and drop of the construct into program leads the trends of teaching programming (Horn, Solovey & Jacob, 2008). Following a long tradition of computer science education, we designed our method for introductory programming using musical metaphor and auditory feedback.

In the case of feedback in learning, most introduced methods for programming education use a familiar feedback of programming results (Lee & Ko, 2011). Based on studies in education, negative feedback discourages learners to proceed on further tasks (see Table 1). As Atlas reports, there was considerable work in the area of self-reported motivation for programming (Atlas, Taggart & Goodell, 2004). This research found that students' awareness to negative feedback have a strong relationship to self-reported performance levels in the course. However, these works failed to track the

status of the program and the notifying errors intuitively, which are significant processes of the basic programming curricular (Wolz, 2009).

**Table 1. *Lahtinen's (2005) checklist in learning programming***

<b>Rank</b>	<b>What kind of issues you feel difficult in learning programming?</b>
<b>1</b>	How to design a program to solve a certain task
<b>2</b>	Dividing functionality into procedures
<b>3</b>	Finding bugs from their own programs

<b>Rank</b>	<b>Which programming concepts have been difficult for you to learn?</b>
<b>1</b>	Recursion
<b>2</b>	Pointers and references
<b>3</b>	Abstract data types

<b>Rank</b>	<b>What kinds of materials have helped you in learning programming?</b>
<b>1</b>	Example programs
<b>2</b>	Interactive visualizations
<b>3</b>	Lecture notes/copies of transparencies

### **2.1.2 Introductory Programming**

Even though everyone uses a computer and interacts with integrated programs, only a very few of them can program their own interactive media. As we live in a society which is a full of interactive objects, familiarity with computers

and programs is becoming significant, and many studies are challenging methods to introduce computer programming (Robins, Rountree & Rountree, 2003). The creation of the program requires challenge for learning about traditional programming language. For experienced programmers who have pursued computer science, it is also challenging for them. Research showed that learning how to program may have a valuable effect on students' achievement, not in problem solving skills, but also in information science education (Clements, 1999). The decades of studies about introductory programming have been diverse, and the methods in which the activities are integrated in with the broader curriculum (Horn, Solovey, Crouser, & Jacob, 2009; Clements, 1999).

There was a previous effort to make programming concepts easier for novice programmers by dedicating some degree of conventional programming language such as BASIC (Kelleher, Pausch & Kiesler, 2007). Today, Python is another good example with many advantages for supporting as an introductory model (Pears et al, 2007). The simple, pseudocode like syntax of Python makes the description of code easier for students. Even though, C, Java, and C++ top the list of the most widely used programming language for both industry and educational area, usages of these languages were considered as traditional views of learning and moving to knowledge about a particular programming language.

Furthermore, increasing focus on initial enthusiasm for introducing programming to children has occurred in a worldwide. Most previous studies mentioned the factors about the difficulties of introductory programming.

There were many factors (Pears et al, 2007), such as “First, early programming languages were too difficult to use. Many children had difficulty mastering the syntax of programming languages. Second, programming was often introduced with activities that were not connected to children’s interests or experiences. Third, children did not have access to a literature of interesting computer programs. Even though young writers are often inspired by reading great works of literature, there was no analogous literature of programming projects to inspire new programmers. Fourth, programming was often introduced in contexts where no one had the expertise needed to provide guidance when things went wrong, or encourage deeper explorations when things went right.”

Moreover, research on computing education also follows a long tradition. Considering the key concepts which is essential for introductory programming would be significant in the process of introductory programming (Pears et al, 2007). Schneider argued that there are the ten essential concepts and objectives of an initial programming course in Computer Science (Schneider, 1978). This research have motivated some of these objectives

- The single most important concept in a programming course is the concept of an algorithm.
- The presentation of a computer language should concentrate on semantics and program characteristics not syntax.

Even though most traditional views of learning programming priorities the



structure and syntax of the language itself (Resnick et al, 2009). However, in contrast to prior work, this study is not structured according to the constructs of the particular programming language used.

There are many contributions for how to success in introductory programming (Porter, Guzdial, McDowee & Simon, 2013). Introduction of introductory programming is important to invite students to take a course in computer science, including students in non-programmers. Most of systems for beginners and children were designed to assist in constructing correct programs. For example, programming metaphors support an accessible programming learning experience. Tarkan designed that cooking scenarios were used as programming metaphor and the programs were created pictorial recipes which controlled in a kitchen environment animation (Tarkan et al, 2010). Using virtual animation, children could strengthen cognitive skills such as planning abilities and experience with problem-solving heuristics. To explore ways in which to provide concrete real-world scenarios, a lot of studies focused on iterative design work around programming. Another metaphor was also suggested by Esper which they referred to as Codespells, using the metaphor of wizardry (Esper, Foster & Griswold, 2013). They created a unique novice experience with a new domain, because it could be considered what expert programmers can do is regarded as “magical”. Codespells allowed getting novice programmers immersed in programming area and a positive view of their ability. The use of this approach has been more child-focused activity and several advantages from programming learning perspective. Without translation from program to real-world scenario,

students would be guided to focus on self-directed learning and to allow creating knowledge through exploration.

## **2.2 Music and Programming**

There was a previous attempt done by Dannenberg who drew analogies between programming and music based on Pascal, and did a lecture on teaching programming to musicians by programmatically creating audio (Dannenberg & Dannenberg, 1984). The goal of the lecture was not focused on programming education, but music composition. Most researchers investigated the approach between music and programming on tasks for composing and designed to examine music universalities in the music culture (Wang & Cook, 2004).

On the other hand, studies on audio-based applications have shown that using auditory feedback can develop and rehearse cognition (Baldis, 2001). Most of these studies focused on the constructing cognition through audio-based interfaces such as short-term memory, abstract memory, spatial abstraction, and haptic perception. Based on this approach, Audio Programming Language for blind learner was also introduced to assist novice blind programmers using text-to-speech system (Sánchez & Aguayo, 2005).

In the field of Human Computer Interaction, auditory feedback can assist exceptional cognition to express the way in which internal and external representations and processing weave together in thought (Card, 1999). This approach was to apply a similar idea to develop a programming education

method that attracts novice programmers to write code. Furthermore, this approach focused on using music to facilitate introductory programming concepts for learning process.

Another area that inspired our research was about students' motivation and performance related to the effects of sensitivity (Atlas, Taggart & Goodell, 2004). This literature considered not programming education, but music education. Regarding responses of music students to performance feedback, performance anxiety appears to be related with levels of enjoyment and self-perceptions of ability in such domain. This approaches explored the relationship between music and education related to the enjoyment of the activity and interaction. Moreover, research on music and education showed that familiar component affects learning. Many of educational technologies with music have focused on increasing learner motivation by educational factors to entice learners to explore new activities.

### **2.3 Summary of Literature Review**

The novelty of this approach is that it utilizes meaningful musical analogies on the programming method for basic programming courses and uses on-line auditory feedback for recognizing the status of the program. Novice programmers can code by 'listening' to the behavior of the implemented program. Our research builds on the ideas from related works by designing method for introductory programming using musical metaphor that will cooperate with the similarities between music and programming. To help

engage students, we find several key concepts in programming language syntax and semantics, and translate them into music notation to help beginner programmers learn them with ease and intuition. For validating that our prototype would make it possible for students to introduce programming, we designed semi-structured interviews and surveys from previous research tradition (Fincher, Tenenbergs & Robins, 2011). Previous researches accomplished its own mission to enabling novice programmers to introduce programming in different ways. All of these researches attempted to entice learners with their intrinsic interest about programming. Although there have been applies to bring together music and programming via music composition area, we wanted to formulate the method for introductory programming focusing on basic concept of computer science education. Our approach therefore, has been on exploring new design directions for computational thinking that can support students who are explicitly experienced in programming.

In the case of programming, students are encouraged to depart from daily life and focus on programming syntax in details. We designed examples and a learning support environment, allowing users to learn to program by them. Moreover, we use music notation as an analogy to programming based on the observation that there are similar attributes between the two, and provide users with on-line auditory feedback to immediately notify the status in a pleasant way.

## **3. Methodology**

### **3.1 System**

In this section, the specification for programming and details of implementation about the system is described. First, we describe about programming style related to conventional programming language, Java, which is one of the most famous language in the world. Furthermore, we describe key structures of this methodology, programming environment and user survey for notation what was used in this research.

#### **3.1.1 Programming Style**

We illustrate several key concepts in programming language syntax and semantics of our approach, and music notation for helping beginners learn them with ease and intuition. It is important to recognize that novice programmers would have difficulties with the syntax of any programming language in early stage. This study proposed new concept of programming language style with music notation (see Table 2). Based on our finding between music and programming syntax, we were interested in the following concepts. First, each musical note was assigned to basic variable for programming and we defined as groups of sounds consisting of at least basic scale in notes itself. For this prototype, we were interested in standard pitch in scales. Second, students are allowed to increase their chances of programming

with melody, which is function or method of traditional programming language. We conducted this approach for broadening knowledge of programming from introductory programming to further in traditional way. Third, repetition, loop, and ending signed were assigned to similarities based on our observation for examples. More details might be advantageous in our discussion of the results.

**Table 2. Data types and syntax**

<b>Data type</b>	<b>Description</b>	<b>Example</b>
<b>Notes</b>	A pitched sound itself	Do Re Mi Fa
<b>Measures</b>	Melody segment	First melody
<b>Sign</b>	Musical symbols	Repetition, ending

There is the syntax of Java which is easily we could see when we learn Java in the class. We translated them into music notation to help beginners learn them with ease and intuition.

```
// add
50 = 49 + 1;
16 = 15++;

// for loop
for (int i = 0; i < 100; i++){
    System.out.println("i Value = " + i);
}

// variables
int var;
long temp;

// method
java.lang.Math.sqrt();
```

**Figure 1. The syntax of Java**

The purpose of musical metaphor was to support students to realize how the

program works by comparing it with realistic alternative methods. We formulated simplified programming language syntax to promote a conceptual understanding of programming and lesson the load of learning programming syntax (see Figure 2, Table 3).

Musical metaphor approach supports programming education because it shares various similar aspects between the two, such as reusable signs, control statements, and the combinations of sequential and algorithmic structures. In addition, the simplicity of musical symbols provides an easy and enjoyable learning experience through familiar notations.

```

// 음 이동
솔 = 파 + 1;
미 = 레++;

// 반복 연주
도돌이표(첫번째줄 세번째 마디부터 ~ 세번째줄 두번째 마디까지) {
    oo멜로디 반복;
}

// 악기 선언
피아노, 바이올린, 플룻;

// 악기 연주
피아노.g장조.멜로디재생();

```

**Figure 2.** The concept of proposed approach

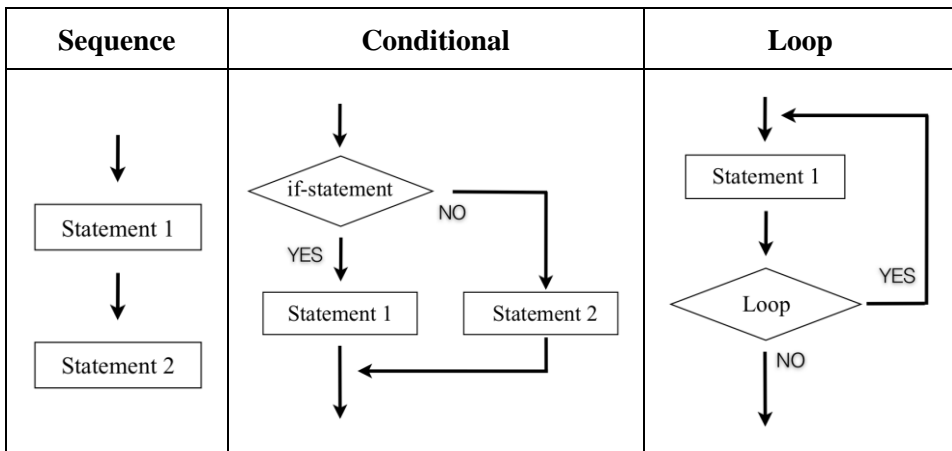
**Table 3.** Comparing between the two of function definitions

Original version	Using melody as method
<pre> int Fibonacci(int n) {     if(n&lt;2)         return n;     else         return             Fibonacci(n-1)         + Fibonacci(n-2); } </pre>	<pre> melody(notes) {     if(Do)         Do;     else         melody(notes-1)         + melody(notes-2); } </pre>

### 3.1.2 Key Structures

This section describes key control structures in the system. It is crucial that the programming code should be written accurately based on precise procedures. We introduced a set of basic algorithms with musical analogies by providing an easy-to-understand methodology (see Table 4). This approach allows learners to concentrate on the achieving the solution to the problems rather than facing difficulties and give up in the early stages (see Table 5).

