Teaching Agency

Subject knowledge requirements for entry into computer science teacher training

Expert group's recommendations

Introduction

To start a postgraduate primary specialist or secondary ITE course specialising in computing or computer science a successful applicant will need to have demonstrated that they already have sufficient subject knowledge to begin the course and the capacity to develop such knowledge where they have identified 'knowledge gaps', before completing the course. The subject knowledge specified below is considered by an expert group, comprising members from the computer and games industry, computer science academics, Naace, ITTE and CAS, to be the minimum necessary for trainees to take full advantage of the training offered and to produce teachers capable of teaching a rigorous course and in the secondary phase leading to a high status qualification in computing.

For a primary specialist entering a post-graduate course this subject knowledge may be demonstrated by the applicant having an 'A' level in Computing/Computer Science or the equivalent - possibly gained through industrial or school-based experience or a subject knowledge enhancement course (SKE) in computing / computer science. Trainees completing an undergraduate course as a primary computing specialist will need to have covered these criteria by the completion of their course. For a secondary specialist this subject knowledge may be demonstrated by the applicant having a degree with a significant content in computing / computer science or the equivalent - possibly gained through industrial or school-based experience or a SKE course in computing/computer science.

These criteria are indicative only and providers are free to develop their own. The criteria[1] below should support institutions in developing the syllabus for a SKE course.

Key concepts

Demonstrate understanding of the key concepts associated with the areas outlined below, at the level appropriate for entry to the relevant teacher training course:

1. **Language, Machines & Computation** - languages, algorithms, machines and computational models
2. **Data and representation** - data representation, data storage, data transmission, data structures, digital and analogue conversion

[1] The work is based on: CAS (2012), Computer Science: A Curriculum For Schools, Computing At Schools Working Group, available at www.computingatschools.org.uk ; discussions of the Computer Science Subject Expert Group, 3 July 2012 (convened by the Teaching Agency) and at Microsoft Victoria, 2 August; and discussions at The Association for Information Technology in Teacher Education (ITTE) Research Conference, Oxford, 5-7 July 2012.

3. **Communication & Co-ordination** - input-process-output, communication protocols, networks and the internet
4. **Abstraction and Design** – hardware, software, simulation & modelling, interfaces, categorisation
5. **Wider Context of Computing** - intelligence & consciousness, looking at the natural world in computational terms, creativity & intellectual property, moral & ethical considerations, uses of computing and jobs/career paths

Key processes

Demonstrate understanding of and apply the key processes associated with Computational Thinking, at the level appropriate for entry to the relevant teacher training course:

1. **Abstraction** – modelling (representing real world issues, systems, situations); decomposing (breaking a problem into sub problems, solving sub problems, putting solutions together); generalising & classifying.
2. **Programming** - design and write programs; abstraction mechanisms; debugging, testing and reasoning about programs

Range and content

Demonstrate knowledge and understanding of, and be able to apply, the following, at the level appropriate for entry to the relevant teacher training course:

