Curriculum Guide for Primary Schools

Contents

Introduction

Computing is a new subject in the English National Curriculum, developed in order to draw together strands of Computer Science, Information Technology and Digital Literacy for pupils.

A high-quality computing education 'equips pupils to use computational thinking and creativity to understand and change the world' (Department for Education, 2013). This guide supports teachers from a broad range of backgrounds who want to understand the best ways to deliver computing and computational thinking within the context of their own subject area. It also aims to provide school leadership with a starting point as they seek to understand the best ways to develop computing and computational skills across the curriculum.

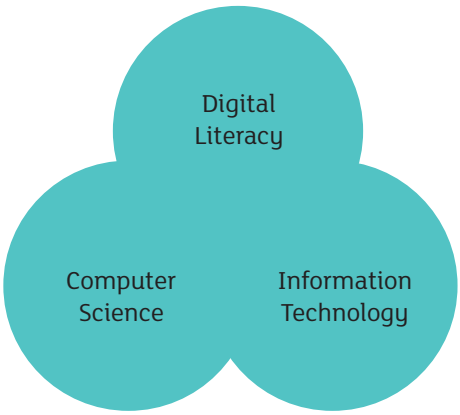


Figure 1: The complementary strands within computing

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 is 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.

Primary

Contains public sector information licensed under the Open Government Licence v3.

Introduction

Computers play a vital role in our lives. From home to work and from research to leisure, there's scarcely a part of modern life untouched by computer technology. Yet, rather than simply become proficient users of computers, it's vital that our pupils learn how such technology works.

Programming, including learning to code, is enjoyable and empowering to learn, as well as rewarding to teach. Unlike other subjects, however, it is one that few teachers studied at school or during their teacher training.

This guide is provided for teachers as an introduction to computational thinking and programming/coding. We hope that the information and support here help you identify and plan Interactive Design activities that support the Explore element of Using ICT, that is, to 'investigate, make predictions and solve problems through interaction with digital tools'.

Section 1
Computational thinking

Understanding problems for computers to provide solutions

Computers are incredible: they greatly expand our mental capacities. We can work faster, process more information and share ideas with people across the globe.

There are two steps to solving a problem with a computer:

1. Think about the necessary steps to solve the problem.
2. Use your technical skills to get the computer to work on the problem.

Think, for example, about using a calculator to solve a word problem in Maths. You need to understand the problem before your calculator can help out with the arithmetic.

Similarly, when making an animation, you have to plan your story and how you'll shoot it before your computer hardware and software can help you do the work.

Note: definitions of the main computing terms throughout this guide are provided in the glossary on page 61.

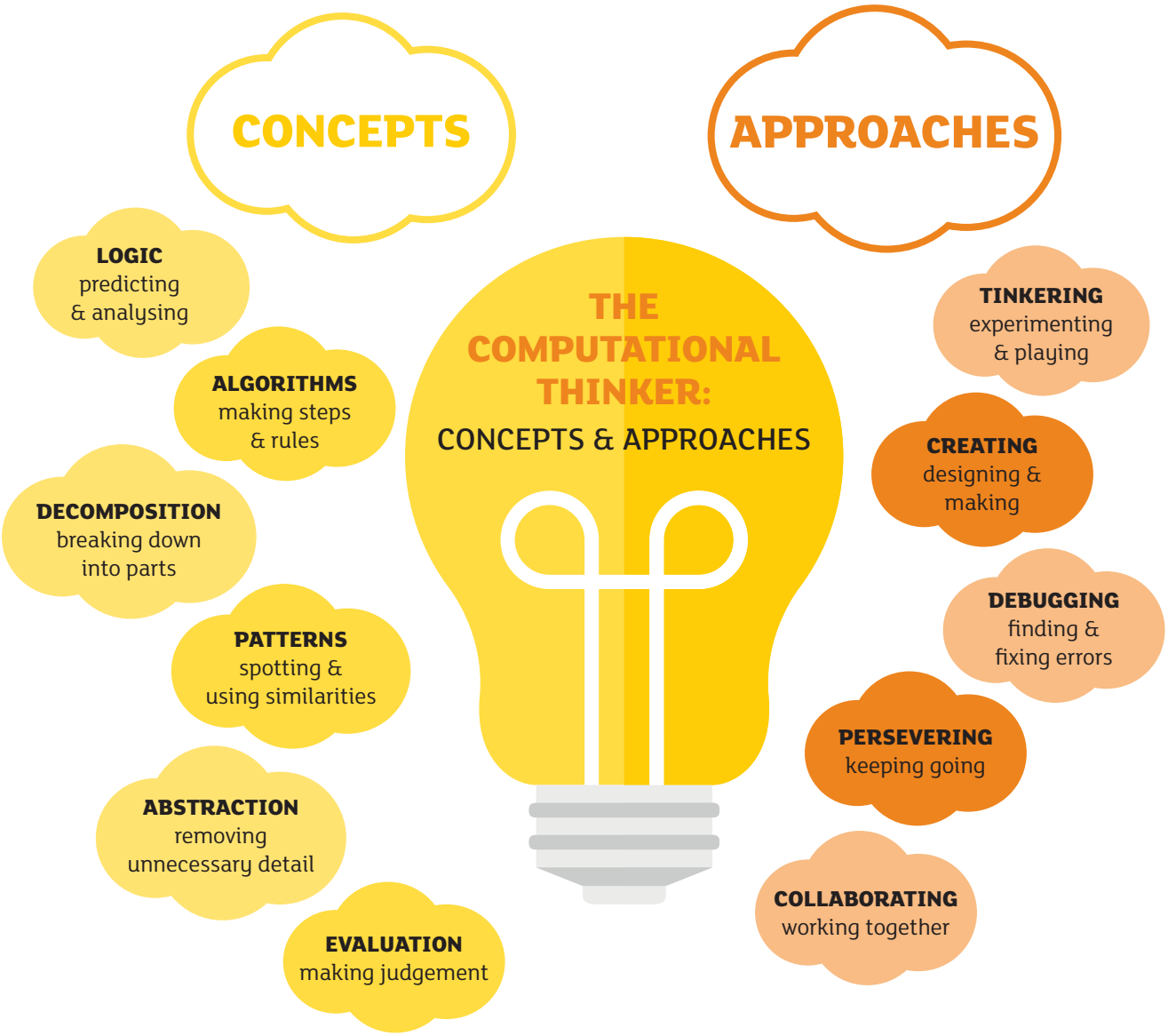
In both examples, the thinking done before starting work on a computer is called computational thinking.

Computational thinking is the term for the processes and approaches used when thinking about problems or systems. It's about considering a problem in ways that mean a computer can help to solve it.

This term was popularised by Jeanette Wing, an American computer scientist. In 2006, she argued that computational thinking should be part of every child's education along with reading, writing and numeracy.

The following are processes used in computational thinking:

- Logical reasoning: predicting and analysing (see pages 11–13)
- Algorithms: making steps and rules (see pages 14–18)
- Decomposition: breaking things down into parts (see pages 19–22)
- Abstraction: removing unnecessary detail (see page 23)
- Patterns and generalisation: noticing and using similarities (see pages 24–26)
- Evaluation: forming judgements



How is computational thinking used?

While computational thinking describes modes of thought that are typical of computer scientists and software developers, many others think this way too – and not just when using computers. The thinking processes and approaches that help with computing are helpful in many other contexts.

For example, the ways in which software engineers go about creating a new social networking platform are not really so different to how you and your colleagues might work as a team to put on a play or organise a trip.

In each case, you must:

- work out the rules or the steps to take for getting things done;
- reflect on the ways in which previous projects were done might help;
- divide the big and complex problem into smaller, more manageable, problems; and
- manage the task's complexity, typically, by focusing on the key details.

How is computational thinking used in a school curriculum?

Logical reasoning, algorithms, decomposition, abstraction and generalisation – all these aspects of computational thinking can help with problem solving across the school curriculum and beyond. As pupils learn these thinking skills in the context of computing work, they usually grow more proficient in applying them to other areas of study.
Your pupils already use computational thinking in many ways and across different curriculum subjects.

Consider these examples.

- When pupils write stories for Language and Literacy, you encourage them to plan first, think about the main narrative events, the settings and the characters.
- In the Arts, you might ask pupils to think about what they are going to create, what steps are needed to create this and how they will work through them. The complex creative process is likely to be broken down into more manageable planned phases.
- When working on a problem in Maths, pupils identify the key information first, before they solve it.

Where does computational thinking fit in?

The National Curriculum has prioritised thinking and problem-solving since its introduction in 2014.

At the heart of the curriculum lies an explicit emphasis on the development of skills and capabilities for lifelong learning and for contributing effectively to society. These whole curriculum skills and capabilities consist of the Cross-Curricular Skills (including Using ICT) and Thinking Skills and Personal Capabilities.

Computational thinking, however, shouldn't be seen as an alternative for 'problem-solving skills'. While it does help to solve problems and it does have wide applications across other disciplines, computational thinking is most evident and probably most effectively learned, through the systematic, creative processes of writing code. This is discussed in the second section of this guide.

While programming (see page 31) is an important part of computing, it would be wrong to see this as an end in itself. Rather, it's through the practical experience of programming that the insights of computational thinking can best be developed.

Classroom activities to develop computational thinking

1

Computational sandwiches

Pupils make a recipe for a sandwich; they consider each step in the process carefully. Teach them that a step-by-step sequence of instructions is called an algorithm. Invite them to share recipes and identify patterns in them – this is called generalisation. Read a range of recipes. Discuss the layers of simplification (abstraction) even in simple recipes, such as those for pizza.

2

Extension work

Challenge able or older pupils, either working to individually or collaboratively, to carry out more complex projects. Examples include researching and writing up aspects of a curriculum topic such as the Viking invasion covered in History, or putting together a class play or a school assembly. In each case, ask pupils to note each step needed for the task and to identify any elements they had to leave out to make the subject matter meet the brief.

Logical reasoning

Explain why something happens

Set up two computers in the same way, give them the same instructions (the program) and the same input, and you can be sure they'll produce the same output.

Computers don't invent things on their own initiative, or work differently because of personal feelings. This means that they are predictable. Because of this we can use logical reasoning to predict exactly what a program or computer system will do.

Children learn this quickly. Watching others use computers and experimenting for themselves allow even the very young to understand how technology works. A child soon recognizes that clicking a button brings up a list of different games to play, or that tapping or stroking the screen produce predictable responses.

Using existing knowledge of a system to predict future behaviour is an important part of logical reasoning. Central to logical reasoning is the ability to explain why something is the way it is. And it's a way to understand why something isn't the way you expected it to be.

Using logical reasoning

Logic is crucial to the operation of any computer. Deep inside its central processing unit (CPU), every operation the computer performs is reduced to logical operations carried out using electrical signals.

Because everything a computer does is controlled by logic, we can use logic to understand program behaviour.

Logical reasoning is used continuously by software engineers. They use their specialist understanding of how computer hardware, the relevant operating system (such as Windows, OS X) and the programming language they're using work so that they can develop new code that will work in the way they want. They also depend on logical reasoning when they test new software and when they seek out and fix the 'bugs' (mistakes) in their thinking (see page 52) or their coding when such tests fail.

Logical reasoning across the curriculum

Children already use logical reasoning across the wider curriculum in many ways.

- In Language and Literacy, pupils predict what a character will do next in a novel, or explain the character's actions in the story so far.
- In Science and Technology, pupils explain how conclusions have been drawn from the results of their experiments.
- In History, pupils discuss the logical connections between cause and effect; they understand that historical knowledge is derived from various sources.

Where does logical reasoning fit into computing activities?

Younger pupils can use logical reasoning to predict the behaviour of simple programs. This can include programs they write themselves, for example those made with a floor turtle, or simple movement commands on screen in a program like Scratch. Other uses of logical reasoning include predicting what happens when playing a computer game, or when using a painting program.

Older pupils can write simple algorithms and code using logical reasoning. They can identify and fix mistakes in their own code or in existing code.

Classroom activities to illustrate logical reasoning

1

Floor turtles

The pupils predict where the turtle device will end up when the go button is pressed. Then they explain why they have made this prediction. Being able to explain thinking is what logical reasoning is all about.

2

Debugging

Logical reasoning is crucial in the process of debugging (identifying and correcting errors in coding). Pupils could test programs made on Scratch or Scratch Jnr to learn this. (See https://scratch.mit.edu/ or http://www.scratchjr.org/) The pupils look at one another's Scratch programs and search for bugs. Encourage them to test the programs to see if they can work out exactly which part of the code is causing a problem. If pupils' programs don't work, ask them to talk through their code, explaining it to a classmate. The action of explaining the process is the important part, so it could even be explained to an inanimate object, such as a rubber duck. This is known as rubber duck debugging.

3

Cracking the Code

Print out some of the rules and regulations that apply to your school, for example its Computer Acceptable Use Policy. Ask pupils to think carefully about some specific rules. By using logical reasoning, the pupils have to explain why the rules are as they are.

4

Reasons for Rules

Give pupils a program that you have made, or one you have downloaded from the Scratch website. The pupils must work backwards from the code and work out what it will do.

5

Games and Logic

Many games draw on the players' skills in making logical predictions. Why not use some to help build their skills? First set up noughts and crosses for the pupils to play using pencil and paper. During the game, each pupil should predict their opponent's next move. After this, set up some computer games – Minesweeper, Angry Birds and SimCity all work well. At certain points during gaming, pupils must pause and predict what will happen when they make their next move. A great game to allow pupils to practise their skills of logical reasoning is chess. You might want to start a chess club if there isn't one already in your school.

Google’s search algorithm is said to be a more closely guarded secret than the recipe for Coca-Cola.

Algorithms

A sequence of instructions, or a set of rules for achieving a result or solving a problem, is known as an algorithm.

For example, you probably know the easiest way to get home from school; this might be turn left, drive for five miles, turn right. This might be considered an algorithm – it's a sequence of instructions for achieving a result: to get you to your chosen destination easily. There are plenty of algorithms that will accomplish the same goal (other routes); in this case, there are even algorithms (such as in those programmed into your satnav) for determining the shortest or quickest route.

Algorithms in the real world

Search engines use algorithms to organise search results, with the aim of putting the result you're looking at the top of the page. Google's search algorithm is said to be a more closely guarded secret than the recipe for Coca-Cola.

Online retailers use different types of algorithms. One type works on the basis of other people's purchases. Once you buy an item, the retailer suggests other purchases based on what other people, who bought the same item as you, went on to buy. A different algorithm can be used to drop the price of a new or untried product every few days or every few hours until the product is purchased by someone; after this, the price goes up.

Credit approvals, store cards, job and dating matches and more are all run on similar principles. The most complicated algorithms are found in science, where they are used to design new drugs or model the climate.

Algorithms across the curriculum

Developing pupils' understanding of algorithms could extend throughout the curriculum. Consider the following examples of ways in which you and your pupils might be using them already.

- A recipe can be considered an algorithm.
- A lesson plan is an algorithm for the sequence of events, activities and intended outcomes in a lesson.
- For many activities there is a sequence of steps or rules for pupils to follow, for example when going for lunch or preparing for a PE lesson.
- In Language and Literacy, we might consider the rules for producing writing in a particular form and then for proofreading and redrafting as a type of algorithm.
- In Science, the method of an experiment could be considered an algorithm.
- Your approach to teaching mental arithmetic might be to implement a simple algorithm.

An example of such an algorithm is the following.

- repeat ten times:
  - ask a question;
  - wait for a response; and
  - provide feedback on whether the response was right or wrong.

Where do algorithms fit with the National Curriculum

The curriculum requirements for Computing set out that pupils should understand what algorithms are, how they are implemented as programs on digital devices. and that programs execute by following precise and unambiguous instructions

There can be many algorithms to solve the same problem, and each of these can be implemented using different programming languages and on different computer systems. For example, Key Stage 1 pupils might usefully compare how they draw a square with a floor turtle and how they would do this on screen in Scratch.

Scratch Jnr

Key Stage 2 builds on this and pupils can design programs with particular goals in mind. Such work draws on their ability to think algorithmically. Pupils also use logical reasoning to explain algorithms and to detect and correct errors in them.

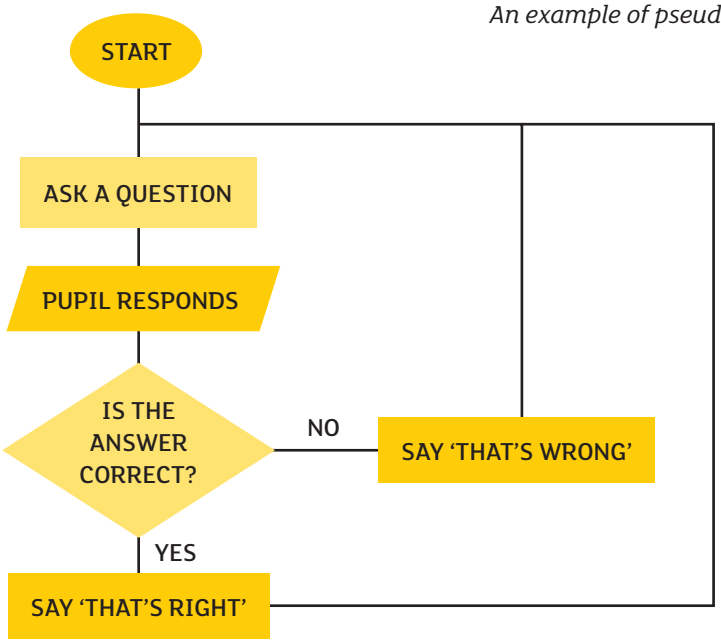
To practise this, encourage pupils to carry out the steps for an algorithm. They must follow the instructions themselves rather than write these as code for a computer. You are likely to see errors and inconsistencies early in the process!

