

**COMPUTING AT SCHOOL SCOTLAND**

EDUCATE · ENGAGE · ENCOURAGE

Part of BCS, The Chartered Institute for IT



# QuickStart Computing Scotland

**Subject Knowledge  
for Primary teachers**

SUPPORTED BY



# Foreword

In 2017, Education Scotland revised the computing science content within the Broad General Education phase, with significant input from Computing At School Scotland, to reflect the developing worldwide understanding of the importance of the subject.

For most of us, technology and computers play a vital role in our lives: at home, at work, for our health and for our informal learning. It is important that our children learn how this stuff works, rather than treating it as magic. Just as we introduce the traditional sciences in primary school, we must now teach computing science, the ‘fourth science’, from the early years, to ensure children leave school equipped with the skills and knowledge they need to participate effectively in society, whether or not they go on to become computing professionals.

*QuickStart Computing* can help primary teachers to pick up the necessary subject knowledge for computing science. It should be read in conjunction with the guide for teachers available at <http://teachcs.scot>, which explains the structure of the Experiences and Outcomes (‘Es and Os’) and Benchmarks in detail, and provides links to a wealth of activities that can be used at primary.

This *QuickStart* resource is focused on primary schools, but there is a companion pack for early secondary.

A word about sponsorship. The *QuickStart* project was funded by Microsoft, with matched funding from the English Department for Education, and further funding from Education Scotland, and it is heartening to see such tangible support for teachers, both from business and from government. I would like to thank them warmly and to emphasise that the *QuickStart* resources were developed for teachers by a Computing At School working group, without the direct influence of the sponsors.

The new computing curriculum is a huge opportunity for schools to make a difference to children’s futures. Let’s make the most of it! I hope that *QuickStart* will help you design, develop and deliver an invigorating computing curriculum that will inspire you and your children.



Quintin Cutts  
Chair, Computing At School Scotland

Every effort has been made to trace copyright holders and obtain their permission for the use of copyright materials. The author and publisher will gladly receive information enabling them to rectify any error or omission in subsequent editions.

Although every effort has been made to ensure that website addresses are correct at time of going to press, Computing At School cannot be held responsible for the content of any website mentioned. It is sometimes possible to find a relocated web page by typing in the address of the home page for a website in the URL window of your browser.

© Crown Copyright 2017  
Published 2017

Author: Miles Berry  
Adjusted for Scotland: Bill Sargent  
Consultants: Yvonne Walker, Graham Hastings  
Cover, text design and typesetting: Burville Riley Partnership

Computing At School is grateful to the following contributors: Bill Mitchell, Lee Goss, Miles Ellison, Roger Davies, Michael Kölling, Hugh Milward, Clare Riley, Phil Bagge, Mark Dorling, Andrea Carr, Becca Law, Tracey Cowell, Sarah Snashall, Jane Jackson, Hilary Beaton, Teresa Watts and Helen Royle from Stuck Ltd, Eruke Ideh-Ichofu.

Photo and text credits:  
With thanks to Mark Dorling and Matthew Walker for permission to use the Progression Pathways. © Mark Dorling and Matthew Walker. Page 7: Barefoot would like to acknowledge the work of Julia Briggs and the eLIM team at Somerset County Council for their contribution to this poster.  
Pages 11, 22, 24–28: Scratch is developed by the Lifelong Kindergarten Group at the MIT Media Lab. See <http://scratch.mit.edu>.  
Page 13: With thanks to iFixit. CC by-nc-sa.  
Page 18, 20: With thanks to TTS.  
Pages 21, 25 and 28: With thanks to Stuart Ball for providing the Kodu screenshots.  
Pages 33–37: With thanks to Advocate Art.  
Page 39: Tim Berners-Lee image CC by Silvio Tanaka.  
Page 41: PageRank example created by 345Kai. Public domain.

To reference this work, please use the following citation: Berry, M. (2017) *QuickStart Scotland Primary Handbook*. Swindon: BCS.

This content is free to use under the Open Government Licence v3.0.

A catalogue record for this title is available from the British Library.  
ISBN: 978-1-78017-440-2

# Contents

|   |           |
|---|-----------|
| <b>User guide</b>                                 | <b>4</b>  |
| <b>QuickStart Computing roadmap</b>               | <b>5</b>  |
| <b>Computational thinking</b>                     | <b>6</b>  |
| <b>Programming</b>                                | <b>18</b> |
| <b>Technology</b>                                 | <b>30</b> |
| <b>Computer networks</b>                          | <b>32</b> |
| <b>Communication and collaboration</b>            | <b>42</b> |
| <b>Productivity and creativity</b>                | <b>44</b> |
| <b>Safe and responsible use</b>                   | <b>46</b> |
| <b>Planning learning, teaching and assessment</b> | <b>50</b> |
| <b>Resources</b>                                  | <b>53</b> |
| <b>Knowledge and skills audit form</b>            | <b>54</b> |
| <b>Glossary</b>                                   | <b>56</b> |



# User guide

Welcome to *QuickStart Computing*: a CPD toolkit for the new primary curriculum. Computing is a new subject. It draws together the strands of computer science and digital literacy, and seeks to equip children with computational thinking skills and the creativity they need to understand and change the world.

Through the programme of study for Computing Science and Digital Literacy, primary school-aged children learn the fundamental principles and processes of computation; they gain repeated, practical experience of writing code to solve problems and to model systems; they become skilled at creating high-quality products and content using digital technology; and they become safe, responsible and critical users of technology<sup>1</sup>. Computing is an enjoyable and empowering subject to learn, and it's a very rewarding one to teach. However, unlike other subjects in the primary curriculum, it's not one many primary teachers learned themselves when they were at school, or were taught about in their teacher training.

## What is QuickStart Computing?

*Quickstart Computing* is a set of resources that addresses the subject knowledge and the subject-specific pedagogy teachers need in order to plan, teach and assess the primary computing curriculum effectively and confidently.

This handbook is broken down into two sections:

- Computing subject knowledge with suggestions for tried-and-tested classroom activities to run in school (see pages 6–49)
- Advice for planning, teaching and assessing the computing curriculum (see pages 50–52).

Both sections are supported by:



weblinks to useful information, including activity ideas from Computing At School, Barefoot Computing and CS Unplugged.

All the resources are available to download free of charge from <http://computingatschool.org.uk/scotland/quickstart>.

## Delivering computing CPD

This toolkit can be used to develop and deliver computing CPD sessions to colleagues and computing coordinators. We have suggested a model for the use of this CPD on the page opposite but you can decide how best to share the training to meet the needs of your school, cluster or hub.

All timings and durations are suggestions only and should be adapted to fit with the needs of both the course leader and session attendees. For example, the diagram opposite outlines two half-day CPD sessions to take place at the beginning and end of a school term, but depending on availability of session leaders and attendees, it may be better to hold three shorter twilight sessions.

These resources are designed to be used flexibly and it is important to spend time reviewing the materials provided and developing these prior to delivering CPD sessions, to ensure they fit the needs of your attendees. The handbook and CPD session presentations are provided in editable format to help you with this.

## Developing computing knowledge and skills

To benefit most from these resources, it's important to engage fully with them.

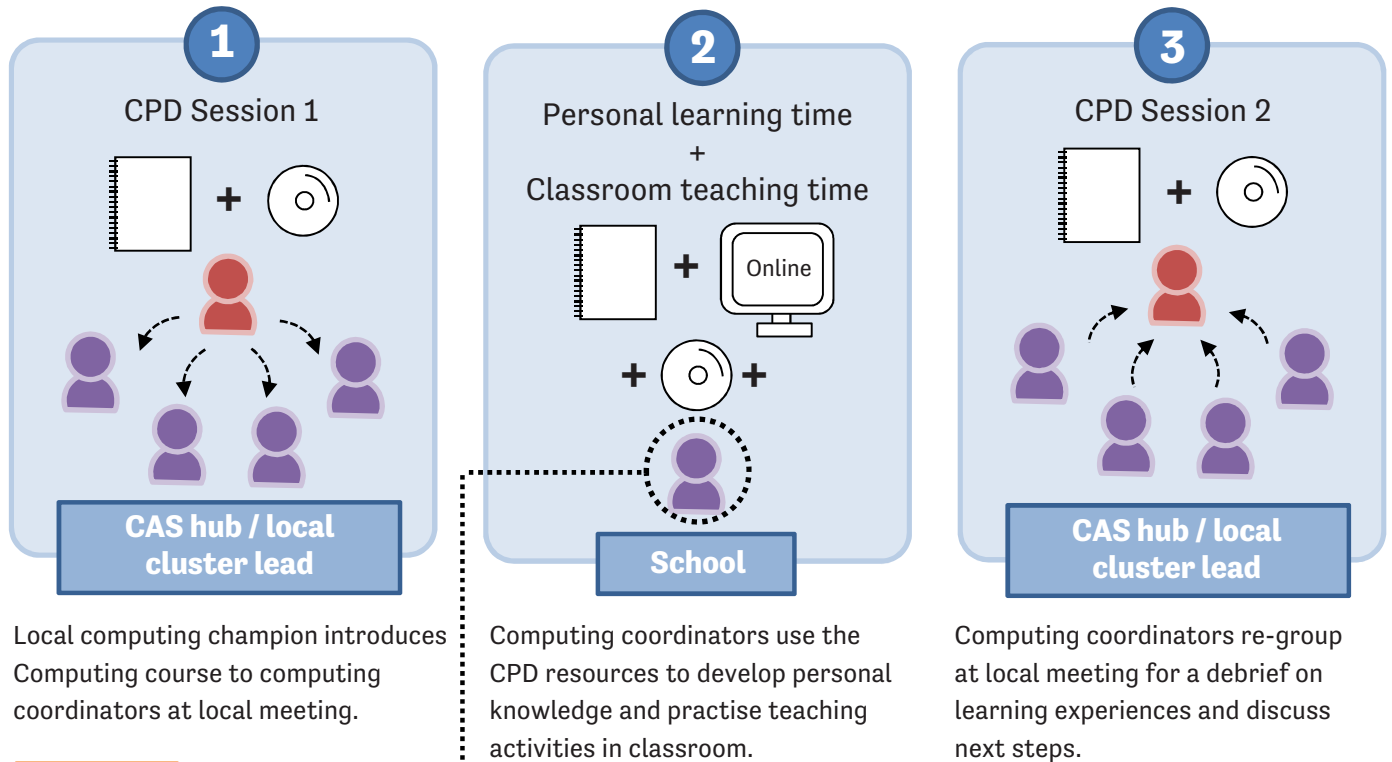
- Read the handbook, particularly those areas of content in which you are less confident (see pages 54–55 for the knowledge and skills audit form, and the interactive audit tool is at <http://computingatschool.org.uk/scotland/quickstart>).
- Have a go at creating some code, for example making games in Scratch or Kodu; think about how you would apply these ideas in school.
- Look for existing examples of computational thinking that you make use of in your job.
- Try out some of the classroom activities described in the handbook.
- Engage with others. Learning as part of a group allows you to share knowledge and ideas. This can be done in school and by joining an online community such as Computing At School: [www.computingatschool.org.uk/](http://www.computingatschool.org.uk/).