**Table 4. Basic algorithms**



#### Sequence

**Sequence** is a linear series where one task is performed sequentially after another. Sequential control is indicated by writing one action after another, the actions are performed in the sequence (top to bottom) that they are



written (Leiserson, Rivest & Stein, 2001).

In this research, we portrayed the sequence algorithm as playing several melodies. Each action on a line by itself and all actions aligned sequentially. Sequence is allowed to make a key structure for music composition, and notes and melodies are arranged based on the consecutive order.

### Conditional / Loop structures

Conditional statements, conditional expressions and structures describe various computations and tasks based on Boolean condition, such as true or false, and perform selectively changing the flow by different condition. Loop statement describes program to be repeatedly executed.

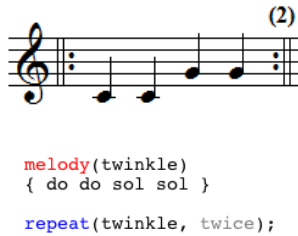
The conditional is considered as the first-and-second ending notation in music and the loop corresponds to its musical analogy as the repetition (see Figure 3, Figure 4). Students can enhance their ability to understand the knowledge of programming through other types of concepts (Esper, Foster & Griswold, 2013).



The image shows a musical staff with a treble clef. The melody consists of several notes: C4, D4, E4, F4, G4, A4, B4, C5. The first ending is marked with a bracket and the number '1.' above it, covering the notes from G4 to C5. The second ending is marked with a bracket and the number '2.' above it, covering the notes from G4 to C5. The staff ends with a double bar line.

```
melody(happy) { do do sol sol }  
melody(first ending) { mi re do }  
melody(second ending) { la la sol }  
  
if(happy once) {  
  first ending;  
} else {  
  second ending;  
}
```

**Figure 3. Simplified programming syntax of conditional structure**



**Figure 4. Simplified programming syntax of loop structure**

### **Arithmetic operations**

Designing approaches for education suggested that instructions should combine both concept of knowledge and strategies for the student in the learning process. To do so, we implemented arithmetic and logic operations using the chords in music.

This strategy would let novice programmers define arithmetic operation functions, which is one of the cognitively complex tasks for novices to acquire correct results. By using this method, students might be able to solve an addition, for example, with on-line auditory feedback. In the traditional method, the process of leading to the correct answer of an arithmetic operation is as follows:


- 1) Defining the method correctly
- 2) Calling in the appropriate position
- 3) Printing out the result

However, this research provides on-line auditory results that correspond with

what the student defined. To make problems easier for novice programmers, the strategy of this research would provide interesting exercises that novices can understand intuitively.

### **Recursion / Solving a certain task**

In the Computer Science curriculum, dealing with program functionality or function definitions is an essential construct in programming concepts. To explore ways to help novice programmers with programming experience and skills, we used musical melodies as method definitions in programming. This could prove to be a useful perception, since the connection between melody and function might support the iterative development of recursion programming experience (see Figure 5, Figure 6).



```
change(note) {  
    note+change(note+1);  
}  
  
change(note) {  
    note+change(note+2);  
}
```

The figure shows a musical staff with a treble clef. The melody consists of seven notes: C4, D4, E4, F4, G4, A4, and B4, all in quarter notes, ascending in pitch. The staff ends with a double bar line.

*Figure 5. Simplified programming syntax of recursive example 1*



```
melody(note) {  
    if(note==do) do;  
    else  
        melody(note-1)+melody(note-2)  
}
```

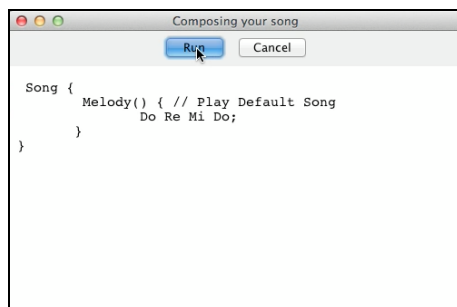
The figure shows a musical staff with a treble clef. The melody consists of seven notes: B4, A4, G4, F4, E4, D4, and C4, all in quarter notes, descending in pitch. The staff ends with a double bar line.

*Figure 6. Simplified programming syntax of recursive example 2*

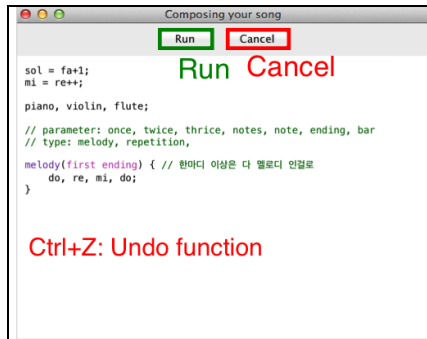
**Table 5. Musical settings as programming metaphors**

<b>Concepts</b>	<b>Musical settings</b>
<b>Arithmetic operation</b>	Using a chord as the addition of notes
<b>Function definition</b>	Using a melody and making a sequence with a variety of predefined melody
<b>Conditional structures</b>	First and second endings for an if-else statement
<b>Loop structures</b>	Repetition for loops
<b>Recursion</b>	Calling a rhythm and melody repeatedly in a sequence
<b>Solving a certain task</b>	Task with sorting example
<b>Understanding the flow of program</b>	Using an on-line auditory feedback for communication

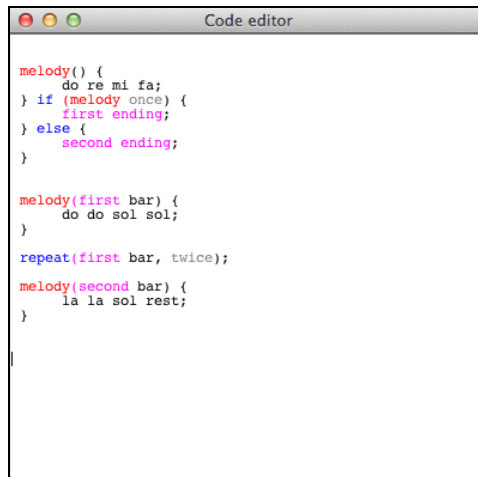
### 3.1.3 Programming Environment



**Figure 7. First prototype of code editor framework**



**Figure 8. Second prototype of code editor framework**



**Figure 9. Third prototype of code editor framework**

To allow for musical metaphor programming in the development environment, we created a prototype of programming language. We were careful to make the simplified programming language to be simply readable by students based on common music notation and the programming environment providing on-line auditory feedback and listening to the results immediately. Moreover, interface should be related by analogy to daily experience and easy to

understand.

### **Java API with audio arguments**

In this research, prototype of programming language provides a code editor as the development environment using JFugue. JFugue is a Java API that allows students to create MIDI files in the Java language. It is an open source programming library for creating and playing music in real-time. To generate music, the programmer defines a series of musical events in their Java application. For example, this is an example of a Java program that plays the C-major scale using JFugue. Programmers, who had previously taken programming courses, who had the ability to write their own computer programs, or who had contributed towards the development of computer programs, could use this library to create an application with a variety of formats. As mentioned previously, however, this research focused on the beginners, and it was safe to assume that they were not aware of this library. We utilize JFugue to implement this method for on-line auditory feedback as a communication medium.

With simple syntax, novice programmers write their program and listen to the results in line by line immediately through the code editor. This makes it possible to notify the programmer about the status of program and aids users in fixing them. We illustrated several key aspects of the code editor we implemented (see Figure 7, Figure 8 and Figure 9). First, unlike previous code editors, this code editor is simple and powerful for students and enables the

creation of codes using music notation. Second, the editor provides an immediate on-line auditory feedback on the status of the program. When the students write the program in the code editor, the string is inserted into the source code, and every line in the source code triggers the system of what we implemented to construct parse tree. By carefully monitoring every change in the code editor, this research approach provides an appropriate on-line auditory feedback of what students define and update for syntax highlighting, which makes it easy and enjoyable for students to use. Finally, we display only the essential functions of the code editor. A simple interface will be less of a burden for beginner programmers in writing their own code.

### **3.2 User Survey**

Before we presented our novel approach to teaching programming using music notation, we reviewed the description by our previous research (Ko & Lee, 2013) – especially the familiarity of music notation. We conducted a survey that asked the level of understanding of music scores and experience in music training. 132 respondents answered that they understood music scores with an average degree of understanding of 3.54 on a scale of 1 to 5 (SD = 1.09). We offered a choice of five scales:

- 0) Not familiar with the notation
- 1) Somewhat unfamiliar
- 2) Undecided

- 3) Somewhat familiar
- 4) Very familiar with the notation

Timestamp	1. 악보를 읽을 수 있습니까?	2. 어떤 종류의 악보를 읽을 수 있습니까? (중복 선택 가능)	3. 서양 음악(5선지) 악보에 대한 이해도를 선택해주세요.	4. 정규 교과과정(초, 중, 고 교 음악교육) 이외의 음악 레슨을 받은 경험이 있습니까?	5. 음악 레슨을 받으신 목적은 무엇이었나요?	응답자 성별	응답자 나이
5/9/2013 0:42:01	네	5선지 악보(서양 음악)		5 10년	전문적인 전공 실기 준비	여	25
5/9/2013 16:48:45	네	5선지 악보(서양 음악)		4 피아노	취미	여	25
5/9/2013 16:49:11	아니오	해당 없음		2 초 피아노 2년	취미	남	30
5/9/2013 16:49:14	네	5선지 악보(서양 음악), 장간보(국악), 코드 악보		5 7년	취미	여	25
5/9/2013 16:50:07	네	5선지 악보(서양 음악)		2 해당 없음	해당 없음	여	26
5/9/2013 16:54:34	네	5선지 악보(서양 음악)		2 1년	취미	남	26
5/9/2013 16:54:37	아니오	해당 없음		1 기타	취미	여	26
5/9/2013 16:54:41		5선지 악보(서양 음악), 코드 악보		4 9년	취미	여	25
5/9/2013 16:58:15	네	5선지 악보(서양 음악)		4 피아노, 바이올린	취미	여	23
5/9/2013 16:59:24	네	5선지 악보(서양 음악)		2		여	만25
5/9/2013 16:59:42	네	코드 악보		5 피아노 9년	취미	여	25
5/9/2013 17:00:37	아니오	해당 없음		3 3년	취미	여	24
5/9/2013 17:01:04	네	5선지 악보(서양 음악)		3 1년	취미	남	27
5/9/2013 17:01:59	네	5선지 악보(서양 음악)		3 2년	취미	여	26
5/9/2013 17:06:13	네	5선지 악보(서양 음악)		4 피아노 4년정도	취미	여	23
5/9/2013 17:07:25	네	5선지 악보(서양 음악)		4	취미	여	22
5/9/2013 17:08:06	네	5선지 악보(서양 음악)		5 5년	취미	여	23
5/9/2013 17:08:23	네	5선지 악보(서양 음악)		4 5년	취미	여	25

**Figure 10. An example of a Google Drive Excel sheet with the responses data from the participants**

We wanted to highlight our own stances under comfortableness with music symbols and indication that music could be a great tool for education and communication medium.