While programming languages like Scratch can minimise the need for a thorough planning stage when writing a program, it's good practice for pupils to prepare thoroughly. They should plan, design write down the algorithm for their program. **This doesn't have to be lengthy writing: they can use rough jottings, a storyboard, pseudocode or a flow chart**. (Pseudocode, as seen in the first illustration below, is a written description of how a program will operate; the second illustration, below, is an example of a flow chart.) This type of writing – in whatever form you choose – makes it easier for pupils to receive feedback on their algorithms before implementing these as code on their computers. Feedback can come from other pupils as well as from you.

Repeat 10 times:
Ask a maths question

If the answer is right then:
Say *'well done!'*
Else:
Say *'think again!'*

An example of pseudocode



An example of a flow chart

Classroom activities to illustrate algorithms

1

Discussion

Discuss what makes one algorithm better than another with your pupils. In early programming lessons, pupils should realise that a Bee-Bot program with fewer steps than another to get to the same place is faster.

2

Guess my number

Play 'Guess my number' to demonstrate the point. Tell the pupils that you have picked a number between 1 and 100 which they have to guess. They can ask questions about the number, but you can only answer 'Yes' or 'No'. Each pupil may ask only one question.

- For the first turn, ask the pupils to guess numbers at random.
- Next, using a new number, ask the pupils to guess the number sequentially, beginning with number one. For example, 'Is the number one?' and so on. Explain that this is called a linear search. Give the class as many turns as they need to guess the number.
- Finally, using a new number, explain the term binary search. Explain that the pupils already know the number is less than 100, so suggest that they ask, 'Is it less than 50?' then, 'Is it less than 25?' or 'Is it less than 75?' depending on the answer. The pupils should keep halving the section they are searching in until they find the number.
- After the game, discuss which approach found the number quickest. When they are familiar with the binary search method, replay the game using a number between 1 and 1000.

3

Sorting weights

The pupils sort a set of unknown weights in order of weight using a simple pan balance. They think carefully about the algorithm they're following to do this. Then they think of a quicker way to accomplish the same task. A demonstration of this may be found at http://csunplugged.org/sorting-algorithms

4

Follow the rules

Explain that not all algorithms are made up of sequences of instructions. Some are based on rules. Write a number sequence on the board to introduce this idea, for example 3, 6, 9, 12 or 2, 4, 8, 16. The pupils must work out the rule for the sequence (adding 3, or doubling the number); then they must predict the next number. Explain the following: the rule for the sequence is the algorithm; the process by which they worked it out was logical reasoning.

Decomposition

Break it to make it

Decomposition is the process of breaking down a problem into smaller manageable parts. This process helps us solve complex problems and manage large projects.

It has many advantages. The process of problem-solving becomes manageable. Large problems are daunting, but a set of smaller, related tasks is much easier to work with. Decomposition also means that a team can work together on the overall task, with each member bringing their own insights, experience and skills.

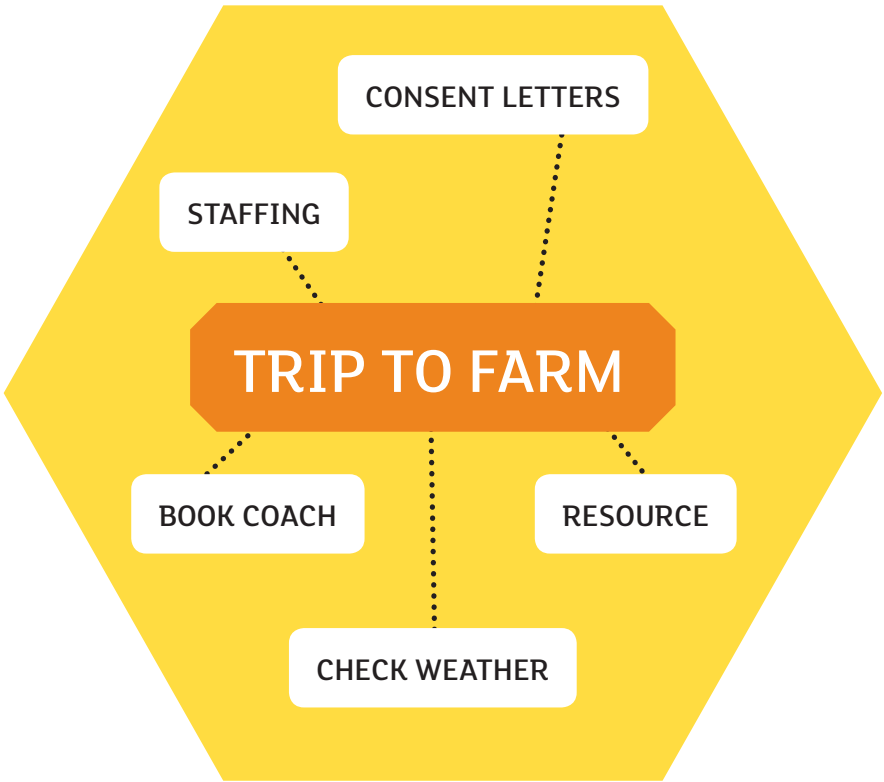
Decomposition in the real world

Breaking down problems into their smaller parts is not unique to computing. Decomposition is standard practice in engineering, design and project management. Software development is a complex process, so being able to split a large project into its component parts is essential. Consider the different elements that are combined to produce a complex program like PowerPoint. Computers themselves are similarly complex. For example, a laptop or a smartphone is made up of many components. These components are often made independently by specialist manufacturers, then assembled to form the finished product with everything working under the control of the operating system and applications.

How can we use decomposition in school?

You'll have used decomposition to tackle big projects at school already. Consider the examples that follow.

Delivering your school's curriculum is a good example. Typically a curriculum would be decomposed as years and areas of learning, then further into terms, units of work and individual lessons or activities. Notice how the project is undertaken by a team working together (your colleagues), and how important it is for the parts to integrate harmoniously. Organising events, such as a school play, a school trip or a school fair are all examples of projects in which decomposition is used.



Decomposition applied to a school trip

Decomposition used across the curriculum

It's likely that you and your pupils already use decomposition in many different ways across the curriculum. Here are just some examples.

- Labelling diagrams in Science or Geography to show the different parts of a plant, or the different nations which make up Europe.
- Planning the different parts of a story in English.
- Breaking down a problem in Maths.
- Planning a research project for any subject, or collaborating on a group presentation. Computers can be beneficial with this sort of collaborative group work, for example collaboration tools are available in Office 365 and other cloud-based software.

Where does decomposition fit in with the Curriculum?

As pupils plan programs to solve problems in a digital environment, encourage them to use decomposition by working out what the different parts of the program or system must do and thinking about how these are interrelated.

For example, a simple educational game needs:

- some way of generating questions;
- a way to check if the answer is right;
- some mechanism for recording progress such as a score; and
- some sort of user interface, which in turn might include graphics, animation, interactivity and sound effects.

Plan opportunities for pupils to work collaboratively on a software development project or on other projects in computing.

This could involve media work such as:

- animations or videos;
- shared online content such as a wiki; or
- a challenging programming project such as making a computer game or even a mobile phone app.

Classroom activities for decomposition

1

Plan a programming project

Plan a large-scale programming project, such as making a computer game in Scratch, through decomposition. Even for a relatively simple game the project would typically be decomposed as follows: planning, design, algorithms, coding, animation, graphics, sound, debugging and sharing. A project like this would lend itself to a collaborative, team-based approach, with development planned over a number of weeks.

2

Decompose a desktop

Remove the casing from an old desktop computer to reveal how computers are made from systems of smaller components connected together. It might be possible to disassemble some components further, though it could be easier to examine illustrations of their internal architecture.

3

Collaborative online project

Set up a collaborative project online. This might take the form of a multi-page wiki site. Pupils could, for example, take the broad topic of e-safety, decompose this into smaller parts and then work collaboratively to develop pages for their wiki, exploring each individual topic. The writing process for these pages might be decomposed further through planning, research, drafting, reviewing and publishing phases.

Abstraction

Abstraction means ignoring or hiding irrelevant details of a problem so you can focus on dealing with the important part.

For the American computer scientist, Jeanette Wing, who is credited with coining the term, abstraction lies at the heart of computational thinking:

'The abstraction process – deciding what details we need to highlight and what details we can ignore – underlies computational thinking.' Jeanette M Wing (2008) 'Computational thinking and thinking about computing'. http://rsta.royalsocietypublishing.org/content/roypta/366/1881/3717.full.pdf

When coders write code to carry out a task, their method may combine other methods or functions built into the programming language but they don't need to know how these functions were created.

While computer programmers find abstraction very useful, it's not something you need to explain to primary pupils when they're carrying out coding activities. It is, though, a powerful way of thinking about problems – so you may wish to introduce the concept in a simple way and not necessarily through computing lessons.

For example, in Maths, solving word problems often involves identifying the important information and setting out how to represent the problem in the more abstract language of arithmetic, for example by using the plus, minus and equals terms and symbols.

If we identify patterns, we can predict outcomes, make rules and solve general problems.

Patterns and generalisation

Making life easier

In computing, searching for a general approach to a class of problems is known as generalisation. If we identify patterns, we can predict outcomes, make rules and solve general problems.

Consider the following example. Pupils are learning about area by working out the area of a rectangle which is drawn on a grid of centimetre squares. While they could count the centimetre squares on the grid, a better method is to multiply the length by the width. Not only is this method faster, it also works for all rectangles, including those that are very small and those that are very big.

Of course, it does take time for the pupils to grasp this formula, but when they know it, this formula can be applied quickly and easily to similar problems that they face in the future. Time spent now is time saved later!

How are patterns and generalisation used in the primary curriculum?

Pupils have probably encountered the idea of generalising patterns in many areas of the primary curriculum already.

- Think about how they respond to nursery rhymes and stories. These contain repeated phrases and structures. Many children grow familiar with the repeated rhymes, rhythms and phrases of nursery rhymes even before they go to primary school. Later, they respond to the familiar narrative structures in traditional stories, such as in fairy tales.
- Songs and other pieces of music often contain patterns and melodies that children recognise and respond to.
- Pupils often work on simple maths problems that involve recognizing patterns and deducing generalised results.
- Pupils recognize patterns in spelling. These are often taught as rules, and exceptions to these general rules are also taught.

Classroom activities for patterns and generalisation

1

Number sequences

In computing lessons, encourage pupils to use a simple programming language such as Scratch to experiment with number patterns and sequences. Challenge them to work out a general program which can generate any linear number sequence.

2

Simpler, quicker, better

Teach pupils to find simpler or quicker ways to solve a problem or achieve a result. Invite them to create geometric patterns, working up to more complex 'crystal flowers' using turtle graphics commands in languages like Scratch, Logo or TouchDevelop. Some of the CCEA Using ICT tasks for Interactive Design could help with this. Emphasise how using repeating blocks of code is much more efficient than writing each command separately. Ask pupils to experiment with how changing one or two numbers in their program produces different shapes.

3

Graphic patterns

Teach pupils to use graphics software to create tessellating patterns to cover the screen. As they work on their patterns, ask the pupils to find faster ways to complete the pattern – copying and pasting groups of individual shapes is one easy way.

4

Sequencing in music

In Music lessons, teach pupils to use simple sequencing software in which patterns of beats are repeated, then allow them to use this to create rhythmic and effective compositions.

Writing software

In addition to the processes of computational thinking outlined above, there are numerous approaches that characterise computational thinking. It's worth helping pupils to develop such approaches to their work; if they do, they'll be much more successful in putting their thoughts into action.

Tinkering/Experimenting

Computer scientists love to play around with technology: to experiment and to explore. Aspects of learning a new programming language or exploring a new system are like the purposeful play that schools often encourage as an effective approach to learning in Foundation Stage classrooms.

Using open source is a great way to play with software and writing code. It's easy to take code written by someone else, to examine how it's constructed and then adapt it to your own purpose. This sharing-based approach is encouraged by programs such as Scratch. In fact, its name comes from the term 'scratching', which, in computing, refers to the process of reusing code – code that can be shared and reworked easily for other purposes and situations.

Rather than explain exactly how a new piece of software works, encourage pupils to play with it, sharing what they discover with each other. Also, set up opportunities for them to use code developed by others – by you, by their peers or from online sources. Use this code as the starting point for their own programming.

Creating

Creativity is at the heart of the programming process. Foster a spirit of creativity in your pupils by seeking out tasks that offer scope for their use of imagination and creativity, rather than simply have them program to find the right answer.

Teach them to reflect on the work they produce, thinking of the strengths and areas for development in their own and others' projects. It is now common practice for software developers to continually seek ways in which they can improve their software. Working with digital music, images, animation, virtual environments and 3D printing are some ways to foreground artistic creativity.

Debugging

A normal part of programming is writing code that initially contains flaws. These flaws, errors or faults are known as bugs, and a big part of writing code is their removal, which is known as debugging.

It's important for you to teach your pupils to think as programmers. Accordingly, they should take responsibility for thinking through their algorithms and code. They should also find and fix their mistakes. There are many areas of the curriculum in which similar processes are used. For example, in Language and Literacy pupils often draft, proofread and redraft their writing and, in Maths, pupils often check through their working.

Just as in Language and Literacy the teacher might ask pupils to proofread each other's writing, ask pupils to debug each other's code. Identifying and correcting mistakes independently is recognised as an important method of learning; debugging code – either a pupil working on their own code, or working on that of a peer – is an excellent way of doing this.

As the teacher, it's important for you to notice the bugs that creep in to your pupils' coding. Often these reveal misunderstanding that you can address directly, either with one or two pupils or with the group as a whole.

Persevering

It's worth reminding ourselves that computer programming is difficult. Yet that difficulty is part of its attraction. Overcoming the challenges along the way to produce code that works effectively brings the programmer tremendous satisfaction. A big part of overcoming those challenges goes beyond technical skills, such as knowing the right algorithms and understanding the language you're working in. A programmer must be willing and able to persevere with a task that's often difficult and sometimes frustrating. Qualities such as perseverance and resilience are vital.

Developing such qualities brings benefits in other areas of the curriculum too. As the work of the psychologist Carol Dweck shows, love of learning and resilience are qualities associated with virtually all high achievers. It is important for pupils to develop what Dweck terms a 'growth mindset', that is to believe that their abilities can grow through hard work and dedication. Meeting the challenges of programming and showing resilience through rewriting code and debugging are excellent ways to do just that.

Rather than allowing them to give up or to ask for your help when they encounter problems in their programming, teach pupils to adopt strategies for dealing with those problems. Typical strategies include identifying the exact nature of the problem, seeking a solution by using a search engine such as KidRex or Swiggle (or by using Bing or Google with the safe search mode locked), or by asking a classmate for help.

Collaborating

Software is developed collaboratively. Teams of programmers and others work together on a shared project. Set up activities that replicate this experience for your pupils. Group work is common in areas of the primary school curriculum; computing lessons – whether discrete, or as part of delivering another subject – should be no different.

'Pair programming' is one useful approach. Two programmers share a screen and keyboard as they work together to write software. One programmer usually adopts the role of driver (taking charge of the detail of the programming), whilst the other adopts the role of the navigator (taking charge of the 'big picture'). The two programmers swap roles regularly, so that each understands both the detail and the big picture.

Larger group work develops additional skills, with each pupil contributing some of their own particular talents or interests to a shared project. It's important, however, that all pupils develop their understanding of each part of the process. Do make sure you plan some sharing of roles or peer-tutoring throughout your activities.

Section 2
Programming

What is programming?

A program is a set of instructions for a computer, which is written in a language that it can understand. Programming is simply the action of designing and writing such instructions. At its simplest level, programming can be a process such as making a toy robot trace a square. At its most complex, it can be writing code to predict the weather.

There are two steps to writing a program:

1) Analyse the problem or system and devise a solution.

   This process uses logical reasoning, decomposition, abstraction and generalisation to design algorithms to solve the problem or model the system. (See Section 1 for further details of these processes.)

2) Express these ideas in a programming language.

   This is called coding. The set of instructions that make up the program is called code.

People study computer science to learn how to code. There's a wonderful sense of satisfaction when you achieve the goal of seeing a computer do just what you've asked because you wrote a precise set of instructions. Programming allows you to test ideas and have immediate feedback on whether something works or not.

Programming in schools

While it is possible to teach computational thinking without coding and vice versa, it's better to teach them together. Teaching computational thinking without allowing pupils to test their ideas by writing code on a computer is like teaching only scientific theories and principles without doing any experiments. Similarly, teaching programming without teaching the processes of computational thinking is like only doing experiments in science without teaching the principles that they exemplify.

