COMPUTING AT SCHOOL
EDUCATE · ENGAGE · ENCOURAGE
In collaboration with BCS, The Chartered Institute for IT

Naace
Learning in a connected world

Computing in the national curriculum

A guide for secondary teachers

FUNDED BY

Department for Education

Computing in the national curriculum

A guide for secondary teachers

Foreword

Computers are now part of everyday life and, for most of us, technology is essential to our lives, at home and at work. 'Computational thinking' is a skill that all pupils must learn if they are to be ready for the workplace and able to participate effectively in the digital world.

The new national curriculum for computing has been developed to equip young people in England with the foundational skills, knowledge and understanding of computing they will need for the rest of their lives. Through the new programme of study for computing, they will learn how computers and computer systems work, they will design and build programs, they will develop their ideas using technology, and create a range of digital content.

But what does all this mean in practice for secondary schools? How should school leaders be planning for the new curriculum and how can teachers develop the additional skills they will need? What qualifications routes are available to computing students at KS4 and where might they lead? This guide has been written especially for secondary teachers. It aims to demystify precise but perhaps unfamiliar language used in the programme of study. It will enable teachers to get to grips with the new requirements quickly and to build on current practice. It includes help for schools with planning and gives guidance on how best to develop teachers' skills.

The new national curriculum for computing provides schools with an exciting new opportunity to reinvigorate teaching and learning in this important area of the curriculum. We hope this guide will help you on your way.

To find out more about Computing At School, please visit us at www.computingatschool.org.uk/secondary. You will also find an eBook version of this guide there, which can be freely shared with colleagues.

Simon Peyton-Jones
Chairman, Computing At School

Computing at School and NAACE would like to acknowledge the following organisations for their support in the development and publication of this guide:

The Raspberry Pi Foundation
The Department for Education
The University of Hertfordshire Schools of Computer Science and Education
Rising Stars UK Ltd.

Published 2014.

Author: Peter Kemp
Consultants for Computing at School: Mark Dorling, Simon Humphreys, Stephen Hunt, Colin Jackson
Consultants for NAACE: Miles Berry, Amanda Jackson (Havering Education Services)

Text design, typesetting and cover design: Burville-Riley Partnership
Photography: Ron Coello

With thanks to the students and teachers of Yavneh College, Borehamwood and Hertswood Academy, Borehamwood.

British Library Cataloguing in Publication Data.
A CIP record for this book is available from the British Library.

ISBN: 978-1-78339-376-3

Printed by Newnorth Print, Ltd. Bedford.

Contents

Introduction

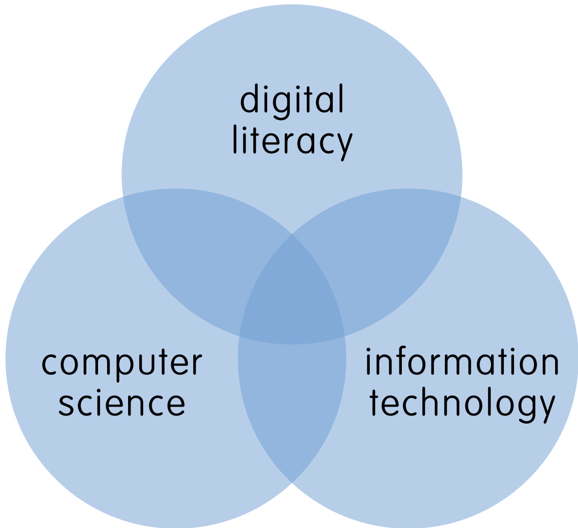
In September 2014, computing is replacing ICT as a national curriculum subject at all key stages. Computing is concerned with how computers and computer systems work, how they are designed and programmed, how to apply **computational thinking**, and how to make best use of information technology. It aims to give pupils a broad education that encourages creativity and equips them with the knowledge and skills to *understand and change the world*.

Computing brings new challenges and opportunities that should excite and empower pupils and teachers. Some of these changes may require you to update existing skills and knowledge. However, there is also a lot of material that should already be familiar.

This guide is intended to support teachers from a broad range of backgrounds. It also aims to provide a starting point for school leadership teams who are looking to understand the best ways to go about delivering computing in the landscape of a changing curriculum, taking into account progression into and out of secondary education.

The three main strands within computing

The Royal Society has identified three distinct strands within computing, each of which is complementary to the others: computer science (CS), information technology (IT) and digital literacy (DL). (See https://royalsociety.org/education/policy/computing-in-schools/report/ for further details.) Each component is essential in preparing pupils to thrive in an increasingly digital world.



Computer science is the scientific and practical study of computation: what can be computed, how to compute it, and how computation may be applied to the solution of problems.

Information technology is concerned with how computers and telecommunications equipment work, and how they may be applied to the storage, retrieval, transmission and manipulation of data.

Digital literacy is the ability to effectively, responsibly, safely and critically navigate, evaluate and create **digital artefacts** using a range of **digital** technologies.

The creation of digital artefacts will be integral to much of the learning of computing. Digital artefacts can take many forms, including digital images, computer **programs**, spreadsheets, 3D animations and this booklet.

The new programme of study

The focus of the new programme of study moves towards **programming** and other aspects of computer science. Programming has been part of the ICT national curriculum for some time but has frequently been overlooked or treated superficially. However, there is more to computer science than programming. Computer science incorporates techniques and methods for solving problems and advancing knowledge, and includes a distinct way of thinking and working that sets it apart from other disciplines. The role of programming in computer science is similar to that of practical work in other sciences – it provides motivation and a context within which ideas are brought to life.

Computational thinking is core to the programme of study. It is the process of *recognising* aspects of computation in the world that surrounds us, and *applying* tools and techniques from computing to understand and reason about both natural and artificial systems and processes. Computational thinking provides a powerful framework for studying computing, with wide application beyond computing itself. It allows pupils to tackle problems, to break them down into solvable chunks and to devise **algorithms** to solve them.

In summary, computational thinking involves:
- **decomposition**
- pattern recognition
- **abstraction**
- pattern generalisation
- algorithm design.

To illustrate these concepts, let's look at how a Snakes and Ladders computer game might be made.

Decomposition is breaking a problem down into its components, each of which can be tackled individually and further decomposed.

> *For example, Snakes and Ladders needs a model that captures a board, the snakes and ladders on the board, player counters, a six-sided dice, and the rules that describe when and how to move counters.*

Pattern recognition is looking for similarities in the behaviours and states of the system you are trying to model.

> *For example, you might specify in the game that every time a counter meets the top of a snake it goes to the bottom of that snake, and to complete a game the player's counter must land on square 100.*

Abstraction helps you only use the detail absolutely necessary for the functioning of the system.

> *For example, Snakes and Ladders might model a snake as two sets of coordinates, ignoring the colour of the snake or the fact that a real snake would need to eat!*

Pattern generalisation allows us to define concepts in their simplest form and to re-use the definition for all instances of that concept.

> *For example, all the snakes in our game could be stored as just two sets of coordinates: one for the top and one for the bottom.*

An algorithm is a precise method for solving a given problem.

> *In this case, the method comprises the steps of rolling a dice, moving a counter, ascending a ladder, changing turns, detecting when the game has finished, etc.*

Algorithms are not just used by computers. For instance, the algorithm for *repair a puncture on your bike* might be: take off the wheel, remove the tyre, remove the inner tube, find the hole, patch it, replace the inner tube, replace the tyre, put the wheel back on.

As teachers, we are competent and confident users of technology in our own personal and professional lives. Few of us have sought to understand how our computers work, or how to program a computer. Many of us are unsure how to teach these things to our pupils.

Now, with help from the web, from new publications and resources, from online communities and from our colleagues (and pupils), it is time to give it a go.

Getting started

There is flexibility in the way you can deliver the computing programme of study but you should plan to cover the whole curriculum – and in many areas to go beyond it.

In the programme of study document, the curriculum for each key stage is expressed as a series of bullet points. The order of these points does not denote their significance, nor should it influence the sequence of your teaching. The amount of time given to any one aspect is up to you. However, it would be unwise to ignore one strand or give too much emphasis to one aspect to the detriment of the others. How you deliver the course content remains in your hands.[1]

The introduction to the programme of study clearly identifies the three strands within computing.

*The core of computing is **computer science**, in which pupils are taught the principles of information and computation, how digital systems work, and how to put this knowledge to use through programming. Building on this knowledge and understanding, pupils are equipped to use **information technology** to create programs, systems and a range of content. Computing also ensures that pupils become **digitally literate** – able to use, and express themselves and develop their ideas through, information and communication technology – at a level suitable for the future workplace and as active participants in a digital world.*[2]

The aims of computing as a whole also reflect the distinction between the three strands.

- *[All pupils] can understand and apply the fundamental principles and concepts of computer science, including abstraction, logic, algorithms and data representation **(CS)***
- *[All pupils] can analyse problems in computational terms, and have repeated practical experience of writing computer programs in order to solve such problems **(CS)***
- *[All pupils] can evaluate and apply information technology, including new or unfamiliar technologies, analytically to solve problems **(IT)***
- *[All pupils] are responsible, competent, confident and creative users of information and communication technology. **(DL)***

The first two of these aims illustrate that computer science has two distinct, but related, aspects. There's a focus on the ideas and principles that underpin computation, and how digital technology works, and this sits alongside the experience of programming – almost certainly the best way for pupils to learn how to apply computer science.

A quick scan of the subject content for KS3 shows expectations for computer science, IT and digital literacy. On the next page, the KS3 content has been adapted opposite to show how it can be viewed in terms of these three strands.