**Note:** throughout the guide we have highlighted computing terms in **blue**. The definitions of these terms are in the glossary.

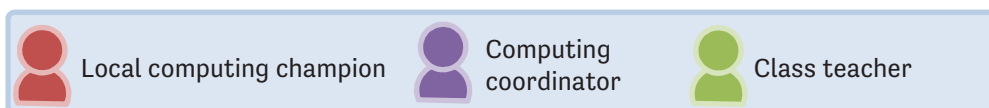
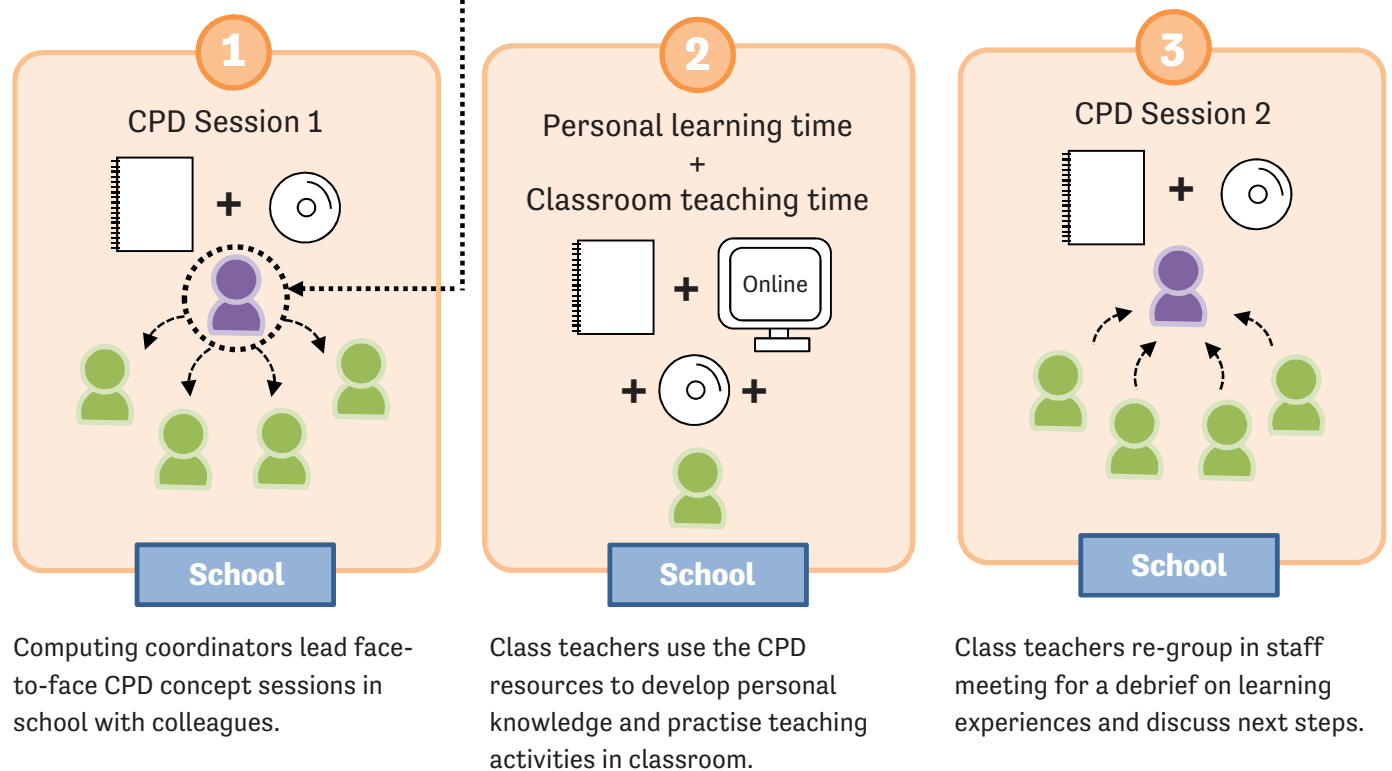
<sup>1</sup> Technologies Benchmarks [Education Scotland 2017]

# QuickStart Computing roadmap

## Wave 1



## Wave 2



# Computational thinking

## How do we think about problems so that computers can help?

Computers are incredible devices: they extend what we can do with our brains. With them, we can do things faster, keep track of vast amounts of information and share our ideas with other people.

### What is computational thinking?

Getting computers to help us to solve problems is a two-step process:

1. First, we think about the steps needed to solve a problem.
2. Then, we use our technical skills to get the computer working on the problem.

Take something as simple as using a calculator to solve a word problem in maths. First, you have to understand and interpret the problem *before* the calculator can help out with the arithmetic bit.

Similarly, if you're going to make an animation, you need to start by planning the story and how you'll shoot it *before* you can use computer **hardware** and **software** to help you get the work done.

In both of these examples, the thinking that is undertaken before starting work on a computer is known as **computational thinking**.

Computational thinking describes the processes and approaches we draw on when thinking about problems or systems in such a way that a computer can help us with these.

Computational thinking is *not* thinking about computers or like computers. Computers don't think for themselves. Not yet, at least!

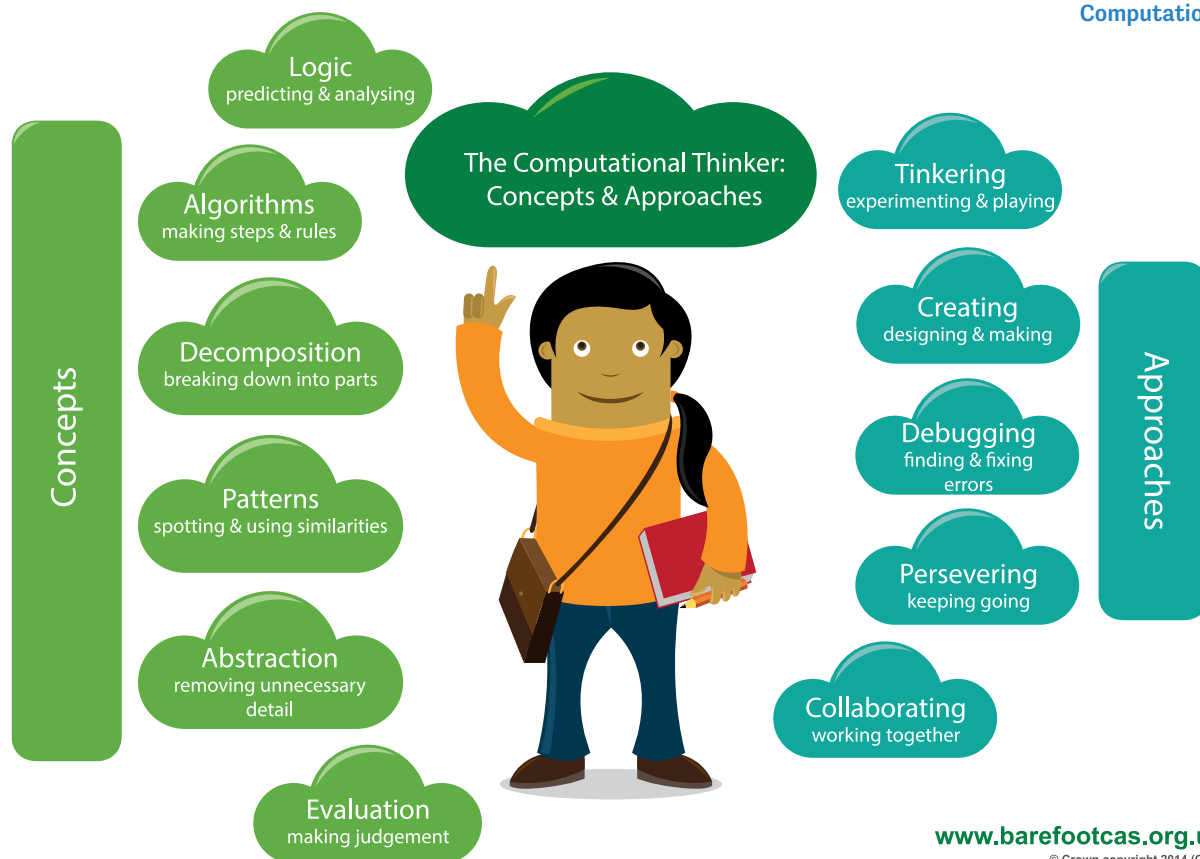
Computational thinking is about looking at a problem in a way that a computer can help us to solve it.