## 4. Experimental Evaluation

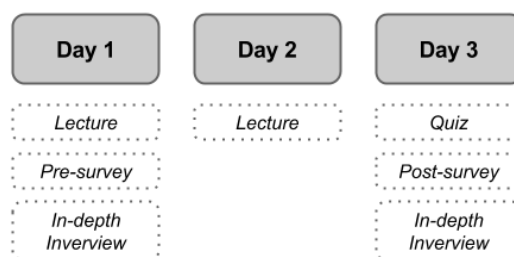
In this section, we described our experimental setting for evaluation. This research conducted two user studies. The first study asked the level of understanding of music scores and experiences in music training. To validate our proposal, we deployed the first study to generalize music notation, which was used in this study could be a great tool for education and communication mediums under comfortableness with music symbols.

Based on the first one, a second study was also conducted. We observed 32 students in two conditions – control and experimental – validating an aim to entice students to program using music notation, as an analogy to programming. By supporting an accessible programming learning experience, this research focused on the first-time programmers' experiences in introductory course. In the control condition, 16 students participated the class for introductory programming using Python. Python was introduced as a programming language itself with the basic concepts of assignment and instruction of reading and writing statements. Python presents many advantages for introductory programming (Pears et al, 2007). The simple syntax like pseudocode of Python allows the beginning students to code easier (Agarwal & Agarwal, 2006). There is a large community that supports Python and many excellent books. Our research followed this tradition of introductory programming to make control condition for user study. In contrast, the experimental condition introduced the methodology of this

research as a gateway to programming language itself with the key concepts in educational programming settings that make use of similar attributes between music notation and programming. 16 students also participated the class for introductory programming using the methodology of this research. To allow for their first experience of programming, researcher focused on peer mentoring. Both of them focused on increasing learner motivation by collaborating various factors to invite learners to participate computational activities (Lahtinen, Ala-Mutka & Järvinen, 2005).

In the rest of this section, we present the process of introductory course in more detail and discuss the preliminary study designed to investigate the hypothesis.

## 4.1 Introductory Course



**Figure 11. Schedule for introductory course**

An introductory programming course should always be taken into account within the structure of a computing curriculum (Kelleher, Pausch & Kiesler,

2007). In tradition, most students who pursue their major in computer science followed a computing curriculum. What are the key aspects in university computing curricula?

Most of introductory computing course embodies a number of assumptions: first, a computing course always makes assumptions regarding computational metaphor and programming paradigm; second, a computing course must focus on the spectrum of possible choices that want to emphasize a particular them throughout the course (Utting et al, 2010). Our intention was to provide an overview of learning about programming: getting broader perspectives on and insight into the students' experience, and being familiarity with the didactic techniques and that have been demonstrated to have a positive impact on learning experience.

One of the most traditional views of introductory programming prioritizes the syntax of the language itself (Resnick et al, 2009). However, some of most influential movements include the component-first approach advocated in. In this research, we put the contents of introductory courses before tradition and designed the process of course specifically for the beginning students. These fall into three steps (see Figure 11):

### **Day 1**

In the introductory course, students were guided through sequence of course with the cognition of basic learning in the introductory programming. During the course, students followed the instruction interacting with researcher, in other words, participated peer mentoring. Quantitative data involving student

behavior were recorded automatically as well as manually by the researcher. The primary activity in the course was to learn how to get a result from the system “hello world”. In experimental condition, we designed default melody for students’ first programming experience. The descriptions, examples, tasks were designed to teach specific aspects of programming. Each of examples focused on obscuring or elucidating the key features in programming domain. In the beginning of the course, we introduced about general concepts about programming, such as the structure and constructs. Using the examples based on curricular, we previewed how we can code and get a result from the code editor. Not only did it explained what action it is taking in each step and heard these changes to the sequence in program, but also students would be allowed to realize the status of system and how it works. In each condition, students achieved the task for introduction, and interactively programmed with the use of a particular language.

The aim of this course was to concentrate on instructional settings for learning about programming. The contents of course were based on the tutorial what we designed for the class and described in the followed section. We implemented the questionnaire for quantitative and qualitative data to get the insights about this methodology. Students were asked about the overall opinion about programming through each condition. In addition, we asked a leading question in the last part, which was “Do you want to continue this introductory programming for gaining more details?”

## **Day 2**

In the second day of the course, students who wanted to get a course continually were guided specifically about the tasks of examples in peer mentoring situation. In both conditions, the tutorial was designed to be easy to access students know how we makes the program with the key structures. A peer mentoring could be a scalable approach to improving control in this tutoring (Lahtinen, Ala-Mutka & Järvinen, 2005). All participants were educated to degree level and came from a variety of programming area including basic concepts using the course materials.

### **Day 3**

Finally, students were guided to participate quick quiz for checking the experience about introductory programming. We examined several students' submissions in detail, to familiarize the programming not only with the process of writing code but reading and understanding that of others.

## **4.2 Tutorials**

In this session, we describe how the programming courses contents were covered with two lectures and five problems sets on each condition. To develop a variety of programming concepts, we then established a set of instructional design principles that can help student deal with each concept.

## Tutorial

### Introduction

초보자가 맨처음 프로그래밍 언어를 접하게 된다면 그것은 영어와 함께 알 수 없는 문법들로 구성되어 있기 때문에 굉장히 어색하게 느낄 수 있다. 아래는 Java라는 프로그래밍 언어로 쓰인 코드다.

```

1 class HelloWorld
2 {
3     public static void main (String [] args)
4     {
5         System.out.println("Hello World!");
6     }
7 }

```

만일 사용자가 위의 프로그램을 실행시킨다면 사용자의 화면에는 "Hello World!" 라는 글씨를 띄우게 될 것이다. 프로그래밍을 접해보지 않은 초보자의 경우 코드를 살펴보면서 유추할 수 있는 모든 것들을 짐작해 보게 될 것이다. { } 이런 괄호들은 뭐지? System.out 이건 뭐고? class HelloWorld 이건 또 뭐야... public static void main(String [] args) 이건???

프로그래밍에 대해 배우는 것은 문제에 대해 논리적으로 생각하고 방법론적으로 접근하는 법을 의미한다. 처음 프로그래밍에 대해 배울 때, Java와 같은 언어를 배우게 된다면, 아무리 간단한 작업을 하고자 할지라도 초보자는 새로운 언어를 배우고 작성하는 법을 배워야 된다.

**Figure 12. Introduction of Tutorial (Study version)**

In the first of the course, we introduced students to the most fundamental constructs of programming, such as statements, boolean expressions, conditions, loops, and functional definition. We first presented each construct in the context of pseudocode based on the traditional methodology. We explained each construct in the context of programs written in each condition, conditional and experimental. With each of examples, students were allowed to ask some questions what make them embarrassed. The tutorial of each condition also included the usage of the environment itself (see Appendix D).

In the first set on Python, students were presented with a challenge of first-programming experience. They were guided:

- 1) Print out the result like "Hello world!"

- 2) Make the program using one condition
- 3) Make the program using one loop
- 4) Define the function using previous example

Throughout we considered assigning several examples and tasks, each focused on one or more concepts, we generally covered the key concepts for introductory programming.

### **4.3 Preliminary Case Study**

Since the scope of this research focused on the introductory programming, we designed two versions of the user study discussing the factor of helping students understand programming concepts with more ease. The control version of the user study used Python, which is simple language as a first language and its potential effect on future learning of more complex languages (Agarwal & Agarwal, 2006); the experimental version used the metaphor of music for programming with on-line auditory feedback based on similar attributes between music score and programming.

We chose to investigate three key aspects of this research that were not examined in most previous programming education. The goal of this research is to examine the role of musical metaphor and on-line auditory feedback on novices' motivation to program. To do this, this study used interview and survey to elicit the parameters held by students in an introductory course.

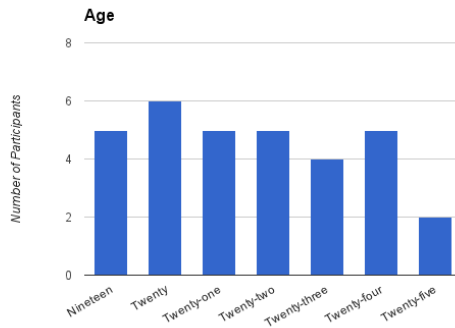
## **4.4 Questionnaire**

The Institutional Review Board of the Seoul National University for Research approved the experimental procedures, and each subject provided written informed consent. We described our questionnaire for preliminary study of this research. The questionnaire was formed based on second topics: expected completeness, comprehensiveness, enjoyment, and overall feedback about the system (see Appendix B).

## **4.5 Participants**

We observed 32 individuals from the undergraduate students at the Ewha Womans University and each participant was randomly assigned one of the two conditions (16 for the control condition and 16 for the experimental condition). Of these, all of the participants were female ranging from 19 to 25 years old ( $M=21.63$ ,  $SD=1.90$ ), and we did in-depth interview with one of participants for collecting intensive observation. Participants who completed tasks in 2 hour session got rewards from researcher. The Institutional Review Board of the Seoul National University for Research approved the experimental procedures, and each subject provided written informed consent (see Appendix C).





**Figure 13. Background information of participants: Age (N=32)**

**Table 6. Background information of participants: Major (N=32)**

	Item	Frequency
College of School	Liberal Arts	14
	Social Sciences	7
	Art and Design	6
	Natural Science	4
	Education	1
<b>Total</b>		<b>32</b>

Students were given a pre-survey and a unique code to receive payment for their submission. The survey was designed to get demographic information (age, gender, major), identify prior music and programming experience, and elicit feedback and attitudes about the methodology. To validate supporting first-time programmers' experience, we conducted a user study with constraint as a novice programmer. We clarified all of the participants have never taken a programming class, part in the development of a computer program. We wanted to interact with the students through think aloud method in observing during the introductory course.

In this study, our null hypothesis was:

H0: There is no difference in influencing a first-time programming experience between the control condition, using traditional, soundless, off-line feedback and the experimental condition, using musical metaphor and on-line auditory feedback.

## **5. Results and Discussion**

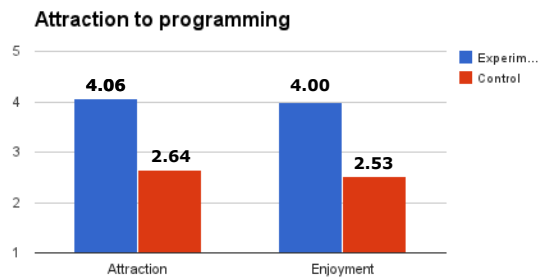
### **5.1 Data Analysis**

Preliminary study was conducted to compare the effectiveness of the method for introductory programming in two groups, control and experimental group. The study was consistent with the observation of participants and analysis of the results based on the data obtained during the course of introductory programming. In this section, we present quantitative and qualitative results that demonstrated measurement about the first-time programming experience. This measures were normally dependent on parametric tests were used for analyses. The level of confidence in this research was set at  $\alpha=0.05$ . Due to the explorative nature of this introductory study, meaningful results were collected from the participants in both conditions.

#### **Difference in the level of attraction to programming**

Based on prior formative evaluation of the introductory programming, we expected this research methodology to be highly inviting. We hoped that the use of familiar objects like musical metaphor in this study would transform an unfamiliar experience into an inviting experience. For comparison, we ran an independent samples t-test for validating this hypothesis. The results, shown in Figure 14, indicated that the form of programming result matters a great deal for inviting. This case rejected the null hypothesis based on the significant difference in the degree of accessibility ( $t(30)=4.87, p<0.0001$ ).