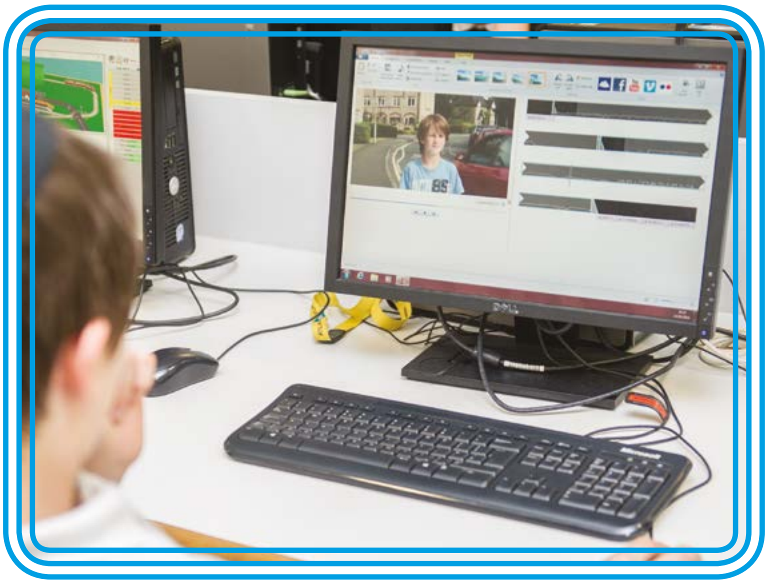
We will look in more detail at the KS3 and KS4 programmes of study later in the booklet.

[1] **All schools maintained by a local authority have a statutory duty to teach the national curriculum programme of study at key stage 3 and key stage 4 as a minimum.**

[2] **www.gov.uk/government/publications/national-curriculum-in-england-computing-programmes-of-study**

	KS3
CS	Design, use and evaluate computational abstractions that model the state and behaviour of real-world problems and physical systems Understand several key algorithms that reflect computational thinking [for example, algorithms for sorting and searching]; use logical reasoning to compare the utility of alternative algorithms for the same problem Use two or more **programming languages**, at least one of which is textual, to solve a variety of computational problems; make appropriate use of **data structures** [for example, **lists**, **tables** or **arrays**]; design and develop modular programs that use procedures or functions Understand simple **Boolean** logic [for example, AND, OR and NOT] and some of its uses in **circuits** and programming; understand how numbers can be represented in **binary**, and be able to carry out simple operations on **binary numbers** [for example, binary addition, and conversion between binary and **decimal**] Understand the **hardware** and **software** components that make up computer systems, and how they communicate with one another and with other systems Understand how instructions are stored and executed within a computer system; understand how data of various types (including text, sounds and pictures) can be represented and manipulated digitally, in the form of binary digits
IT	Undertake creative projects that involve selecting, using, and combining multiple applications, preferably across a range of devices, to achieve challenging goals, including collecting and analysing data and meeting the needs of known users Create, re-use, revise and re-purpose digital artefacts for a given audience, with attention to trustworthiness, design and usability
DL	Understand a range of ways to use technology safely, respectfully, responsibly and securely, including protecting their online identity and privacy; recognise inappropriate content, contact and conduct and know how to report concerns

At first glance, the KS3 subject content appears to be weighted towards the computer science strand. In fact, there is considerable overlap between the three strands, and these labels are just placeholders, based on where we believe the principal emphasis lies. The statutory requirements are intentionally not labelled under any headings in the programme of study. When planning a scheme of work it is best to avoid categorising by strand, for instance, by saying, "Today, we are doing IT". Instead, you should aim to devise activities that include all three strands and cover the content in a balanced, stimulating and creative way.



There are big changes in assessment, as with other national curriculum subjects. The old system of levels has been abolished and is not being replaced. How your school chooses to assess, record and report pupils' mastery of the curriculum content is your decision. We explore some options in the Assessment section of the booklet.

A final thought: every core principle can be taught or illustrated without relying on the use of a specific technology – or indeed any technology at all – and the increasing range of *unplugged* activities should be considered as part of a balanced delivery.

Subject knowledge

The statements in the programme of study are brief and, in many cases, very dense. In this section, we unpack the concepts each statement refers to.

Key stage 3

Design, use and evaluate computational abstractions that model the state and behaviour of real-world problems and physical systems

Modelling is the process of developing a representation of a real-world system or situation that captures the aspects of the situation that are necessary for a particular purpose, while omitting the unimportant. The London Underground map, storyboards for animations and an animation showing the masses and velocities of planets orbiting the sun are all examples of models.

Different purposes need different models. For instance, the well-known (topological) London Underground map is great for route planning, but no use for estimating travel times.

Modelling real-world problems and physical systems can provide engaging projects for pupils. Work that you have done involving spreadsheets in the past may provide some examples of modelling real-world problems.

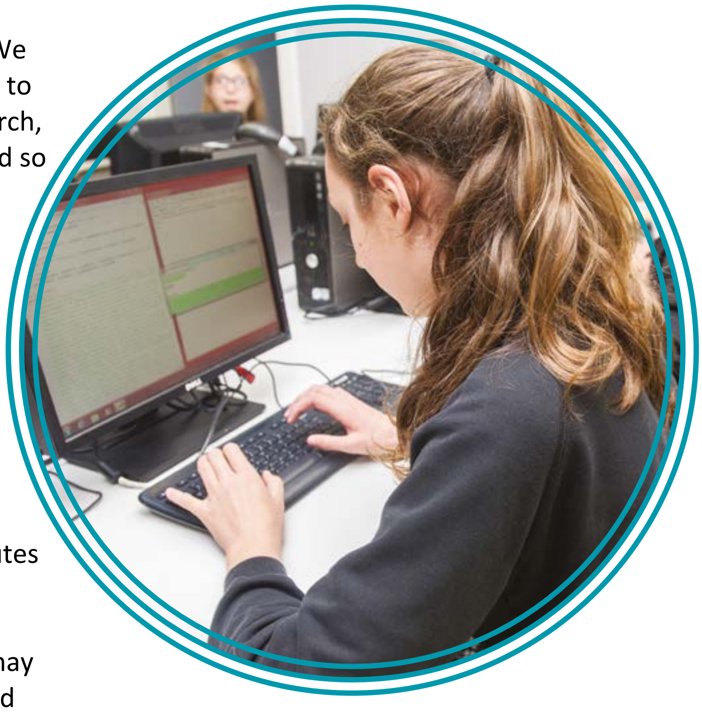
Computational abstractions are models (often pieces of code) that include sufficient information to represent the computational aspects of a situation without describing absolutely everything. You can have several layers of abstraction. For example, you might represent a car as a red rectangle in a model of a road network. You might have a more detailed abstraction in a simple game – adding wheels, a windscreen and the ability to move. An even more detailed abstraction would have fuel consumption, engine noise and suspension. Computational abstraction only includes the detail necessary for the model you are building.

By designing and using computational abstractions, pupils will be able to evaluate them, looking at how closely they match real life and how useful they are in making predictions.

Understand several key algorithms that reflect computational thinking [for example, algorithms for sorting and searching]; use logical reasoning to compare the utility of alternative algorithms for the same problem

An algorithm is a precise method for solving a given problem (for example, a recipe for baking a loaf of bread or instructions for constructing a flat-pack desk). Some algorithms are written to be run on a computer, and some are meant to be carried out by a person. A common misconception is that algorithms, programs and code are all the same thing. This isn't the case. Programs *contain* algorithms. Programs are written in programming languages. Code is a generic term for any set of statements written in a programming language, whether or not they constitute a program. An algorithm can also take the form of steps written in structured English (pseudocode) or it can be expressed as a flowchart.

Algorithms help shape the world around us. We have algorithms to find the best route from A to B, algorithms that rank pages from a web search, algorithms that establish our credit rating, and so on. Algorithms are everywhere!



Often, there are several possible algorithms to solve the same problem, which differ in complexity, efficiency or generality. These may just be different ways of arriving at the same result (for example, different procedures for adding together three-digit numbers). Where the problem is more complex, different algorithms may lead to different solutions (for example, different routes from A to B).

It is important that pupils understand there may be more than one way to solve a problem, and that some problems have more than one 'right' solution. This can be challenging, but it is an essential lesson in life. Pupils should be prepared to evaluate an algorithm using factors such as correctness (in the sense that it solves the problem) and speed, but also the quality of the solutions that it yields.

For example, to find the way out of a maze, one (simple but slow) algorithm might be to walk around at random until you find the exit. Another (more complicated) algorithm would involve remembering where you had been in order to avoid going down the same path twice. Another might be to keep your left hand on the wall and walk until you find the exit. Each algorithm gives a different solution to the same problem (a different route out of the maze). Other qualities might be taken into consideration: maybe one solution gets your feet wet and another gets them muddy.

The programme of study requires that pupils understand several key algorithms, giving searching and sorting as examples. These examples are important because the tasks (searching and sorting) are easy to understand, but there is a particularly rich variety of algorithms for solving them, vividly illustrating the kind of trade-offs discussed above. Pupils can play the part of a machine executing searching and sorting algorithms, and they can translate algorithms into program code.

Searching

Linear and binary searches are the main searches pupils should learn.

A linear search starts at the beginning of a list and goes through every item until it finds the one you are looking for (or you come to the end of the list without finding the item). A real-life example of this might be looking for a picture of your cousin's wedding in a pile of unordered printed photographs. You keep flicking through until you find the photograph, or until you've looked at all the photographs and conclude that the picture isn't there.

A binary search is much quicker, but only works if you have a list of items that have already been sorted in order. You start with the middle item. If you find it first time, well done! Otherwise, the item you are looking for will either be in the top or bottom half of the list. Go to the middle of that half and eliminate the other half of the list. Continue chopping the list in half until you have found the item – or worked out that the item you're looking for is not in the list.