When we do computational thinking, we use the following processes to tackle a problem:

- Logical reasoning: predicting and analysing (see pages 8–10)
- Algorithms: making steps and rules (see pages 10–12)
- Decomposition: breaking down into parts (see pages 12–14)
- Abstraction: removing unnecessary detail (see pages 14–15)
- Patterns and generalisation: spotting and using similarities (see pages 15–16)
- Evaluation: making judgements.

### What can you do with computational thinking?

Although computational thinking describes the sort of thinking that computer scientists and software developers engage in, plenty of other people think in this way too, and not just when it comes to using



[www.barefootcas.org.uk](http://www.barefootcas.org.uk)

© Crown copyright 2014 (OGL)

computers. The thinking processes and approaches that help with computing are really useful in many other domains too.

For example, the way a team of software engineers goes about creating a new computer game, video editor or social networking platform is really not that different from how you and your colleagues might work together to put on a school play, or to organise an educational visit.

In each case:

- you take a complex problem and break it down into smaller problems
- it's necessary to work out the steps or rules for getting things done
- the complexity of the task needs to be managed, typically by focusing on the key details
- the way previous projects have been accomplished can help.

## How is computational thinking used in the curriculum?

Ideas like logical reasoning, step-by-step approaches (algorithms), decomposition, abstraction and **generalisation** have wide applications to solving problems and understanding systems across (and beyond) the school curriculum. There are many ways to develop these in school beyond the computing

curriculum, but as pupils learn to use these in their computing work, you should find that they become better at applying them to other work too.

You will already use computational thinking in many different ways across your school.

- When your pupils write stories, you encourage them to plan first: to think about the main events and identify the settings and the characters.
- In art, music or design and technology, you will ask pupils to think about what they are going to create and how they will work through the steps necessary for this, by breaking down a complex process into a number of planned phases.
- In maths, pupils will identify the key information in a problem before they go on to solve it.

## Where does computational thinking fit in the new computing curriculum?

As their digital literacy becomes more sophisticated they embed computation to solve problems. Increasingly they apply the core principles underpinning digital technologies to develop their own ideas<sup>1</sup>

<sup>1</sup> How Good is our School? [Education Scotland 2015]

Whilst programming (see pages 18–29) is an important part of the new curriculum, 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.*

Computational thinking shouldn't be seen as just a new name for 'problem-solving skills'. It does help to solve problems and it has wide applications across other disciplines, but it's most obviously apparent, and probably most effectively learned, through the rigorous, creative processes of writing code – as discussed in the next section.



### Classroom activity ideas

- Ask your pupils to write a recipe for a sandwich, thinking carefully about each step that needs to be carried out. Point out that the step-by-step sequence of instructions is an algorithm. Ask them to share each other's recipes and spot patterns in them (this is called generalisation). Read a range of recipes and discuss the layers of simplification (abstraction) present in even relatively simple recipes, such as for pizza.
- Plan a traditional 'design, make and evaluate' project for design and technology, drawing out the parallels with computational thinking. For example, plan the process for making a musical instrument. Tell the pupils to break this complex problem down into smaller stages, such as:
  - » planning their design (an abstraction – a simplified version – capturing the key elements of this)
  - » sourcing their materials (using decomposition to identify the different components)
  - » assembling the materials to create the instrument (a systematic, step-by-step approach – an algorithm)
  - » evaluating (testing) the instrument.
- Challenge older pupils to work individually or collaboratively on more complex projects, for example researching and writing up aspects of a curriculum topic such as the Viking invasion, or putting together an assembly or a class play. In each case ask them to note down the individual steps needed for the task and to think about what they have left out to make the subject fit their brief.



### Further resources

- Barefoot Computing, 'Computational Thinking', available at: <http://barefootcas.org.uk/barefoot-primary-computing-resources/concepts/computational-thinking/> (free, but registration required).
- Berry, M., 'Computational Thinking in Primary Schools' (2014), available at: <http://milesberry.net/2014/03/computational-thinking-in-primary-schools/>.
- Computer Science Teachers Association, 'CSTA Computational Thinking Task Force' and 'Computational Thinking Resources', available at: <http://csta.acm.org/Curriculum/sub/CompThinking.html>.
- Computing At School, 'Computational Thinking', available at: <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', available at: [www.google.com/edu/computational-thinking/index.html](http://www.google.com/edu/computational-thinking/index.html).
- Wing, J., 'Computational Thinking and Thinking about Computing' (The Royal Society, 2008), available at: <http://rsta.royalsocietypublishing.org/content/366/1881/3717.full.pdf+html>.

## Logical reasoning

Can you explain why something happens?

If you set up two computers in the same way, give them the same instructions (the program) and the same **input**, you can pretty much guarantee the same **output**.

Computers don't make things up as they go along or work differently depending on how they happen to be feeling at the time. This means that they are predictable. Because of this we can use *logical reasoning* to work out exactly what a program or computer system will do.

Children quickly pick this up for themselves: the experience of watching others and experimenting



for themselves allows even very young children to develop a mental model of how technology works. A child learns that clicking the big round button brings up a list of different games to play, or that tapping here or stroking there on the screen produces a reliably predictable response.

This process of using existing knowledge of a system to make reliable predictions about its future behaviour is one part of logical reasoning. At its heart, logical reasoning is about being able to explain why something is the way it is. It's also a way to work out why something isn't quite as it should be.

### How is logical reasoning used in computing?

Logic is fundamental to how computers work: deep inside the computer's central processing unit (CPU), every operation the computer performs is reduced to logical operations carried out using electrical signals.

It's because everything a computer does is controlled by logic that we can use logic to reason about program behaviour.

Software engineers use logical reasoning all the time in their work. They draw on their internal mental models of how computer hardware, the **operating system** (such as Windows 8, OS X) and the programming language they're using all work, in order to develop new code that will work as they intend. They'll also rely on logical reasoning when testing new software and when searching for and fixing the 'bugs' (mistakes) in their thinking (known as **debugging** – see page 17) or their coding when these tests fail.

### How is logical reasoning used across the curriculum?

There are many ways in which children will already use logical reasoning in their computing lessons and across the wider curriculum.

- In English, pupils might explain what they think a character will do next in a novel, or explain the character's actions in the story so far.

- In science, pupils should explain how they have arrived at their conclusions from the results of their experiments.
- In history, pupils should discuss the logical connections between cause and effect; they should understand how historical knowledge is constructed from a variety of sources.

### Where does logical reasoning fit in the new computing curriculum?

In the computing curriculum, at Early Level, pupils are expected to use logical thinking to identify and correct errors in a set of instructions.

At Level 1, they should be able to identify when a program does not do what was intended, and to correct errors/bugs; and, at Level 2, to identify any mismatches between the task description and the programmed solution, indicating how to fix them.<sup>2</sup>



### Classroom activity ideas

- Provide pupils with floor turtles and ask them to make predictions of where the robot will end up when the 'go' button is pressed. Then ask them to explain *why* they think that. Being able to give a reason for their thinking is what using logical reasoning is all about.
- In their own coding, logical reasoning is key to debugging (finding and fixing the mistakes in their programs). Ask the pupils to look at one another's Scratch or Kodu programs and spot bugs. Encourage them to test the programs to see if they can isolate exactly which bit of code is causing a problem. If pupils' programs fail to work, get them to explain their code to a friend or even an inanimate object (e.g. a rubber duck).
- Give pupils a program of your own or from the Scratch or Kodu community sites and ask them to work backwards from the code to work out what it will do.
- Ask pupils to think carefully about some school rules, for example those in the school's computer Acceptable Use Policy. Can they use logical reasoning to explain why the rules are as they are?

<sup>2</sup> Technologies Benchmarks [Education Scotland 2017].

- There are many games, both computer-based and more traditional, that draw directly on the ability to make logical predictions. Organise for the pupils to play noughts and crosses using pencil and paper. As they are playing, ask them to predict their opponent's next move. Let them play computer games such as Minesweeper, Angry Birds or SimCity, as appropriate. Ask them to pause at certain points and tell you what they think will happen when they move next. Consider starting a chess club if your school doesn't already have one.



### Further resources

- Barefoot Computing, 'Logic: Predicting and Analysing', available at: <http://barefootcas.org.uk/barefoot-primary-computing-resources/concepts/logic/> (free, registration required).
- Computer Science for Fun, 'The Magic of Computer Science', available at: [www.cs4fn.org/magic/](http://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](http://www.cs4fn.org/teachers/activities/braininabag/braininabag.pdf).
- The P4C Co-operative, a co-operative providing resources and advice on philosophy for children, available at: [www.p4c.com/](http://www.p4c.com/).
- PhiloComp.net, website highlighting the strong links between philosophy and computing, available at [www.philocomp.net/](http://www.philocomp.net/).

## Algorithms

### What's the best way to solve a problem?

An algorithm is a sequence of instructions or a set of rules to get something done.

You probably know the fastest route from school to home, for example, turn left, drive for five miles,

turn right. You can think of this as an 'algorithm' – as a sequence of instructions to get you to your chosen destination. There are plenty of algorithms (i.e. routes) that will accomplish the same goal; in this case, there are even algorithms (such as in your satnav) for working out the shortest or fastest route.

### How are algorithms used in the real world?

Search engines such as Bing or Google use algorithms to put a set of search results into order, so that more often than not, the result we're looking for is at the top of the front page.

Your Facebook news feed is derived from your friends' status updates and other activity, but it only shows that activity which the algorithm (EdgeRank) thinks you'll be most interested in seeing. The recommendations you get from Amazon, Netflix and eBay are algorithmically generated, based in part on what other people are interested in.

Given the extent to which so much of their lives is affected by algorithms, it's worth pupils having some grasp of what an algorithm is.

### How are algorithms used across the curriculum?

Helping pupils to get an *idea* of what an algorithm is needn't be confined to computing lessons. You and your pupils will already use algorithms in many different ways across the school.