It can be useful for pupils to analyse problems using terms such as algorithms and decomposition, and have repeated practical experience of writing computer programs in order to solve problems.

Programming activities for Key Stage 1 pupils could include looking at how simple algorithms are implemented as programs on digital devices. The phrase 'digital devices' includes tablets, laptop computers and programmable toys such as a Bee-Bot. It can be useful for pupils to be able to look at their algorithms, in whatever way they've recorded these, and their code side by side.

Pupils also should have the opportunity to create and debug their own programs as well as to predict what a program will do.

In Key Stage 2, pupils can move on to learning how to design and write programs that accomplish specific goals such as making a game in Scratch in which sprites interact. Other programs can include controlling or simulating physical systems, for example making and programming a robot. They can be taught to use sequence, selection and repetition in their programs as well as variables to store data (for example, using a counter to keep score in a Scratch game).

Pupils can also learn to use logical reasoning to detect and fix the errors in their programs.

Here are some ideas for extended programming projects:

Key Stage 1

- Solve a maze using a floor/screen turtle.
- Create a simple animation in Scratch.

Key Stage 2

- Create a question and answer maths game.
- Create more complex computer games and animations in which there are more sophisticated interactions between sprites. For example, in Scratch, using the 'broadcast' and 'receive' commands to make characters in an animation talk to each other.

© TSS Group Ltd

Using arrow cards to record algorithms for programmable toys.

How do you program a computer?

Programming a computer means writing code. The code is the set of instructions written in a language the computer can understand.

But don't despair: not all computer languages are highly complex and specialised. In fact, those that we teach and use at primary school are a halfway house. They're written in a computer language that is expressed in a version of English that we can understand, which then gets translated by the computer into a more pure computer language called machine code. Machine code is a set of instructions that can be run directly on the silicon chips of the machine. It is a language that the computer can respond to directly.

Programs are made up of precise instructions. When writing a line of code, there's no room for ambiguity or debate over meaning. We can only write code using the clearly defined vocabulary and grammar of the programming language, but typically these are words taken from English, so code is something that people can write and understand, but the computer can also follow.

Programming languages for primary school

There are hundreds of computer languages. While most of these are too complex for those beginning to learn programming, there are many that can be used effectively in the primary classroom. Many of these are also well resourced and some are backed by the support of online communities. Choose a language that you know already, or one that you'll find easy to learn.

When choosing a programming language, consider the following:

- Not all languages run on all computer systems.
- Select a language that suits your pupils. (There are computer languages that are readily accessible to primary pupils, such as Scratch. In most cases these have been written for pupils, or at least adapted to make them easier to learn.)
- Consider the learning resources that are available. Why not pick one that is supported by a good range? It's even better still if it has online support communities available, both for teachers and for learners.
- It is beneficial for the pupils to be able to continue working in the chosen language on their home computer. It's even better if they can continue work easily on the same project via the internet.
- Some say that some languages are better at fostering good programming habits than others. It's probably better to focus on the teaching than the language in that regard. At this early stage, good teaching – where computational thinking is taught alongside coding – will help to prevent pupils from developing bad coding habits.

The right language for the right Key Stage

The table below suggests which programming languages are suitable at which stage of primary school.

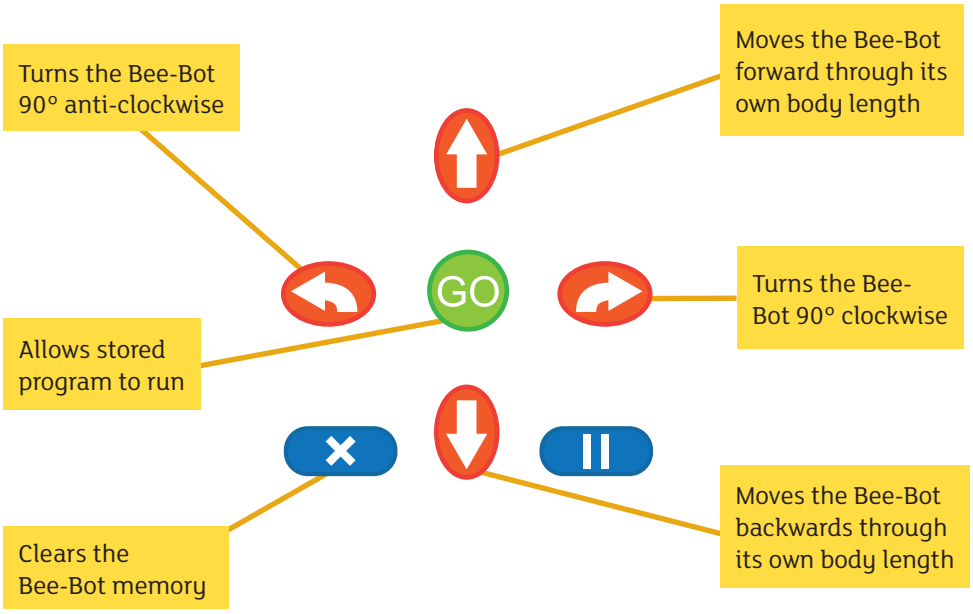
Key Stage	Language Type	Language
Foundation Stage	Device-specific	Bee-Bot
		Programmable toys
KS1	Limited instruction	Scratch Jr
		Scratch
KS2	Game programming	Kodu
	Block-based	Scratch, Hopscotch
	Text-based	BitsBox
		HTML/CSS

Programming a floor turtle

Rather than using computers, programming work at Foundation Stage and Key Stage 1 is more likely to use simple programmable toys.

Pupils learn the techniques of programming more quickly and easily when they use a simple language and a simple interface. Using a floor turtle, such as a Bee-Bot or a Roamer-Too, makes it easy for them to plan and check programs. Since these devices work on the floor, pupils can, quite literally, put themselves in the place of the device they're programming.

The programming language used by the Bee-Bot has five commands: forward, back, turn left, turn right and pause. All you need to do to program one is simply press the buttons in the desired order to build a sequence of commands, with new commands being added to the end of the sequence.



The Bee-Bot functions

© TSS Group Ltd

Pupils can enjoy a range of fun activities using a Bee-Bot, both for computing lessons and for work across the curriculum. Foundation pupils are likely to start programming the device one instruction at a time, but older children often become skilled at writing longer sequences.

When pupils are ready to move from programming floor turtles to programming on screen, why not use an on-screen simulation of a Bee-Bot? You can make, or adapt, one using Scratch.

Classroom activities

1. **Experiments with a turtle**
   Let very young pupils experiment with a floor turtle. As they play, encourage them to develop their understanding of the link between pressing buttons and running their program.

2. **Simple programming**
   The pupils plan a sequence of instructions to achieve a specific goal, such as moving the floor turtle from one 'flower' to another on an illustrated grid. Then the pupils must demonstrate logical reasoning by predicting what will happen when their program runs, and explain their thinking.

3. **More complex challenges**
   For more complex challenges, provide pupils with the code for a floor turtle's route from one place to another, including an error in the code. Ask the pupils to work out where the bug is in the code and then fix this, before testing out their code on the floor turtle.

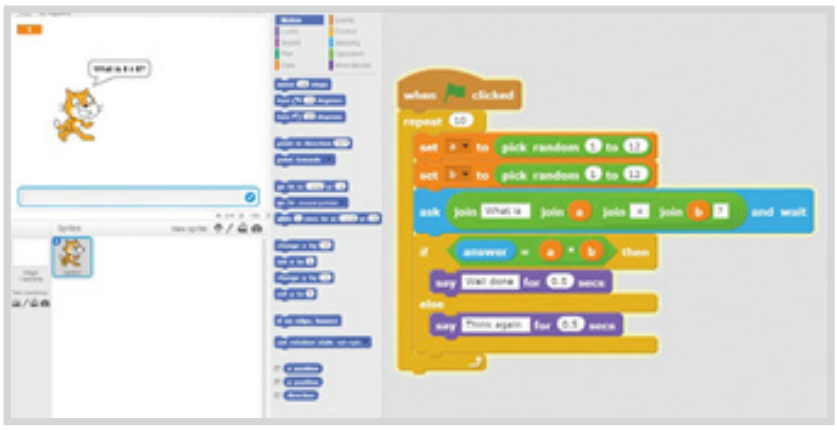
Programming movement on screen

Graphical programming toolkits make learning to code easy. In most of these, programs are developed by dragging or selecting blocks or icons which represent particular instructions in the programming language.

These can normally only fit together in ways that make sense, so the potential for errors in spelling or punctuation is minimised. In this way, the programmer focuses on the ideas of their algorithm and the intended outcome, rather than becoming bogged down in the words and the grammar of the programming language. Programming in this way usually means that pupils can become more active learners, who require much less support from their teacher.

Scratch

Devised by the prestigious Massachusetts Institute of Technology (MIT), Scratch is a versatile and free programming language that helps children learn programming. It has two versions: Scratch, which is aimed at students from the age of eight; and Scratch Jnr, which is aimed at children between the ages of five and seven. The programmer creates their own graphical objects, including the stage background where the action of the Scratch program takes place and moving objects, known as sprites, such as the characters in an animated story or a game.



Programming with Scratch

Each object can have one or more scripts, built using the blocks of the Scratch language (see the illustration above). To program an object in Scratch, drag the block you want from the different palettes then snap it into place within the other blocks on the screen. This forms a script. Scripts can run in parallel, or be set in motion by particular events.

Numerous other projects use Scratch as a starting point for their own platforms. For example, ScratchJr can be used in the form of an iPad app designed for young programmers at Foundation Stage and the start of Key Stage 1. The same sort of building block interface is used by Snap!, a language developed by the University of California at Berkeley, which allows even more complex programming ideas to be explored.

Scratch programmers can access a vibrant online community to download and share their projects globally. With such a network of support, pupils can learn much more about programming than what is required at primary school. Similarly, teachers can take advantage of help, support and high quality resources from Scratch's educator community. Scratch is available as both a web-based editor or as a standalone desktop application. You can move files between online and offline versions easily.

Kodu

Kodu was developed by Microsoft. It is visual programming language for creating simple, interactive 3D games. Each object in the Kodu game world can have its own program. Kodu programs are 'event driven'. This means that are made up of sets of commands set up in a 'when [this happens], do [that]' format. Actions are triggered when things happen, such as a key being pressed, one object hitting another or the score reaching a certain level.

As with Scratch, programmers can share their programs – in this case, games – with others in the Kodu community. This facilitates informal and independent learning. Again, like Scratch, pupils may download and modify games created by others. This can be an effective way to learn programming, which can encourage pupils to develop games with a strong sense of audience and purpose.

Classroom activities

1 **Scratch animations**
Pupils create simple scripted animation using Scratch. Each animation could have at least two programmed characters, who help to act out a story. Designing the algorithm for this sort of program is similar to storyboarding in video work.

2 **Kodu games**
First pupils play a selection of games from those on the Kodu community site. Then they are set the task of developing their own game. As a starting point, you might ask them to create a game in which Kodu (the player's avatar) is guided through a hostile landscape, where he encounters enemies.

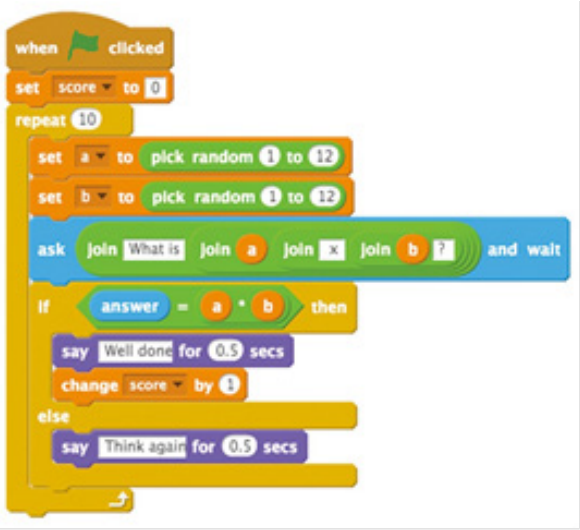
What's inside a program?

The details vary from language to language, but each one contains some of the same structures and ideas. Programmers use these over and over again – in different languages and to tackle different problems. The following is a summary of the structures.

- Sequence: running instructions in order (see p41);
- Selection: running one set of instructions or another, depending on what happens (see p43);
- Repetition: running some instructions several times (see p45); and
- Variables: a way of storing and retrieving data from the computer's memory (see p49).

It's important that pupils learn these concepts as they progress.

This Scratch script below shows sequence, selection, repetition and variables. Before we consider them in detail, can you work out which is which?



Identify the elements in this Scratch script

Sequence

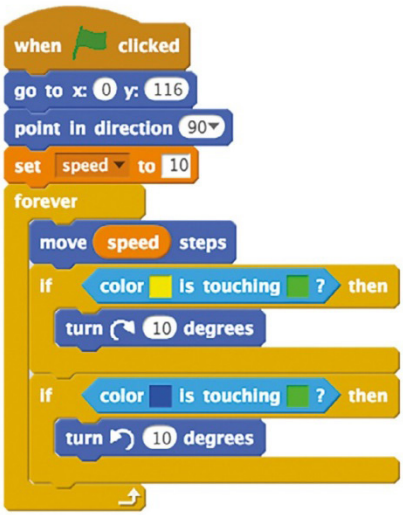
Programs are sequences of instructions. For example, the sequences of instructions in programs for floor turtles are built up as the stored sequence of button presses for what the turtle should do.

The instructions – as they would be for any program – are precise and unambiguous. The floor turtle will take each of the instructions (in the form of the stored button presses) and convert that instruction into a signal for the motors driving its wheels.

As they start to program, pupils might simply type a single instruction at a time, clearing the memory after each. Yet, as they gain experience as programmers, or need to solve a problem more speedily, their sequences grow in complexity. For example,

Forward
Forward
Forward
Turn left
Forward
Forward

When working with Scratch, pupils' early programs are likely to be made up of simple sequences of instructions too. These instructions must be precise, unambiguous and correctly ordered. In creating algorithms, pupils should have worked out the exact order in which to put the steps in order to complete the task.



Identify the elements in this Scratch script

Classroom activities

1. **Problem solving with a turtle**
   Give pupils progressively more complex problems to solve with a floor turtle. Ask them to plan their algorithm for solving these problems, then create single programs on the floor turtle.

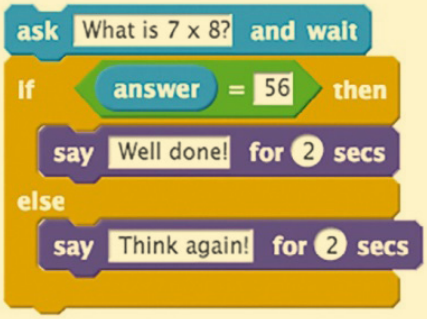
2. **Scratch remixed**
   Assign pupils existing projects from Scratch (see Further resources on page 55). Invite them to modify these projects by changing the code and observing the ways in which their modifications affect the program.

3. **Scratch animations**
   Using Scratch, pupils design, plan and code scripted animations. If possible, they should use a timeline or a storyboard to devise their algorithm before converting it into instructions for the sprites (characters) in Scratch.

Selection

Selection is the programming structure through which a computer executes one or other set of instructions according to whether a particular condition is met or not.

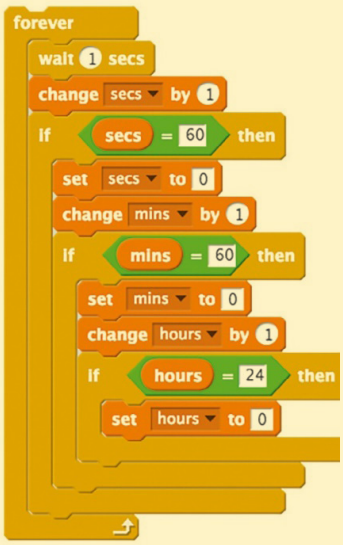
This ability to do different things depending on what happens in the computer as the program is run or out in the real world lies at the heart of what makes programming such a powerful tool.In Scratch, as with many other languages, we can build selection into our sequence of instructions. This allows the computer to follow different instructions according to whether a condition is met or not. The example below shows how this works in practice.



Selection being used in Scratch

A simple selection instruction lies at the heart of many educational games: if the answer is right then give a reward, else say the answer is wrong. (See the Scratch script for the times tables game later in this section.)

Selection statements can be 'nested' inside one another. This allows more complex sets of conditions to be used to decide what happens in a program.



Nested selection statements in a clock program

Look at how some 'if' blocks are inside others in this script written in Scratch. It is to model a clock in Scratch. Note how the script also uses repetition and three variables for the seconds, minutes and hours.

When designing a game in Kodu, selection is also vital. A set of conditions control an object's behaviour in a game. For example, WHEN you press the left arrow, the object moves left. Similarly, interaction with other objects, variables and environments in Kodu are programmed as a set of WHEN … DO … conditions. For example, WHEN I bump the apple DO eat it AND add 2 points to score.