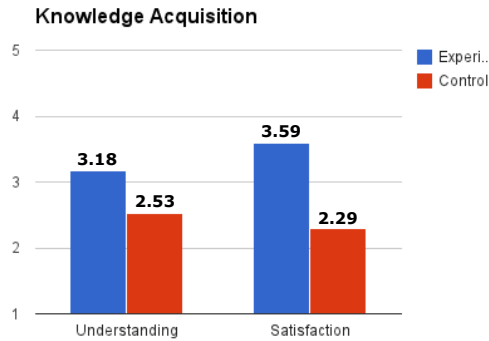
Overall, students were significantly more likely satisfied with the result using on-line auditory feedback (M=4.06, SD=0.90) rather than with a text form (M=2.64, SD=0.79). This was especially true that students preferred to form of auditory feedback. The self-reported average level of enjoyment in the experimental condition was 4.00 (SD=0.94). The self-reported average level of assistance in the control condition was 2.53 (SD=0.72). The difference between two condition was significant ( $t(30)=5.14, p<0.0001$ ). To gain some insight into this effect, during the course, I asked the students' which form of the results was more enjoyable. Ten of the students who answered questionnaire that auditory feedback were more attractive than the conventional way. We want to discuss more about this insight in the discussion section.



**Figure 14. Comparison of control and experimental group rating on self-reported level of assistance and enjoyment**

Similarly, the ratio of participation in the next step was 56.25% (experimental group), while the ratio in another group was 12.5%.

## Difference in the level of knowledge acquisition

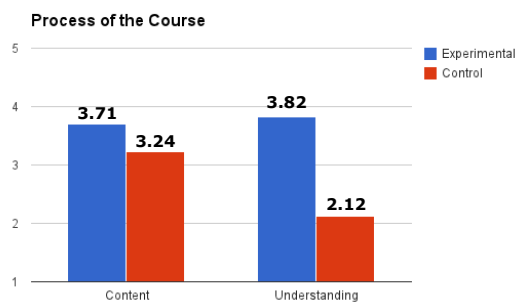


**Figure 15. Comparison of control and experimental group rating on self-reported level of understanding and satisfaction**

As we mentioned in introduction, one of our hypothesis was that the experimental condition would be easier for students to understand than the control condition. This hypothesis was caused by musical metaphor because we felt that students would be familiar with it. Despite early expectations, when participants were asked about “How much do you understand about programming through this class?” the results of the study showed that there was no significant difference between the two conditions ( $t(30)=1.87, p=0.07$ ) (see Figure 15). For the experimental group, 41% of students answered that musical metaphor greatly help them visualize the overall picture of the program. Students had a higher rating on understanding ( $M=3.18, SD=1.13$ ) compared to conditional group ( $M=2.53, SD=0.87$ ). Based on this introductory programming, participants in each group had similar level of understanding. On the question of the measurement of satisfaction of the

system, the overall rating was high with experimental group ( $M=3.59$ ,  $SD=1.28$ ), and different in conditional group ( $M=2.29$ ,  $SD=0.99$ ). In general, participant felt satisfaction in the case of experimental group ( $t(30)=3.31$ ,  $p=0.002$ ). This again supports the observation that even in introductory course focused on intermediate level, understanding the overall picture of the program can affect their satisfaction.

### **No Difference in the process of the course**



**Figure 16. Comparison of control and experimental group rating on self-reported level of content and understanding about the process of the course**

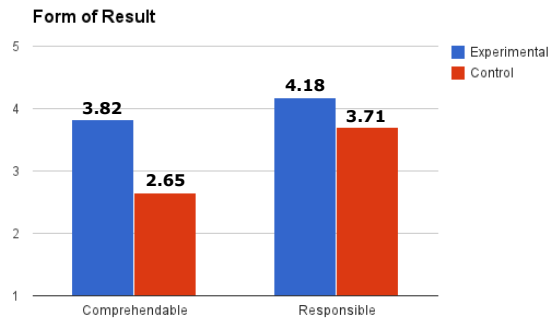
We measured the contentment in terms of the overall session for each group. The average contentment of the course was 3.71 ( $SD=0.92$ ) for the experimental condition and 3.24 ( $SD=0.90$ ) for the control condition. A two sample t-test showed no significant difference between the two means ( $t(30)=1.50$ ,  $p=0.14$ ) (see Figure 16). To put these results into perspective, previous research on contentment in introductory programming showed that

the contents of the course were really important to novice programmer (Pears et al, 2007). Likewise, there was no significant difference in the contentment participants felt about the course. There was, however, a significant difference in participants' reporting that they understood programming through the course ( $t(32)=4.96$ ,  $p<0.0001$ ). Participants in the experimental condition ( $M=3.82$ ,  $SD=1.19$ ) were significantly more likely than those in the control condition ( $M=2.12$ ,  $SD=0.78$ ) to agree to the statement.

### **Reaction-Time**

During periods of observation, quantitative data involving student behavior were obtained automatically as well as manually by the researcher. We measured the time of reaction when students got the result from the program. The minimum time spent recognizing the status of program for the control and experimental condition was 4.84 seconds and 4.08 seconds, respectively. The median overall reaction time for the control and experimental conditions were 4.35 seconds and 4.1 seconds, respectively. There was no significant difference in the length of time participants in either condition programmed tasks overall. We addressed possible explanations for this in the discussion.

### **Difference in the form of result**



**Figure 17. Comparison of control and experimental group rating on self-reported level of response about result form**

To measure the level of understandable, we assigned a level of understandable and responsible from the logs. The distribution of ‘comprehensible’ showed that a number of participants from experimental group (M=3.82, SD=1.19) and conditional group (M=2.65, SD=0.93) checked difference between the two ( $t(30)=3.22$ ,  $p=0.003$ ) (see Figure 17).

The findings demonstrated that musical metaphor and auditory feedback can increase student’s motivation to program. More specifically, music notation as an analogy to programming supported positive effects on students’ acquisition in learning a simple programming language. Overall, on the six measures, this methodology was more inviting, more supportive of being apprehendable, and more effective to learn. One of the goals of this research was to notify people that “Programming is not that hard to learn”. In fact, learning how to program is a sort of learning any other language. We wanted to invite novice



programmers to make their own ideas a reality based on the experience of programming. In this study, we focused on using music to facilitate introducing programming concepts for learning programming.

## **5.2 Survey Feedback**

### **Positive**

The purpose of musical metaphor was to enable students to figure out how the methodology works by relating it to music. Student answered, for example, that they could imagine the overall feature of the program based on the music structure.

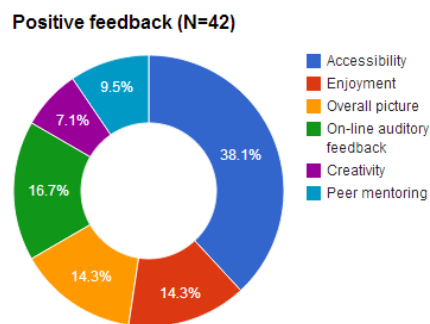
There are the six different issues noticeable in the context of learning, and each issue is explained in Table 7 with quotes from the students: Accessibility, creativity, enjoyment, on-line auditory feedback, overall picture, peer tutoring, and music notation.

**Table 7. Inductive categorization of positive feedback from the student**

Issue	Description	Quotes
<b>Accessibility</b>	Lowering barriers to programming	<p>“I am contented with getting lesson from this approach. I think this approach is helpful for novice programmers who have problem in controlling the level of difficulty during the tutoring.”</p> <p>“I think it affects introductory programming to explore possibility of education methodology. Students who are not familiar with programming can use this approach effectively for entering to programming area.”</p>
<b>Creativity</b>	Relationship between music and programming	<p>“I am interested in the novelty of this research especially the relationship between music and programming because it seems like easy to access to programming.”</p> <p>“I like the creativity of association between music (or instrument) and programming. Students who have interest in programming might consider this approach fresh because they didn't know about the connection between programming and other fields.”</p>
<b>Enjoyment</b>	Playful experience with programming	<p>“I think this introduction is good start for programming. I had fun with music and preferred to jump into programming.”</p> <p>“As a gateway to languages like traditional programming, then, this study appears feasible choice.”</p>
<b>On-line auditory feedback</b>	Helping people notify the status of program	<p>“It is fun to use the system because I can hear what I programmed.”</p> <p>“I like the form of result which is music attracting me a lot.”</p>
<b>Overall picture</b>	Getting the overall picture of the program with the assistance of musical metaphor	<p>“The experience with this introduction helped me establish a general idea of how to think like a programmer.”</p> <p>“I can guess overall image of program through this approach. To novice programmer, it is hard to think about the overall picture and status of program.”</p>
<b>Peer tutoring</b>	Teaching programming with similar status as the students being tutored	<p>“I feel peer tutoring influenced me during the tutoring time. Whenever I wanted to use some computer program, I felt nervous because I don't want to make some problems or something wrong.”</p> <p>“In this study, I think importance of peer tutoring is really essential.”</p>
<b>Music notation</b>	Being familiar with music notation	<p>“In the case of traditional programming language, I have to know about the meaning of English. However in this case, I don't have to translate them.”</p> <p>“This notation can lower the barriers to programming.”</p>

As shown in Figure 18, Accessibility (38.1%) appeared to be the most common issue the students experienced in the introductory programming. On-

line auditory feedback (16.7%) was another issues that were chosen considerably frequently in the context of introductory programming. Enjoyment and Overall picture (14.3%) were other issues, and the issue of peer mentoring and creativity counted for 12.5% and 9.4% respectively.



**Figure 18. Frequency of positive issues about the system**

## Negative

There are also the seven different negative issues noticeable in the context of learning, and each issue is explained in Table 8 with quotes from the students: Error notification, range of the system, weakness, prior background, motivation, unacceptability, and syntax.

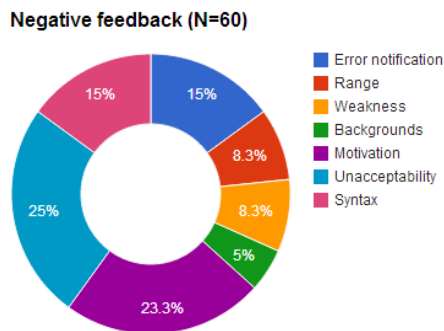
**Table 8. Inductive categorization of negative feedback from the student**

Issue	Description	Quotes
Error notification	Erasing the function of error notification to decrease negative factor of runtime error	<p>“I am afraid about errors. However, even if it gives me negative effects, I think I have to know about the existence of error in program.”</p> <p>“When we learn something, realizing what is problem in this situation is really important.”</p>
Range	Focusing on the introductory programming	<p>“I am not sure this is real programming. I think there is more serious thing beyond this study.”</p> <p>“Novice programmers also should know about the key concepts in programming. There is no excuse for learning only a few things.”</p>
Weakness	Being loaded from the music	<p>“People who don’t like music at all might feel overwhelmed, because they have to know about programming and even music.”</p> <p>“We can meet people quite often someone who can’t read music score.”</p>
Backgrounds	Basic backgrounds about programming	<p>“It is hard to understand what researcher wants to talk about.”</p> <p>“So what? What is programming?”</p>
Motivation	Willing to learn about programming	<p>“There is no much relationship between enjoyment and willingness to learn about programming.”</p> <p>“I think it is totally different between just enjoying and being eager to learn.”</p>
Unacceptability	Barriers to learning about programming	<p>“For me, it is really hard to type something on keyboard.”</p> <p>“It is so complicated.”</p>
Syntax	Barriers to English language for Korean	<p>“During the course, I was overwhelmed by the syntax. I’ve never seen that kinds of things in my life.”</p> <p>“Why do I have to type only in English?”</p>

**Range** The purpose of musical metaphor was to enable beginners to figure out how the system works by relating it to music. However, as we developed this metaphor, some problems arose, including inconsistencies with music structure and complexities that do not exist in authentic music. We plan to improve on our research along with the metaphor of music and engage students such that they can feel positive about their experiences when learning

their further languages.