Range and Content	Algorithms	Programming	Data	Computers & Social Informatics	Communication and the Internet
A student about to embark on a primary teacher training course as a CS specialist should know, understand and be able to:	• Explain that an algorithm is a precise way of solving a problem which can be followed by humans and computers. • Give examples of algorithms met in everyday life. • Explain that computers need more precise instructions than humans and the need for precision to avoid errors. • Explain and show how algorithms can use selection (if), repetition (loops), procedures (sub-algorithms within an algorithm). • Explain the need for accuracy of algorithms. • Distinguish between an algorithm and the	• Code competently in at least two programming languages, which may both be 'visual'; at least one of these must allow the use of programming concepts such as selection, repetition, procedures, variables and relational operators. • Explain and use programming concepts such as selection, repetition, procedures, variables, and relational operators. • Review and assess the quality of code. Find and correct errors in syntax and meaning. • Explain that computers are controlled by sequences of precise instructions known as programs • Explain that computers follow instructions/ blindly; hence the need for care and precision.	• Explain how computers represent all data in binary, with a variety of examples: unsigned integers, text representation (e.g. ASCII), different sound file data/types, and different graphics data/file types. • Explain how the same binary data can be interpreted in different ways e.g. an 8-bit value could be a character or a number. • Explain how the same information can be represented in a computer in a variety of ways e.g. sound as mp3 or MIDI. • Explain that data can have errors, how this might affect results and decisions based on the data and how errors can be	• Explain what a computer is and give examples of devices that include computers. • Explain and describe the key characteristics of basic computer architecture (eg CPU, memory, hard disk, mouse, display etc) . • Explain why there are sometimes different operating systems and application software for the same hardware. • Explain and use common troubleshooting techniques. • Explain Moore's Law and multitasking by computers. • Discuss social and ethical issues raised by the role of computers in the world. • Explain the importance of human-computer interface design	• Explain what the World Wide Web and the Internet are, and the difference. • Outline the key features of the World Wide Web and their relationships– eg browsers, URLs, navigation methods • Outline how data are transported on the Internet, including packets and the notion of a protocol. • Explain the role of search engines and what happens when a user requests a web page in a browser. • Explain the technological perspective on safety and security.

	programs that implements that algorithm	• Represent algorithmic steps in multiple programming languages (e.g. logo, scratch). • Explain how and use programs to simulate environments to test hypothesis. • Explain and show how programs can be planned, tested and corrected and documented. • Explain how HTML constructs the rendering of a web page	reduced. • Explain the need for and content of the Data Protection Act, Computer Misuse Act and Copyright legislation (and other relevant legislation).	• Discuss career paths for those studying Computing.	
In addition to the above, a student about to embark on secondary teacher training as a CS specialist should know, understand and be able to:	• Explain how the choice of an algorithm should be influenced by the data. • Be able to explain and use several key algorithms (e.g. sorting, searching, shortest path). • Explain how algorithms can be improved, validated, tested and corrected.	• Program competently in a least two programming languages, at least one of which must be 'textual'. • Explain and use programming concepts such as selection, repetition, procedures, constants, variables, relational operators, logical operators and functions. • Explain and use truth	• Explain the difference between data and information. • Explain the need for and use of hexadecimal, two's complement, signed integers, and string manipulation. • Explain the need for data compression, and be able to describe simple compression methods.	• Explain the use of logic gates and registers. • Explain Von Neumann architecture. • Explain the fetch-execute cycle. • Explain and use low level instruction sets and assembly code. • Explain what compilers and interpreters are and do and give some examples of when they are used.	• Explain the concepts of: client/server models; MAC addresses, IP addresses and domain names; and cookies. • Explain a 'real protocol' e.g. using telnet to interact with an HTTP server. • Explain routing; redundancy and

	• Explain that a single problem could be solved by more than one algorithm. • Explain and show how different algorithms can have different performance characteristics for the same task. • Successfully apply algorithms in solving GCSE and A level type problems.	tables and Boolean valued variables. • Explain and use two-dimensional arrays (and higher). • Explain and use nested constructs (e.g. a loop that contains a conditional, and vice versa) • Explain the concept of procedures that call procedures;. • Explain how low level languages work and when they are used, being able to give simple examples. • Explain that a program can be written to satisfy requirements and that they should be corrected if they do not meet these. • Successfully apply programming in solving Computing/Computer Science GCSE and A level type problems	• Explain the need for analogue to digital conversions and how this works. • Explain the limitations of using binary representations – eg rounding errors, sampling frequency and fractional numbers. • Explain how structured data can be represented in tables in a relational database, and simple database queries	• Explain the main functions of operating systems.	error correction; encryption and security.

Any enquiries regarding this publication should be sent to us at https://www.education.gov.uk/help/contactus.

This document is also available from our website at www.education.gov.uk.