Selection statements in Kodu

Classroom activities

1

Scratch quiz

Pupils use Scratch to design simple question and answer games. Teach them to devise the overall algorithm for their game before coding. Then encourage them to develop the user interface, making it more engaging than just a cat asking questions. Teach them to have a target audience in mind.

2

Kodu games

Pupils devise simple games using Kodu. Encourage them to experiment with the different conditions that a character in Kodu can respond to in its event-driven programming. Teach pupils consider how they might use such conditions when developing their own games. Allow them sufficient time to code and to redraft their code. Encourage them to think carefully about the algorithms they design – that is, the rules of their game. iCompute has further, easy to follow lessons for this activity. See www.icompute-uk.com for further details.

Repetition

In programming, repetition is when the program repeats the execution of certain commands.

Such an occurrence is also referred to as a loop, since the computer keeps looping through the commands one at a time as they're carried out. Using repetition makes a long sequence of instructions shorter and, usually, easier to understand.

Writing code that uses repetition typically involves noticing that some of the instructions you want the computer to follow are very similar or the same. It therefore draws on the computational thinking process of generalisation (see pages 24–26).

Consider the following two examples of writing a program to make a square.

Example 1:
A program for a Bee-Bot
(forward, left, forward, left, forward, left, forward, left).

Example 2:
As you can see, for each side we first move forward, then turn left. On a Roamer-Too or a Pro-Bot, we could use the repeat command to simplify the coding by using the built-in repeat command, replacing this code with, for example: **repeat 4 [forward, left].**

The same rules apply in Logo, which is the language that the Roamer-Too and Pro-Bot programming device-specific languages are derived from.

Compare the next two examples. Both are to draw equilateral triangles. The second uses repetition; the first does not.

1) Drawing a triangle (without repetition)

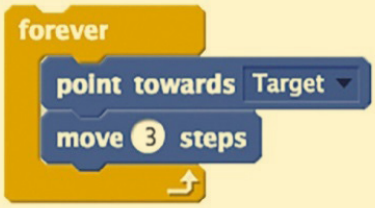
FORWARD 100
LEFT 120
FORWARD 100
LEFT 120
FORWARD 100
LEFT 120

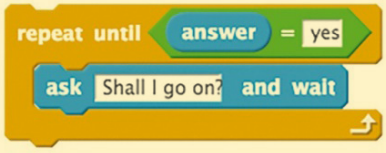
2) Drawing a triangle (using repetition):

REPEAT 3 [FORWARD 100
LEFT 120]

Notice how repetition reduces the amount of typing and how it makes the program reflect the underlying algorithm more clearly. In these examples, the repeated code is run a certain number of times. It's also possible to repeat code continually. This is useful in real world systems, for example in a control program for a digital thermostat. The program checks room temperature continually, and sends a signal to turn on the heating when it drops below a certain value. Such techniques are also common in games programming.

Consider, for example, the following Scratch code which makes a sprite continually chase another around the screen:



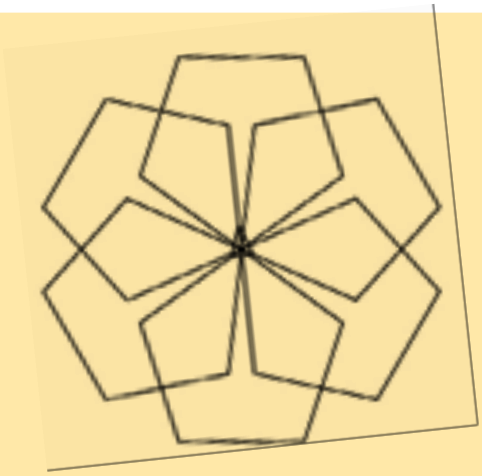


Repetition can be combined with selection. In this way a repeating block of code runs as many times as necessary until a condition is met. Look at the following example, which is a fragment in Scratch:

One repeating block can be nested inside another. 'Crystal flower' programs written in Logo use this idea. Consider the following example:

```
REPEAT 6 [
REPEAT 5 [
FORWARD 100
LEFT 72 ]
LEFT 60 ]
```

draws



Classroom activities

1

Fish tanks

Pupils produce a fish tank animation in Scratch. They use simple repetition commands to make each sprite independently with its own set of repeating motion instructions. They can add additional complexity by including some selection commands to alter the sprites' behaviour when they touch. See Scratch 2.0 Fishtank Game tutorial: www.youtube.com/watch?v=-qTZ5bFEdC8

2

Crystal flowers

Pupils experiment with crystal flower programs. They may use Scratch, Logo or other languages that support turtle graphics. They investigate the effect of changing the number of times a loop repeats as well as the effect of changing the parameters for the commands inside the loop. This activity offers plenty of scope to make links between computing and cultural education.

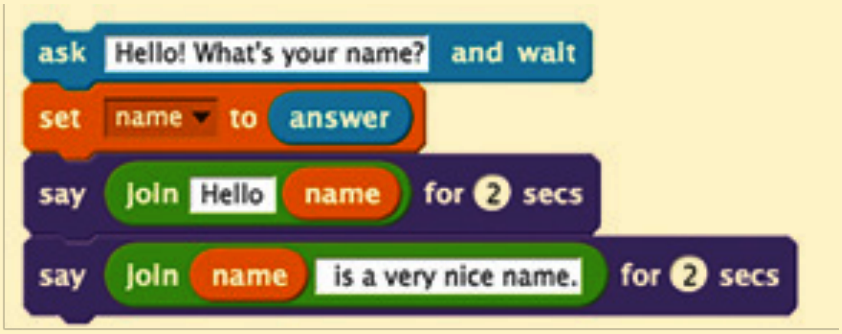
Variables

A variable is a simple way of storing a piece of information in the computer's memory while the program is running, and retrieving that information later.

For pupils in primary school, variables are a sophisticated concept. So if you want to introduce variables to your pupils, it's a good idea to show them plenty of examples to help them understand. A classic example, which pupils are likely to be familiar with from computer games, is score. If you were writing code to make a computer game, you would most likely make a variable called 'score'. This variable would store information about the points gained during a game. When, for example, the character collects a piece of treasure, you instruct the program to increase the variable 'score' by one. As points are collected, the variable keeps changing.

You can use variables to store data input by the person using your program and then refer to that data later on.

The following example, in which the user's name is remembered, comes from code written in Scratch.

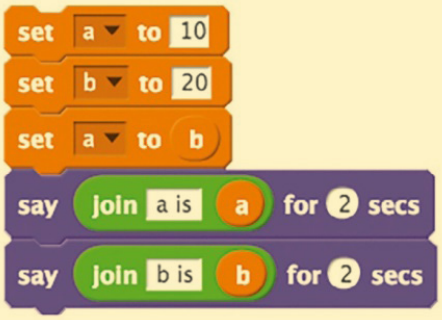


Name as a variable in Scratch

Here, 'name' is a variable. The computer stores whatever the user types in, then uses it twice in Scratch's response; 'answer' is a temporary variable used by Scratch to store for the time being whatever the user types in. As you can see from this example, variables can store text as well as numbers. You can store other types of data in variables too. This depends to a degree on the programming language you're working with.

Inexperienced programmers sometimes find it challenging to grasp the idea that the contents of the 'box' still remain there after the variable is used.

Read the following code and work out what will be displayed on the screen:

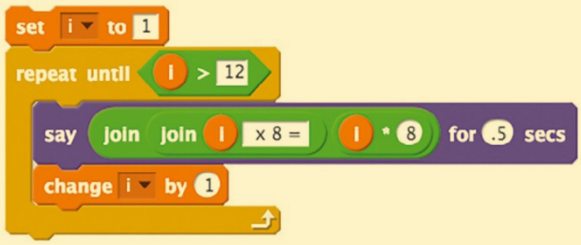


Using variables in Scratch

You should see 'a is 20' followed by 'b is 20'. Try it!

One way to make good use of variables in a program is to use them as an iterator. This counts the number of times a repeating loop has been completed. Simply set a counter to zero or one at the start of the loop, then add one each time the loop is completed.

For example, the following script in Scratch calculates the eight times tables.



Using an iterator

An iterator can also work with words and sentences (or strings as they're called in computing) one letter at a time, or through lists of data one item at a time. Be careful with the start and the finish. Beginning or ending too late or too soon is a common mistake when setting up iterators.

Classroom activities

1

Mystery function machine

Pupils create a mystery function machine in Scratch. This accepts an input, stores it in a variable, then uses mathematical operators to produce an output which is shown on screen. With the display set to full screen, pupils can challenge one another (and you) to work out what the program does by trying different inputs.

2

Games

Pupils use variables in their Scratch games programs. They use scoring to reward a player for achieving particular goals (for example, collecting apples), and they set a time limit.

Debugging is a vital part of writing code, but doing it can be more time consuming than writing the code in the first place!

Debugging: Can we fix it?

‘Bugs’ is the term given to mistakes in code and in algorithms. The process of searching for and repairing bugs is known as ‘debugging’.

Debugging is a vital part of writing code, but doing it can be more time consuming than writing the code in the first place! Bear this in mind as you plan coding activities. And remember that while debugging successfully and completing a working program brings great satisfaction, poring over code that refuses to work can cause great frustration. Be prepared to manage this in class.

It’s useful for pupils in Key Stage 2 to use logical reasoning to find and fix errors in algorithms and programs. Pupils can correct their code and explain what went wrong and how they fixed it.

In programming classes, pupils writing a program for a particular purpose might want you or others to repair their programs. While your first instinct might be to help, please remember that the objective is not so much to have pupils create a working program, as for them to learn how to program. Debugging skills are a crucial part of that.

How do you debug?

It’s good practice to provide a robust, general set of debugging strategies which pupils can use for any programming activity.

Logical reasoning should underpin debugging. This is apparent in the approach of iCompute. They suggest a logical debugging sequence of four steps.

1. Predict what should happen.
2. Find out exactly what happens.
3. Work out where something has gone wrong.
4. Fix it.

One way to help predict what should happen is to get pupils to explain their algorithm and code to someone else. In doing so, it’s quite likely that they’ll spot a flaw in the way they’re thinking about the problem or in the way they’ve coded the solution.

In finding out exactly what happens, it can be useful to work through the code, line by line. Seymour Papert described this as ‘playing turtle’. So, in a turtle graphics program in Logo (or similar) pupils could act out the role of the turtle, walking and turning as they follow the commands in the language.

In working out where something has gone wrong, encourage pupils to look back at their algorithms before they look at their code. Before they can begin to fix bugs, they need to establish whether it was flaw in their thinking or with the way they've implemented that as code.

Some programming environments allow you to step through code one line at a time – you can do this in Scratch by adding (wait until [space] pressed) blocks in liberally. Scratch will default to showing where sprites are and the contents of any variables as it runs through code, which can also be useful in helping to work out exactly what caused the problem.

Debugging is a great opportunity for pupils to learn from their mistakes and to get better at programming.

Classroom activities

1. **Fixing pre-bugged programs**
While pupils will probably make plenty of authentic errors in their own code, it's also worth training them to debug by giving them other programs to fix. This can help them learn strategies for debugging. It also helps you assess their skills in logical reasoning as well as their programming knowledge. Create some programs containing deliberate mistakes, perhaps using a range of semantic errors (errors in the logic of the code). Set pupils the challenge of finding and fixing these.

2. **Peer Debugging**
Pupils debug each other's programs. One way to manage this is that pupils work on their own programs for the first part of the lesson then take over their partner's project. Next they complete this project then they debug it for their partner.

3. **Deliberate peer errors**
Pairs of pupils to write code containing deliberate mistakes. They challenge to their partner to discover and repair the errors in the code.

Further reading and resources

General

The following sites contain a wealth of practical and engaging ideas, information and approaches.

iCompute for Primary Schools

https://icompute-uk.com

iCompute supports primary school teachers, equipping them with the confidence, knowledge, skills and resources to teach computing creatively and with confidence. BETT and ERA Award nominated each year since 2013, featured on BBC Bitesize and the Hour of Code this site includes step-by-step lesson plans and thousands of primary computing resources.

BBC Bitesize

http://www.bbc.co.uk/education

The Bitesize site has a range of learner guides and programmes to support primary school pupils on all aspects of computing. Follow the links from the main site to KS2 and then to Computing.

Wider reading

Liane O'Kane

https://icompute-uk.com/news
The blog of Liane O'Kane, Primary Computer Science Master Teacher, Teacher/ Trainer, contains many of her articles, resources and other publications on computing education.

Topics

Computer Science Teachers Association, 'CSTA Computational Thinking Task Force' and 'Computational Thinking Resources':
http://csta.acm.org/Curriculum/sub/CompThinking.html

Computing at School, 'Computational Thinking':
http://community.computingatschool.org.uk/resources/252

Curzon, P, Dorling, M, Ng, T, Selby, C and Woollard, J, 'Developing Computational Thinking in the Classroom: A Framework' (Computing at School, 2014), available at:
http://community.computingatschool.org.uk/files/3517/original.pdf

Google for Education, 'Exploring Computational Thinking':
www.google.com/edu/computational-thinking/index.html

Wing, J, 'Computational thinking and thinking about computing' (The Royal Society, 2008):
http://rsta.royalsocietypublishing.org/content/366/1881/3717.full.pdf+html

Logical reasoning

Computer Science for Fun, 'The Magic of Computer Science', available at:
www.cs4fn.org/magic/

Computer Science Unplugged, 'Databases Unplugged', available at:
http://csunplugged.org/databases

McOwan, P and Curzon, P (Queen Mary University of London), with support from EPSRC and Google, 'Computer Science Activities with a Sense of Fun', available at:
www.cs4fn.org/teachers/activities/braininabag/braininabag.pdf

Decomposition

NRICH, 'Planning a School Trip', available at:
http://nrich.maths.org/6969

Glossary

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

Acceptable Use Policy (AUP): An Acceptable Use Policy comprises a set of rules applied by the owner/manager of a network, website or large computer system that defines the ways in which the network, site or system may be used.

Algorithm: An unambiguous set of rules or a precise step by step guide to solve a problem or achieve a particular objective.

Command: An instruction for the computer to execute, written in a particular programming language.

Computational thinking: Thinking about systems or problems in a way that allows computer systems to be used to model or solve these.

Computer networks: The computers and the connecting hardware (wifi access points, cables, fibres, switches and routers) that make it possible to transfer data using an agreed method ('protocol').

Data: A structured set of numbers, possibly representing digitised text, images, sound or video, which can be processed or transmitted by a computer, also used for numerical (quantitative) information.

Debug: To fix the errors in a program.

Decomposing/ decomposition: The process through which problems or systems are broken down into their component parts, each of which may then be considered separately.

E-safety: Used to describe behaviours and policies intended to minimise the risks to a user of using digital technology, particularly the internet.

Generalisation: A computational thinking process in which general solutions or models are preferred to or derived from particular cases.

Hardware: The physical systems and components of digital devices; see also software.

Input: Data provided to a computer system, such as via a keyboard, mouse, microphone, camera or physical sensors.

Interface: The boundary between one system and another – often used to describe how a person interacts with a computer.

Loop: A block of code repeated automatically under the program's control.

Open source software: Software in which the source code is made available for others to study, and typically adapt, usually with few if any restrictions.

Operating system: The programs on a computer which deal with internal management of memory, input/output, security and so on, such as Windows 8 or iOS.

Output: The information produced by a computer system for its user, typically on a screen, through speakers or on a printer, but possibly through the control of motors in physical systems.

Packets of data: A small set of numbers that get transmitted together via the internet, typically enough for 1000 or 1500 characters.

Program: A stored set of instructions encoded in a language understood by the computer that does some form of computation, processing input and/or stored data to generate output.

Programming (or coding) is the term given to the writing of programs.

Programmable toys: Robots designed for children to use, accepting input, storing short sequences of simple instructions and moving according to this stored program.

Repetition: Executing a section of computer code a number of times as part of the program.

Script: A computer program typically executed one line at a time through an interpreter, such as the instructions for a Scratch character.

Selection: A programming construct in which one section of code or another is executed depending on whether a particular condition is met.

Sequence: To place program instructions in order, with each executed one after the other.

Simulation: Using a computer to model the state and behaviour of real-world (or imaginary) systems, including physical or social systems; an integral part of most computer games.

Software: computer programs such as Office programs, web browsers, games and the computer operating system. The term also applies to apps running on mobile devices and to web-based services.

Sprite: A computer graphics object that can be controlled (programmed) independently of other objects or the background.

Variables: A way in which computer programs can store, retrieve or change data, such as a score, the time left, or the user's name.

Technology

Technology

What is a computer?

The term 'computer' originally referred to *people* whose job it was to perform repeated numerical calculations according to a set of instructions (i.e. an algorithm). Since the 1940s it has been used to refer to digital machines that accept data input, process this according to some set of stored instructions (i.e. a program) and output some sort of information.

The power of digital computers comes from their ability to run through these stored instructions incredibly quickly. The silicon chip at the heart of a modern smartphone can execute over a billion instructions per second!

A digital computer comprises two inter-related systems.

- Hardware: the physical components, including the processor, memory, power supply, screen, etc.
- Software: the core **operating system**, embedded control programs, compilers or interpreters and many application programs.