You may find it useful to look at http://csunplugged.org/searching-algorithms, which provides lots of non-computer-based searching activities.

Sorting

There are dozens of competing algorithms for sorting data. It is important that pupils know that there are lots of ways of achieving the same result, and that some methods are more efficient than others.

Giving pupils physical sorting tasks, and asking them to record the algorithms they use, can be a useful way of introducing the subject. There are some helpful examples at http://csunplugged.org/sorting-algorithms, which provides lots of non-computer-based sorting activities. There are also sorting dance videos available, for example, see www.youtube.com/watch?v=lyZQPjUT5B4.

Use two or more programming languages, at least one of which is textual, to solve a variety of computational problems; make appropriate use of data structures [for example, lists, tables or arrays]; design and develop modular programs that use procedures or functions

Computer programs, like algorithms, are comprised of sets of rules or instructions, but they differ in that they need to be written in a precise (or formal) language that a computer can interpret. There are thousands of programming languages – some are appropriate for teaching purposes, some are not. Language choice is discussed in more detail in the Planning section.

The programme of study states that pupils need to use two or more programming languages. Why is this necessary? When pupils can use more than one language they are better able to understand the concepts behind programming. This allows them to easily switch to other languages in the future. Learning to use a programming language can be compared with learning the grammar of a foreign language – once you have learned one, you should more easily pick up another from the same family.

Pupils may have some experience of programming from primary school, probably through using a visual programming language such as Scratch (http://scratch.mit.edu/).

To solve computational problems, a set of basic programming concepts need to be mastered, beginning with sequence, selection and repetition. An example of these in use might be in creating a password log-in system.

- **Sequence**: putting instructions in the right order to make something happen. For example, "Enter username; Enter password; Check details".
- **Selection**: using conditions to control the flow of a program. For example, "IF username = "Sam" and password = "j377y78" THEN display welcome message ELSE display error message".
- **Repetition**: the ability to execute a sequence of instructions many times until a certain condition has been met. For example, "WHILE username incorrect or password incorrect DO ask for username and password".

A **data structure** is a way of storing and organising related data items so that they can be treated as a single, more abstract, item. Structured data is an important tool when solving computational problems. Pupils should be familiar with the idea of a named **variable** from KS2 – a small-sized short-term data store used in a program to store a single value such as *test_score* or *name*. A single name may also be used to refer to an organised collection of simple variables (a data structure), such as an **array**, a **table** or a **list**.

An array is a sequence of data items of fixed length, in which each item is referred to by its position. A list is a sequence of data items whose length can vary over time.

Pupils should be familiar with tables if they have used spreadsheets. Tables allow data to be structured and sorted by user-defined labels. For example, pupils might create a table to record the food that people ordered in a restaurant.

Person's name	Starter	Main	Dessert	Drink
Abby	Olives	Pasta	Chocolate cake	Lemonade
Johannes	Garlic bread	Pizza	Meringue	Sparkling water

Note that the mechanisms for structuring data, and for naming and referencing data structures, differ from language to language.

Modular design is very important when solving complex problems as it allows programmers to decompose problems into manageable chunks (modules) that can be independently tackled and tested. These modules are then combined to make a whole functioning program.

A solution to a problem can be broken down into sub-problems. These can be called procedures, functions, methods or subroutines, depending on what they do and what language they are written in.

For example, in our Snakes and Ladders game we have modules for DISPLAYBOARD and TAKEATURN. These can be further broken down, for instance, into:

- function ROLLDICE, which may return a random number between 1 and 6
- procedure MOVE(2), which may follow the instructions to move the counter 2 squares.

Understand simple Boolean logic [for example, AND, OR and NOT] and some of its uses in circuits and programming

Boolean statements can have one of two values: **TRUE** or **FALSE**. Saying that "6 is greater than 10" is a FALSE statement. Saying that "4 + 4 is equal to 8" is TRUE. Pupils should be aware that they use **Boolean logic** in their everyday lives. For example, the statement "It is raining outside" is either TRUE or FALSE, depending on the weather.

AND, **OR** and **NOT** are given as examples of Boolean operations. I might say:

> "4 is an odd number" AND "5 is an odd number".

The first part of the statement is FALSE, so the overall statement is FALSE. But if I were to change the operator to an OR:

> "4 is an odd number" OR "5 is an odd number",

the overall statement is now TRUE, as we only require one part to be TRUE.

If I add a NOT operator, the truth value will be reversed, so the following is FALSE:

> "NOT (5 is an odd number)".

These operations can be defined using truth tables. Pupils might be asked to come up with their own logical statements and test if they are TRUE or FALSE. You can also combine multiple statements using AND, OR and NOT operations.

Why are these concepts important for pupils to understand? Boolean logic is the means by which computers perform all of their calculations, as computer chips are built out of electronic circuits made up of logic 'gates'. These gates are small pieces of electronics where the output voltage depends on whether the voltage is on or off at the inputs. Their operation can be demonstrated through software simulations or on paper.

Truth values and Boolean logic are fundamental to how computer programs work. When pupils use IF or WHILE statements they will always use truth values, and often need Boolean logic. For example:

> IF Health < 50 AND Character is touching enemy THEN Game Over.

It is useful to encourage pupils to plan their programs and break down the logic involved using pseudocode or flowcharts.

Understand how numbers can be represented in binary, and be able to carry out simple operations on binary numbers [for example, binary addition, and conversion between binary and decimal]

Digital computers store and process all data (such as text, pictures, video) as sequences of ones and zeros, for example 10100101. A single 1/0 value is known as a bit.

A sequence of bits may directly represent a binary (base 2) number. The digits in a binary number have place values much like the digits in a decimal number but, in the case of binary, each place value is a power of two. For example, 1101 in binary is another way of representing the decimal number 13.

Binary
8s 4s 2s 1s
1 1 0 1
1 × 8 = 8
1 × 4 = 4
0 × 2 = 0
1 × 1 = 1
8 + 4 + 0 + 1 = 13

Addition in binary works in the same way as addition in decimal, except that the maximum any column can hold is 1 rather than 9.

Understand the hardware and software components that make up computer systems, and how they communicate with one another and with other systems

Computer systems are made up of hardware and software. Hardware includes the physical components of the computer such as memory, graphics cards, display screens and disk drives. Software involves the computer programs that run on the hardware, including **operating systems** and application programs such as word processors and games.

Computers function as **input**, **process** and **output** systems. Data is input, computation is performed and an output response is given. For example, when you type in your username and password, they are checked against stored values, and the output is either a failure or success message. Pupils should be aware of a range of input and output devices, what they are used for and, at a very basic level, how they provide an interface between the rich, analogue real world and the digital domain of the computer.

Computers are often connected through local **networks** and the biggest network of them all, the **internet**. The internet is a network of networks, where data of all types is exchanged between computers across wired and wireless connections. Connected computers, including smart hardware such as **routers** and **switches**, have numeric IP (internet protocol) addresses, which are used to identify where to send data items and where they have come from. The data items are packaged up into regular-sized chunks known as **packets**, each of which contains information about where it came from and where it is going. A single message may be made up of many packets, which may or may not take the same route through the network.

There are multiple routes between any two computers. This makes the internet very robust. When there is a blocked road, drivers find alternative roads. The internet is the same – if one connection is broken, data packets can usually find alternative routes to their destinations.

The **World Wide Web** is not the internet. It is a collection of linked data files. These are transmitted across the internet and appear as web pages. The language that defines the contents of a web page is called **HTML**. Email and voice conversations are other types of data sent across the internet. Networking can be explored through role-play, with pupils acting out the functions of the internet and how data is sent across it.

Understand how instructions are stored and executed within a computer system; understand how data of various types (including text, sounds and pictures) can be represented and manipulated digitally, in the form of binary digits

For most modern computers to run programs they must first load the necessary data and instructions into their main memory. Getting data and instructions into the main memory usually requires them to be copied from secondary storage devices such as USB memory sticks, DVDs or hard disks. Once in the main memory, the central processing unit (**CPU**) can fetch and execute the instructions, giving outputs where necessary. The process by which this happens is called the fetch-execute cycle.

When we give a computer instructions through a visual or a textual programming language, these instructions are translated by a **compiler** or **interpreter** into a sequence of simpler instructions that can be carried out by the CPU. Most programming languages use English-like statements that are several steps removed from CPU instructions – hence the need for the translation process. **Assembly code**, however, is different. Each assembly code instruction corresponds to one CPU instruction, so it can be used to illustrate how programming language statements are implemented in practice. Using a tool such as the Little Man Computer helps pupils to see this in action (www.yorku.ca/sychen/research/LMC/).

All data and instructions required by the computer are stored as sequences of bits, including text, images and sounds. Text is stored character by character using a system such as **Unicode**. In one version of Unicode the sequence 0100 0001 (65) represents ‘A’, 0100 0010 (66) = ‘B’ and 0100 0011 (67) = ‘C’, and so on. Other variants of Unicode allow a computer to display larger varieties of characters, including non-Latin alphabets. Pupils can send each other secret Unicode messages and try to convert them into text.

Bitmaps may be used to store pictures on a computer. Bitmaps break an image into a grid of squares, known as **pixels**, and use a fixed number of bits to represent the colour of each pixel. The more bits allocated to each pixel, the larger the range of colours, or **colour depth**, but also the larger the file size. The more pixels that make up an image, the higher the **resolution** of that image. Pupils could experiment with creating and manipulating bitmaps using image-editing programs. You could also encourage them to create their own bitmaps using paper or spreadsheets. See, for example, http://community.computingatschool.org.uk/resources/8 (note that you need to log in to the Computing at School community to view this web page). There is a useful spreadsheet task for simulating bitmaps at http://kershaw.org/processing/index.php?js=bitmap.js.