- A lesson plan can be regarded as an algorithm for teaching a lesson.
- There will be a sequence of steps pupils follow for many activities, such as getting ready for lunch or going to PE.
- In cookery, we can think of a recipe as an algorithm.
- In English, we can think of instructional writing as a form of algorithm.
- In science, we might talk about the method of an experiment as an algorithm.
- In maths, your approach to mental arithmetic (or many computer-based educational games) might be an implementation of a simple algorithm.

An example of this might be:

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

## Where do algorithms fit in the new computing curriculum?

The computing curriculum expects pupils at Level 1 to have an understanding of algorithms as a sequence of steps.

There can be many algorithms to solve the same problem, and each of these can be implemented using different programming languages on different computer systems: it can be useful for pupils to compare how they draw a square with a floor turtle and how they would do this on screen in Logo or ScratchJr.



Scratch Jnr programming for drawing a square.

Level 2 builds on this: pupils are expected to design programs with particular goals in mind, which will draw on their being able to think algorithmically, as well as using logical reasoning (see pages 8–10) to explain algorithms and to detect and correct errors in them. To practise this, encourage pupils to carry out the steps for an algorithm: to follow the instructions themselves rather than writing these as code for the computer. Errors and inconsistencies should become apparent!

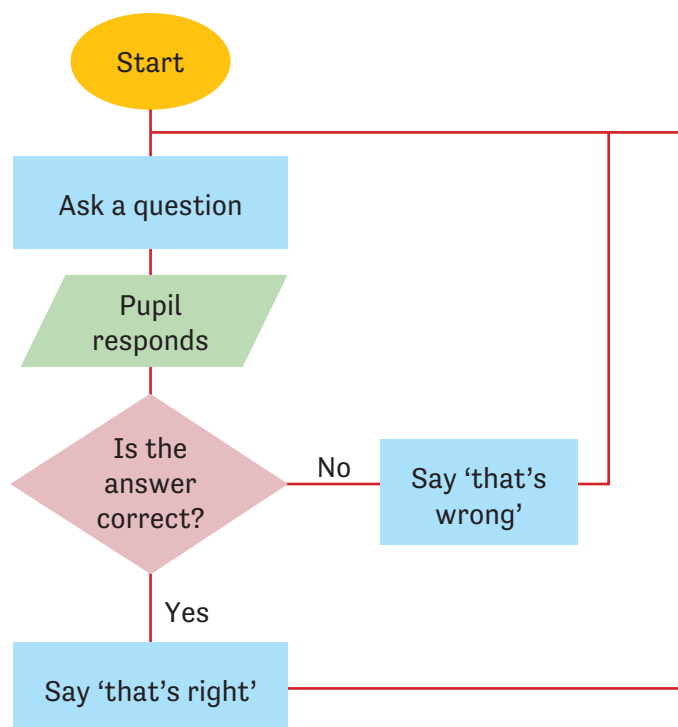
Whilst programming languages like Scratch and Kodu (see pages 21–22) can make it seem unnecessary to go through the planning stage of writing a program,

it is good practice for pupils to write down the algorithm for a program, perhaps as rough jottings, a storyboard, pseudocode (a written description of how a program will operate) or even as a flow chart (see below). This makes it far easier for them to get feedback from you or their peers on their algorithms before implementing these as code on their computers.

**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 activity ideas

- Talk to the pupils about what makes one algorithm better than another. In early programming work, pupils will come to realise that a Bee-Bot program which uses fewer steps than another to get to the same place is quicker to type and quicker to run.
- Play the classroom game 'Guess my number' to demonstrate this. Tell the pupils that you have

chosen a number between 1 and 100 and they are to guess what it is. Tell them that they can ask you questions about the number but that you can only answer 'yes' or 'no', and that they can only ask you one question per pupil.

- » For the first go, ask the pupils to guess numbers randomly.
- » Next, using a new number, ask the pupils to guess the number sequentially from one, e.g. 'Is the number one?' and so on. Explain that this is called a linear search. Allow them to have as many goes as needed to guess the number.
- » Finally, using a new number again, explain how to use a binary search. Explain to the learners that they already know the number is less than 100, so suggest they ask 'Is it less than 50?'. then 'Is it less than 25?' or 'Is it less than 75?' depending on the answer. Tell the pupils to keep halving the section they are searching in until the number is found.
- » Afterwards, talk about which approach found the number quicker. When they are familiar with using a binary search method, replay the game using a number between 1 and 1000.
- Organise the pupils to sort a set of unknown weights into weight order using a simple pan balance, thinking carefully about the algorithm they're following to do this, and then to think of a quicker way to accomplish the same activity. See <http://csunplugged.org/sorting-algorithms> for a demonstration of this.
- Explain to the pupils that not all algorithms are made of sequences of instructions: some are rule based. Introduce rule-based algorithms by writing a number sequence on the board, e.g. 3, 6, 9, 12 or 2, 4, 8, 16. Ask the pupils to work out the rule for the sequence (adding 3, or doubling the number) and to predict the next number. Explain that the rule for the sequence is the algorithm and the process by which they worked it out was logical reasoning.

## www Further resources

- Bagge, P., 'Flow Charts in Primary Computing Science', available at: <http://philbagge.blogspot.co.uk/2014/04/flow-charts-in-primary-computing-science.html>.
- Barefoot Computing, 'KS2 Logical Number Sequences Activity', available at: <http://barefootcas.org.uk/programme-of-study/use-logical-reasoning-explain-simple-algorithms-work/ks2-logical-number-sequences-activity/> (free, but registration required).

- Cormen, T., 'Algorithms Unlocked' (MIT Press, 2013).
- Peyton Jones, S. and Goldberg, A. (Microsoft Research), 'Getting from A to B: Fast Route-Finding Using Slow Computers', available at: [www.ukuug.org/events/agm2010/ShortestPath.pdf](http://www.ukuug.org/events/agm2010/ShortestPath.pdf).
- Slavin, K., 'How Algorithms Shape Our World', available at: [www.ted.com/talks/kevin\\_slavin\\_how\\_algorithms\\_shape\\_our\\_world?language=en](http://www.ted.com/talks/kevin_slavin_how_algorithms_shape_our_world?language=en).
- Steiner, C., 'Automate This: How Algorithms Came to Rule Our World' (Portfolio Penguin, 2013).

## Decomposition

How do I solve a problem by breaking it into smaller parts?

The process of breaking down a problem into smaller manageable parts is known as decomposition. Decomposition helps us solve complex problems and manage large projects.

This approach has many advantages. It makes the process a manageable and achievable one – large problems are daunting, but a set of smaller, related tasks is much easier to take on. It also means that the task can be tackled by a team working together, each bringing their own insights, experience and skills to the task.

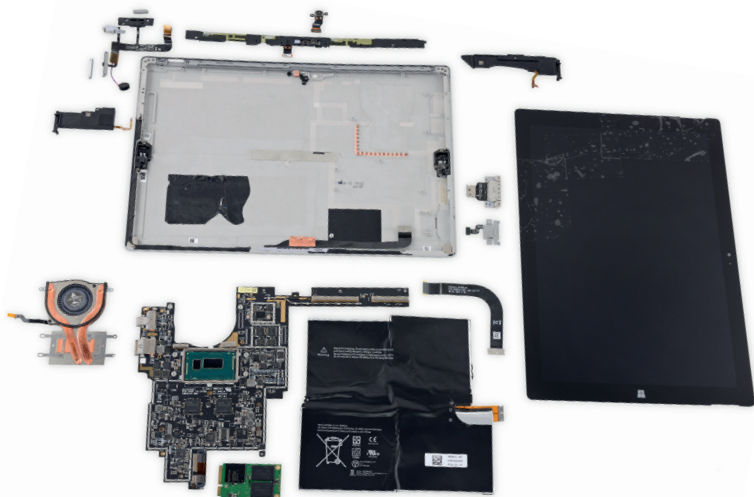
How is decomposition used in the real world?

Decomposing problems into their smaller parts is not unique to computing: it's pretty standard in engineering, design and project management.

Software development is a complex process, and so being able to break down a large project into its component parts is essential – think of all the different elements that need to be combined to produce a program like PowerPoint.

The same is true of computer hardware: a smartphone or a laptop computer is itself composed of many components, often produced independently by specialist manufacturers and assembled to make the finished product, each

under the control of the operating system and applications.

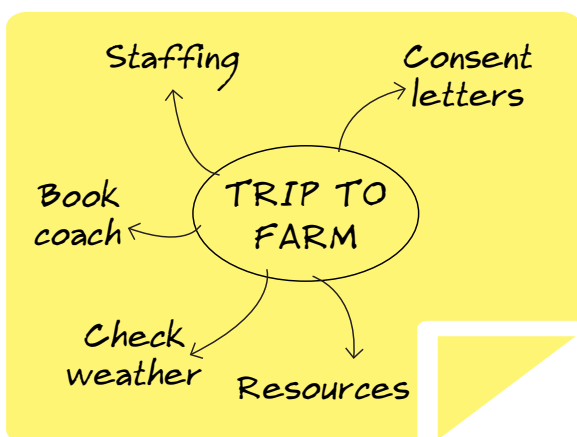


A tablet can be broken down (decomposed) into smaller components.

## How is decomposition used in school?

You'll have used decomposition to tackle big projects at school, just as programmers do in the software industry. For example:

- Delivering your school's curriculum — typically this would be decomposed as years and subjects, further decomposed into terms, units of work and individual lessons or activities. Notice how the project is tackled by a team working together (your colleagues), and how important it is for the parts to integrate properly.
- Putting on a school play, organising a school trip or arranging a school fair



A task such as organising a school trip can be decomposed into smaller chunks.

## How is decomposition used across the curriculum?

You and your pupils will already use decomposition in many different ways across the curriculum.

- In science or geography, labelling diagrams to show the different parts of a plant, or the different nations which make up the UK.
- In English, planning the different parts of a story.
- In general project planning, planning a research project for any subject or working collaboratively to deliver a group presentation. Technology can help with this sort of collaborative group work, or can even be a focus for it, and great collaborative tools are available in Office 365 and other software.
- In maths, breaking down a problem to solve it.