There is an incredible variety of electronic devices that contain some sort of digital computer. There are two different types of device:

Computer-controlled for specific purpose

- digital watch
- digital television
- digital camera ...

Programmable computer – can do many different things

- laptop
- tablet
- smartphone ...

How do computers remember things?

The memory of a computer stores both the programs it needs to operate and the **data** that it processes. There are different types of computer memory and usually there's a trade-off between speed and cost. These days, high capacity storage has become very cheap, so that data centres can provide users with vast amounts of storage for little or no cost through services such as Microsoft OneDrive and Google Drive.

Irrespective of where programs or data are stored in computer memory, they are always stored in a digital format. Information is represented as sequences of numbers. The numbers themselves are stored in a binary code, represented using just two symbols: 0 and 1 (this number system is called base 2). Each 0 or 1 is called a bit.

A range of standard codes are used to convert machine code, images, sound or video into a digital format. These provide standard ways to represent information of different types in binary. Text data is encoded in Unicode. A byte is a group of eight bits; it's used as a unit of memory. Eight bits are more than enough to store one character from the Latin alphabet, in upper or lower case, a punctuation symbol, a digit, etc. One thousand bytes make a kilobyte: enough to store 1000 characters (a short paragraph).

Images, sound and video have their own accepted standards for being encoded digitally, such as bitmaps for images or 'WAV' files for audio. These typically take up much more room than text, so often a form of compression is used (where patterns in the data help reduce the amount of storage space needed). If the original data can be recovered perfectly this is

called lossless compression. If some of the original information is thrown away, the original image, sound or video can be stored in a much more compact format, although some of the original quality is lost in the process: this is 'lossy' compression.

Interestingly, the key stage 2 programme of study is more concerned with how information is communicated than how it's stored, but binary representation should be covered in:

- 'work with … various forms of input and output'
- 'understand **computer networks**, including the internet'.[1]

How do computers interact with the real world?

In order for a computer to be able to do anything in the real world, it needs some form of input (to receive data) and some form of output (to push information back out).

The form of input will vary:

Laptop inputs
- keyboard
- trackpad/touchpad
- microphone
- webcam
- through a port (e.g. USB mouse)
- via a network connection …

Smartphone inputs
- touch-sensitive screen
- buttons
- microphone
- camera
- GPS receiver
- accelerometer
- barometer
- through a port
- via a network connection …

A computer will need to convert the analogue, real-world data it receives into a digital format before it can be processed, stored or transmitted. We call this process digitisation and it inevitably involves throwing away some of the fine detail of the real-world information.

Computers can produce many different forms of output:

Laptop/desktop PC outputs
- screen
- speakers
- printer
- headphones
- network connections …

Smartphone/tablet outputs
- screen
- speakers
- small motor to produce vibrations
- bright LEDs used as a flash
- network connections …

What is a robot?

A robot is a computer that can move. This could be a single, integrated system such as a **programmable toy**, or it could be a motor under a computer's control, such as a robotic arm in manufacturing.

Robots are used widely in industry, where repetitive tasks can be performed effectively and efficiently by machines. As 'smarter' algorithms have been developed by computer scientists, more and more decision-making capabilities can be built in to the robot, so that it can autonomously react to changes in its environment.

Further resources

- 'Arduino the cat, Breadboard the mouse and Cutter the Elephant': video of a group of girls planning and programming soft toys, available at: http://vimeo.com/4313755.
- BBC Cracking the Code: Miniature computers, available at: www.bbc.co.uk/programmes/p01661f7.
- BBC Cracking the Code: Robots, available at: www.bbc.co.uk/programmes/p01661tn.

[1] National Curriculum in England, *Computing Programmes of Study* (Department for Education, 2013).

Networks

How do computers communicate?

Connecting computers to form computer networks and the internet (a network of networks) has had a huge impact on our lives.

Think about how limited our use of technology in school would be if we had no access to the local network or the internet. Think about how frustrating it is when we have no data signal for our smartphones or wifi for our laptops.

The internet has made it possible to communicate and collaborate with a richness and immediacy never experienced before. And yet, it's something that most of us take for granted.

The new computing curriculum sets out to change this. It requires that pupils are taught:

- in key stage 1: to 'recognise common uses of information technology beyond school'
- in key stage 2: to 'understand computer networks, including the internet [and] how they can provide multiple services, such as the world wide web'
- in key stage 2: to 'use search technologies effectively, appreciate how results are selected and ranked, and be discerning in evaluating digital content'.[1]

[1] National Curriculum in England, *Computing Programmes of Study* (Department for Education, 2013).

How does the internet work?

The internet is a physical thing: it's the cables, fibre, transmitters, receivers, switches, **routers** (and all the rest of the **hardware**) that connects computers, or networks of computers, to one another.

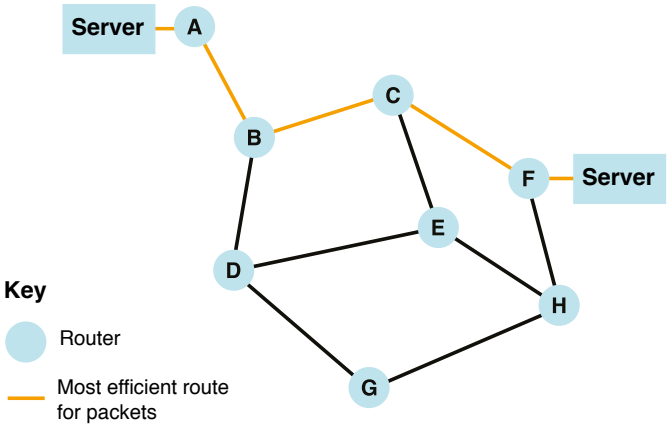
The internet has been designed to do one job: to transport **data** from one computer to another. This information might be an email, the content of a web page, or the audio and video for a video call.

The data that travels via the internet is digital: this means it is expressed as numbers. All information on the internet is expressed this way, including text, images and audio. These numbers are communicated using binary code, which is made up of 1s and 0s, using on/off (or low and high) electrical or optical signals. Binary code is similar to the Morse code used for the telegraph in Victorian times, but it's much, much faster. A good telegraph operator could work at maybe 70 characters (letters) a second, but even a basic school network can pass data at 100 million on/off pulses a second, enough for some 12.5 million characters per second. One transatlantic fibre connection has the capacity for up to 400 billion characters per second!

Digitised information needs to be broken down into small chunks by the computer before it can be sent efficiently. These smaller chunks of data are known as 'packets'.

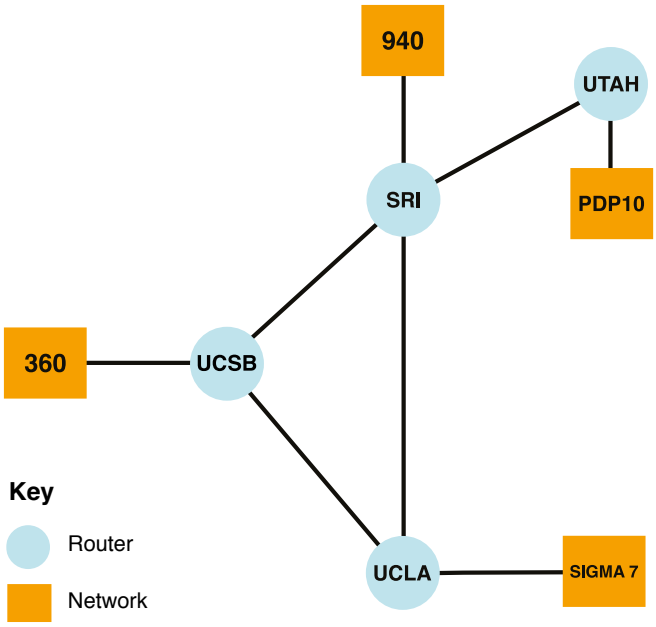
The small **packets** can be passed quickly through the internet to the receiving computer where they are re-assembled into the original data. The process happens so quickly that high definition video can be watched this way, normally without any glitches.

The packets don't all have to travel the same way through the internet: they can take any route from sender to recipient. However, there is generally a most efficient route, which all the packets would take.



A sample network: note there is more than one route for packets to travel.

It's perhaps easier to understand how the internet works now by looking at a picture of how it worked in 1969 when it started:



SRI Stanford Research Institute
UTAH University of UTAH
UCSB University of California, Santa Barbara
UCLA University of California, Los Angeles

Here you see the internet made up of just four routers: UCLA, SRI, UCSB and UTAH. Each router is a piece of hardware that passes packets of data from the networks they are connected to (in the case of UTAH, PDP10, in the case of UCLA, SIGMA 7) to any of the other three networks.

So if you were using the PDP10 network at the University of UTAH and sent a message to someone at UCLA, your message would be passed first to your router at UTAH, then on to the router at Stanford Research Institute (SRI), then (normally) to UCLA's router, where it would be passed on to whichever recipient it was intended for on their SIGMA 7.

The internet is obviously much, much bigger than this example. In real life, the journey of a packet of data from your home computer to one of Microsoft's server farms might look something like this:

your home wifi access point
↓
your home switch and router
(usually all in the same black box)
↓
switches in your nearest BT green cabinet
↓
more switches in your local telephone exchange
↓
London internet exchange
↓
routers near Porthcurno in Cornwall
↓
fibre optics under the Atlantic
↓
further switches and routers in the USA
until Microsoft's internet connection at whichever of its data centres you are communicating with

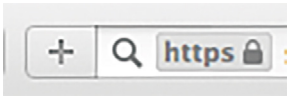
When you type a URL (such as www.bbc.co.uk or www.icompute-uk.com) into your browser you send a packet of data requesting the content of these pages to be returned to you. But before this can happen, the domain name first needs to be converted into numbers. This is the job of the **Domain Name Service (DNS)**, which converts these familiar web addresses into numbers known as **IP (Internet Protocol) addresses**. The DNS itself uses the internet to look up (in the equivalent of huge phone books) the numeric address corresponding to the domain names.

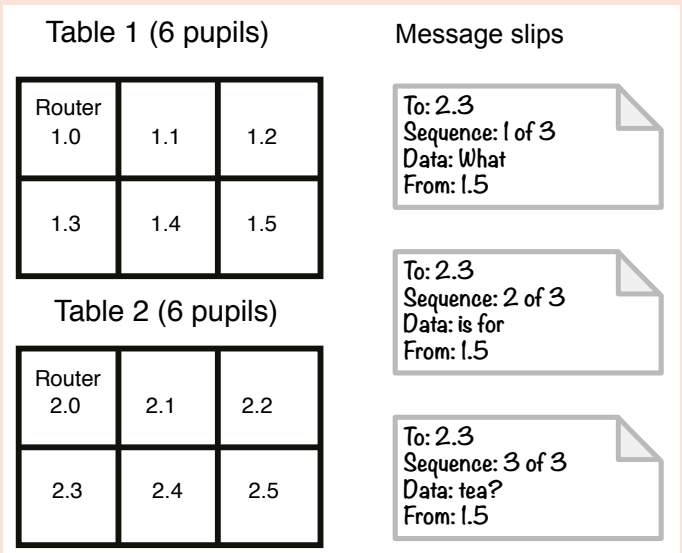
Each packet has a destination IP address on it. With it the router can easily look up which way to pass the packet on.

Who can see the data we transmit?

There's nothing to stop routers from looking at the data in the packet before they pass it on (just as there was nothing to stop telegraph clerks reading the messages they passed on in Morse code).

To be able to send information, such as passwords or bank account details, secretly via the internet, it's important to **encrypt** the data first. This happens automatically when using the 'https' version of websites (see page 37). In these situations, you'll see a little green padlock displayed in your browser's address bar. The data is decrypted when it reaches its destination.





Role-playing a computer network in class.

Classroom activity ideas

- Ask pupils to draw a picture of the internet. This will allow you to spot any misconceptions they have, and provide an opportunity for pupils to share their understanding.
- Carry out this 'unplugged' activity to model how the internet passes packets of data.
  - » Organise all but four of your pupils into groups.
  - » Tell the pupils to choose one pupil in their group to be the 'group router'. The rest of the group will be 'computers'.
  - » Ask the remaining four pupils to take on the role of 'internet routers', which connect the group routers together.
  - » Give each 'computer' a numerical address, comprising a group number and a computer number (e.g. 1.1, 1.2, 1.3; 2.1, 2.2, 2.3, etc.).
  - » Ask each 'computer' to write a short message to another 'computer' in a different group, splitting their message over three different slips of paper and marking their slips '1 of 3', '2 of 3' and '3 of 3'. Tell them to write their numerical address and the numerical address of the recipient, e.g. 'To: 2.2; From 3.4; 2 of 3.' This is the 'packet header'.
  - » Ask the 'computers' to pass their slips to their 'group router', who can pass these on one at a time to the 'internet routers'. They in turn pass them to the correct 'group router' who passes them to the recipient themselves, who can reassemble the message as their other packets arrive.
- Investigate the physical infrastructure of the school network. Tell the pupils to walk from their laptop to the local wifi point, or to follow the network cable from the computer to the classroom switch. Next, walk together to the school's main network switch, firewall and router. If you can, then walk down to the nearest BT green cabinet, and perhaps to your local telephone exchange, depending on how close this is to you.
- Explore the steps on the journey of a packet using the 'tracert' command at the Windows command prompt, if you have access to this. Also see the Visual traceroute reference in Further resources.
- Ask your school network manager to talk pupils through how the school network connects their computers to the rest of the internet.

Further resources

- iCompute., 'iWeb & iNetwork', available at: www.icompute-uk.com
- BBC Bitesize clip 'Computer networks – LAN and WAN', available at: www.bbc.co.uk/learningzone/clips/computer-networks-lan-and-wan/4381.html.
- Blum, A., *Tubes: Behind the Scenes at the Internet* (Viking, 2012).

- Andrew Blum's talk 'Discover the physical side of the internet', available at: www.ted.com/talks/andrew_blum_what_is_the_Internet_really.
- Artistic representations of what the internet means, available at: www.canyoudrawtheinternet.com.
- Naughton, J., *From Gutenberg to Zuckerberg: What You Really Need to Know About the Internet* (Quercus, 2012).

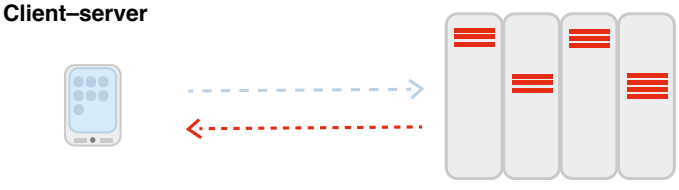
What can you do with the internet?

Picture the train network, efficiently routing trains of all kinds from one point to another, irrespective of what those trains contain. Some will have passengers, others freight, others are perhaps maintenance stock. In the same way, the infrastructure of the internet can be used for lots of different things.

The services which run on computer networks, including the internet, fall into roughly two groups:

1. client–server: one computer (the client) accesses services or content running or stored on another, typically larger, computer (the server)
2. peer-to-peer: two computers communicate directly as equals, passing data directly to and from each other.

Client–server



Client computer — Server computer

Peer-to-peer

Client computer — Client computer

The World Wide Web (see page 36) fits into the client–server model, but so do lots of other services which use computer networks and the internet as a means of communicating.

A school network will often have one or more computers acting as servers, responding to requests from the desktop, laptop and tablet computers which act as clients. On a local area network (LAN) like this, the servers might provide: central storage and backup for files, access to documents, etc. from any computer on the network, a management information system (such as SIMS), local email accounts, access to printers, username and password authentication, filtering and logging of access to the web and even locally stored copies of frequently visited web pages.

Email is a good example of a client–server system using the internet (although many people's experience of email is as webmail accessed through a browser like Internet Explorer). The journey of an email might be something like this:

- Alice opens up Outlook and starts typing in her email to Bob. She includes Bob's email address, bob@builders.com, in the 'To' line of the email and clicks 'send'.
- The email is transmitted via the internet (or the local network) to her outgoing mail server. If the email is intended for another domain (builders.com here) rather than Alice's own (lookingglass.org) then Exchange will forward the email as packets of data via the internet, which routes these through to the incoming mail server for builders.com as discussed above.
- The inbound mail server at builders.com (again perhaps running Exchange) re-assembles the message from the packets of data, accepts this and stores this ready for Bob to collect.
- Later on, Bob's email client (perhaps also Outlook) connects to his mail server and asks if there are any messages for him. The one from Alice gets transmitted to Bob's computer via the local network or the internet, where Bob can read it in his email software.

Although it might look to Alice and Bob as though they are communicating directly with each other, all their emails are going via the outbound and inbound mail servers. Notice that the contents of their emails aren't encrypted, so the organisations running the two mail servers can read the contents of these messages if they wish.

Not all communication on the internet uses a client–server model. For example, peer-to-peer communication is a model used for Skype and a number of other video conferencing or voice over internet systems. Although Skype uses a server to maintain a list of logged-in users and the IP address of their computers, when a call is

connected the packets of data that make up the digitised video and audio for the call are routed directly through the internet between the two parties.