Image formats such as JPEG use compression techniques to make smaller files at the expense of some of the ‘finer detail’ captured in the original bitmap. There is no need for pupils to understand how thisis done.

Sound sampling is used to convert sound waves into a digital data stream. An analogue sound wave is picked up by a microphone and sent to an analogue-to-digital converter. The analogue-to-digital converter takes samples from the sound wave many times a second, storing each sample as a binary (digital) value on the computer.

The more samples taken per second and the more bits used to store each sample, the closer the digital representation of a sound can be to the original waveform. Once in digital form the sound can be manipulated and played back through a digital-to-analogue converter, which changes the binary values back into sound waves. Storage formats such as MP3 take the sampled sound and use compression techniques to make smaller files at the expense of some of the ‘finer detail’ in the original, digitised sample. There is no need for pupils to understand how this is done.

Pupils should have access to a variety of tools that allow them to experiment with different sampling settings and their effect on the quality and size of sound files, as well as manipulating and remixing sound.

Undertake creative projects that involve selecting, using, and combining multiple applications, preferably across a range of devices, to achieve challenging goals, including collecting and analysing data and meeting the needs of known users

Pupils should become confident and discerning users of technology, selecting, using and combining applications. Projects could mix topics from computer science, digital literacy and information technology, allowing pupils to see

how the three areas relate. For example, creating a computer game to teach people about the Norman Invasion could involve **coding**, creating the graphics and sound, making a video advert and producing a promotional poster.

Facilitating creative projects is something most computing teachers will be familiar with, and you will probably have examples that can be re-used. There is no doubt that using real-world problems that require real-world data sets (see, for instance, http://data.gov.uk/data/search and www.bloodhoundssc.com/education) is motivating and challenging for all pupils.

Pupils should have the opportunity to express their creativity. Once you have helped them learn the fundamentals, they should be free to create solutions with the tools they feel are most suitable for the task. Where possible, pupils should set their own challenging goals, creating inventive and original solutions that push their specific abilities. They should evaluate the effectiveness of their solutions in terms of goals and suitability, and reflect on the process they followed, including the software they used.

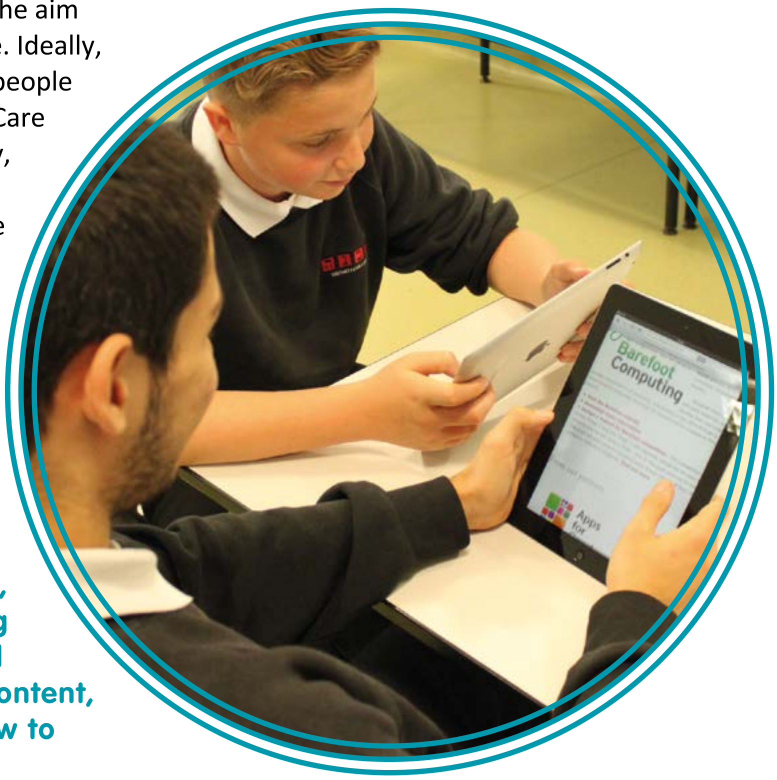
Most pupils will use a range of devices in their everyday lives. The computing programme of study provides a framework to experiment with how these devices can be combined, bringing together cameras, voice recorders, mobile phones, tablet computers, laptops, desktop computers and internet tools through project work.

Create, re-use, revise and re-purpose digital artefacts for a given audience, with attention to trustworthiness, design and usability

All pupils will be using computers to create digital artefacts. The ability to use web search engines effectively is essential in finding and choosing existing artefacts that are available and appropriate – and a great opportunity for pupils to practise their Boolean logic!

You should discuss copyright, including 'all rights reserved' and open licensing. Open licences such as Creative Commons (see http://creativecommons.org/) allow for re-use and re-purposing of digital artefacts as long as the open licence is applied to any modified work. Pupils might want to look at the implications of publishing some of their work online under such a licence, with the potential for global audiences.

Pupils' work should be purposeful, with the aim of meeting the needs of a given audience. Ideally, the audience should be a real person or people with whom the pupil can communicate. Care should be taken over design and usability, including accessibility for people with disabilities and inclusion of people whose first language is not English. Prototyping can be used instead of extensive design documentation. For example, pupils could create a prototype and make changes to it following feedback, to help them move closer to a solution.



Understand a range of ways to use technology safely, respectfully, responsibly and securely, including protecting their online identity and privacy; recognise inappropriate content, contact and conduct and know how to report concerns

People are living more of their lives online. Pupils need to be aware of the dangers that exist on the internet from their own personal conduct, from contact with other people, and from their access to different types of content. They need to be aware of their legal and ethical responsibilities, especially regarding their conduct towards others and their respect of intellectual property rights.

Pupils should be aware of their online identity and take steps to protect it. They should understand what strong passwords are and how they are used to protect their computers and online data from threats. Pupils should also be aware of dangers such as trojans and viruses, and of the increased risk when using pirated software. They should understand the importance of keeping their computers and software up to date, and of using tools such as virus scanners.

Pupils should understand what constitutes safe practice when accessing websites and opening email attachments. They should be familiar with secure websites that use https; this might be a good opportunity to teach basic cryptography such as the Caesar cipher: see http://csunplugged.org/cryptographic-protocols.

A digital footprint is the data that is stored about a person's online activities. It can include information that people have willingly uploaded or that others have recorded about them without their knowledge. All this data may be available to people, such as future employers, and pupils should consider taking steps to limit access to their personal data.

Pupils should be aware that the internet is not an anonymous space; their activities can be tracked through their **IP address** and **browser cookies**. On individual computers, **web browser** settings such as internet history can be cleared.

Dangers such as sexting, grooming and cyberbullying may affect pupils in your school, and parents are often ill-prepared to deal with these threats. Pupils may report concerns about their digital activity to you. Make them aware of the support available through CEOP (www.ceop.police.uk/safety-centre/) and ChildLine. There is further information for you, pupils and parents/carers on the Thinkuknow website (see www.thinkuknow.co.uk/).

The overall aim is that pupils become responsible and resilient users of technology, able to make confident and safe use of the web and of other internet-based services, and able to detect and deal with issues when they arise.

Key stage 4

All pupils must have the opportunity to study aspects of information technology and computer science at sufficient depth to allow them to progress to higher levels of study or to a professional career.

All pupils should be taught to:

- **develop their capability, creativity and knowledge in computer science, digital media and information technology**
- **develop and apply their analytic, problem-solving, design, and computational thinking skills**
- **understand how changes in technology affect safety, including new ways to protect their online privacy and identity, and how to identify and report a range of concerns.**

Schools maintained by local authorities have a statutory duty to provide pupils with the opportunity to study computing at KS4; this can be done through examinable or non-examinable provision. Care must be taken that KS4 provision is planned and relevant to pupils. Where a school is unable to offer any computing qualifications or a pupil chooses not to pursue a computing qualification at KS4, the school still needs to further develop the capability and knowledge pupils acquired in KS1–KS3, either through computing classes or through other subjects.

Ideally, pupils should have the opportunity to specialise in an area of computing such as taking a qualification in information technology, computer science or digital media. This parallels the science curriculum, where pupils have the option to take joint or single sciences. Qualifications might be GCSEs or a more professionally oriented qualification in network management or database administration. If a school is unable to offer a range of qualifications, it could aim to provide the opportunity to study topics from IT, CS and DL.

Computer science is now recognised as a science, along with biology, chemistry and physics, and is therefore part of the English Baccalaureate (EBacc). Several exam boards now offer GCSE qualifications that cover sufficient computer science to count towards the EBacc, though not all of them are called GCSE Computer Science. These GCSEs, like those in other subjects, offer opportunities for pupils with a range of abilities to progress. Schools should aim to help pupils make informed choices, rather than discriminating based on factors such as their KS3 scores in mathematics.

It is important that any qualification offered by a school should be valued by employers and higher education institutions, and should support progression into A-levels, to further and higher education, or to a professional career.

Whether or not your school offers qualifications, it is desirable that pupils' skills in computing are applied and further developed across the curriculum. Examples include creating animations to support geography lessons, learning about online safety in PSHE, and creating programs to simulate science experiments or solve maths problems. Other subjects can provide a rich source of topics and tasks both for those preparing for examinations and for non-examinable provision.

Planning

How can we turn the requirements of the programme of study into engaging lessons?

There are four things to keep in mind.