## Where does decomposition fit in the new computing curriculum?

The computing curriculum expects that, at Level 1, pupils simplify problems by breaking them down into smaller, more manageable parts<sup>3</sup>.

As pupils plan their **programs** or systems, encourage them to use decomposition: to work out what the different parts of the program or system must do, and to think about how these are inter-related. For example, a simple educational game is going to need 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 get some experience of working as a collaborative team on a software development project, and indeed other projects in computing. This could be 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.

<sup>3</sup> Technologies Benchmarks [Education Scotland 2017]



## Classroom activity ideas

- Organise for the pupils to tackle a large-scale programming project, such as making a computer game, 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.
- Take the case off an old desktop computer and show the pupils how computers are made from systems of smaller components connected together. Depending on the components involved, some of these can be disassembled further still, although it's likely to be better to look at illustrations of the internal architecture of such components.
- Organise for the pupils to carry out a collaborative project online, for example through developing a multi-page wiki site. For example, pupils could 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 process of writing these pages can be further decomposed, through planning, research, drafting, reviewing and publishing phases.



## Further resources

- Apps for Good, available at: [www.appsforgood.org/](http://www.appsforgood.org/).
- Barefoot Computing, 'Decomposition', available at: <http://barefootcas.org.uk/sample-resources/decomposition/> (free, but registration required).
- Basecamp (professional project management software) can be used by teachers with their class (free), available at: <https://basecamp.com/teachers>.
- Gadget Teardowns, available at: [www.ifixit.com/Teardown](http://www.ifixit.com/Teardown).
- NRICH, 'Planning a School Trip', available at: <http://nrich.maths.org/6969>.
- Project Management Institute Educational Foundation, 'Project Management Toolkit for Youth', available at: <http://pmief.org/learning-resources/learning-resources-library/project-management-toolkit-for-youth>.

# Abstraction

## How do you manage complexity?

For American computer scientist Jeanette Wing, 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.<sup>4</sup>*

Abstraction is about simplifying things, identifying what is important without worrying too much about the detail. Abstraction allows us to manage complexity.

We use abstractions to manage the complexity of life in schools. For example, the school timetable is an abstraction of what happens in a typical week: it captures key information such as who is taught which subject where and by whom, but leaves to one side further layers of complexity, such as the learning objectives and activities planned in any individual lesson.

## How is abstraction used across the curriculum?

Abstraction is such a powerful way of thinking about systems and problems that it seems worth introducing pupils to this whilst they're still at primary school. This doesn't have to be just in computing lessons.

- In maths, working with 'word problems' often involves a process of identifying the key information and establishing how to represent the problem in the more abstract language of arithmetic, algebra or geometry.
- In geography, pupils can be helped to see a map as an abstraction of the complexity of the environment, with maps of different scales providing some sense of the layered nature of abstraction in computing.
- In history, pupils are taught world history or national history as an abstraction of the detail present in local histories and individual biographies, which are themselves abstractions of actual events.

- In music, the piano score of a pop song might be thought of as an abstraction for that piece of music.

## Where does abstraction fit in the new computing curriculum?

Abstraction as a separate concept appears at Level 3, where pupils are:

*developing their understanding of information and can use an information model to describe particular aspects of a real-world system.*<sup>5</sup>

In computing lessons, pupils can learn about the process of abstraction from playing computer games, particularly those that involve interactive simulations of real-world systems (see Classroom activity ideas). Encourage pupils' curiosity about how things work, helping them to think about what happens inside the computer or on the internet as they use software or browse the web.

When pupils put together a presentation or video on a topic they know about, they'll need to focus on the key information and think about how this can be represented, whilst leaving to one side much of the detail of the subject: this too involves abstraction.



### Classroom activity ideas

- Encourage pupils who are learning to program to create their own games. If these are based on real-world systems then they'll need to use some abstraction to manage the complexity of that system in their game. In a simple table tennis game, e.g. Pong, the **simulation** includes the ball's motion in two dimensions and how it bounces off the bat, but it ignores factors such as air resistance, spin or even gravity. Ask your pupils to think really carefully about what detail they need to include, and what can be left out when programming a similar game.



### Further resources

- Barefoot Computing, 'Abstraction', available at: <http://barefootcas.org.uk/barefoot-primary-computing-resources/concepts/abstraction/> (free, but registration required).
- BBC Bitesize, 'Abstraction', available at: [www.bbc.co.uk/education/guides/zttcrdm/revision](http://www.bbc.co.uk/education/guides/zttcrdm/revision).
- BBC Cracking the Code, 'Simulating the Experience of F1 Racing Through Realistic Computer Models', available at: [www.bbc.co.uk/programmes/p016612j](http://www.bbc.co.uk/programmes/p016612j).
- Google for Education, 'Solving Problems at Google Using Computational Thinking', available at: [www.youtube.com/watch?v=SVVB5RQfYxk](http://www.youtube.com/watch?v=SVVB5RQfYxk).
- 'The Art of Abstraction – Computerphile', available at: [www.youtube.com/watch?v=p7nGcY73epw](http://www.youtube.com/watch?v=p7nGcY73epw).

## Patterns and generalisation

### How can you make things easier for yourself?

In computing, the method of looking for a general approach to a class of problems is called generalisation. By identifying patterns, we can make predictions, create rules and solve more general problems. For example, in learning about area, pupils *could* find the area of a particular rectangle by counting the centimetre squares on the grid on which it's drawn. But a better solution would be to multiply the length by the width: not only is this quicker, it's also a method that will work on *all* rectangles, including really small ones and really large ones. Although it takes a while for pupils to understand this formula, once they do it's so much faster than counting squares.

### How are patterns and generalisation used in the Scottish curriculum?

Pupils are likely to encounter the idea of generalising patterns in many areas of the primary curriculum.

<sup>5</sup> Technologies Benchmarks [Education Scotland 2017]

- From an early age, they'll become familiar with repeated phrases in nursery rhymes and stories; later on they'll notice repeated narrative structures in traditional tales or other genres.
- In music, children will learn to recognise repeating melodies or bass lines in many musical forms.
- In maths, pupils typically undertake investigations in which they spot patterns and deduce generalised results.
- In English, pupils might notice common rules for spellings, and their exceptions.



### Classroom activity ideas

- In computing, encourage pupils to always look for simpler or quicker ways to solve a problem or achieve a result. Ask pupils to explore geometric patterns, using turtle graphics commands in languages like Scratch, Logo or TouchDevelop to create 'crystal flowers' (see pages 26–27). Emphasise how the use of repeating blocks of code is much more efficient than writing each command separately, and allow pupils to experiment with how changing one or two of the numbers used in their program can produce different shapes.
- Organise for the pupils to use graphics software to create tessellating patterns to cover the screen. As they do this, ask them to find quicker ways of completing the pattern, typically by copying and pasting groups of individual shapes.
- Help the pupils to create rhythmic and effective music compositions using simple sequencing software in which patterns of beats are repeated.
- Ask the pupils to experiment with number patterns and sequences using Scratch or other programming languages. Can they work out a general program which they could use to generate any linear number sequence?



### Further resources

- Barefoot Computing, 'Patterns', available at: <http://barefootcas.org.uk/barefoot-primary-computing-resources/concepts/patterns/> (free, but registration required).
- Isle of Tune app, available at: <http://isleoftune.com>.
- Laurillard, D., *Teaching as a Design Science: Building Pedagogical Patterns for Learning and Technology* (Routledge, 2012).
- Pattern in Islamic art, available at: [www.patterninislamicart.com](http://www.patterninislamicart.com).

- M. C. Escher website, available at: [www.mcescher.com](http://www.mcescher.com).

## How does software get written?

As well as the above processes, there are also a number of approaches that characterise computational thinking. If pupils are to start thinking computationally, then it's worth helping them to develop these approaches to their work, so they can be more effective in putting their thoughts into action.

### Tinkering

There is often a willingness to experiment and explore in computer scientists' work. Some elements of learning a new programming language, or exploring a new, system look quite similar to the sort of purposeful play that's seen as such an effective approach to learning in the best nursery and reception classrooms.

**Open-source software** makes it easy to take someone else's code, look at how it's been made and then adapt it to your own particular project or purpose. Platforms such as Scratch and TouchDevelop positively encourage users to look at other programmers' work and use this as a basis for their own creative coding.

In class, encourage pupils to play with a new piece of software, sharing what they discover about it with one another, rather than you explaining exactly how it works. Also, look for ways in which pupils can use others' code, from you, their peers, or online, as a starting point for their own programming projects.

### Creating

Programming *is* a creative process. Creative work involves both originality and making something of value: typically something that is useful or at least fit for the purpose intended.

Encourage pupils to approach tasks with a creative spirit, and look for programming tasks that allow some scope for creative expression rather than merely arriving at the right answer.

Encourage pupils to reflect on the quality of the work they produce, critiquing their own and others' projects. The process of always looking for ways to



improve on a software project is becoming common practice in software development. Look for projects in which artistic creativity is emphasised, such as working with digital music, images, animation, virtual environments or even 3D printing.

## Debugging

Because of its complexity, the code programmers write often doesn't work as it's intended.

Getting pupils to take responsibility for thinking through their algorithms and code, to identify and fix errors, is an important part of learning how to think and work like a programmer. It's also something to encourage across the curriculum: get pupils to check through their working in maths, or to proofread their stories in English. Ask pupils to debug one another's code (or indeed proofread one another's work), looking for mistakes and suggesting improvements. There's evidence that learning from mistakes is a particularly effective approach, and the process of pupils debugging their own or others' code is one way to do this. Keep an eye on the bugs that your pupils do encounter, as these can sometimes reveal particular misconceptions that you may need to address (see pages 28–29).