Some online gaming websites use a similar peer-to-peer system, as does BitTorrent, a protocol which allows large files to be shared between many computers by allowing direct peer-to-peer connections. Because peer-to-peer connections are harder for large organisations to monitor, they are favoured by those using the internet for criminal purposes, for example the use of the BitTorrent protocol for illegally sharing copyrighted material.

Classroom activity ideas

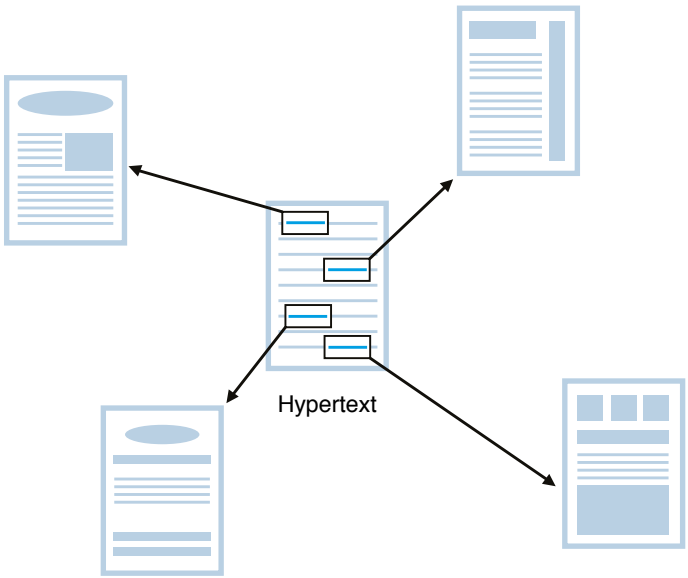
- Role-play can be used very effectively to teach how email works and issues with email security. Explain to pupils that email addresses can be 'spoofed' or accounts hacked. So, not all emails are from who they appear to be. Warn them that files attached to emails can contain viruses. Also explain that links in emails can sometimes point to websites that are set up to capture personal information such as passwords. You might like to run this as part of a larger topic looking at the effective and safe use of email, perhaps in a twinning project with a class in this or another country.
- Share and write a range of emails and written letters. Discuss the advantages and disadvantages of each type of communication.
- Use a video conferencing system to allow experts to talk to the class or to allow two classes to communicate. As you set up the computer, talk through the technical aspects of the call with your pupils. Note: Skype and most other video conferencing systems don't allow children to register for accounts, so you will need to run this as a whole-class activity.
- Encourage pupils to talk about how they and their families use the internet to communicate, highlighting any services they use in addition to the World Wide Web.

Further resources

- Guha, S., Daswani, N. and Jain, R., 'An Experimental Study of the Skype Peer-to-Peer VoIP System' (2006), available at: http://saikat.guha.cc/pub/iptps06-skype.pdf.
- The journey of a letter, available at: www.anpost.ie/anpost/schoolbag/primary/our+people/the+journey+of+your+mail/.
- 'Story of Send on Google Green' (a short cartoon about the journey of a gmail), available at: www.youtube.com/watch?v=5Be2YnlRIg8.

What is the World Wide Web?

In 1989, British computer scientist Tim Berners-Lee decided to combine the capabilities of the internet with the functions of hypertext (documents that include hyperlinks that allow connections to be made between different files) to manage information systems at CERN where he was working.

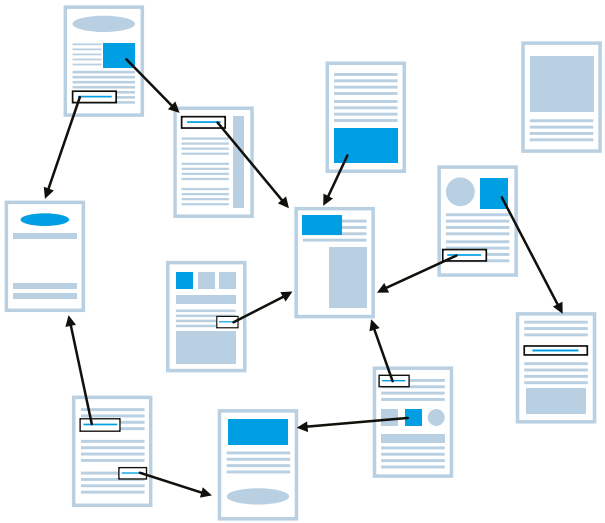


The links in the hypertext take the reader to different documents which extend or support the information in the original document.

Berners-Lee developed a specification for how an internet-based version of hypertext would work and then wrote the software for the first **web servers** and web browsers. The result was the **World Wide Web**.

The internet is about connecting computers together, but the World Wide Web is about the connections between documents. When you click on a web link, another web page is requested from (typically) a different web server somewhere else on the internet.

The content of this web page is then delivered to your web browser.



The World Wide Web is about the connection (the links) between documents.

What standards does the World Wide Web use?

To ensure that all computers could communicate with one another, Berners-Lee developed a set of standards (called protocols) for the Web. Versions of these are all still used today.

1. HTTP (HyperText Transfer Protocol)

This is the process that computers use to request and transfer hypertext to one another.

The Web is a client–server system: we use a web browser on our computer to request a web page from one of the many, many web servers connected to the internet. The request travels as a packet of data via switches and routers until it reaches the intended web server. The server responds by sending back the content of the page, together with any images and formatting instructions and mini programs (typically in JavaScript) needed for the page. If the page isn't there, it sends back a '404: Not found' error message – sometimes you'll see other error messages too.

Remember that the internet doesn't encrypt packets of data: there's another version of HTTP, called HTTPS, where the request for a page, the contents of the page and any information entered into a form (such as a password) are sent over the internet in an encrypted form. This encryption can sometimes be bypassed by network managers and government agencies.

2. URL (Uniform Resource Locator)

URLs are the precise location on the Web where web pages or their components are stored. It's what you type in to your browser's address bar to request a page.

Each bit of a URL means something. Let's look at the URL of one of the first web pages – Berners-Lee's home page for the World Wide Web project itself – to work out what each bit means:

http://info.cern.ch/hypertext/WWW/TheProject.html

- **http:** this is the protocol we're using to request hypertext and the content that comes back – see above.
- **://** is just punctuation – Berners-Lee now thinks it would have been better if he'd skipped the // bit!
- **info** is the name of the web server we're connecting to. Often this will be www these days, or this is just omitted as the main web server for the organisation will be assumed.
- **cern** is the name of the organisation, in this case the European Centre for Nuclear Research.
- **ch** is an abbreviation for the country where the organisation has registered their domain name, in this case Switzerland. Some countries also show what sort of organisation it is registered as, e.g .co.uk for a commercial site and .sch.uk for a school site in the UK. If no country is shown, then it will be registered in the USA: .com for commercial sites, .edu for university sites, and so on.
- **/hypertext** is a directory (folder) on the web server.
- **/WWW** is a directory inside the /hypertext directory on the web server.
- **TheProject** is the name of the actual file we're requesting, in this case a web page about the World Wide Web project. Sometimes you don't see a file name at the end of a URL, in which case the web server will send back the default file for the directory, often an index page such as index.html.
- **.html** is the file extension, which shows what format the page is written in, in this case **HTML** (see page 38). This is like .doc or .docx for a Word file, or .jpg or .jpeg for an image.

Although it is often convenient to use search engines like Google or Bing to find pages rather than typing in URLs, the URL is a good way to check that you're connecting to the web server you think you are (rather than a spoof website). URLs are also useful when acknowledging sources of information, and for creating links between pages (and so building more of the connections that make the Web so useful).

HTML (HyperText Mark-up Language)

HTML is the computer language (code) in which the content and structure of a web page are described or 'marked up'.

The content of web pages is stored in HTML format on web servers. Creating a web page involves writing (or getting a computer to generate) the HTML that describes the page. HTML can be read, and written, by humans as well as computers. You can view the HTML source code for any web page using tools built into your web browser. (There's a menu command to do this, or you can press 'ctrl-u' in Internet Explorer.)

These days, the HTML for a web page might not be stored as a file on the web server: in content management systems, when a page is requested it will be generated automatically using a database of content, a template and some programs running on the web server. For example, every time you visit www.bbc.co.uk/newsround/ the page will be generated using the latest news in the database.

More recently, a couple of other languages have come to play an important part in developing for the Web.

CSS (Cascading Style Sheets)

CSS provides formatting information alongside the content and structure of HTML, allowing designers and developers to specify exactly how the content of the page should be displayed in the web browser on a computer, tablet, smartphone or printer.

JavaScript

JavaScript is a programming language that can be interpreted by the web browser itself, allowing interaction with the content of a page to be handled by the user's computer (the client) rather than on the server itself. The web-based version of Office 365 relies heavily on JavaScript.

What's the most amazing thing about the Web?

The amazing thing about the Web isn't really these technologies though. It's that, from its early days as the preserve of academic scientists, so many organisations and individuals have connected their own web servers to the internet and added their own content to the Web. In part this was because Berners-Lee created a system that was accessible, scalable and extensible, capturing the imagination of many. But it's also because he and CERN gave it to the world for free – the standards and the technology were entirely open, without any central authority or commercial company licensing or charging for their use.

Classroom activity ideas

- The national curriculum for history suggests that key stage 1 pupils could look at William Caxton (who introduced the printing press to England in the fifteenth century) and Tim Berners-Lee as examples of 'the lives of significant individuals in the past who have contributed to national and international achievements'. Compare the life, work and influence of these two figures.
- Encourage pupils to look at the different parts of the URLs for the web pages they visit, asking them to explain what each part of the URL means. Make a display showing the different parts of some interesting or common URLs.
- Ask pupils to talk to their parents, grandparents or carers about the difference the World Wide Web has made in their lives.
- Tell pupils to keep a diary of the different ways they use the Web over a week.

Further resources

- BBC Bitesize 'What is the world wide web?', available at: http://www.bbc.co.uk/guides/z2nbgk7.
- Tim Berners-Lee 'Answers for Young People', available at: www.w3.org/People/Berners-Lee/Kids.html.
- The original CERN home page for the Web, available at: http://info.cern.ch/hypertext/WWW/TheProject.html.
- Codecademy curriculum materials, available at: www.codecademy.com/schools/curriculum (registration required).
- Mozilla Web Literacy whitepaper, available at: http://mozilla.github.io/webmaker-whitepaper/.
- Wayback Machine to search for historic web pages, available at: http://archive.org/web/.

How do you make a web page?

There are plenty of tools available for you and your pupils to create your own content for the Web.

Your school's learning platform or VLE provides one way to get content online, as do blogging platforms

like WordPress. These platforms usually include a 'WYSIWYG' (what you see is what you get) editor. This makes writing content for the Web similar to using Microsoft Word, with a range of formatting controls built in. In most of these editors, you can swap into code (or source view), seeing and editing the HTML itself. This can be a good introduction to working directly in HTML, as you can always swap back to the WYSIWYG view to see the effects of editing the code.

Giving pupils some experience of writing content for the Web through editing HTML 'by hand' is well worth doing although it isn't, strictly speaking, programming. It adds to their understanding of networks including the internet that the national curriculum at key stage 2 expects, and is one more way of using software on a range of devices to create content. It is also a good way to get pupils used to working in a formal, text-based computer language. As with other text-based languages, working in HTML helps reinforce the importance of spelling, punctuation and grammar: mistakes in the mark-up of the page usually become quite apparent in the way the browser displays the page.

Many pupils are likely to find these skills useful in the long term too, both at secondary school and beyond: developing content for the Web is part of many jobs, teaching included.

What does HTML look like?

Let's compare the HTML code for a simple web page and the page itself.

```
<!doctype html>
<html>
  <head>
      <meta charset="utf-8">
    <title>A simple webpage</title>
  </head>
  <body>
    <h1>Origins of the Web</h1>
    <p>Tim Berners-Lee started working on
the world-wide web project in 1989.</p>
    <p>He was working at <a href="http://
home.web.cern.ch/">CERN</a> in Switzerland
at the time.</p>
    <img src="http://upload.wikimedia.org/
wikipedia/commons/thumb/7/7e/Tim_Berners-
Lee_CP_2.jpg/320px-Tim_Berners-Lee_CP_2.
jpg">
  </body>
</html>
```

Can you see where the content for the page comes from in the code? Can you see what effect some of the HTML tags (the bits in the <...> angle brackets like <h1> and) have on how the content is structured?

Notice how most of the tags come in matched pairs, e.g.

- <html> and ending </html> for the whole page
- <head> to </head> for the information about the page, such as its character set and title
- <body> to </body> for the content of the page
- <h1> to </h1> around the main heading for the page
- to around each paragraph.

Compare the underlined link in the web page with the corresponding code. In the code, <a> to show where the link should be and `href="http://home.web.cern.ch/"` inside the <a> tag detail where the link should point to.

An image is inserted from elsewhere on the web, using a single <img> tag, this time without a matched closing tag, and again giving the location of the image using `src="http://upload.wikimedia.org/wikipedia/commons/thumb/7/7e/Tim_Berners-Lee_CP_2.jpg/320px-Tim_Berners-Lee_CP_2.jpg"` inside the <img> tag.

How do I get started with HTML?

Mozilla's Thimble tool for creating websites (available at: https://thimble.webmaker.org/) makes it easy to get started with coding in HTML, as it displays the source code alongside the resulting web page.

Rather than starting from a blank page, pupils can try editing other web pages, exploring the structure and HTML code of these pages and seeing what effect changing the code has on how the page is displayed in the browser.

On Internet Explorer, you can use the Developer Tools (hit F12, or launch via the menu) to view and edit the source code (the HTML code which describes the content and structure) for a page. Alternatively, you can install Mozilla's X-Ray Goggles as an active bookmarklet (see Further resources) to remix and share edited web pages.

Classroom activity ideas

- When using their learning platform, VLE or class blog, encourage pupils to swap from the normal WYSIWYG (what you see is what you get) mode of the built-in editor into the code, source or HTML mode and try writing their post or page in that. Remind them that they can swap back and forth to see how the code relates to the page that's displayed. Give pupils a list of some common HTML tags to try out for themselves.
- Set pupils the challenge of making a parody of a web page by using either the Developer Tools in Internet Explorer or X-Ray Goggles to edit the code for the page. It's wise to decide some ground rules for this activity in advance. Show pupils how easily a spoof page can be created this way, and explain why it's so important to check the address of the page they're visiting to confirm it is authentic rather than merely one which looks convincing.
- Rather than asking pupils to write up a story or a report using Word, challenge them to do this using HTML code to make a web page. Emphasise that they need to concentrate on the content and structure of their page, which is what HTML is designed for. Encourage them to add in links to supporting material using the `<a>` tag if they're creating a non-fiction account, and perhaps to add in some images from elsewhere on the Web using the `<img>` tag.

Further resources

- Learn to code tutorials from Codecademy, available at: www.codecademy.com/ (registration required).
- Shay Howe 'Learn to Code and CSS' tutorials, available at: http://learn.shayhowe.com/.
- 'App Design Basics: Learn to code using HTML and CSS' from Playto, available at: https://learn.playto.io/html-css/lesson/0.
- Thimble: https://thimble.webmaker.org/.
- Tutorials on a wide range of computer languages from w3schools, available at: www.w3schools.com/.
- See the source code behind web pages using X-Ray Goggles, available at: https://goggles.webmaker.org/.

How does a search engine work?

Search engines like Google and Bing have transformed the way we use the Web. Instead of having to remember URLs for the pages we want, or following the links from one page to another, we can normally rely on these web-based programs to give us the most relevant results for our query.

Given how much we use search engines, it's important to use them effectively and efficiently, to show some discernment in deciding how far a particular page can be trusted, and to have some grasp of the **algorithms** that underpin them.

In order for Bing or Google to be able to respond to a search query, they use their index of the Web. A search engine builds its index by using specially written programs called 'web crawlers'. The web crawlers create a huge copy of the publicly accessible bits of the Web (called a cache) which is stored on the search engine's servers.

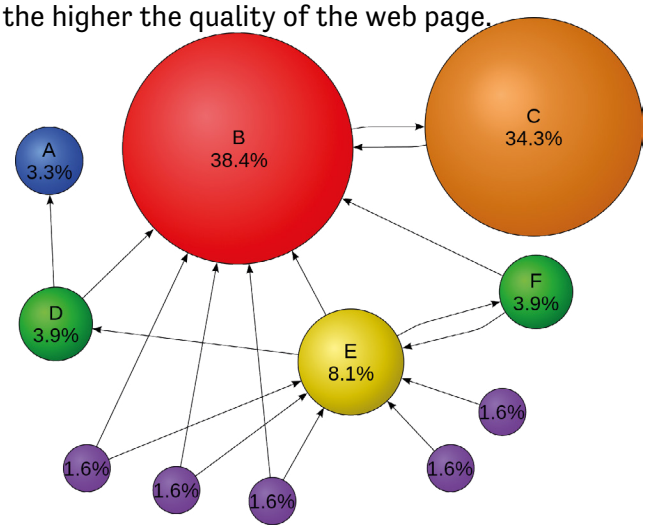
When a new or updated copy of a web page is added to the cache, an entry for the page will be added to, or updated in, the search engine's index of the Web for each of the words on the page (typically ignoring small, common words like 'and', 'the' and so on). The web crawlers continue to build and update the cache by following all the hyperlinks in the page, requesting and making copies of those pages too, adding or updating index entries for them, and following the links on those pages too. And so on.