- The programme of study is a minimum entitlement – there's nothing that imposes any limits on what schools, teachers or pupils cover in computing.
- The programme of study is not a scheme of work – it's up to you, as a school, to determine how you cover this content, in what order, in what contexts and with what resources.
- Schemes of work are not lesson plans – that level of planning comes later, with the ideas for each unit of work getting translated into the detail of specific objectives, resources, activities and assessment.
- There is a far greater focus now on learning about computers and computation, rather than on learning how to use technology.

The computing curriculum brings new teaching opportunities to sit alongside those familiar from the ICT programme of study. There is an increased emphasis on computer science; not only on how to use technology, but on how to make it and how it works. Planning needs to bring together the three strands of computing (CS, IT and DL) and there are many opportunities to have exciting and creative lessons. This section will help you plan for the new curriculum in terms of schemes of work and resourcing.

Discrete or embedded?

Computing is a powerful interdisciplinary subject that has connections to other subjects in the curriculum. The new programme of study is challenging and will require dedicated time in order to deliver it. However, finding cross-curricular links can be of real benefit, especially where they involve pupils applying computational thinking in other subjects. Also, while digital literacy is part of computing, the provision of e-safety education, and the safeguarding of pupils, is an institution-wide issue.

National curriculum or national curriculum 'plus'?

The curriculum is a minimum entitlement – what more can you add to enhance your provision? There are many interesting and exciting ways in which you could extend and build upon the programme of study. For example you could use a school drama production as a source of inspiration. Your pupils could record the movements of the actors on stage and use them to create an animated version of the production. Or they could design a media-rich website for the production, containing images, videos of the play, interviews with the cast and crew, and so on, and implement it using a content management system such as WordPress. Or they could write a program to control the lighting and sound cues. Or create an inventory system for props and scenery. Or design an app that helps actors to rehearse their parts. You could run a robotics course and talk about whether computers can ever be conscious. You could make use of small programmable robots in your teaching, or set up a school robotics club or a school team to enter Robocup. You could get your pupils thinking about Asimov's Three Laws of Robotics, and about the nature of consciousness. And about the plausibility (or otherwise) of claims made about the capabilities of machines. There are so many possibilities!

Themes?

Using CS, IT and DL as themes is limiting. The most interesting problems involve elements of all three. A more engaging approach might be to utilise themes running through the curriculum.

Pupil-centred?

Encouraging pupils to learn what they need to know by working with the tools and creating the digital artefacts that excite them most is a good way to get them engaged. Building a set of differentiated modular projects for pupils to choose from will allow them to match the curriculum to their own interests and to map their progress against any criteria you provide. Alternatively, you could give a half-termly briefing and let pupils define their own learning goals through enquiry-based learning. You can then plan out a unit to meet the needs and interests of the class.

Progression

Pupils will arrive at KS3 with differing knowledge and experiences of computing, so early assessment and intervention may be needed. KS3 is a stepping stone for future qualifications and careers. Planning needs to cover all parts of the curriculum in sufficient depth so that pupils can make informed choices about their future.

As pupils move through KS3, planning should take account of the fact that many computing concepts take time to master. Computational thinking and programming must be practised and key ideas, such as algorithmic problem solving, need to be revisited again and again. For further information about progression, see the Assessment section of this booklet.

Pupils in your classes will have a range of programming and digital creating abilities. However able they might appear, they have a lot more to learn. Can your planning support topics beyond the curriculum, as well as encouraging and supporting pupils beyond school?

Assessment

Your school might have an assessment framework that would influence the structure of a computing course. How will your methods of assessment influence the topics and tasks you set? (Assessment is covered in more detail later in this booklet.)

Timings

As with mathematics or history, schools need to provide weekly computing lessons so that pupils can develop and practise their computational thinking across the whole of KS3.

We recommend you do not try to fit provision into less time for two main reasons. First, pupils need time and regular practice to acquire the skills, knowledge and experiences they require to understand computing and to make informed choices about their future. Second, you will be unable to support pupils who are learning the subject independently outside school.

Schemes of work

There are several options available for creating computing schemes of work (SoWs).

- *Top down*, starting from the programme of study itself.
  The programme of study gives a clear list of the content that should be covered at KS3 (remember, you are free to add to this content). Starting from the programme of study makes it relatively easy to translate the curriculum content into learning outcomes because it's clear what needs to be covered.
- *Bottom up*, starting with ideas for units of work, perhaps project-based.
  Using ideas and themes for units can provide a structure, making it easier to ensure progression and continuity in each year. Whichever themes you select, the topics will need to be revisited in each year of the course. There should be a clear sense of what pupils have already experienced and what subsequent steps in learning are likely to involve.
- *Off the shelf*, using a commercial, free or crowd-sourced scheme, which can be adapted as necessary.
  Whether you choose to plan your own scheme, or to adapt an existing scheme, we recommend that you look at resources and lesson ideas available through other routes first. Use an existing plan as a starting point and then edit it so that it draws on the expertise and enthusiasm of colleagues and works well for your pupils.
- A more *pupil-centred, enquiry-led approach*.
  A scheme of work in this context might just suggest possible projects, resources and a consistent approach to monitoring achievement and curriculum coverage.

Collaborating on documents is made easier with the internet, so there is no need to plan on your own. Joining with a like-minded colleague through a teaching school alliance or a local authority, or in informal groups via Twitter or other social networks, will allow you to draw on others' insights and experience, and your contribution may impact on pupils' learning beyond your own school. The Computing at School community is an ideal place to start – either online or through its network of local hubs: see http://www.computingatschool.org.uk/index.php?id=regions.

Resourcing

Programming languages

Each of the major programming languages used at KS3 has strong arguments to its use as a medium of instruction. Teaching programming should aim to give pupils the underlying skills of sequence, selection, repetition, etc. and not just the ability to use a particular programming language.

Which programming language should I use?

Consider the following when deciding which language to use when teaching programming.

- **How well do I know the language?** You need to support pupils from the least to the most able. Choosing a language you know well or can learn quickly makes you better prepared to handle questions and fix problems, and to signpost extra help for the most able pupils.
- **Are there high-quality resources and a community to support teaching and learning?** Can you effectively teach the curriculum in the way you want to, and seek help if needed? Finding pre-made and differentiated resources will make it easier to support your class and help you plan/adapt great lessons.
- **Can pupils easily access the language at school and at home?** Installing programming languages at school can be problematic. Your network managers may prefer certain products and it is a good idea to discuss your preferred language with them. You will almost certainly have pupils who want to do more programming at home. Is there a free, portable and/or web-based version of the language you are using, so they can develop their interest?

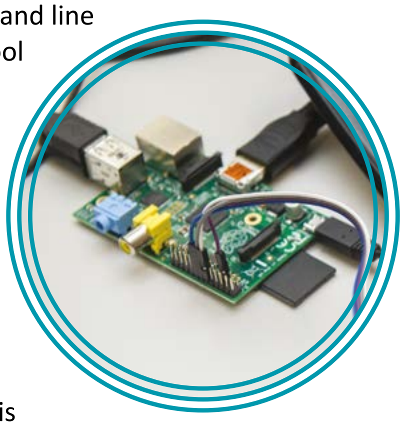
There are a broad range of languages available and you must pick at least one that is textual.

- Visual languages such as Scratch and Kodu are an excellent starting point, but be aware that many pupils may have experience of these from primary school.
- General-purpose textual languages such as Python, Visual Basic.NET and Java allow for a wide range of project work, including graphics, apps and games.
- The formula language of spreadsheets is a textual programming language (albeit a limited one) that provides some insight into functional programming. Most spreadsheets offer a textual scripting language such as VBA or JavaScript to create more complex functionality.
- Several languages (such as Logo) control the behaviour of a turtle or robot while at the same time introducing the idea of a textual language. Many general-purpose languages also offer this functionality.
- Javascript is a fully featured programming language and can be used to teach programming. HTML and CSS are examples of specialised and declarative text-based markup languages that cannot be used to solve computational problems.

Hardware

Most school computer labs can provide the hardware and software necessary to deliver the new curriculum. See www.computingatschool.org.uk/data/uploads/CASInfrastructure.pdf for detailed requirements. Taking apart old computers can aid pupils' understanding of hardware – check with your IT department to see if they have any spare. Be aware of health and safety considerations and ensure you undertake a risk assessment for such activities.

Small computers such as the Raspberry Pi can provide an excellent teaching tool. Pupils can see individual hardware components and the Raspberry Pi offers access to operating system functionality such as the command line that might be restricted in school computer labs. The command line allows pupils to give the computer textual commands and to query details such as network settings. Some basics of networking can be taught using a router, cables and a Pi (see www.ocr.org.uk/qualifications/by-subject/computing/raspberry-pi/ for this exercise and other teaching resources). You should carefully consider how you will incorporate small computers such as the Raspberry Pi into your teaching, and the extra costs involved in connecting a monitor, keyboard, mouse and power supply.



The curriculum suggests that pupils should use a range of devices, which may include digital cameras, online servers and sound recorders (and potentially pupils' own digital devices such as mobile phones and tablets).

Software

There are many IT tools to support delivery of the computing curriculum, such as office and design software suites, software development environments, animation tools, simulators and emulators. Consider their suitability for the course. Will they do what you need them to? Are they easy to use? How will they affect your pupils' chances of gaining qualifications or employment? Will all pupils be able to access them away from school?

Finally, do not forget the unplugged approach. Many of the concepts and principles of computer science can be taught without using any hardware or software at all!

Teaching

Seymour Papert (1928–) is seen by many as the pioneer of computing in schools. He is probably best known as the co-developer of the Logo programming language in the late 1960s.

Logo introduced turtle graphics, in which a computer-controlled robot 'turtle', equipped with a pen, moves, turns and draws to make shapes on paper. Papert saw Logo as more than a programming language; he believed it was a powerful tool for pupils to develop their thinking skills.