## Persevering

Computer programming *is* hard. This is part of its appeal – writing elegant and effective code is an intellectual challenge requiring not only an understanding of the ideas of the algorithms being coded and the programming language you're working in, but also a willingness to persevere with something that's often quite difficult and sometimes very frustrating. Carol Dweck's work on 'growth mind-sets' suggests that hard work and a willingness to persevere in the face of difficulties can be key factors in educational outcomes. Encourage pupils to look for strategies they can use when they do encounter difficulties with their programming work, such as working out exactly what the problem is, searching for the solution on Bing or Google (with the safe search mode locked), KidRex or Swiggle, or asking a friend for help.

## Collaborating

Software is developed by teams of programmers and others working together on a shared project. Look for ways to provide pupils with this experience in computing lessons too. Collaborative group work has long had a place in primary education, and computing should be no different.

Many see 'pair programming' as a particularly effective development method, with two programmers sharing a screen and a keyboard, working together to write software. Typically, one programmer acts as the driver, dealing with the detail of the programming, whilst the other takes on a navigator role, looking at the bigger picture. The two programmers regularly swap roles, so both have a grasp of both detail and big picture. Working in a larger group develops a number of additional skills, with each pupil contributing some of their own particular talents to a shared project. However, it's important to remember that all pupils should develop their understanding of each part of the process, so some sharing of roles or peer-tutoring ought normally to be incorporated into such activities.



## Further resources

- Barefoot Computing, 'Computational Thinking Approaches', available at: <http://barefootcas.org.uk/barefoot-primary-computing-resources/computational-thinking-approaches/> (free, but registration required).
- Briggs, J., 'Programming with Scratch Software: The Benefits for Year Six Learners' (Bath Spa MA dissertation, 2013), available at: [https://slp.somerset.gov.uk/cypd/elim/somersetict/Computing\\_Curriculum\\_Primary/Planning/MA\\_JBriggs\\_Oct2013.pdf](https://slp.somerset.gov.uk/cypd/elim/somersetict/Computing_Curriculum_Primary/Planning/MA_JBriggs_Oct2013.pdf).
- DevArt: Art Made with Code, available at: <https://devart.withgoogle.com/>.
- Dweck, C., 'Mindset: How You Can Fulfil Your Potential' (Robinson, 2012).
- Education Endowment Foundation toolkit, available at: <http://educationendowmentfoundation.org.uk/toolkit/>.
- Papert, S. and Harel, I., 'Situating Constructionism' (Ablex Publishing Corporation, 1991), available at: [www.papert.org/articles/SituatingConstructionism.html](http://www.papert.org/articles/SituatingConstructionism.html).

# Programming

## What is programming?

Programming is the process of designing and writing a set of instructions (a program) for a computer in a language it can understand.

This can be really simple, such as the program to make a robot toy trace out a square; or it can be incredibly sophisticated, such as the software used to forecast the weather or to generate a set of ranked search results.

Programming is a two-step process.

- First, you need to analyse the problem or system and design a solution. This process will use logical reasoning, decomposition, abstraction and **generalisation** (see pages 6–17) to design algorithms to solve the problem or model the system.
- Secondly, you need to express these ideas in a particular programming language on a computer. This is called coding, and we can refer to the set of instructions that make up the program as ‘code’.

Programming provides the motivation for learning computer science – there’s a great sense of achievement when a computer does just what you ask it, because you’ve written the precise set of instructions necessary to make something happen. Programming also provides the opportunity to test out ideas and get immediate feedback on whether something works or not.

## What should programming be like in schools?

It’s possible to teach **computational thinking** without coding and vice versa, but the two seem to work best hand-in-hand.

Teaching computational thinking without giving pupils the opportunity to try out their ideas as code on a computer is like teaching science without doing any experiments. Similarly, teaching coding without helping pupils to understand the underlying processes of computational thinking is like doing experiments in science without any attempt to teach pupils the principles which underpin them.

This is reflected in the new computing curriculum, which states that pupils should not only know the principles of information and computation, but should also be able to put this knowledge to use through programming. One of the aims of the Scottish curriculum for computing is that pupils can analyse problems in computational terms and have repeated practical experience of writing computer programs in order to solve problems.

At Level 1, pupils should be taught *how* simple **algorithms** are implemented as programs on digital devices. The phrase ‘digital devices’ encompasses tablets, laptop computers, **programmable toys**, and perhaps also distant **web servers**. It can be useful for pupils to be able to see their algorithms (in whatever way they’ve recorded these) and their code, side by side.

Children can use simple arrow cards to record algorithms for programmable toys.



Pupils also should have the opportunity to create and **debug** (see pages 28–29) their own **programs**, as well as predicting what a program will do.

At Level 2, pupils should be taught to design and write programs that accomplish specific goals, which should include controlling or simulating physical systems, for example making and programming a Lego robot. They should be taught to use **sequence**, **selection** and **repetition** in their programs (see pages 24–28), as well as variables to store **data**. They should also learn to use logical reasoning to detect and fix the errors in their programs.



### Classroom activity ideas

- There are simple activities on the Barefoot Computing website; see Further resources below.
- Here are some ideas for extended programming projects:
  - » Year 2: solve a maze using a floor/screen turtle
  - » Year 3: create a simple animation
  - » Year 4: create a question-and-answer maths game
  - » Year 5: create more complex computer games
  - » Year 6: develop a simple app for a tablet or smartphone.



### Further resources

- Barefoot on 'Programming', available at: <http://barefootcas.org.uk/barefoot-primary-computing-resources/concepts/programming/> (free, but registration required).
- BBC Bitesize: Controlling physical systems, available at: [www.bbc.co.uk/guides/zxjsfg8](http://www.bbc.co.uk/guides/zxjsfg8).
- BBC Cracking the Code, for examples of source code for complex software systems such as robot footballers and a racing car simulator, available at: [www.bbc.co.uk/programmes/p01661pj](http://www.bbc.co.uk/programmes/p01661pj).
- CAS Chair, Prof Simon Peyton Jones' explanation of some of the computer science that forms the basis for the computing curriculum: <http://community.computingschool.org.uk/resources/2936>.
- Code.org for activities and resources, available at: <http://code.org/educate>.
- Rushkoff, D., *Program or be Programmed: Ten Commands for a Digital Age* (OR Books, 2010).

## How do you program a computer?

Programming a computer involves writing code. The code is the set of instructions for the computer written in a programming language that the computer understands. In fact, the programming languages we use are a halfway house – they're written in a language we can understand which then gets translated by the computer into the 'machine code' of instructions that can be run directly on the silicon chips which control it.

Programs comprise precise, unambiguous instructions – there's no room for interpretation or debate about the meaning of a particular line of computer code. 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.

### What programming languages should you use?

There are many languages to choose from. The majority are more complex than necessary for those just getting to grips with the ideas of programming, but there are plenty of simple, well supported languages that can be used very effectively in the primary classroom. Try to pick a language that you'll find easy to learn, or better still, know already.

Consider these points when choosing a programming language:

- Not all languages run on all computer systems.
- Choose a language that is suitable for your pupils. There are computer languages that are readily accessible to primary pupils – in most cases this will mean one that has been written with pupils in mind, or at least adapted to make it easier to learn.
- Choose a language supported by a good range of learning resources. It's better still if it has online support communities available, both for those who are teaching the language and those who are learning it.
- It is beneficial to the pupils if they can continue working in the language on their home computer, or, even better, if they can easily continue work on the same project via the internet.

There's a view that some languages are better at developing good programming 'habits' than others. Good teaching, in which computational thinking is emphasised alongside coding, should help to prevent pupils developing bad coding habits at this stage.

## Which language is right for which Level?

The table below illustrates a progressive approach to programming languages in a primary setting.

| Level       | Language type       | Language     | See                     |
|-------------|---------------------|--------------|-------------------------|
| Early Level | Device-specific     | Bee-Bot      | Page 20                 |
|             |                     | Roamer-Too   | Page 20                 |
| Level 1     | Limited instruction | ScratchJr    | Page 21                 |
|             |                     | Lightbot™    | Pages 20–21             |
| Level 2     | Game programming    | Kodu         | Pages 21, 22, 25 and 28 |
|             | Block-based         | Scratch      | Pages 21–22, 24–28      |
|             | Text-based          | Logo         | Pages 22–23, 26–27, 29  |
|             |                     | TouchDevelop | Page 23                 |

Whilst there's much to be said for letting pupils explore several programming languages, it's important that they develop a degree of fluency in one fairly general-purpose language, so that this becomes a medium in which they can solve problems, get useful things done and work creatively.

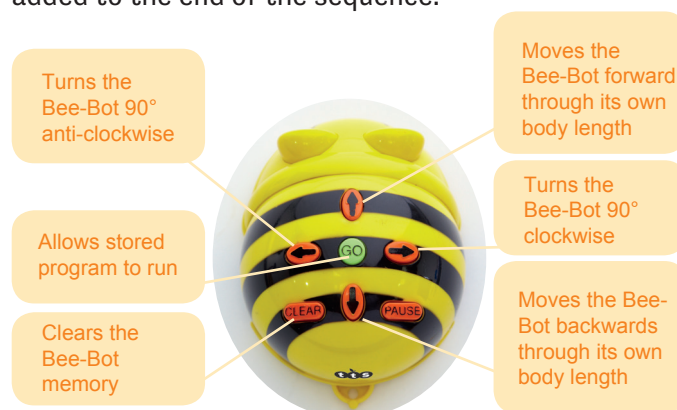
## www Further resources

- Iry's 'brief, incomplete and mostly wrong history of programming languages': <http://james-iry.blogspot.mx/2009/05/brief-incomplete-and-mostly-wrong.html>.
- Utting, I., Cooper, S., Kolling, M., Maloney, J. and Resnick, M., (2010) 'Alice, Greenfoot, and Scratch – A Discussion', available at: <http://kar.kent.ac.uk/30617/2/2010-11-TOCE-discussion.pdf>.
- Wikipedia list of 'Hello, world!' in many programming languages, available at: [http://en.wikipedia.org/wiki/List\\_of\\_Hello\\_world\\_program\\_examples](http://en.wikipedia.org/wiki/List_of_Hello_world_program_examples).