So when we type in a keyword such as 'dog' into a search engine, it consults the index and returns a list of all the web pages on which that keyword appears. Typing in several keywords, e.g. 'dog' and 'bowl' would only return pages with both of these keywords, which helps to narrow down the set of results.

How are search results ranked?

The really clever bit about web searches is not the list of results, but the right rank order the results are put into. How do the search engine algorithms decide what to put top of the list?

Google's founders, Larry Page and Sergei Brin, recognised that the key to determining how relevant a particular result was likely to be lay in the links between other pages and the result. They realised that a high quality page is a page that has lots of links *pointing to it* from other web pages, particularly if they too were high quality results. This is shown in the illustration below, where the larger the circle, the higher the quality of the web page.



The cached and indexed copy of the (publicly accessible) Web on the servers of search engines also includes the links between them. This allows Page and Brin's PageRank algorithm to work out which pages are considered the highest quality to other web developers (as they add links to those into their own content). Thus, for many queries the Wikipedia entry will often be at the top of, or at least high up, the results list, not because of its accuracy or authority, or even because people click on this more than other results, but because lots of the other high quality search results link to it.

The actual algorithms that search engines use can be very complicated and are frequently tweaked to keep one step ahead of the 'search engine optimisation' (SEO) industry that tries to improve the ranking for its clients' pages. These days, the ranking of results is typically personalised: based on location, the history of what the user's searched for and clicked on before, and close on 200 other factors or 'signals'.

When teaching pupils about how search engines work, point out the 'sponsored' results which are shown above or to the side of those generated using this relevance algorithm. The sponsored results are also algorithmically generated, based on the keyword, some quality measure for the advert, the page it points to and often your search history. They're placed on a 'pay per click' basis: the search engine doesn't charge for showing the advert, but the advertiser pays when you click on it, so it's in their interests to only show the most relevant adverts here.

The mechanics will vary from one search engine to another, but a good search engine should also: filter out explicit content automatically, allow you to search within a particular site, allow results to be filtered by their location (e.g. just the UK) and by date range (e.g. just pages created or edited in the last year). Some search engines even allow results to be filtered by reading level, for example restricting the results to those written using shorter words or less complex sentences.

Classroom activity ideas

- Encourage pupils to use search engines for independent or guided research projects. Get pupils to experiment with the effect that adding in additional keywords or searching for phrases (by putting quotation marks around the phrase) has on a set of results.
- Demonstrate, and ask pupils to use, some of the more advanced search features, such as filtering by date and reading level. Show pupils how they can view the cached copy of a web page (for both Google and Bing this is hidden under the green drop-down next to the URL on the results page).
- Read through the Digital Schoolhouse notes on a simulation of how a search engine works, based on Google engineer Doug Aberdeen's presentation at the 2012 CAS Conference (see Further Resources below). Print off the resources and run this as an activity with your class.

Further resources

- Doug Aberdeen's simulation from the CAS conference, available at: www.computingatschool.org.uk/index.php?id=aberdeen.
- Useful list of advanced search keywords in Bing, available at: http://onlinehelp.microsoft.com/en-us/bing/ff808421.aspx.
- Short animated presentation 'How Search Works' by Matt Cutts, available at: www.youtube.com/watch?v=BNHR6IQJGZs.
- Peter Dickman's lecture 'How Google Search Works', available at: www.youtube.com/watch?v=C8v7AM1o7uM.
- Digital Schoolhouse simulation of how a search engine works: http://community.computingatschool.org.uk/files/3874/original.pdf.
- Eli Pariser's talk 'Beware online "filter bubbles"' (how individually focused our search results are), available at: www.ted.com/talks/eli_pariser_beware_online_filter_bubbles?language=en.

Communication and collaboration

How can you use computers to work with others?

There's more to computing than computer science. With the use of digital technology such as smartphones and the internet, it's hard to think of any sphere of life which hasn't been changed by the near ubiquitous nature of communication technology.

The national curriculum seeks to ensure that all pupils learn about some of the opportunities that networks offer for communication and collaboration.

Young people are usually comfortable using a range of digital technologies to communicate with one another (although you should not presume that they act safely and responsibly when doing so: see Safe and responsible use. They are perhaps less skilled in using technologies to work collaboratively on shared projects.

Different technologies work with different-sized groups:

One-to-one
email
video calls
instant messaging

One-to-many
blogging
personal website
publishing on YouTube
podcasting
posting to social media

Many-to-one
searching the web
watching YouTube
browsing social media

Many-to-many
discussion forums
Wikipedia

We need to develop pupils' *understanding* of these technologies (and some critical discernment about their use) rather than just their ability to use any particular platform. The implementation of communication technology will change, but underlying principles are likely to remain the same.

Can communication technology be embedded across the whole curriculum?

Yes! Many schools are now using digital communication and collaboration technologies as part of their day-to-day work.

Can pupils communicate with other schools?

Again, yes! The internet can provide many opportunities for pupils in one class to communicate with or work collaboratively with pupils in another class.

There's so much that can be gained through even a simple, email based eTwinning project. Think of the scope for exploring 'contrasting localities' in

geography, for practising other languages, or looking at a period in history from a global perspective.

What audience can pupils reach?

These days, it's easy for a teacher to set up a class blog, perhaps as open access so that a child's work can reach an audience, potentially, of close on three billion others. Blogs are also a great way to share what's happening in your class with your pupils' parents and with other teachers.

Blogs can be used as a basis for partnership projects with another class or group of classes, taking turns to respond to work that's posted. However, it's really important that comments posted to a class or school blog are moderated by a teacher before they're seen by pupils.

Blogging can be easily used to record and share pupils' work in computing. Even without blogging, pupils could share their programming work through community sites for tools such as Scratch and Kodu (taking care that all involved observe the terms and conditions that apply to these platforms).

How can pupils work collaboratively?

The internet makes it easy for pupils to work collaboratively online, just as they have always been able to do in class.

Web-based platforms such as Office 365 mean that pupils can work on files together, either by inviting comment and review from others, or through real-time collaboration. The efficiency with which joint projects can be undertaken and reviewed can make this a very exciting mode of work.

Teachers and pupils alike will be aware of the collaborative nature of Wikipedia. This can provide a good opportunity for pupils to become more discerning in evaluating digital content, and indeed to correct errors or add content to Wikipedia when they can. The Simple English Wikipedia is far less 'complete' than the main edition, and so it's practical for primary classes to 'adopt' pages here, editing or monitoring these for other users. Alternatively, teachers can set up their own wiki for their class, using one of a number of online tools.

Online collaborative working is a very important part of software development. Pupils themselves can get some experience in collaborative software development through the re-mix feature built into platforms such as Scratch, TouchDevelop and Kodu.

What ground rules should we establish?

It's important to establish an agreed set of rules for any online activities. Pupils need to be aware that terms and conditions do apply to them, even if they are rarely written in accessible language. You should brief pupils on what is expected of them. The key stage 2 programme of study expects pupils to recognise acceptable and unacceptable behaviour.

It's helpful to have a set of guiding principles here: pupils should behave online just as they would offline. This would include:

- not being deliberately hurtful
- taking care of shared resources
- being prepared to stand up for doing the right thing, even if it's unpopular
- not talking to strangers
- being honest.

Explain to pupils that most online systems automatically log the activities that take place in them: someone (or something) is watching what they do online!

Further resources

- eTwinning: connect with classes across Europe, available at: www.eTwinning.net.
- 100 Word Challenge: carry out and share short literacy projects, available at: http://100wc.net/.
- Quadblogging®: collaborative blogging in groups of four classes across the world, available at: http://quadblogging.com/.
- Simple English Wikipedia, available at: http://simple.wikipedia.org/wiki/Main_Page.
- Wikipedia: Five pillars: the guiding principles behind Wikipedia, available at: http://en.wikipedia.org/wiki/Wikipedia:Five_pillars.
- Wikispaces Classroom: creating wikis in school, available at: www.wikispaces.com/content/classroom.

Productivity and creativity

Can we teach our old ICT topics?

The answer to this is yes! There are few, if any, topics from the old ICT curriculum that don't appear in the new computing curriculum.

At key stage 1 pupils should be taught to: 'use technology purposefully to create, organise, store, manipulate and retrieve digital content'.

At key stage 2 pupils should be taught to: 'select, use and combine a variety of software (including internet services) on a range of digital devices'. They design and create digital content as well as **programs** and systems, and they accomplish given goals, including 'collecting, analysing, evaluating and presenting data and information'.[1]

See pages 50–51 for more information on reusing old ICT units when planning a computing scheme of work.

How can we make ICT activities more meaningful for pupils?

David Jonassen and others coined the term 'meaningful learning'. They were thinking particularly about learning activities that involved using technology, but the principles can be applied more broadly. Jonassen's list[2] was:

- **active:** pupils should *do* something
- **constructive:** pupils should *make* something
- **intentional:** pupils should have some *say* in what they do or how they accomplish something
- **authentic:** link to pupils' direct experience, including that of school: look for connections with other areas of the curriculum
- **cooperative:** look for activities where pupils can learn with and from one another.

For example, pupils could work together to create and then analyse the results from an online survey of other pupils about their views on the breadth of the school's curriculum, choosing for themselves how they might present the results of their survey.

How should pupils go about project work?

It's important to find a balance between getting things done in the time available and developing good working habits for extended projects.

It's probably best to mix a range of short activities with more extended projects in which the processes of planning, implementing, revising and evaluating are fully explored. Working through the stages of a project in detail is good experience for this sort of work elsewhere.

Look for ways to get pupils involved in managing projects. This can include deciding what programs and equipment they'll need to use. The project management skills involved in creative media work are very similar to those required in software development.

What digital tools should pupils work with?

The programmes of study are quite careful not to specify particular digital media. Technology currently

[1] National Curriculum in England, *Computing Programmes of Study* (Department for Education, 2013).

[2] Jonassen, D. H. *et al.*, *Meaningful Learning with Technology* (Upper Saddle River NJ: Pearson, 2008).

available in most schools can be used for work across a very wide range of media including: text, images, sound, animations, video and 3D. Ensure that your pupils experience working across this full range. A PowerPoint presentation is likely to include text and images, and perhaps video, audio and animations.

Also aim to ensure that your pupils work on a variety of devices and are able to draw on web-based services, tablets, smartphones, digital cameras or other systems rather than just using traditional Windows PCs in their IT work.

How can creativity be taught?

Sir Ken Robinson defines creativity as 'the process of having original ideas that have value'[3]: creative work should be original, and this should at least mean that it's a pupil's own work, not something where they've simply filled in a blank or copied something. Creative work should also be of value: at the very least to the pupils themselves, but also to a wider audience.

As well as originality and value, creative work also implies that the pupil has made something. An emphasis on creativity recognises how powerful the process of making things for others is as a means to learning.

In the classroom, help pupils to become masters of the software tools and digital devices they use, helping them to develop confidence, competence and independence. Then encourage them to use them, playfully or experimentally, as a way of helping them express their own insights and ideas.

What can pupils do with data?

The computing curriculum includes a requirement for pupils to work with numerical **data**. This is an important application of computer systems and seems likely to become even more so in the future.

There's much you can do to provide pupils with an authentic experience of working with both small and large datasets. Pupils can generate interesting sets of data, or access large, open data repositories.

Online survey tools, such as Google Forms or Excel Online, allow pupils to design and deploy quick opinion polls or surveys, and then analyse, evaluate and present the results. Choosing topics of genuine interest to pupils, perhaps concerned with aspects of school life, can make activities like this much more engaging. Pupils should think about privacy and ethical aspects of such surveys. Good practice includes principles of informed consent and anonymity; the latter is particularly important as otherwise data protection legislation would apply when processing personal data.

Classroom activity ideas

- Carry out activities that draw on automatically generated data, perhaps using sensors (e.g. a Scratch script to record the level of sound in class; see Further resources).
- Organise your pupils to analyse some *big* datasets made publicly available on the internet. Help them to use n-gram viewer to search for the occurrence of words or phrases in the vast number of books that Google have digitised and see how this changes over time (see Further resources). Analyse how search term popularity has changed over time, e.g. look at the relative popularity of searches for 'Britain's Got Talent' and 'The X Factor' over time in searches performed in the UK using Google Trends (see below).
- Discuss the ethical implications of data processing (i.e. what others do with our data). Ask pupils to think about the detailed profile which internet, email or search engine providers build up through analysing each user's activity, as well as to what uses this information might be put.

Further resources

- 'A picture is worth a thousand words: what we learned from 5 million books' lecture, available at: www.youtube.com/watch?v=5l4cA8zSreQ; see also n-gram viewer: https://books.google.com/ngrams.
- Classroom sound monitor on Scratch, available at: http://scratch.mit.edu/projects/20968943/.
- Google forms (www.google.co.uk/forms/about) or Excel Surveys (http://blogs.office.com/2012/11/16/excel-surveys/) for creating online surveys.
- Jonassen, D. H. *et al.*, *Meaningful Learning with Technology* (Upper Saddle River NJ: Pearson, 2008).
- Monte Carlo Method, available at: http://en.wikipedia.org/wiki/Monte_Carlo_method.
- Robinson, K., *Out of Our Minds – Learning to Be Creative* (Capstone, 2011).
- Using Google searches to predict flu: www.youtube.com/watch?v=uEt8NuqBvPQ; see also Google Trends: www.google.com/trends/.

[3] Robinson, K., *Out of Our Minds – Learning to Be Creative* (Capstone, 2011).

Safe and responsible use

Keeping children safe

How can we keep children safe online?

Schools have a responsibility to keep pupils safe. The Byron Review,[1] Ofsted and others have emphasised that the best way to achieve this is to teach pupils how to keep themselves safe. Think of pupils cycling to school: the pupils are exposed to risks which could otherwise be avoided, but these risks are balanced by a range of benefits (independence, health, environment, road congestion, etc.). We do all we can to outweigh the risks by teaching pupils to cycle well and safely.

The new computing curriculum goes beyond just teaching **e-safety**, and states that key stage 2 pupils should be taught to:

> *use technology safely, respectfully and responsibly; recognise acceptable/unacceptable behaviour; identify a range of ways to report concerns about content and contact.*[2]

It's important to recognise that these requirements are a whole school responsibility. They should be taught across the curriculum and become part of the life of the school – this isn't just something for computing lessons.

By moving from a risk mitigation approach to a values-based approach that promotes the responsible use of technology, we can help develop the pupils' sense of moral responsibility and the 'grit' necessary for pupils to stand up for doing the right thing. Pupils will then be far better at coping with the challenges of secondary education and adolescence, and far less likely to fall prey to the more sinister aspects of the internet and other technologies.

What are the risks?

In *The Byron Review* Professor Tanya Byron outlined three broad categories of risk which children are exposed to through their use of digital technology: content, contact and conduct.

	Commercial	Aggressive	Sexual	Values
Content (child as recipient)	Adverts Spam Sponsorship Personal info	Violent/hateful content	Pornographic or unwelcome sexual content	Bias Racist Misleading info or advice
Contact (child as participant)	Tracking Harvesting personal info	Being bullied, harassed or stalked	Meeting strangers Being groomed	Self-harm Unwelcome persuasions
Conduct (child as actor)	Illegal downloading Hacking Gambling Financial scams Terrorism	Bullying or harassing another	Creating and uploading inappropriate material	Providing misleading info/ advice

Table taken from *Safer Children in a Digital World: The Report of the Byron Review*, p.16 (www.education.gov.uk/publications/eOrderingDownload/DCSF-00334-2008.pdf)**. Contains public sector information licensed under the Open Government Licence v2.0: see** www.nationalarchives.gov.uk/doc/open-government-licence/version/2/**.**

[1] **Byron, T., *Safer Children in a Digital World: The Report of the Byron Review* (London: DCSF, 2008).**

[2] **National Curriculum in England, *Computing Programmes of Study* (Department for Education, 2013).**

Content

Children are naturally curious, and as teachers we hope to develop that curiosity – to establish a life-long love of learning. The Web has provided almost instant access to a wealth of information that pupils can access to further their learning and satisfy their curiosity.

Schools have effective filters that minimise exposure to inappropriate material in school, but this does not prevent pupils accessing such material outside of school, including on tablets or smartphones.

Both Bing and Google have safe-search modes (which can be locked in place) and these help prevent pupils from accessing particularly inappropriate content. In addition, a number of organisations have developed search engines targeted at children (for example www.swiggle.org.uk/), often through a combination of safe-search and custom-search tools in Google search.

Encourage parents to use the safe search filters on their search engine, and to request filtered internet access at home and on mobile devices, explaining how to do this and why it is a good idea.

But, even *with* filters in place, children may still encounter content that concerns them. Establish a 'no blame' culture in school so they feel they can alert you, or their parents, to such content. Many schools teach children to close the laptop, switch off the monitor or turn the tablet over if they find content they know they shouldn't see or that concerns them; again it's worth explaining this to parents and suggesting they do the same at home.