> *I began to see how children who had learned to program computers could use very concrete computer models to think about thinking and to learn about learning and in doing so, enhance their powers as psychologists and as epistemologists.*[3]

Inspired by his work with Logo was Papert's theory of learning: constructionism. Put simply, this is the theory that people learn best through making things for other people.

Pupils learn more when they write about a topic than when they read about it, especially if they know that you, and perhaps others, will be reading what they write. It seems likely that this is true of every aspect of computing.

- Pupils will learn computer science more effectively by writing programs and creating theory questions to teach others.
- Pupils will learn to use information technology more effectively if they're doing something creative, such as making a presentation, website or video, especially if this is to be shown to others.
- Pupils will develop a richer digital literacy if they document what they know and learn for others through blog posts, audio recordings or screencasts.

When teaching the computing curriculum, look for practical, creative projects for pupils to work on individually or in groups, ideally bringing together computer science, information technology and digital literacy topics. The projects you set are more likely to be motivating if they're linked to your pupils' own interests. These might allow for cross-curricular work, projects on school life, or interests beyond school.

Also, look for a real audience for pupils' work, whether they're presenting to one another, writing for a public blog, creating software or digital content for younger pupils, or planning to upload their work for others to see, via Scratch or through school GitHub and YouTube accounts (pupils need to be 13 before they can register to use GitHub and YouTube independently).

Project work is sometimes taught in a very formal way, with one particular model of the system life cycle (the waterfall model) being rigidly adhered to. The waterfall model has its strengths, not least of which is the structure it provides to the inexperienced and the less able. However, it also has significant weaknesses, and it does not guarantee success. Experienced developers tend to adopt more flexible approaches, where multiple prototypes are designed, created, tested, evaluated and improved. Introducing this way of working will help prepare pupils for their future professional lives.

Computing without computers

It is tempting to use computers for almost everything in computing lessons. However, providing activities such as role-play and creative writing, and using pencil and paper, can help clarify topics that can later be reinforced through computer use. For example, you could act out a binary search. The pupils can then discuss the algorithm used and try to implement it. CS Unplugged (www.csunplugged.org) and CS4FN (www.cs4fn.org) both have a range of resources for engaging pupils in active non-computer-based tasks.

Teaching programming

For some pupils, the fact that there are often several possible answers to a problem can be daunting. Others aren't used to the 'rapid fail – correct – fail better' model of computer programming. Aim to create a classroom environment of mutual respect, and acceptance that people learn through their mistakes. It is not unusual for professional programmers to spend over 50% of their time locating and fixing mistakes in their programs. This can be very challenging

[3] Papert, S., *Mindstorms: Children, Computers, and Powerful Ideas*, (Basic Books, 1993), page 21.

for the novice, and it is important to teach pupils techniques for locating and correcting the mistakes they have made. This is not the same as testing, which can tell us that the program does not do what we intended, but cannot tell us why.

When pupils begin programming, they often need assistance in **debugging**. This can quickly become chaotic if their default is to immediately ask the teacher for help. One way to alleviate this problem is to implement a 'brain, buddy, book (or internet), then teacher' model, where pupils can only seek help from the teacher once they have exhausted the other routes of support.

There are lots of websites to teach programming, but you might find that making a custom support site or video will help your pupils progress at their own pace. Encourage pupils to show their understanding by explaining their code line by line to one another. This is sometimes called rubber ducking or rubber duck debugging: see www.c2.com/cgi/wiki?RubberDucking.

You will not know (and do not need to know) the answers to all the questions raised by pupils in a computing classroom. It is important that pupils see you using strategies to debug program code, to find answers and to model different possible solutions. As a teacher, you know how to structure and enable learning, and in the computing classroom it is important for you to research ways to support your pupils' learning. One method is to purposely insert mistakes in your code and ask pupils to use the techniques you have taught them to find those mistakes.

Let your pupils explore. Much learning happens through guided exploration. Giving pupils the basic instructions to change the colour of text or create simple graphics will allow them to customise tasks and put their stamp on their work, even when you are only asking for simple functionality such as working out the average of some numbers.

Inclusion

The digital age has seen the web, interactive whiteboards, virtual learning environments, video conferencing, blogs, wikis, podcasts, video and mobile devices have a transformative impact on both learning and teaching. Using technology draws on and enhances pupils' digital skills, and has opened up subject areas previously unavailable to many pupils.

The following section is based on Naace/CAS joint guidance: see http://naacecasjointguidance.wikispaces.com/Terminology.

The digital divide

Pupils in your school probably come from a range of backgrounds, with access to digital technology influenced by social, cultural and economic factors. Computing can be used as a vehicle for social mobility, with those who excel in the subject being in high demand across large parts of the economy. When selecting resources and technologies to deliver computing, take care to ensure that all pupils have the opportunity to study outside the classroom and become independent learners. Ways to provide access include running after-school clubs, having computers and software in libraries, and using licensing agreements or open source software so that resources can be used at home.

Gender

Computing can appeal to pupils of both genders. Take care to counter stereotypes within school (and society in general) that computing is a male-only field. There are many organisations supporting women in technology, for example, see http://casinclude.org.uk/ and www.entrepreneurfirst.org.uk. Highlight the positive contributions of female role models such as Ada Lovelace, Grace Hopper, Jeannette Wing and Dame Wendy Hall. Lesson examples and project topics should be carefully considered to appeal to both genders. Be wary of pursuing activities that appeal to one gender or another – for instance, certain types of computer game.

Assistive technology

Computing can be made accessible to pupils with special educational needs or disabilities

(SEND) through the use of assistive technologies, including hardware and software. Examples include adapted mice or keyboards, Braille displays, screen readers and adjusted system settings for dyslexia. These technologies can also be applied to other subjects, allowing access across the curriculum. When pupils are designing digital artefacts they should consider building in support for SEND users, for example through the use of colour schemes, layouts and support for screen readers.

English as an additional language

Computing offers a range of tools for EAL pupils. Many software products and websites used in the classroom have in-built internationalisation settings to allow pupils to use them in their first language. Machine translation of documents and websites may also be used, although the accuracy of translation cannot be relied upon.

Gifted and talented pupils

Computing is a subject where pupils often voluntarily spend a significant amount of time outside school learning independently. It may be difficult for you to stay ahead of your pupils in all aspects of computing, and pupils may show knowledge and skills beyond that expected by a scheme of work. For example, pupils might be using 3D animation software at home, something not typically covered in lessons.

As a teacher, it is important that you encourage pupils who are displaying exceptional and esoteric skills to share their knowledge with others. You can seek out advice from subject support groups such as CAS and Naace on how to guide them. For example, you might advise a programmer with an interest in maths to look at extended tasks on Project Euler (http://projecteuler.net/), or to teach themselves a functional programming language (see www.haskell.org/haskellwiki/Learn_Haskell_in_10_minutes). Your role in this situation is to structure and facilitate pupil learning, guiding the pupil to relevant material and external support.

Gifted and talented pupils should not be rushed through the curriculum. There is plenty of opportunity to develop depth and enrich their learning of a particular topic. This might include looking at more efficient algorithms to solve a task, or looking at how the task could be solved using different technologies.

Informal learning

There are ample opportunities both locally and online for pupils to learn more about computing. Much of the software and support material for becoming a successful **digital creator** is available free on the web; some commercial software offers complimentary pupil licences. The web hosts multiple communities where pupils and developers share digital artefacts and learning. Examples include YouTube and the curated MIT ScratchEd (http://scratch.mit.edu/educators/). ScratchEd is a support community for educators using Scratch with their pupils. Note that owing to COPPA legislation, pupils need to be 13 to register to use many American-based online communities (for more information, see www.coppa.org/).

There are opportunities across England for digital creators to meet each other face to face, through events such as Raspberry Jam, Young Rewired State and Coder Dojos. There are also competitions open to pupils, including the University of Manchester Animation competition, Kodu Kup, Robocup Junior and the Informatics Olympiad (see the Resources section under Extended learning and competitions for more information).

After-school clubs are an excellent way of harnessing extra enthusiasm in your pupils. They provide an opportunity for pupils to continue programming or to work on particular projects.

Assessment

Formative assessment

There are several challenges to assessing computing.

- Work delivered through projects can be open-ended, with pupils achieving very different but relevant outcomes.
- If pupils work collaboratively, how do you assess an individual's contribution?

Despite these challenges, it is possible to use assessment for learning (AfL) techniques that you're familiar with from other subjects to assess computing.

Self-assessment

Effective digital creators are independent learners. Part of the process of becoming an independent learner is being able to assess your own progress and evaluate your work. Self-assessment goes hand in hand with pupils setting their own goals. Reporting self-assessment can take the form of a learning journal, blog or screencast.

Peer assessment

Building on the idea of constructionism and making digital artefacts for other people, peer assessment provides discussion and feedback, helping the creator and assessor to understand what a finished product would look like, and how to improve it. Peer assessment can happen in the classroom but it also takes place online, through communities such as YouTube and Scratch. Pair programming and code reviews are industry techniques that can be used in the classroom.

Target setting

Setting challenging targets can help pupils recognise areas for development, an important step in becoming an independent learner. Make sure that targets are realistic, manageable and fully evaluated.

Open questioning

The theory elements of computing run the risk of being taught in a 'tell and recall' way, with pupils being passive receivers of information. Open questioning ("Why?" and "How?") allows pupils to understand the implications of theory. Programming and IT project tasks can be assessed by asking questions such as, "Why did you choose to do it this way and not another?" and "Can you explain how this works?". For more information, see the assessment guidance from Naace at www.naace.co.uk/curriculum/assessment.