**Motivation** A lot of students mentioned about the relationship between enjoyment and motivation. They mentioned that there is no much relationship between excitement and willingness to learn about programming. Although students currently had a critical attitude in the field of computer science, Kelleher pointed out that previous experience programming and time spent programming would lead to provide them performance and interest in programming (Kelleher, Pausch & Kiesler, 2007). We could demonstrate this concern with empirical evidences and the importance of supporting collaboration to engage students.



**Figure 19. Frequency of negative issues about the system**

**Error notification** It is true that the experience of programming is characterized by occasions of failure (Fitzgerald et al, 2008). The initial experience with new programming language often leads to unexpected results and unknown error messages from the program. Even though these

forms of feedback are important to supporting a programmer recognize what status is and how program works, the experience might be quite embarrassing and discouraging (Lee & Ko, 2011). Following a tradition of previous studies, this research was also implicated in considering the form of error messages. For most novice programmers, we decided that the system do not notify any kinds of feedback when error occurs in program. Traditional studies have shown that the system scold users for errors affect users' performance negatively (Kelleher, Pausch & Kiesler, 2007). After the user study, we got a lot of feedbacks about the form of feedback including not only on-line auditory feedback, but also nonoccurrence of errors in process. In future work, we will expand the capabilities of the prototype to account for error finding issue.

### **5.3 In-depth Interview**

For the purposes of this study, we defined active interaction as simultaneous active participation, and I measured it by writing down every comment during the course. In particular, we provided the result of in-depth interview from one participant, who interested in learning about programming. Her major is in statistics and she had weak backgrounds in statistical programming with R. She gave clear comments that the approach of this research did indeed excite: "This approach, programming with musical metaphor and on-line auditory feedback, was a full of fun, and chance to think about understanding introductory concepts through exploration. It was really nice having auditory

rewards for the results instead of colorful text those randomly generated by the environments. When I learned about programming with R, the purpose of the lectures was testing statistical hypothesis and drawing the figure from the data. However, in a two hour session, I had never before programmed but I was able use a tutorial to learn how to program in this approach, and then continue to next step for additional programming courses.” This, in itself, would be a success.

## **6. Conclusion**

In this paper, we presented a study for introductory programming with musical metaphor and on-line auditory feedback. We believe the effects of using music notation as an analogy for introductory programming would improve the performance and comprehension of students. Our results provided empirical evidence that thoughtfully designed the methodology for introductory course could offer significant advantages over the introduction of traditional programming language in the context of computer science education. Among these advantages, musical metaphors can be more inviting and more conducive to be introduced programming. Moreover, in this study, the musical metaphors are better at encouraging students to take an active role in exploring and learning, an effect that seems especially who have interests in music.

The results of this research suggest several ways for future work. We want to further implement the methodology to better understand exactly. We are planning to evaluate the methodology and following hypothesis will be investigated: this research can assist students in enabling to identify the program's flow easily in the learning process.

### **6.1 Contributions**

We discuss this research on programming education as an expanded concept of computational thinking. Moreover, the National Research Council



announced that digital fluency as “the ability to reformulate knowledge, to express oneself creatively and appropriately, and to produce and generate information goes beyond traditional notions of computer literacy requires a deeper, more essential understanding and mastery of information technology for information processing, communication, and problem solving than does computer literacy as traditionally defined.” Due to the NRC report, ability associated with programming take an important role in the development of fluency (Resnick et al, 2009).

We conducted a user study comparing the methodology of this research with other traditional programming language as a gateway to learn and introduce programming. This study assessed the methodology and yielded results about which features contributed to differences and statistically better. Especially, we chose to examine two key features of the system that are not found in traditional programming languages: musical metaphor and on-line auditory feedback.

This study has several contributions. First, on the basis of previous work, more key concepts in educational programming settings were improved that make use of similar attributes between music notation and programming. Second, the study described system design incorporating with JAVA API for on-line auditory feedback. Third, the study provided the possible lessons learned from encouraging novice programmers with music, which makes stressful situations become more pleasant and enjoyable.

## **6.2 Limitation and Future Work**

Some researchers in Computer Science, Education, and Human-Computer Interaction (HCI) gave considerable and valuable comments to our previous works. However, they troubled because this research might be different with other programming systems. They concerned that students learning this methodology might be not learning essential computer science concepts and programming skills that would be needed in the future. We provided some rebuttals that considered in the context of computer science education.

First, this research has introduced some valuable ideas that would impact how programming is taught and what kinds of topics are considered. Some of students would start to learn programming and computer science through this research. Second, the purpose of this research was to enable a wide range of students to visualize their ideas and challenge the opportunities of programming. Moreover, it is significant that novice programmers' early experiences are fulfilling ones. The problem is that most programming systems are so complex to learn and use that most students are hard to fulfill their goals. In this study, we focused on dealing with this problem and helping students get familiar with the experience.

## **Acknowledgements**

I would like to gratefully thank Prof. Kyogu Lee for his guidance, understanding, insight, and his leadership during my graduate studies at Music and Audio Research Group (MARG). He encouraged and inspired me to explore my own individuality and self-sufficiency with independence. I am very grateful for the many hours he spent discussing and his untiring feedback in my research.

I would also like to thank my co-advisor, Prof. Joongseek Lee, and the other advisors of Graduate School of Convergence Science and Technology including Prof. Namjun Kang, Joonhwan Lee, and Bongwon Seo for their guidance and support. Prof. Joonhwan Lee was especially generous to suggesting the direction of the user studies and goal of the research. I was extremely lucky to interact with advisors who cared so much about my study, and who gave me valuable advice and support for my study. I would also like to thank all the members of students at MARG for helping me during my graduate studies and also providing for unforgettable memories in my life. In particular, I would like to thank Boyeon Son, Ara Kim, and Yeonhwa Kim for giving me heartfelt consideration, assistance and friendship.

I must express appreciation to my sincere friends, Jimin Choi, Joowon Park, Hyunjin Joo, Julie Kim, and Shia Kim who have helped me in various ways in my life. A special thanks to Jiexin Wang and Jason Koh whose counsel and friendship were essential to make it possible to complete this study.

Finally, and most importantly, I would like to thank my family for their

unending encouragement and support, special thanks to my parents who have given me quiet patience and unwavering love with their faith in me. I would like to express my gratitude to my sister Hyun for her continued encouragement. I am glad you have made many progresses in your life and wish you the best of luck with your work!

## Reference

- Agarwal, K. K., & Agarwal, A. (2006). Simply python for cs0. *Journal of Computing Sciences in Colleges*, 21(4), 162-170.
- Atlas, G. D., Taggart, T., & Goodell, D. J. (2004). The effects of sensitivity to criticism on motivation and performance in music students. *British Journal of Music Education*, 21(01), 81-87.
- Baldis, J. J. (2001). Effects of spatial audio on memory, comprehension, and preference during desktop conferences. In *Proceedings of the SIGCHI conference on Human factors in computing systems* (pp. 166-173). ACM.
- Bramwell-Dicks, A., et al. (2013). *Affective Musical Interaction: Influencing Users' Behaviour and Experiences with Music, Music and Human-Computer Interaction*. Springer London, 67-83.
- Card, S. K., Mackinlay, J. D., & Schneiderman, B. (Eds.). (1999). *Readings in information visualization: using vision to think*. Morgan Kaufmann.
- Carter, L. (2006). Why students with an apparent aptitude for computer science don't choose to major in computer science. In *ACM SIGCSE Bulletin* (Vol. 38, No. 1, pp. 27-31). ACM.
- Clements, D. H. (1999). The future of educational computing research: The case of computer programming. *Information Technology in Childhood Education Annual*, 147-179.
- Dannenberg, F. K., Dannenberg, R. B., & Miller, P. L. (1984). *Teaching Programming to Musicians*.
- Esper, S., Foster, S. R., & Griswold, W. G. (2013). CodeSpells: embodying the metaphor of wizardry for programming. In *Proceedings of the 18th ACM conference on Innovation and technology in computer science education* (pp. 249-254). ACM.
- Fincher, S., Tenenberg, J., & Robins, A. (2011). Research design: necessary bricolage. In *Proceedings of the seventh international workshop on Computing education research* (pp. 27-32). ACM.

- Fitzgerald, S., Lewandowski, G., McCauley, R., Murphy, L., Simon, B., Thomas, L., & Zander, C. (2008). Debugging: finding, fixing and flailing, a multi-institutional study of novice debuggers. *Computer Science Education*, 18(2), 93-116.
- Horn, M. S., Solovey, E. T., Crouser, R. J., & Jacob, R. J. (2009). Comparing the use of tangible and graphical programming languages for informal science education. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 975-984). ACM.
- Horn, M. S., Solovey, E. T., & Jacob, R. J. (2008). Tangible programming and informal science learning: making TUIs work for museums. In *Proceedings of the 7th international conference on Interaction design and children* (pp. 194-201). ACM.
- Huron, D. (2006). *Sweet Anticipation: Music and the Psychology of Expectation*. MIT Press.
- Kelleher, C., Pausch, R., & Kiesler, S. (2007). Storytelling alice motivates middle school girls to learn computer programming. In *Proceedings of the SIGCHI conference on Human factors in computing systems* (pp. 1455-1464). ACM.
- Ko, A. J., Myers, B. A., & Aung, H. H. (2004). Six learning barriers in end-user programming systems. In *Visual Languages and Human Centric Computing, 2004 IEEE Symposium on* (pp. 199-206). IEEE.
- Ko, E., & Lee, K. (2013). *Using Music Notation for Teaching Computer Programming*. ICCE.
- Lahtinen, E., Ala-Mutka, K., & Järvinen, H. M. (2005). A study of the difficulties of novice programmers. In *ACM SIGCSE Bulletin* (Vol. 37, No. 3, pp. 14-18). ACM.
- Leiserson, C. E., Rivest, R. L., & Stein, C. (2001). *Introduction to algorithms*. T. H. Cormen (Ed.). The MIT press.
- Lu, J. J., & Fletcher, G. H. (2009). Thinking about computational thinking. In *ACM SIGCSE Bulletin* (Vol. 41, No. 1, pp. 260-264). ACM.
- McIver, L., & Conway, D. (1996). Seven deadly sins of introductory programming

- language design. In *Software Engineering: Education and Practice. Proceedings. International Conference* (pp. 309-316). IEEE.
- Pears, A., Seidman, S., Malmi, L., Mannila, L., Adams, E., Bennedsen, J., & Paterson, J. (2007). A survey of literature on the teaching of introductory programming. In *ACM SIGCSE Bulletin* (Vol. 39, No. 4, pp. 204-223). ACM.
- Porter, L., Guzdial, M., McDowell, C., & Simon, B. (2013). Success in introductory programming: what works?. *Communications of the ACM*, 56(8), 34-36.
- Resnick, M., Flanagan, M., Kelleher, C., MacLaurin, M., Ohshima, Y., Perlin, K., & Torres, R. (2009). Growing up programming: democratizing the creation of dynamic, interactive media. In *Proceedings of the 27th international conference extended abstracts on Human factors in computing systems* (pp. 3293-3296). ACM.
- Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education*, 13(2), 137-172.
- Sánchez, J., & Aguayo, F. (2005). Blind learners programming through audio. In *CHI'05 extended abstracts on Human factors in computing systems*(pp. 1769-1772). ACM.
- Schneider, G. M. (1978). The introductory programming course in computer science: ten principles. *ACM SIGCSE Bulletin*, 10(1), 107-114.
- Swan, K. (1991). Programming objects to think with: Logo and the teaching and learning of problem solving. *Journal of Educational Computing Research*, 7(1), 89-112.
- Tarkan, S., Sazawal, V., Druin, A., Golub, E., Bonsignore, E. M., Walsh, G., & Atrash, Z. (2010). Toque: designing a cooking-based programming language for and with children. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 2417-2426). ACM.
- Utting, I., Cooper, S., Kölling, M., Maloney, J., & Resnick, M. (2010). Alice, greenfoot, and scratch--a discussion. *ACM Transactions on Computing Education (TOCE)*, 10(4), 17.