## How do you program a floor turtle?

Programming in Early Years and Level 1 is much more likely to involve working with simple programmable toys than using computers. It's much easier for pupils to learn the idea of programming when working with a really simple language and **interface**, and for them to plan and check their programs when they can, quite literally, put themselves in the place of the device they're programming.

A programmable floor turtle, such as the Bee-Bot or Roamer-Too, is ideal for this. The Bee-Bot programming language consists of five commands: forward, back, turn left, turn right and pause. Programming a Bee-Bot is simply a process of pressing buttons in the desired order to build a sequence of commands, with new commands being added to the end of the sequence.



This simple device can be used as a basis for many engaging activities, both for early programming and across the curriculum. Younger pupils will often work with the Bee-Bot one instruction at a time, whilst older children will become adept at creating longer sequences of instructions.

A number of tablet or smartphone apps and web-based tools are based on the idea of device-specific languages like these. These are often in the form of a game with a sequence of progressively harder levels in which players create ever more complex sequences of instructions to solve challenges. For example: Bee-Bot, Lightbot™, A.L.E.X and Cargo-Bot.

One approach for scaffolding the transition from floor-turtle programming to programming on-screen is to use an on-screen **simulation** of a Bee-Bot: it's relatively easy to make (or adapt) one yourself in Scratch 2.0.



## Classroom activity ideas

- Allow very young pupils to play with a floor turtle, tinkering with it so they can develop their own sense of the relationship between pressing buttons and running their program.
- Encourage pupils to plan a sequence of instructions for a particular objective, such as getting the floor turtle from one 'flower' to another. Ask pupils to predict what will happen when they run their program, and to explain their thinking (logical reasoning).
- 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.



## Further resources

- BBC on how to program a robot, available at: [www.bbc.co.uk/learningzone/clips/programming-robots/4391.html](http://www.bbc.co.uk/learningzone/clips/programming-robots/4391.html).
- Bee-Bots are available from TTS. Other programmable toys include Roamer-Too (by Valiant) and Pro-Bot (by TTS).
- Barefoot on 'KS1 Bee-Bots, 1, 2, 3 Programming Activity', available at: <http://barefootcas.org.uk/barefoot-primary-computing-resources/concepts/programming/ks1-bee-bots-12-3-programming-activity/> (free, but registration required).
- Bee-Bot and Roamer-Too simulator activities, available at: <http://scratch.mit.edu/projects/19799927/>.
- Lightbot™, available at: <http://lightbot.com/>.

# How do you program things to move around the screen?

There are a number of graphical programming toolkits available; these make learning to code easier than ever. 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, and the amount of typing, and thus the potential for spelling or punctuation errors, is kept to an absolute minimum.

With toolkits like these it's easy to experiment with creating code. By letting programmers focus on the ideas of their algorithm, rather than the particular vocabulary and grammar of the programming language, learning to program becomes easier and often needs less teacher input.

## Kodu

Microsoft's Kodu is a rich, graphical toolkit for developing simple, interactive 3D games.

Each object in the Kodu game world can have its own program. These programs are 'event driven': they are made up of sets of 'when [this happens], do [that]' conditions, so that particular actions are triggered when certain things happen, such as a key being pressed, one object hitting another, or the score reaching a certain level.

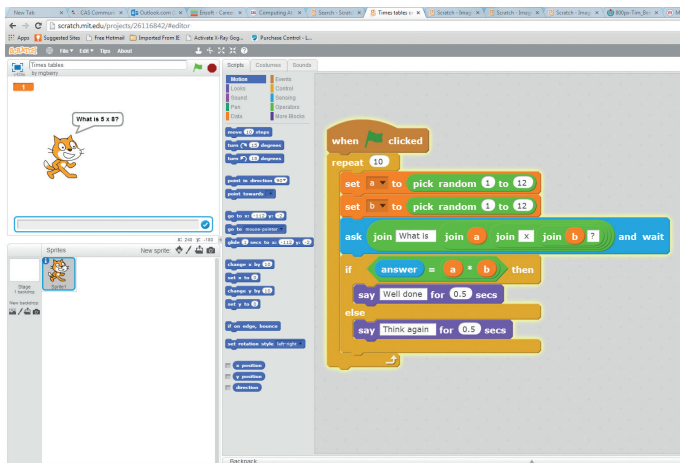


Kodu interface.

Programmers can share their games with others in the Kodu community, which facilitates informal and independent learning. There's plenty of scope for pupils to download and modify games developed by others, which many find quite an effective way to learn the craft of programming. This can also offer pupils a sense of creating games with an audience and purpose in mind.

## Scratch

In MIT's Scratch, the programmer can create their own graphical objects, including the stage background on which the action of a Scratch program happens, and a number of moving objects, or **sprites**, such as the characters in an animation or game.



Screenshot of a Scratch program.

Each object can have one or more scripts, built up using the building blocks of the Scratch language. To program an object in Scratch, you drag the colour-coded block you want from the different palettes of blocks, and snap this into place with other blocks to form a script. Scripts can run in parallel with one another or be triggered by particular events, as in Kodu.

A number of other projects use Scratch as a starting point for their own platforms; for example ScratchJr is an iPad app designed for young programmers (Level 1), and Berkeley's Snap! allows even more complex programming ideas (such as functions) to be explored through the same sort of building-block interface.

There's a great online community for Scratch developers to download and share projects globally, making it easier for pupils to pursue programming in Scratch far beyond what's needed for the Scottish curriculum. There's also a supportive educator community, which has developed and shared high-quality curriculum materials.

Scratch is available as a free web-based editor or as a standalone desktop application. Files can be moved between online and offline versions.



### Classroom activity ideas

- Pupils could develop a game in Kodu, taking inspiration from some of the games on the Kodu community site. As a starting point, tell them to create a game in which Kodu (the player's avatar in the game) is guided around the landscape bumping into (or 'zapping') enemies.

- Ask your pupils to create a simple scripted animation in Scratch, perhaps with a couple of programmed characters who take turns to act out a story. Designing the algorithm for a program like this is very similar to storyboarding in video work.



### Further resources

- Armoni, M. and Ben-Ari, M., 'Computer science concepts in Scratch', available at: [http://stwww.weizmann.ac.il/g-cs/scratch/scratch\\_en.html](http://stwww.weizmann.ac.il/g-cs/scratch/scratch_en.html).
- Berry, M., 'Scratch across the curriculum', available at: <http://milesberry.net/2012/06/scratch-across-the-curriculum/>.
- Creative Computing, 'An Introductory Computing Curriculum Using Scratch', available at: <http://scratched.gse.harvard.edu/guide/>.
- Kelly, J., *Kodu for Kids* (Que Publishing, 2013).
- Kinect2Scratch, to program Microsoft Kinect with Scratch, available at: <http://scratch.saorog.com/>.
- Kodu Game Lab Community, available at: [www.kodugamelab.com/](http://www.kodugamelab.com/).
- Other graphical programming environments for education include Espresso Coding, 2Code from 2Simple, and J2Code.
- Scratch, available at <http://scratch.mit.edu/>.
- ScratchEd online community for educators, available at: <http://scratch.mit.edu/educators/>.
- ScratchJr: [www.scratchjr.org/](http://www.scratchjr.org/).
- Snap!, available at: <http://snap.berkeley.edu/>.

## What is real programming?

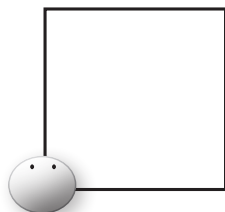
Most software development in academia and industry takes place using text-based languages, where programs are constructed by typing in the commands in a programming language using a keyboard.

Historically, text-based programming has been a real barrier for children when learning to code, and there's no need to rush into text-based programming as part of the primary curriculum. It is, however, worth considering text-based programming for an extra-curricular programming club or even in class, if you or your colleagues feel confident with this. Possible text-based programming languages for primary schools could include Logo and TouchDevelop.

## Logo

Logo was developed by Seymour Papert and others at MIT as an introductory programming language for children. It's probably best known for its use of 'turtle graphics' – an approach to creating images in which a 'turtle' (either a robot or a representation on-screen) is given instructions for drawing a shape, such as:

```
REPEAT 4 [
  FORWARD 100
  RIGHT 90 ]
```



Papert saw Logo as a tool for children to *think* with, just as programming is both the means to, and motivation for, computational thinking.

In Logo programming, more complex programs are built up by 'teaching' the computer new words. These are called procedures; for example, defining a procedure to draw a square of a certain size using the key words of the language. Once you have defined the procedure 'square', typing it in will then result in the turtle drawing a square. For example:

```
TO SQUARE :SIDE
  REPEAT 4 [
    FORWARD :SIDE
    RIGHT 90 ]
END
SQUARE 50
```

## TouchDevelop

Typing code on a tablet computer or a smartphone is not easy, and this can be problematic for schools that use these devices extensively.

Developed by Microsoft Research, TouchDevelop is a programming language and environment, which takes into account both the challenges posed and the opportunities offered by touch-based interfaces, such as those on tablets and smartphones.

TouchDevelop makes it quite easy to develop an app for a smartphone or tablet on the device.

Although TouchDevelop is a text-based language, programmes aren't *typed* but are created by choosing commands from the options displayed in a menu system. In this way, TouchDevelop is a halfway house between graphical and text-based programming.

As with Logo, turtle graphics commands are available as standard. On many platforms, TouchDevelop can also access some of the additional **hardware** built into the device, such as the accelerometer or GPS locator, allowing more complex apps to be developed; these can be hosted online as web-based apps, or installed directly on the device if it's a Windows phone.

```
action main ()
  for 0 ≤ i < 4 do
    turtle → forward(100)
    turtle → right turn(90)
  end for
end action
```

Program to draw a square using a turtle.

A particularly nice feature of TouchDevelop is the use of interactive tutorials to scaffold pupils' learning of the language.