KWL

Asking pupils to state what they already *know*, what they *want* to learn and what they have *learned* provides a perfect platform for pupil self-assessment and target setting. It can also inform your future lesson planning.

Technology-enhanced learning

Using technology-enhanced learning can be particularly effective in AfL. Here are a couple of examples.

Blogs and online communities: by publishing work in open or school-specific communities, pupils can share practice, create work for an audience and peer assess. Most virtual learning environments will have the ability to implement this within your school, and online communities such as YouTube and Scratch allow for wider audiences. Assessing work through blogs and online communities also links well with several of the key points around digital literacy. Pupils should think carefully about allowing completely open comments on their work – they need to be resilient if adverse comments are received and your role may be to support them.

Machine assessment: platforms exist to teach and automatically assess some aspects of computing. For example, OCR has built a computing MOOC for their GCSE: see www.cambridgegcsecomputing.org/. Commercial websites and virtual learning environments may provide tools to create self-marking tests.

Summative assessment

The levels from the previous ICT curriculum have been removed, leaving assessment at KS3 to the responsibility of individual schools.

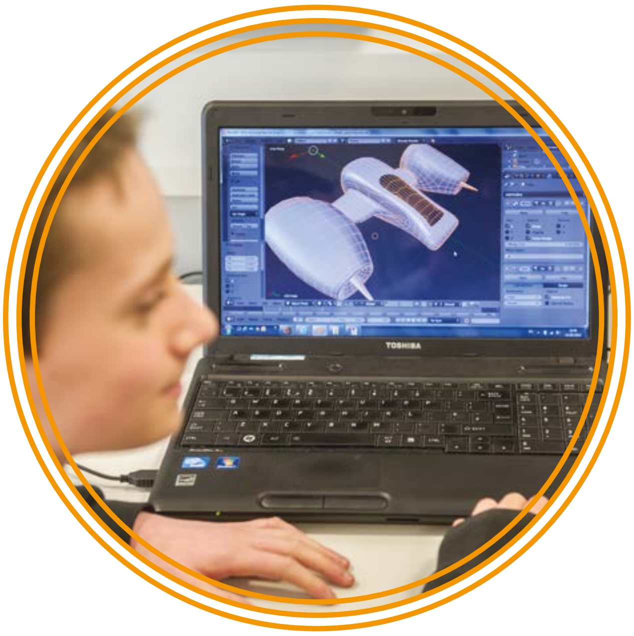
By the end of each key stage, pupils are expected to know, apply and understand the matters, skills and processes specified in the relevant programme of study.

The programme of study should form the basis of any assessment scheme. An obvious solution is to gather evidence from individual pupils as to whether they have met the requirements through a portfolio or record of achievement. This doesn't need to be too onerous a task. For example, as a pupil demonstrates mastery of a particular point, the evidence could be collected on a blog and their progress recorded by the teacher in the school assessment system.

There are a number of frameworks available for assessing computing. One such is the *Progression Pathways Assessment Framework (KS1 to KS3)* published by Computing at School (community.computingatschool.org.uk/resources/1692). This framework provides guidance on what to look for at different stages in the development of knowledge under a range of subject headings. An alternative framework is *Assessing Attainment in Computing* (community.computingatschool.org.uk/resources/2078) which takes a slightly different approach. It is up to you whether to adopt a published framework or to devise one of your own.

It makes sense to split the programme of study into individual clauses that can be easily understood by teachers, pupils and parents or carers, and against which progress can be tracked. To demonstrate progress, pupils could take a 'snapshot' showing the parts of the programme of study that have been met, along with the evidence. This could be compared with another 'snapshot' from a later date.

Achievement of each part of the programme of study could be rewarded with an item such as a badge. Mozilla's OpenBadges system (http://openbadges.org/) offers a potential digital solution. DigitalMe breaks down the criteria for awarding a badge into skills, knowledge, behaviours and evidence: see www.digitalme.co.uk/badges.



Here is an example of how this might work in practice.

> *Understand how … text … can be represented … digitally, in the form of binary digits*
>
> *Skills* – convert from Unicode to text; convert from text to Unicode; be able to predict the Unicode value of a character given another character close to it in the alphabet.
> *Knowledge* – know the need for coding systems; know why Unicode is used; know there are different types of Unicode; know how binary is used to represent a capital letter in the English alphabet.
> *Behaviours* – ability to work in pairs to create Unicode messages; peer assessment of work.
> *Evidence* – three messages decoded from Unicode into text; one message written in Unicode; completed worksheet on Unicode.

The example above is matched to a clause from *Assessing Attainment in Computing*. This assessment framework is derived directly from the programme of study, organised as eight bands across KS1–KS3 for CS, IT and DL. It builds on Computing at School's primary national curriculum guidance.

A similar approach can be followed if using the *Progression Pathways Assessment Framework (KS1 to KS3)* published by Computing at School. On the right-hand side of the web page, there is a PDF document available for download that explains how the assessment framework can be used with digital badges. An alternative version shows progression under the CS, IT and DL strands.

Concluding remarks

Understanding how computers work and being able to use them creatively gives pupils the power to shape the world around them. The new computing curriculum offers a firm foundation in this exciting and important subject. Computing gives pupils the opportunity to access technologies and ideas previously unavailable, helping them to make informed choices about their futures.

Computing can be a hugely fun and rewarding subject for teachers and pupils. Some of the ideas might be new to you and, at first, daunting. But treat this as an opportunity to learn and teach the concepts that are shaping the modern world. You may start to see the world differently, developing your knowledge of computing and skills in creating digital artefacts. Through computational thinking you will start to think like a computer scientist. And you are not alone. There are thousands of people across the country – and some great support networks and communities – wanting to help bring the computing curriculum to life. Together, we can bring about a generation of computational thinkers and digital creators.

Glossary

abstraction (process) – the act of selecting and capturing relevant information about a thing, a system or a problem.

abstraction (product) – a representation of a thing, a system or a problem that contains only selected (relevant) details about it; for example, a diagram is an abstraction.

algorithm – a set of unambiguous rules or instructions to achieve a particular objective.

array – a data structure comprising a collection of values of the same type, accessible through an index.

assembly code – a human-readable programming language in which each instruction corresponds to a single executable instruction for a CPU.

binary – a method of encoding data using two symbols, 1 and 0.

binary number – a number written in the base 2 number system.

bit – a basic unit of data that stores one binary value, 1 or 0.

bitmap – a collection of pixels forming an image.

Boolean – a data type with only two values, TRUE or FALSE.

browser cookie – a small piece of text recording activity about websites you visit, stored on your computer.

circuit – a grouping of electronic components that allow for operations to be performed.

code – any set of instructions expressed in a programming language.

coding – the act of writing computer programs in a programming language.

colour depth – the number of different colours that may be used in an image, dictated by the number of bits used to represent the colour of each pixel.

compiler – a program that converts programs written in one language (source code) into equivalent programs written in a different language (often in the form of instructions that a processor can execute).

computational thinking – a philosophy that underpins computing through decomposition, pattern recognition, abstraction, pattern generalisation and algorithm design.

CPU – central processing unit; the device within a computer that executes instructions.

data structure – a particular way to store and organise data within a computer program.

debugging – the process of finding and correcting errors in programs.

decimal – the base 10 number system.

decomposition – breaking a problem or system down into its components.

digital – using discrete binary values.

digital artefact – digital content made by a human with intent and skill.

digital creator – a person who makes digital artefacts.

digital media – media encoded in a computer readable form.

hardware – the physical components that make up a computer.

HTML – hypertext mark-up language; the language used to create web pages.

input (noun) – an input is a data value passed from the outside world to a computer.

input (verb) – to input is to send data from the outside world into a computer system.

internet – a network of interconnected networks.

interpreter – a program that converts instructions written in one language into equivalent instructions in another language, and executes each instruction as soon as it is translated.

IP address – Internet Protocol address; a unique numeric value that is assigned to a computer or other device connected to the internet so that it may be identified and located.

lists – a data structure for storing ordered values.

model – a representation of (some part of) a problem or a system.

modelling – the act of creating a model.

modular design – the practice of designing a system or program as a set of independent but interacting units (modules) that may be implemented and tested separately before bringing them together to solve the overall problem.

network – more precisely, a computer network; a collection of computational devices (personal computers, phones, servers, switches, routers, and so on) connected to one another by cables or by wireless media, and arranged so that data may be sent between devices either directly or via other devices.

operating system – a set of programs that manage the functioning of, and other programs' access to, hardware.

output (noun) – a response from a system.

output (verb) – to generate an output.

packet – more precisely, a network packet. A formatted unit of data for transmission across a network. Each packet contains part of a message plus some additional data, including where it is from and where it is going.

pixel – the smallest controllable element of picture/display.

process (noun) – a process is a running program.

process (verb) – the act of using data to perform a calculation or other operation.

program – a set of instructions that the computer executes in order to achieve a particular objective.

programming – the craft of analysing problems and designing, writing, testing and maintaining programs to solve them.

programming language – formal language used to give a computer instructions.

repetition – the process of repeating a task a set number of times or until a condition is met.

resolution – a measurement of the number of pixels needed to display an image.

router – more precisely, a network router. A router is a device that connects networks to one another (typically one or more local area networks (LANs) to a wide area network (WAN)), and directs packets between networks. A home broadband router performs the functions of a switch while allowing computers to connect to the internet.

selection – using conditions to control the flow of a program.

sequence (noun) – an ordered set of instructions.

sequence (verb) – to arrange a set of instructions in a particular order.

server – a computer or program dedicated to a particular set of tasks that provides services to other computers or programs on a network.

software – the programs that run on the hardware/computer system.

switch – more precisely, a network switch. This is a device that connects multiple computers to one another on a single local area network (LAN), and directs packets from machine to machine.

table – a data type storing organised sets of data under column headings.