- Wang, G., & Cook, P. R. (2004). On-the-fly programming: using code as an expressive musical instrument. In Proceedings of the 2004 conference on New interfaces for musical expression (pp. 138-143). National University of Singapore.
- Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, 49(3), 33-35.
- Wolz, U., Leitner, H. H., Malan, D. J., & Maloney, J. (2009). Starting with scratch in CS 1. In *ACM SIGCSE Bulletin* (Vol. 41, No. 1, pp. 2-3). ACM.



## Appendix A: Online Survey Questionnaire

### 악보 친숙도 조사

안녕하세요. 저는 서울대학교 Music and Audio Research Group 석사과정 고은정입니다.  
악보에 관한 친숙도를 알아보기 위해 간단한 설문조사를 진행하고자 합니다.

여러분들께서 응답해 주시는 모든 내용은 통계적인 목적으로만 집계되어 사용됩니다.  
기타 문의 사항 또는 피드백은 메일로 부탁드립니다.

설문에 참여해주신 모든 분들께 감사드립니다. (고은정: [konzung@gmail.com](mailto:konzung@gmail.com))

응답자 나이

응답자 성별

- 남  
 여

1. 악보를 읽을 수 있습니까?

- 네  
 아니오

2. 어떤 종류의 악보를 읽을 수 있습니까? (중복 선택 가능)

- 해당 없음  
 5선지 악보(서양 음악)  
 정간보(국악)  
 타브 악보  
 Other:

3. 서양 음악(5선지) 악보에 대한 이해도를 선택해주세요.

1 2 3 4 5

전혀 모름      충분히 이해함

4. 정규 교과과정(초,중,고교 음악교육) 이외의 음악 레슨을 받은 경험이 있습니까?

(레슨 받으신 개월 또는 년 수를 적어주세요)

- 해당 없음  
 Other:

5. 음악 레슨을 받으신 목적은 무엇이었나요?

- 해당 없음
- 취미
- 전문적인 전공 실기 준비
- Other:

6. 평소 음악에 관하여 본인의 친숙함의 정도를 자유롭게 선택해 주세요.

1 2 3 4 5

매우 친숙하지 않음      매우 친숙함

7. 평소 즐기는 음악의 장르에 관하여 자유롭게 선택 또는 작성해주세요. (복수 가능)

- 클래식 (Classic)
- 팝 (Pop)
- 재즈 (Jazz)
- 힙합 (Hiphop)
- 국악 (Korean traditional music)
- Other:

Never submit passwords through Google Forms.

Powered by  


This content is neither created nor endorsed by Google.  
[Report Abuse](#) - [Terms of Service](#) - [Additional Terms](#)

## Appendix B: Offline Survey Questionnaire

---

IRB No. 1311/001-003

유효기간: 2014년 11월 6일

### 초보자를 위한 프로그래밍 교육 방안 관련 설문

안녕하세요.

서울대학교 융합과학기술대학원 Music and Audio Research Group에서 초보자를 위한 프로그래밍 교육 방안을 위한 설문을 진행하고 있습니다.

본 설문은 약 15분 정도 소요될 것으로 예상되며, 여러분의 소중한 정보는 본 연구목적 이외에 절대로 사용되지 않을 것을 약속드립니다.

문의 사항은 [konzung@snu.ac.kr](mailto:konzung@snu.ac.kr) 으로 이메일 보내주시면 답변 드리겠습니다. 감사합니다!

#### 기본 정보

Q1. 성별      1. 남성                      2. 여성

Q2. 나이      \_\_\_\_\_

#### 음악에 관한 친숙도

Q3. 귀하는 악기를 배우거나 음악 연주를 처음 접해본 적이 언제인가요?

1. 5세 이전
2. 5-10세
3. 10-15세
4. 15-20세
5. 20세 이후



**Q4.** 귀하는 정규 교과과정(초,중,고교 음악교육)이외의 음악 교육을 받은 경험이 있으신가요?

- 1. 네
- 2. 아니오

**Q5.** 만약 있다면, 어떤 형태의 음악 교육인지 간단히 적어주세요.

---

**Q6.** 서양 음악(5선지) 악보에 대한 이해도를 선택해주세요.

1	2	3	4	5
Terrible			Excellent	

**Q7.** 서양 음악(5선지) 악보를 평소 어느정도 접하시는지 선택해주세요.

1	2	3	4	5
Never			Always	

사전 설문을 모두 완료하셨습니다.  
이제 프로그래밍 학습을 진행하겠습니다.



**Main task 질문**

- 본 실험은 두 명이 한 팀이 되어 진행하는 실험으로, Task가 주어집니다.
- 프로그래밍을 진행하는 과정마다 직접 말로 소리내어 설명해주세요. 예를 들면 타이핑, 예제 활용, 소리 확인 등 머리 속에 떠오르는 생각들을 모두 소리내어 친구와 대화하시면 됩니다.

**연구방법에 관한 평가****[완성도(Completeness)]**

Q1. 해당 프로그래밍 언어를 활용하여 위의 Tasks를 순조롭게 이행하였나요?

1	2	3	4	5
Terrible				Excellent

Q2. 본 연구방법은 처음 학습 시작 단계부터 최종 단계까지의 단계가 적절하였나요?

1	2	3	4	5
Terrible				Excellent

Q3. 해당 프로그래밍 언어는 프로그래밍 결과를 빠르게 제공하였나요?

1	2	3	4	5
Terrible				Excellent



Q4. 프로그래밍을 진행하면서 결과물이 기대에 어느정도 만족하시나요?

1	2	3	4	5
Terrible				Excellent

**[이해도(Comprehensiveness)]**

Q5. 본 연구방법이 제공하는 학습 과정은 이해하기 쉬웠나요?

1	2	3	4	5
Terrible				Excellent

Q6. 본 연구방법의 프로그래밍 결과의 형태는 이해하기 쉬웠나요?

1	2	3	4	5
Terrible				Excellent

Q7. 본 연구방법을 통한 프로그래밍에 관해 어느정도 이해하셨다고  
생각하시나요?

1	2	3	4	5
Terrible				Excellent





Q12. 본 연구방법이 제공하는 학습법을 프로그래밍 학습을 위해 다시 이용하실 의향이 있으신가요?

1. 예
2. 아니오

Q13. 본 연구방법의 장점은 무엇입니까? 자유롭게 적어주십시오

Q14. 본 연구방법의 단점은 무엇입니까? 자유롭게 적어주십시오.

Q15. 프로그래밍을 진행하면서 도움이 되는 요소가 있었다면 자유롭게 적어주십시오.

Q16. 프로그래밍을 진행하면서 혼란을 유발하는 요소가 있었다면 자유롭게 적어주십시오.





## Appendix C: Materials from SNU-IRB

IRB No. 1311/001-003

유효기간: 2014년 11월 6일

### 컴퓨터 프로그래밍 교육 연구 참여자를 모집합니다.

본 연구에서는 초보 프로그래머를 대상으로 친숙하고 비교적 단순한 교육 방법을 제공하여 향후 프로그래밍 언어에 대한 접근성을 높이고 초보 프로그래머의 실력 향상 및 프로그래밍 학습에 긍정적인 연구 방법을 개발하고자 합니다.

- 연구명은 무엇인가요?  
초보자를 위한 프로그래밍 교육 방안 연구
- 누가 참여할 수 있나요?  
선정기준: 만 18 세 이상 65 세 이하이며, 평소 프로그래밍 언어를 학습하는 데 관심과 흥미가 있는 자, 과거 프로그래밍 언어 학습과 관련된 수업을 수강한 적이 없는 자, 프로그래밍 관련 프로젝트에 참여한 경험이 없는 자, 음악을 좋아하고 음악 표기법이 익숙한 자, 독립적으로 설문지를 읽고 작성할 수 있는 자  
제외기준: 한 개 이상의 프로그래밍 언어가 능통한 자, 한글을 읽지 못하고 이해하지 못하는 자, 영어를 읽지 못하고 이해하지 못하는 자, 키보드 타이핑이 불가능한 자, 음악에 부정적 인식이 있는 자
- 본 연구에 참여하게 되면 무엇을 하나요?  
본 연구에 참여하시면, 연구 방법을 통해 개발된 시스템과 교육 방법을 적용한 프로그래밍 교육이 이루어지게 됩니다. 참여자께서는 프로그래밍 언어 학습의 입문 단계에서 다루는 주제들을 일정 기간동안 학습하게 됩니다. 본 연구 참여 후에는 프로그래밍에 대한 기초 지식 학습 효과를 얻을 수 있으며, 프로그래밍 지식의 정도와 교육 효과 또한 평가 받게 됩니다. 학습 과정 이후에는 자가 설문지를 작성하며 설문지의 작성 시간은 약 15 분 정도로 예상됩니다. 학습 기간은 연구 참여 희망자와 추후 조정할 계획입니다.
- 어디서 하나요?  
서울대학교 융합과학기술대학원 D 동
- 참여 혜택은 무엇인가요?  
프로그래밍 언어에 관한 지식과 자가설문 평가 결과에 대한 연구원의 설명  
5000 원 상당의 음료 교환권 증정
- 추가 경비나 비용이 있나요?  
없습니다



IRB No. 1311/001-003

유효기간: 2014년 11월 6일

1 차 모집 기한: 2013 년 11 월 30 일 까지

모집 대상: 20 명

궁금한 사항이 있거나 참여를 원하는 분은 문자나 이메일로 문의하여 주시기 바랍니다.

문의 시 나이, 성별, 전화번호를 기재해주시면 담당연구원이 연락을 드리겠습니다.

서울대학교 디지털정보융합전공

연구 책임자: 이교구 교수

담당 연구원: 고은정

전화: 010-2800-4369

이메일: konzung@snu.ac.kr



## 참여자 동의를 위한 설명서

### 초보자를 위한 프로그래밍 교육 방안 연구

-서울대학교 융합과학부 디지털정보융합전공 음악/오디오 연구그룹-

본 연구는 서울대학교 융합과학부 디지털정보융합전공 음악/오디오 연구그룹에서 초보 프로그래머를 대상으로 프로그래밍 학습 효과를 증진시킬 수 있는 교육 방법을 개발하고, 교육 효과를 측정하고자 하는 연구입니다. 본 연구에는 2013년 12월 31일까지 귀하와 같은 총 32명의 일반인이 참여하게 될 것입니다.

참여자의 선정 및 배제기준은 다음과 같습니다.