Byron identified commercialisation as another risk associated with exposing pupils to the internet. As teachers, we must help pupils to become discerning and critical about commercial aspects of the content they come across. For example, teach them about spam in email and how this can be filtered semi-automatically, as well as asking them to think about what sort of algorithms might be used in doing so.

Talk to pupils about advertising on the web and how this can be avoided through the use of browser plugins such as AdBlock, as well as the difference between sponsored and other results from search engines. It's also important to help pupils become aware of the difference between altruistically created content such as Wikipedia and many blogs, and content created with a perhaps hidden or implicit commercial purpose, e.g. apparently free online services that are sustained through using the user's data to help target advertising.

Contact

The new curriculum requires that pupils are taught who they can turn to if they have concerns over contact online. In most cases, pupils should talk to their parents or their teachers about such contact: if pupils report such concerns to you, this is likely to be covered by your safeguarding policy, so make sure you follow this carefully. Sometimes pupils might be too embarrassed to turn to either you or their parents, so it's worth introducing them to ChildLine and, in the case of key stage 2 pupils, CEOP (see Further resources).

Traditionally e-safety work in schools has included clear advice to children on not sharing personal information online. The curriculum includes this at key stage 1. Online privacy is an increasing matter of concern and there are broader issues here than 'stranger danger'. Pupils should be aware of their 'digital footprint', the data about them that is created by deliberately sharing content and through the automatic logging of all online activity. Whilst such logs are kept securely, many people are concerned about the uses to which such data could be put.

Classroom activity ideas

- Challenge older pupils to consider how algorithms can be designed to filter search results from a search engine to make them safe for children.
- Ask older pupils to think about the long-term implications of the data trails they leave behind them when they search the internet. Ask them to discuss: 'Who do you want to keep your data private from?' (From internet predators? From future employers? From the providers of search, internet and email services? From advertisers? From the school network manager? From government agencies?)

Conduct

The curriculum at key stage 1 requires that pupils learn to use technology 'respectfully'. At key stage 2 this is extended to 'responsibly', and pupils should also learn to recognise acceptable and unacceptable behaviour. Supporting children's moral development is a vital part of primary education, as well as a

statutory requirement for a school's curriculum and, as part of 'spiritual, moral, social and cultural development', an element of all Ofsted inspections.

Lawrence Kohlberg's stages of moral development[3] offers one model for thinking about this:

1. Obedience and punishment orientation (How can I avoid punishment?)
2. Self-interest orientation (What's in it for me?)
3. Interpersonal accord and conformity (The good boy/girl attitude)
4. Authority and social-order maintaining orientation (Law and order morality)
5. Social contract orientation (Do unto others...)
6. Universal ethical principles (Principled conscience)

Under this model, we would hope to see pupils taking increasing responsibility for their own moral and ethical decisions and behaviour whilst at primary school. If schools take moral education seriously, many aspects of pupils' inappropriate conduct using technology can perhaps be avoided, or their consequences reduced.

Cyber-bullying

Even in primary schools, cyber-bullying is a common problem. Whilst this is more likely to happen outside of school, it's common for both bully and victim to be members of the same class or school and the cause and consequences may often be connected to school. As with bullying in general, a clear zero tolerance message is vital, together with a culture in which this can be reported in the knowledge that swift and effective action will follow. Alongside this, it's worth building up pupils' resilience to off-hand, unintentionally hurtful remarks from others and some recognition that not every online disagreement or critical comment constitutes bullying.

Copyright

There are generous exemptions from much copyright legislation for clearly specified educational use, but it's still important to teach and show best practice in the use of copyright material. This includes children (and teachers!) properly acknowledging the source of content and respecting any associated licence terms.

Creative Commons (see Further resources) provide a range of licences that allow those who create work to license it for re-use under a range of different conditions. You can teach pupils about this approach to sharing online and show them how they can search for, acknowledge and re-use Creative Commons licensed content in their own work. Both Google and Bing image search allow results to be filtered to show just images that have been licensed in this way.

Pupils own the copyright in their own work including the work they produce in school. As teachers, we should respect this by seeking permission from pupils and their parents before publishing pupils' work online. Asking parents to license this use of their children's work might seem over the top, but it's important that pupils learn about their rights as well as their responsibilities.

Terms and conditions

It's important that pupils and teachers respect the terms and conditions of any websites or other online services that they use. The terms and conditions of most online services run to many pages, but when signing up for new services, or asking pupils to do so, it's well worth checking through the sections on any age-restrictions as well as those on copyright and data privacy. US-based companies are required to abide by American COPPA (Children's Online Privacy Protection Act) legislation, which prevents their storing personal data on under 13s without parental consent. Thus, many US-based internet services prohibit under 13s from using the service. Primary school pupils using these services would be doing so without the operators' permission, which might be considered in breach of the UK Computer Misuse Act. Some services, including Office 365 and Google Apps for Education, allow schools to create accounts on behalf of children. Other websites, such as Scratch, allow teachers to create multiple accounts in their own name and share these with pupils.

Passwords

As more and more aspects of pupils' learning and life are mediated through online systems, it's important that they learn to protect their own online identity and respect the online identity of others. The sooner pupils can memorise and type in their own password (even a simple, short one) the better. Later on, encourage pupils to use long passwords that can't easily be guessed (e.g. CorrectBatteryHorseStaple), to use different passwords for different sites or services and to change passwords regularly. Discourage pupils from sharing passwords with one another (as this is usually their only way to prove who they are in any online system) or with their parents; many difficulties could arise through one parent impersonating their son or daughter in an otherwise secure 'walled garden' environment such as a school VLE or learning platform.

[3] Kohlberg, L., *Essays on Moral Development: Vol. 2, The Psychology of Moral Development* (Harper & Row, 1984).

Time to turn off

Finally, discuss with your pupils the opportunity cost associated with screen time. Time spent using a computer is time not spent doing other things, such as reading a (paper-based) book, learning a musical instrument, playing in a team and socialising face-to-face with family and friends. Whilst digital technology is seen by many as transformative of so many aspects of learning and life, many would count it a great shame if it came to dominate childhood to a greater extent than it already has. Helping children to become more discerning users of technology, knowing when it would be useful, and when it might be more of a distraction, is perhaps also one of our responsibilities as teachers.

Further resources

- Byron, T., *Safer Children in a Digital World: The Report of the Byron Review* (DCFS, 2008), available at: http://webarchive.nationalarchives.gov.uk/20130401151715/http://www.education.gov.uk/publications/eOrderingDownload/DCSF-00334-2008.pdf.
- Childnet's SMART rules: www.kidsmart.org.uk/beingsmart/.
- Creative Commons, for information and free licences to use, available at: http://creativecommons.org/.
- 'Digital Literacy & Citizenship from the South West Grid for Learning', teaching resources, available at: www.digital-literacy.org.uk/Home.aspx.
- Ofsted: 'Inspecting safeguarding in maintained schools and academies – Briefing for section 5 inspections', available at: www.ofsted.gov.uk/resources/inspecting-safeguarding-maintained-schools-and-academies-briefing-for-section-5-inspections.
- Thinkuknow.co.uk (CEOP), information and teaching resources for keeping children safe online, available at: www.thinkuknow.co.uk/Teachers/.
- UK Safer Internet Centre, for information and teaching resources, available at: www.saferinternet.org.uk.
- UNCRC (United Nations Convention on the Rights of the Child), for information and training on children's rights, available at: www.ohchr.org/en/professionalinterest/pages/crc.aspx.

Teaching

Approaches for teaching computing

What makes a good computing lesson?

There's a wealth of learning theory, academic research and professional practice, including Ofsted's expectations, that we can draw on to help address this crucial question. Good practice in computing is unlikely to be different from good practice across the primary curriculum. We can draw on what's effective in other subject areas which have much in common with computing: science, design and technology, art and design and music teaching.

What approaches are useful for teaching computing?

Educational theory can be mined for insights into how a new subject like computing might be taught. The pragmatic teacher is likely to draw on a blend of these approaches.

- **Experimenting:** Provide pupils with a chance to explore and tinker with new software or hardware when they first encounter it, so they can figure out their own mental model for how it works. This can be particularly effective with younger pupils.
- **Making:** A lot can be learnt through the process of making things to show to or share with others. This might be computer code, but it might also be PowerPoint presentations, web pages, edited video, digital photographs, etc.
- **Discussion:** Make the most of pupils' different insights, experiences and backgrounds by allowing them to share their ideas with one another and with others. Paired programming activities in class and online discussion forums are just two ways to facilitate this.
- **Connecting:** Learning from others need not be limited to the classroom: encourage pupils to explore others' solutions to problems on the Kodu or Scratch community sites, for example, or to search online for solutions to problems.
- **Direct instruction:** For some ideas in computing, the traditional, direct instruction approach can work well. Complex ideas such as variables, how the internet works or how search engines operate could be learnt using discovery-based approaches, but direct teaching is likely to be more effective.
- **Practise:** Don't assume that once pupils have demonstrated they can do something or understand an idea that their learning is secure. Provide opportunities for them to practise applying their skills, knowledge and understanding.

What does Ofsted expect?

Ofsted's expectations of good or outstanding lessons are the same irrespective of subject, and are outlined in the current edition of the *School inspection handbook* (Ofsted 2014).

The *School inspection handbook* makes clear the importance of inclusion, as discussed in relation to planning for computing (page 51). Thus, for teaching to be considered outstanding:

> *almost all pupils currently on roll in the school, including disabled pupils, those who have special educational needs, disadvantaged pupils and the most able, are making sustained progress that leads to outstanding achievement.*[1]

In his presentation on inspecting computing, David Brown made some suggestions for what good or outstanding teaching in computing might look like.

He recommended that:

- *it is informed by excellent subject knowledge and understanding of continuing developments in teaching and learning in computing*
- *it is rooted in the development of pupils' understanding of important concepts and progression within the lesson and over time; it enables pupils to make connections between individual topics and to see the 'big picture'*
- *lessons address pupils' misconceptions very effectively; teachers' responses to pupils' questions are accurate and highly effective in stimulating further thought*
- *teachers use a very wide range of innovative and imaginative resources and teaching strategies to stimulate pupils' active participation in their learning and secure good or better progress across all aspects of the subject.*[2]

When commenting on pupils' achievement in computing, David Brown suggested that this would be good or outstanding if:

- *pupils demonstrate excellent understanding of important concepts in all three strands of the computing curriculum and are able to make connections within the subject because they have highly developed transferable knowledge, skills and understanding*
- *pupils show high levels of originality, imagination, creativity and innovation in their understanding and application of skills in computing*

but would be regarded as inadequate if:

- *pupils rarely demonstrate creativity or originality in their use of computing but seem confined to following instructions.*[3]

What about beyond the classroom?

Look for ways in which pupils can take their learning further. Here are just a few examples.

- Code Club (see Further resources) run after-school coding clubs in around 2,000 primary schools and make their activities available for others to use as they wish. Typical clubs are run by volunteers, although teacher involvement is also needed. Code Club provide an introductory Scratch programming course and a more advanced Scratch coding course, as well as courses that introduce the basics of HTML coding for web-development and Python programming.
- Many schools have found it helpful to institute a system of pupil 'digital leaders', who can help teachers and other pupils with some limited tech support, as well as being the first to try out new software and hardware and even advising on the school's technology policies. Typically, schools run an open application process for these roles, inviting potential digital leaders to set out in writing why they would be well suited to the role.
- Activities such as CoderDojo and Young Rewired State (see Further resources) rely on parental support for primary aged pupils and involvement from those working in the software industry.
- Some primary pupils might, with parental permission, become active participants in online communities, such as those around Kodu, Scratch or even YouTube and blogging. Others might pursue more advanced coding skills, perhaps using online interactive tutorials such as those offered by Codecademy (see Further resources), or teaching themselves how to develop for Windows Phone, Android or iOS.

Try to find ways in which this sort of advanced learning beyond school can be brought in to class and shared with other pupils.

Further resources

- David Brown (Ofsted) 'Inspecting computing' (Computing Conference) slides.
- O'Kane, L., 'Computing Pedagogy' (2019), available at: www.icompute-uk.com/news/computing-pedagogy
- O'Kane, L., 'Achieving Computing Mastery' (2019), available at: www.icompute-uk.com/news/computing-mastery-for-primary-schools
- Cousin, G., 'An introduction to threshold concepts' (2006), available at: www.et.kent.edu/fpdc-db/files/DD%2002-threshold.pdf.
- Hattie, J., *Visible Learning for Teaching: Maximising Impact on Learning* (Routledge, 2009).
- Papert, S., *Mindstorms: Children, Computers, and Powerful Ideas* (Basic Books, 1980).
- The Sutton Trust Education Endowment Foundation Teaching and Learning Toolkit, for information on research and guidance on using resources for disadvantaged pupils, available at: http://educationendowmentfoundation.org.uk/toolkit/.
- Young Rewired State community of young digital makers, available at: www.yrs.io/.

Assessment

Assessing and tracking progress

How can we collect evidence of learning?

When Ofsted reported on ICT in schools, assessment came in for particular criticism. Assessing computing can provide some particular challenges.

- It's too easy to focus on the outcomes of a task at the expense of assessing the learning that takes place in the process.
- It's too easy to focus on assessing pupils' skills in using particular software instead of assessing their knowledge and understanding.
- If pupils have worked with a partner or in a group to complete a project, how can you assess each individual's learning?

You can do much to meet these challenges and develop robust approaches to assessment, so that you can form a judgement about what individual pupils can do, know and understand, as well as helping pupils themselves reflect on how they've applied computational thinking.

Blogs and Screencasts for showcasing, reflection and feedback

Probably, the most effective thing you could do is to start a class blog. Ask pupils to use this to upload the outcomes of their work and document the computational thinking processes they worked through, focusing on any challenges they overcame.

Blogs provide a way for pupils to get feedback through the comments section. Invite pupils to respond to any questions raised. You can create a tagging system so you and your pupils can use their blog to track progress. A blog begun in Year 1 and continued up to Year 6 would provide rich evidence of both progress and attainment.

Other approaches

Naace suggest using an interview at the end of a project. A pupil might explain the computational thinking they used in solving a problem, but could also reflect on what and how they have learnt.

There's a place for formal testing in computing. In programming work, code tracing and debugging challenges are useful ways of assessing both specific knowledge of a programming language as well as logical reasoning and other problem-solving skills.

Evidence for pupils' computational thinking will be found in how they approach projects, but well-designed questions might provide one way of assessing this more directly.

How can we track progress?

iCompute's Computing Assessment Toolkit is used for both planning and monitoring progression in computing. It provides detailed treatment of progression across the three strands of the National Curriculum.

Comprehensive end of unit assessment guidance is provided with each unit plan the results of which teachers input into the interactive pupil progress trackers.

The pupil progress trackers assign each pupil a colourway allowing teachers to identify any gaps and plan for next steps.

How can we assess attainment?

The old national curriculum levels have been removed and not replaced. The statutory attainment target is clear:

> *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.*[6]

iCompute Tests & Tasks support teacher assessment with end of unit online diagnostic tests and open-ended project tasks with teacher mark scheme. The assessment data feeds into pupil progress trackers.

The advantage of a method like this is that it shows pupils, parents and teachers exactly what has been achieved and what aspects of the curriculum remain targets for subsequent work.

Assessment Strategies

Evidence – Blogging and using e-Portfolios such as SeeSaw or maintain individual folders on the school network for each pupil to contain digital work.

Teacher Feedback – Face-to-face or using digital 'marking' strategies such as adding text comments in digital work or adding audio of your comments.

Self/Peer – Blogging, Vlogging or Video Screencasting provides excellent opportunities for pupils to reflect on work.

Diagnostic Testing – Creative online interactive quizzes provide engaging opportunities to assess pupil understanding and bring a gamification aspect to assessment.

Assessment Projects – Using end-of-unit open-ended project tasks allow pupils to demonstrate learning.

Progress Tracking – Understanding where pupils are and planning next steps to meet age-related expectations

Even just one Scratch script provides evidence of attainment for the key stage 2 programme of study.

From the Scratch scripts themselves, we have evidence of:

- write programs that accomplish specific goals
- use sequence in programs
- work with various forms of input (keyboard and mouse in this case)
- design programs that accomplish specific goals
- design and create programs
- use repetition in programs (forever loop, two different repeat until loops)

- simulate physical systems
- use selection in programs (if ... then ... else)
- work with variables (score).

If pupils had also explained how they'd solved the problems, then you might also have evidence of a number of the 'logical reasoning' statements.

SEN

Pupils with special educational needs working below the level of the programme of study for their key stage should be assessed using the P-scale statements, as in the past. iCompute has pupil progress trackers for the Early Years adapted from P-Scales.

Further resources

- O'Kane, L., 'How to Assess Primary Computing' (2019), available at: www.icompute-uk.com/news/computing-assessment
- O'Kane, L., 'Primary Computing Assessment' (2019), available at: www.icompute-uk.com/news/primary-computing-assessment
- O'Kane, L., 'Computing Tests & Tasks' (2019), available at: www.icompute-uk.com/news/computing-tests-and-tasks