Unicode – a standardised system for representing individual characters as sequences of bits.

variable – a data store used in a program.

web browser – a computer program to view websites.

World Wide Web – a service made of connected hypertext documents linked together across the internet.

Resources

Here is a small selection of resources for computing at KS3 and KS4. A much more complete curated list can be found at http://community.computingatschool.org.uk/resources/1787.

Background

Computing at School Working Group, *Computer Science: A Curriculum for Schools* (Computing at School, 2012), available at: www.computingatschool.org.uk/data/uploads/ComputingCurric.pdf.

Helsper, E.J. and Eynon, R., 'Digital Natives: Where is the Evidence?' *British Educational Research*, (2010) 36(3), 503–520.

Papert, S., *Mindstorms: Children, Computers, and Powerful Ideas* (Basic Books, 1993).

The Royal Society, *Shut Down or Restart? The Way Forward for Computing in UK Schools* (London, 2012), available at: http://royalsociety.org/uploadedFiles/Royal_Society_Content/education/policy/computing-in-schools/2012-01-12-computing-in-Schools.pdf.

Rushkoff, D., *Program or be Programmed: Ten Commands for a Digital Age* (OR Books, 2009).

Teaching Agency, *Subject Knowledge Requirements for Entry into Computer Science Teacher Training* (London, 2012), available at: http://academy.bcs.org/sites/academy.bcs.org/files/subject%20knowledge%20requirements%20for%20entry%20into%20cs%20teacher%20training.pdf.

Subject knowledge

Bentley, P.J., *Digitized: The Science of Computers and How it Shapes our World* (Oxford University Press, 2012).

Berners-Lee, T., 'Answers for Young People', available at: www.w3.org/People/Berners-Lee/Kids.html.

Brennan, K. and Resnick, M., *New Frameworks for Studying and Assessing the Development of Computational Thinking* (2012), available at: http://web.media.mit.edu/~kbrennan/files/Brennan_Resnick_AERA2012_CT.pdf.

Computing at School, *The Raspberry Pi Education Manual* (Computing at School, 2012), available at: http://pi.cs.man.ac.uk/download/Raspberry_Pi_Education_Manual.pdf.

Kemp, P. *et. al.* (2011–) *A-level Computing* (Wikibooks), available at: https://en.wikibooks.org/wiki/A-level_Computing/AQA.

O'Byrne, S. and Rouse, G., *OCR Computing for GCSE* (Hodder Education, 2012).

Extended learning and competitions

The British Informatics Olympiad: a computer programming competition for pupils under 19. Finals of the competition take place in Cambridge: see www.olympiad.org.uk.

CoderDojo: organisation promoting computer programming and technology; locations spread across the UK: see www.coderdojo.com.

Make Things Do Stuff: campaign and website providing pupils with links to clubs, communities, competitions and events; provides online projects: see www.makethingsdostuff.co.uk.

RaspberryJam: monthly meeting for Raspberry Pi enthusiasts of all ages; locations spread across the UK: see www.raspberryjam.org.uk.

UK Schools Computer Animation Competition: run by the University of Manchester, open to UK pupils aged 7–19: see http://animation14.cs.manchester.ac.uk (note the web address changes each year).

Young Rewired State: organisation promoting computer programming through online networking and free camps of varying lengths for pupils aged 18 and under; locations spread across the UK: see www.youngrewiredstate.org.

Teaching resources

New Zealand-based **Computer Science (CS) Unplugged** produce an excellent collection of resources exploring computer science ideas through classroom-based, rather than computer-based, activities: see http://csunplugged.org/.

Computing at School (CAS) hosts a large resource bank of plans, resources and activities. CAS is free to join: see www.computingatschool.org.uk.

Naace (the ICT association) and **CAS** have developed joint guidance on the new computing curriculum: see http://naacecasjointguidance.wikispaces.com/home.

A group of teachers and teacher trainers convened by the **NCTL** worked together to curate resources for initial teacher training for the computing curriculum, many of which may be useful for CPD and classroom use: see https://sites.google.com/site/primaryictitt/.

The 2008 **Royal Institution Christmas Lectures** were given by computer scientist Chris Bishop. These can be watched at www.richannel.org/christmas-lectures/2008/2008-chris-bishop.

Excellent resources are available for teaching with MIT's Scratch programming toolkit, together with an online support community, on the **ScratchEd** site: see http://scratched.media.mit.edu/.

Resources for teaching **safe, respectful and responsible use of technology** are widely available. Good starting points for exploring these topics are www.childnet.com/teachers-and-professionals and www.thinkuknow.co.uk/teachers/.

Support

Computing at School (CAS), as the subject association for computer science, has been a key influence on the development of the new computing curriculum. CAS has a vibrant support community, including members from industry and from all phases of education. There's a dedicated forum for members in secondary education, and many local and regional events, including training. See www.computingatschool.org.uk for more information or to join (free membership).

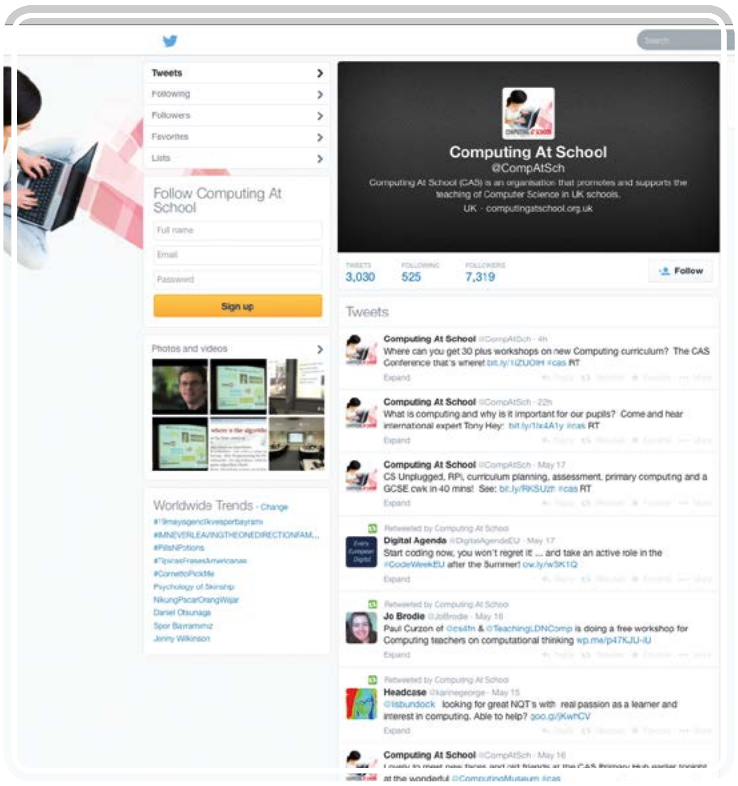
Naace is the ICT association concerned with advancing education through the use of technology, both within and beyond the computing curriculum. Naace members share a vision for the role of technology in transforming learning and teaching. Its members include teachers, school leaders, advisors and consultants working within and across all phases of UK education. Membership requires an annual subscription but many resources are available for free. See www.naace.co.uk/.

CAS has worked in collaboration with the British Computer Society (BCS) to establish a **Network of Teaching Excellence in Computer Science**. The network coordinates and provides training opportunities for serving and trainee teachers. The initiative is supported by the DfE, CPHC (Council of Professors and Heads of Computing), Microsoft and Google. The programme aims to build a high-quality, sustainable CPD infrastructure at low cost by nurturing long-term collaboration between employers, universities, professional bodies, schools and teachers. See www.computingatschool.org.uk/index.php?id=noe.

Many local authorities and **CLCs (City Learning Centres)** provide support and advice for schools and teachers on all aspects of the curriculum, including computing. Contact your local advisors or consultants for details of events and support in your area.

The **Science Learning Centres** offer CPD and other support to teachers and other school staff working in STEM disciplines including computing and ICT. Their national portal is at www.sciencelearningcentres.org.uk, and training opportunities may be found by searching there or by following links to the five regional Science Learning Consortia.

Twitter is a great informal source of ideas and advice once you've built up a useful list of contacts. The CAS Twitter account: **@compatsch**, its followers: https://twitter.com/CompAtSch/ and those it follows: https://twitter.com/CompAtSch/following may be helpful in developing your own personal learning network.



Facebook has groups for computing teachers teaching KS3: www.facebook.com/groups/254185994653238/ and teaching GCSE: www.facebook.com/groups/411684088866398/.

COMPUTING AT SCHOOL
EDUCATE · ENGAGE · ENCOURAGE
In collaboration with BCS, The Chartered Institute for IT

Computing At School promotes the teaching of computing in schools. Our aim is to support all teachers and all schools, and to develop excellence in the teaching of computing in their classrooms. We provide resources, training, local conferences and workshops, regional hub meetings, online community forums and so much more! **Computing At School** is free to join. Sign up and find out about events in your area by visiting us at **www.computingatschool.org.uk/secondary**.

Naace
Learning in a connected world

Naace promotes the appropriate use of computing to support learning, teaching and school organisation. Our aim is to support and challenge all teachers and schools and also those who provide services to schools. Naace has existed as an advocate in this area for 30 years and makes a small charge for annual membership.

Visit **www.naace.co.uk/membership** to join and to find out more about the ICT Quality Mark and Third Millennium Learning Award.

An eBook version of this guide, which can be freely shared with colleagues, is available at:
www.computingatschool.org.uk/secondary

9 781783 393763 >