선정기준: 만 18세 이상 65세 이하이며, 평소 프로그래밍 언어를 학습하는 데 관심과 흥미가 있으나 경험이 부족한 자, 음악을 좋아하거나 익숙한 자, 독립적으로 실문지를 읽고 작성할 수 있으며 프로그래밍 교육에 참여하기 적합하다고 판단되는 자

제외기준: 한 개 이상의 프로그래밍 언어가 능통한 자, 한글을 읽지 못하고 이해하지 못하는 자, 영어를 읽지 못하고 이해하지 못하는 자, 키보드 타이핑이 불가능한 자, 교육 과정에 참여하기 어렵다고 판단되는 자

참여하는 방법은 본 동의서를 읽고 설명을 들으신 후 자발적 의사로 결정하여 서명하시면 됩니다. 참여하시면 자가 실문지를 작성하고 서면 교육 후 본 연구 방법을 통해 개발된 시스템과 교육 방법을 적용한 프로그래밍 교육이 이루어지게 됩니다. 서면 교육과 실문 작성의 소요시간은 15분 정도로 예상되며, 본 연구 참여 후에는 프로그래밍에 대한 기초 지식 학습 효과를 얻을 수 있으며, 효과의 정도 또한 평가 받게 됩니다.

구체적으로 다음과 같은 평가가 이루어질 것입니다.

- 인구학적 정보 - 나이, 성별
- 연구참여에 관한 정보 - 프로그래밍 경험 유무, 음악 기초지식 및 교육 관련 경험
- 연구방법에 관한 사용자 평가

본 연구에 참여하시면 프로그래밍 언어에 관한 지식과 자가실문 평가 결과에 대한 연구원의 설명 및 상담을 받으실 수 있습니다. 본 연구에서는 프로그래밍 교육 과정에서 기초적인 입문 단계에 해당하는 교육이 진행되게 됩니다. 본 연구를 통해 학습을 통한 초보 프로그래머들의 실력 향상 및 프로그래밍에 관한 호감도를 측정할 예정이고 이를 연구 개선에 반영할 계획입니다. 이로서

2014년 11월 6일

공극적으로 프로그래밍 언어에 대한 일반인들의 접근성을 높이고 친숙하고 단순한 교육 방법을 통해 프로그래밍 학습에 긍정적으로 작용하리라 기대합니다.

개인정보관리 책임자는 서울대학교 융합과학기술대학원 이교구 교수(031-888-9150)입니다. 개인의 사적인 정보는 보호됩니다. 개인의 신상정보는 수집 및 보관되지 않습니다. 하지만 성별, 연령, 자가설문평가 결과 및 프로그래밍 코드 결과는 학술적으로 이용될 수 있습니다. 설문평가 결과와 피험자께서 작성하신 프로그래밍 코드 결과는 안정성 확보를 위해 외부로부터 접근이 제한된 컴퓨터를 통해 기술적/물리적으로 연구 책임자가 정보를 관리합니다. 연구 책임자와 담당자는 개인 정보 보호를 위해 수시로 관리 및 운영하고 있으며, 연구 목적으로 보존된 결과를 취급하는 연구자는 연구팀 이외에는 존재하지 않습니다. 연구팀 이외에는 결과를 열람할 수 없습니다. 또한 생명윤리 및 안전에 관한 법률에 근거하여 연구와 관련된 자료들은 3년간 보관 후 폐기됩니다. 귀하의 정보를 열람할 수 있는 본 연구팀은 귀하의 사적 정보에 대한 기밀성을 유지해야 할 의무가 있으며, 만일 개인정보유출로 인한 문제가 발생한 경우에 본 연구팀에게 책임이 있습니다.

본 연구 참여로 인해 추가 경비나 비용도 발생하지 않습니다. 동의 후에도 언제든지 연구참여에 대한 동의를 철회할 수 있음도 알려드립니다.

본 연구에 대한 문의사항이 있으시면 언제든지 담당 연구자에게 문의하실 수 있습니다. 또한 피험자의 권리와 복지에 관한 문의사항이 있을 경우 연구윤리심의위원회 사무국으로 문의하실 수 있습니다.

담당연구자: 이 교 구 (서울대학교 융합과학부 디지털정보융합전공)  
고 은 정 (서울대학교 융합과학부 디지털정보융합전공), 031-888-9139

서울대 생명윤리 위원회 사무국  
전화: 02-880-5153

www.irs.or.kr

서면 동의서

**초보자를 위한 프로그래밍 교육 방안 연구**

1. 본인은 이 연구에 전적으로 자발적인 의사에 따라 참여합니다.
2. 본인은 연구의 목적과 절차, 본인에게 예상되는 것에 대해 충분한 설명을 들었습니다.
3. 언제든지 자유롭게 연구 참여를 중지할 수 있음을 알고 있습니다.
4. 본인은 자가 설문 평가 결과 및 프로그래밍 코드 결과가 이 연구에 이용됨을 이해하고 있습니다. 본인은 이 정보가 비밀 유지되어 처리된다는 것을 알고 있으며 이를 근거로 하여 이 정보의 활용에 동의합니다. 본인은 이 연구와 관련된 보고서에 본인의 이름이 거명되지 않음을 이해합니다.
5. 본인은 자료 보호에 관한 법률이 규정하는 바의 권리가 있음을 알고 있습니다. 본인은 연구 목적을 위해 본인의 개인적인 자료가 수집, 처리, 발표, 이동되는 것에 대해 동의하였습니다.
6. 자료 보호에 관한 관련 법률로 보호 받는 권리와 상충되지 않는 한, 본인은 이 연구로 얻어진 개인적인 자료나 결과를 이용할 수 있음을 확인합니다.

연구참여자	서명	날짜
연구자	서명	날짜



## Appendix D: Examples of tutorial

### Tutorial for experimental condition

#### Introduction

초보자가 맨처음 프로그래밍 언어를 접하게 된다면 그것은 영어와 함께 알 수 없는 문법들로 구성되어 있기 때문에 굉장히 어색하게 느낄 수 있다. 아래는 Java라는 프로그래밍 언어로 쓰인 코드다.

```
1 class HelloWorld
2 {
3     public static void main (String [] args)
4     {
5         System.out.println("Hello World!");
6     }
7 }
```

만일 사용자가 위의 프로그램을 실행시킨다면 사용자의 화면에는 "Hello World!" 라는 글씨를 띄우게 될 것이다. 프로그래밍을 접해보지 않은 초보자의 경우 코드를 살펴보면서 유추할 수 있는 모든 것들을 짐작해 보게 될 것이다. { } 이런 괄호들은 뭐지? System.out 이건 뭐고? class HelloWorld 이건 또 뭐야... public static void main(String [] args) 이건???

프로그래밍에 대해 배우는 것은 문제에 대해 논리적으로 생각하고 방법론적으로 접근하는 법을 의미한다. 처음 프로그래밍에 대해 배울 때, Java와 같은 언어를 배우게 된다면, 아무리 간단한 작업을 하고자 할지라도 초보자는 새로운 언어를 배우고 작성하는 법을 배워야 된다.

당신이 외국인과 한국어로 대화하게 된다고 가정했을 때 외국인이 한국말을 다소 어눌하게 할 지라도 어느정도 이해할 수 있을 것이다. 그러나 컴퓨터는 한 치의 실수도 용납하지 않는다. 단 한 개의 세미 콜론(:) 실수라도 존재한다면 프로그램은 제대로 동작하지 않는다.

#### About this method

반면, 우리는 꽤 어린 나이 부터 음악과 친숙하게 지내며 악보를 보고 연주하는 것을 즐긴다. 예를 들어, 악보를 통해 같은 멜로디가 반복적으로 진행되고 (특정 기능이 반복적으로 수행되는 프로그램), 악보에서 조건에 따라 다른 마디로 이동하는 (상황에 따라 수행되는 기능이 달라지는 프로그램) 과정들이 진행되게 된다. 또한 음표 하나하나는 (특정 변수) 악보를 구성하는 요소가 된다.

위와 같은 간단한 구조를 배우는 데에도 수수께끼 같은 문법들로 이루어진 프로그래밍 언어는 초보자의 학습에 방해가 된다. 우리는 이 부분을 개선하고자 음악에 초점을 맞춰보고자 한다. 음악을 이용하고 음악에 비유한 프로그래밍 공부 방법은 멜로디를 작성하고, 음계를 이용하여 프로그램과 비슷한 구조의 악보를 만드는 과정을 제공할 수 있다. 악보와 프로그래밍 코드의 유사점을 이용한 프로그래밍 방

법은 프로그래밍을 배우고 싶어하는 예비 또는 초보 프로그래머들에게 유용한 방법이 될 것이다. 자신의 프로그래밍 결과를 소리로 확인하고 악보에 비유하여 프로그램의 기초 개념에 대해 학습하는 과정을 통해 초보자는 프로그래밍 학습에 대한 장벽을 낮출 수 있으며 기존의 프로그래밍 언어와 유사한 구조를 통해 실제 프로그래밍 단계와 연결시킬 수 있는 학습 방법을 제안하고자 한다. 프로그래밍에서 문법은 굉장히 중요한 요소이며 기존 언어와 비슷한 외형을 프로그래밍 학습 단계에서 다룬다면 학습하는 학생들 입장에서 좀 더 문법에 대해 거부감 없이 다가갈 수 있을 것이다. 내용과 구조 두 가지 요소를 모두 고려한 방법에 대해 지금부터 알아보자! 악보에 하나하나 끼워맞추면서 프로그래밍을 진행하다 보면 수수께끼 같던 프로그램을 훨씬 더 쉽게 이해할 수 있을 것이다.

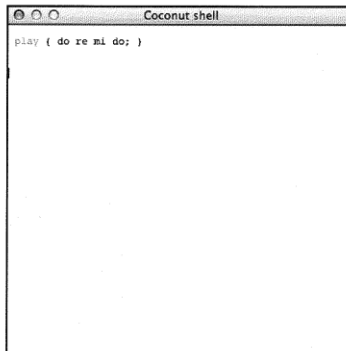


Figure 1. Code editor

이 튜토리얼은 음악에 비유하여 프로그래밍을 진행하는 과정을 학습하는 방법을 위한 튜토리얼이다. 해당 학습에 앞서 사용자는 음악에 대해 친숙함을 느끼는 것으로 가정하며, 일반적으로 프로그래밍을 어떻게 진행할 지에 관한 학습을 진행하게 된다. 이 튜토리얼은 프로그래밍을 이해하기 위한 전반적인 내용을 다루며, 지금부터는 프로그래밍에 대해 알아보자.

### Statements

프로그래밍에서 statement는 프로그래밍이 해야되는 일을 간단히 지시한다. 이것은 언급사항 또는 지시사항이라고 생각할 수 있다. 해당 방법에서 `play { }` 이라고 정의된 구간은 하나의 재생 구간이 된다. 하나의 재생 구간은 다음과 같이 표현된다:

```
play { do re mi do; }
```

또 다른 구간은 다음과 같이 정의된다:

```
melody { do re mi mi; }
```

또한 당신은 특정 상황에서 프로그램을 실행하고 싶을 것이다. 그러한 조건은 Boolean expressions로 정의될 수 있다. Boolean은 음에 관한 계획보다는 맞고 틀리고에 대한 표현 방법이다. 하나의 블록은 다음과 같이 구성되어 있다:

만약 어떤 음이 특정 음보다 작거나 그 조건을 충족시키지 못하면 음악은 재생되지 않는다. (박자 기호: 음표가 모자라거나 악보 구조상 재생이 안되는 식의 구조)

#### Conditions (조건문)

조건문은 결과가 참일때만 실행되는 형태이다. 조건문은 진행하기에 참이다 또는 진행하기에 거짓이다 두가지 상황이 존재한다. 해당 방법에서는, if { }, else if { }, else { } 와 같은 구문들은 조건문을 나타낸다.

예를 들면,

```
melody(happy) { do do sol sol; }
melody(first ending) { mi re do; }
melody(second ending) { la la sol; }

if(happy once) {
  first ending;
} else {
  second ending;
}
```

위의 구조는 if-else 구조 이다. 악보에서 조건에 따라 다른 멜로디가 재생되는 구조이다. happy 멜로디를 정의하고, 아래의 조건문에서 해당 조건에 맞는 마디가 재생되는 형태이다. 위의 코드에서는 조건에 따라 first ending 멜로디 또는 second ending 멜로디가 재생된다. 중첩 또한 가능하다. 멜로디 안에서 다시 그 멜로디를 형성하는 구조이다.

이번에는 반복문에 대해 소개한다.  
Loops (반복문)

프로그래밍에서 반복문은 명령을 통해 여러가지 실행을 반복적으로 수행할 수 있다. 본 연구방법에서는 repeat 이라고 시작하는 명령문이 반복문이 될 수 있다.

예를 들면,

```
repeat { sol mi sol mi; }

melody(twinkle) { do do sol sol; }
repeat(twinkle,twice);
```



이 구조는 조건에 따라 반복문을 수행한다. 또다른 repeat() 블록의 경우에는 반복문을 진행할 횟수를 정해서 수행한다. 또다른 구문인 repeat() 의 경우에는 조건에 따라 반복문을 수행한다. (중중, 당신은 특정 명령문을 여러번 수행하고 싶을 것이다.)

#### Variables (변수)

변수는 특정 값을 저장할 수 있는데, 수학에서 x, y로 표현하는 것과 같은 맥락이다. 본 연구 방법에서는 변수들이 각각 하나의 음으로 존재한다. 변수는 지역변수와 전역변수가 있다. 본 연구방법에서 지역 변수는 하나의 노래 안에서만 사용될 수 있다.

#### Thread ()

프로그래밍에서 쓰레드는 프로그램 안에서의 하나의 미니 프로그램으로, 다른 쓰레드와 동시에 진행될 수 있다. 여러 개의 쓰레드로 구성되어 있는 프로그램은 한 번에 여러가지 일을 할 수 있다. 음악에서는 마치 화음쌓는 것과 같은 구조이다.

#### Function Definition

```
change(note) {
  note + change(note+1);
}
change(note) {
  note + change(note+2);
}
```

#### Recursive Example

```
melody(note) {
  if(note == do) do;
  else
    melody(note-1) + melody(note-2)
}
```

## Tutorial for control condition

### Introduction

초보자가 맨처음 프로그래밍 언어를 접하게 된다면 그것은 영어와 함께 알 수 없는 문법들로 구성되어 있기 때문에 굉장히 어색하게 느낄 수 있다. 아래는 Java라는 프로그래밍 언어로 쓰인 코드다.

```
1 class HelloWorld
2 {
3     public static void main (String [] args)
4     {
5         System.out.println("Hello World!");
6     }
7 }
```

만일 사용자가 위의 프로그램을 실행시킨다면 사용자의 화면에는 "Hello World!" 라는 글씨를 띄우게 될 것이다. 프로그래밍을 접해보지 않은 초보자의 경우 코드를 살펴보면서 유추할 수 있는 모든 것들을 짐작해 보게 될 것이다. { } 이런 괄호들은 뭐지? System.out 이건 뭐고? class HelloWorld 이건 또 뭐야... public static void main(String [] args) 이건???

프로그래밍에 대해 배우는 것은 문제에 대해 논리적으로 생각하고 방법론적으로 접근하는 법을 의미한다. 처음 프로그래밍에 대해 배울 때, 예를 들어 Java와 같은 언어를 배우게 된다면, 아무리 간단한 작업을 하고자 할지라도 초보자는 새로운 언어를 배우고 작성하는 법을 배워야 된다.

당신이 외국인과 한국어로 대화하게 된다고 가정했을 때 외국인이 한국말을 다소 어눌하게 할 지라도 당신은 어느정도 감안하고 이해할 수 있을 것이다. 그러나 컴퓨터는 한 치의 실수도 용납하지 않는다. 단 한 개의 세미콜론(; ) 실수라도 존재한다면 프로그램은 제대로 동작하지 않는다.

### About Python

Python은 1991년 프로그래머인 Guido van Rossum이 발표한 고급 프로그래밍 언어로, Python의 특징은 다음과 같다.

- 코드를 간결하고 가독성이 높게 작성 가능
- C, C++, Java 보다 간단한 문법 구조
- 상호 대화적 프로그래밍 진행 가능
- 초보자부터 전문가까지 폭넓은 사용자층 보유



Figure 1. python 개발 환경

당신은 개발 창의 '>>>' 입력 부분에 직접적으로 코드를 입력할 수 있다. 당신이 코드 조각을 완성할 때마다 각각은 알맞게 실행할 것이다. 예를 들어, 입력:

```
>>> print ("hello world")
```

엔터를 누르면, 아래와 같이 보일 것이다.

```
hello world
```

개발 환경을 다음과 같이 계산기로도 사용할 수 있다:

```
>>> 4 + 4
8
>>> 8 ** 3
512
>>>
```

더하기(+), 빼기(-), 곱하기(\*), 나누기(/), 나머지 연산(%), 제곱(\*\*) 계산 등은 파이썬 언어 자체에 내장되어 있기 때문에 바로 사용할 수 있다. 예를 들면, 당신이 루트 값을 사용하고 싶으면, 바로 다음과 같이 계산할 수 있다.

만약 아래의 명령이 무엇인지 알 수 없으면 미리 걱정하지는 말기!

```
>>> 16 ** 0.5
4.0
>>> import math
>>> math.sqrt(16)
4.0
```

### Statements (코드 작성)

프로그래밍에서, statement는 프로그래밍이 해야되는 일을 간단히 지시한다. 이걸 언급사항 또는 지시 사항이라고 생각할 수 있다. 해당 방법에서, print ( ) 이라고 정의된 구간은 화면에 출력할 코드를 나타낸다. 하나의 출력 구간은 다음과 같이 표현된다:

```
>>> print ("My name is EJ!");
```

또 다른 구간은 다음과 같이 정의된다:

```
>>> def return5():
    print (5)
```

def 을 통해 함수를 정의하고 특정 결과를 출력하는 기능을 수행한다.

가끔, 당신은 특정 상황에서 프로그램을 실행하고 싶을 것이다. 그러한 조건은 Boolean expressions 로 정의될 수 있다. Boolean은 숫자 비교에 대한 맞고 틀리고에 대한 표현이다. 만약, 수치의 비교가 올바르게 이루어지지 않으면, 뒤따르는 코드는 제대로 실행되지 않는다. 하나의 블럭은 다음과 같이 구성되어 있다:

블럭 안의 수치비교를 진행한 후에 해당 비교가 올바르게 이루어지지 않으면 아래의 구문이 실행되지 않는다.

### Conditions (조건문)

```

>>> # Boolean math
>>>
>>> # Examples of if statements
>>> # General format:
>>> # if <condition is True>:
>>> #     <code to execute if condition is True>
>>>
>>> if 9 > 5:
>>>     print ("Yes, 9 greater than 5")
Yes, 9 greater than 5
>>>
>>> if 9 != 5:
>>>     print ("Yes, 9 not equal to 5")
Yes, 9 not equal to 5

```

```

>>> # An example of an if/else statement
>>> # General format:
>>> # if <condition is True>:
>>> #     <code to execute if condition is True>
>>> # else:
>>> #     <code to execute if condition is False>
>>>
>>> if 9 < 5: print ("Yes 9 less than 5")
else: print ("No 9 is not less than 5")
No 9 is not less than 5

```

조건문은 결과가 참일때만 실행되는 형태이다. 조건문은 진행하기에 참이다 또는 진행하기에 거짓이다 두가지 상황이다. 해당 방법에서는, if { } 와 같은 구문들은 조건문을 나타낸다.

Loops (반복문)

```

>>> # For loop examples using range:
>>>
>>> # Range with 1 argument goes from 0 through n-1
>>> for num in range(10):
>>>     print (num)

0
1
2
3
4
5
6
7
8
9
>>> # Range with 2 arguments goes from the first number through the last-1
>>> for num in range(7, 15):
>>>     print (num)

7
8
9
10
11
12
13
14

```

프로그래밍에서, 반복문은 명령을 통해 여러가지 실행을 수행할 수 있다. 파이썬에서는 for 구문으로 시작하는 명령문이 반복문이 될 수 있다. 종종, 당신은 특정 명령문을 여러번 수행하고 싶을 것이다. 이번에는 변수에 대해 알아보자.

#### Variables (변수)

프로그래밍에서, 변수는 특정 값을 저장할 수 있다. 대수에서는 x, y로 표현하는 것과 같은 맥락이 될 수 있다.

변수가 참, 거짓 값을 갖게 된다면 Boolean 변수가 된다. 실행문, Boolean statement, 조건문, 반복문, 변수를 활용해서 불력을 만들고, 이번에는 thread를 위해 한 단계 더 나아가는 프로그래밍 구조를 진행하자.

#### Thread ()

프로그래밍에서, 쓰레드는 프로그램 안에서의 하나의 미니 프로그램이다. 다른 쓰레드와 동시에 진행 될 수 있다. 여러개의 쓰레드로 구성되어 있는 프로그램은, 한 번에 여러가지 일을 할 수 있다.

```

>>> import threading
>>> import datetime
>>> #make thread class
>>> class ThreadClass(threading.Thread):
    def run(self):
        now = datetime.datetime.now()
        print ("%s says Hello World at time: %s\n" %(self.getName(),now))

>>> for i in range(2):
    t = ThreadClass()
    t.start()

Thread-3 says Hello World at time: 2013-11-04 20:53:04.991912
Thread-4 says Hello World at time: 2013-11-04 20:53:04.992292

```

### Recursion Notes

non-recursive example:

```

>>> def it_sum(a_list):
    result = 0
    for x in a_list:
        result += x
    return result

>>> it_sum([1,2,3])
6

```

recursive example:

```

>>> def rec_sum(a_list):
    if a_list == []:
        return 0
    else:
        return a_list[0] + rec_sum(a_list[1:])

>>> rec_sum([1,2,3,4,5])
15

```

계산 과정:

```

rec_sum([1, 2, 3, 4, 5])

= 1 + rec_sum([2, 3, 4, 5])

= 1 + (2 + rec_sum([3, 4, 5]))

= 1 + (2 + (3 + rec_sum([4, 5])))

= 1 + (2 + (3 + 4 + rec_sum([5])))

= 1 + (2 + 3 + 4 + 5 + rec_sum([]))

```

```
= 1 + (2 + 3 + 4 + 5 + 0)
```

```
= 1 + 14
```

```
= 15
```

## Functional Programming

```
>>> # Defining Functions
>>> # def starts a function definition
>>> # names of functions follow variable naming conventions
>>> # functions can take zero or more parameters
>>>
>>> base = 10
>>> exp = 4
>>>
>>> def hello_world():
>>>     base = 20
>>>     print ("inside of helloworld base is", base)
>>>     return "Hello, world!"

>>> print (hello_world())
inside of helloworld base is 20
Hello, world!
>>> print ("outside of helloworld base is", base)
outside of helloworld base is 10
```

```
>>> def add(x,y): return x+y
>>> add(2,3)
5
```

## for exercise (Task)

```
>>> sum = 0;
>>> for i in range(1,15):
>>>     sum = sum+i

>>> print(sum)
105

>>> money=2000
>>> if money >= 3000: print("taxi")
>>> else: print("walking")

walking
```



```
>>> a = "hello"
>>> b = "baby"
>>> print(a+b)
hellobaby
>>>
>>> c = "coconut"
>>> print(c*2)
coconutcoconut
>>>
```

Task 1. 문자열 합치기

Task 2. 문자열 곱하기

Task 3. range 합

Task 4. conditional

```
>>> a,b = 'python', 'coconut'
>>> print(a,b)
python coconut
>>> a,b = b,a
>>> print(a,b)
coconut python
>>>
```

python function definition

def 함수이름(입력값):

<수행문장1>

<수행문장2>

return result

```
>>> def sum(a,b):
        result=a+b
        return result
```

```
>>> sum(1,5)
6
>>>
```