### Classroom activity ideas

- Revisit the turtle graphics activities you might have been using for programming in the past.
- Explore how different programming languages can be used to simulate dice being rolled. First, ask pupils to think about how they would do that in Scratch. Then, challenge your pupils to create an app in TouchDevelop which simulates rolling a dice when the phone or tablet is shaken, or when the screen is tapped. Ask pupils to think about how deterministic computers can simulate random events such as these.



### Further resources

- Archived lesson plan from DfES for creating crystal flowers: <http://webarchive.nationalarchives.gov.uk/20090608182316/http://standards.dfes.gov.uk/pdf/primaryschemes/itx4e.pdf>.
- Horspool, N. and Ball, T., *TouchDevelop: Programming on the Go* (APress, 2013), available at: [www.touchdevelop.com/docs/book](http://www.touchdevelop.com/docs/book).
- Logo, available at: [www.calormen.com/jslogo/](http://www.calormen.com/jslogo/) and elsewhere.
- Papert, S., *Mindstorms: Children, Computers, and Powerful Ideas* (Basic Books Inc., 1980), available at: <http://dl.acm.org/citation.cfm?id=1095592>.
- TouchDevelop interactive tutorials for Hour of Code™: [www.touchdevelop.com/hourofcode2](http://www.touchdevelop.com/hourofcode2).
- TouchDevelop from Microsoft Research: [www.touchdevelop.com/](http://www.touchdevelop.com/).

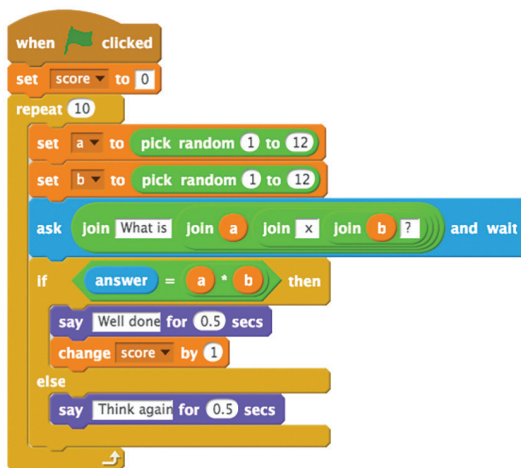
# What's inside a program?

Whilst the detail will vary from one language to another, there are some common structures and ideas which programmers use over and over again, from one language to another and from one problem to another:

- Sequence: running instructions in order (see below)
- Selection: running one set of instructions or another, depending on what happens (see pages 25–26)
- Repetition: running some instructions several times (see pages 26–27)
- Variables: a way of storing and retrieving data from the computer's memory (see pages 27–28).

These are so useful that it's important to make sure all pupils learn these.

This Scratch **script** shows sequence, selection, repetition and variables. Can you work out which bit is which, before we look at these ideas in detail?



## Further resources

- BBC Bitesize programming tutorial 'How do we get computers to do what we want?' (covering sequence, selection and repetition), available at: [www.bbc.co.uk/guides/z23q7ty](http://www.bbc.co.uk/guides/z23q7ty).
- Cracking the Code clip, available at: [www.bbc.co.uk/programmes/p016j4g5](http://www.bbc.co.uk/programmes/p016j4g5).
- Scratch multiplication test, available at: <http://scratch.mit.edu/projects/26116842/#editor>.

## Sequence

Programs are built up of sequences of instructions. When pupils start programming with floor turtles, their programs consist entirely of sequences of instructions, built up as the stored sequence of

button presses for what the floor turtle should do. As with any program, these instructions are precise and unambiguous, and the floor turtle will simply take each instruction (the stored button presses) and turn that into signals for the motors driving its wheels.

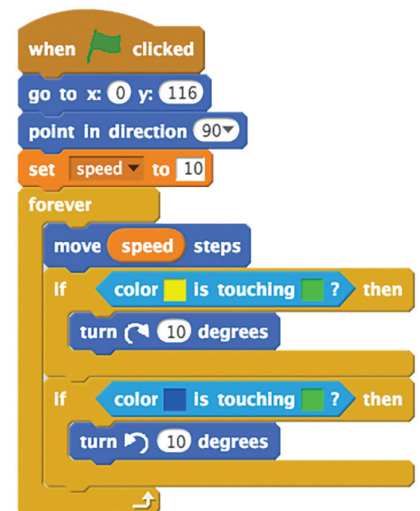
Initially, pupils might type in just one instruction at a time, clearing the memory after each, but as they become more experienced as programmers, or want to solve a problem more quickly, sequences become more complex.

**Forward**  
**Forward**  
**Forward**  
**Turn left**  
**Forward**  
**Forward**

Pupils' first Scratch programs are also likely to be made up of simple sequences of instructions. Again, these need to be precise and unambiguous, and of course the order

of the instructions matters. In developing their algorithms, pupils will have had to work out exactly what order to put the steps in to complete a task.

A program that children might create in Scratch.



## Classroom activity ideas

- Give pupils progressively more complex problems to solve with a floor turtle, asking them first to plan their algorithm for solving these before creating single programs on the floor turtle.
- Provide pupils with existing projects from Scratch (see Further resources on page 26). Allowing them to 'remix' these projects, by changing the code and seeing how this affects the program, is a useful learning experience.
- Ask pupils to design, plan and code scripted animations in Scratch, perhaps using a timeline or storyboard to work out their algorithm before converting this into instructions for sprites in Scratch.





## Further resources

- Animation 14: UK Schools Computer Animation Competition (Level 2), available at: <http://animation14.cs.manchester.ac.uk/gallery/winners/KS2/>.
- Barefoot on 'Sequence', available at: <http://barefootcas.org.uk/barefoot-primary-computing-resources/concepts/programming/sequence/> (free, but registration required).
- Cracking the Code clip on programming a robotic toy car: [www.bbc.co.uk/programmes/p01661yg](http://www.bbc.co.uk/programmes/p01661yg).
- Viking invasion animation in Scratch from Barefoot Computing (for Level 2), available at: <http://barefootcas.org.uk/programme-of-study-use-sequence-in-programs/upper-ks2-viking-raid-animation-activity/> (free, but registration required).

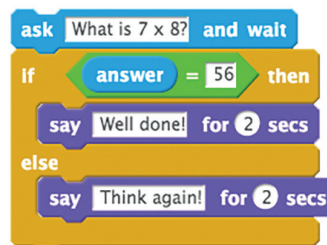
## Selection

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.

Selection is an important part of creating a game in Kodu. An object's behaviour in a game is determined by a set of conditions; for example: WHEN the left arrow is pressed, the object will move left. Similarly, interaction with other objects, variables and environments in Kodu is programmed as a set of WHEN ... DO ... conditions; for example, WHEN I bump the apple DO eat it AND add 2 points to score.



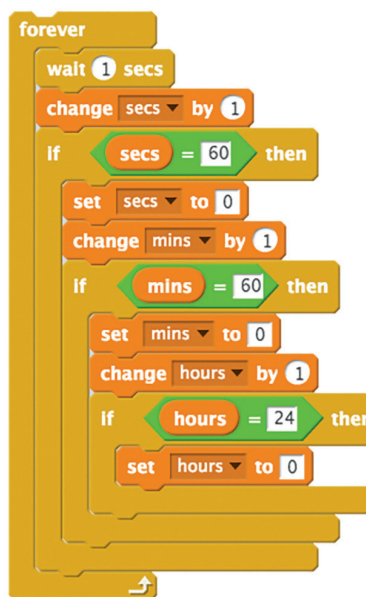
In Scratch (and other programming languages), you can build selection into a sequence of instructions, allowing the computer to run different instructions depending on whether a condition is met.



Examples of how selection can be used to start a script in Scratch.

At the core of many educational games is a simple selection command: if the answer is right then give a reward, else say the answer is wrong. See the Scratch script for the multiplication game in the Further resources on [page 24](#).

It's also worth noting that selection statements can be nested inside one another. This allows more complex sets of conditions to be used to determine what happens in a program. Look at the way some *if* blocks are inside others in the following script to model a clock in Scratch. The script also uses repetition and three variables for the seconds, minutes and hours of the time:



## Classroom activity ideas

- Encourage pupils to explore the different conditions to which the character in Kodu can respond in its event-driven programming. Get pupils to think creatively about how they might use these when developing a game of their own. Give them time to design their game, thinking carefully about the algorithm, i.e. the rules they're using.

- Ask pupils to design simple question-and-answer games in Scratch. Encourage them to first think about the overall algorithm for their game before coding this and working to develop the user interface, making this more engaging than just a cat asking lots of questions. It's helpful if pupils have a target audience in mind for software like this.



## Further resources

- Barefoot Computing on 'Selection', available at: <http://barefootcas.org.uk/programme-of-study/use-selection-programs/selection/> (free, but registration required).
- Papert, S., 'Does Easy Do It? Children, Games, and Learning', available at: [www.papert.org/articles/Doeseasydoit.html](http://www.papert.org/articles/Doeseasydoit.html).

## Scratch projects to remix

- Analogue clock by mgberry on Scratch, available at: <http://scratch.mit.edu/projects/28742256/#editor>.
- Addition race by mgberry on Scratch, available at: <http://scratch.mit.edu/projects/15905989/#editor>.

## Repetition

Repetition in programming means to repeat the execution of certain instructions. This can make a long sequence of instructions much shorter, and typically easier to understand.

Using repetition in programming usually involves spotting that some of the instructions you want the computer to follow are the same, or very similar, and therefore draws on the computational thinking process of pattern recognition/generalisation (see pages 15–16). You'll sometimes hear the repeating block of code referred to as a loop, i.e. the computer keeps looping through the commands one at a time as they're executed (carried out).

Think about the Bee-Bot program for a square (**forward, left, forward, left, forward, left, forward, left**). Now, for each side, we move forward and then turn left. On a Roamer-Too or a Pro-Bot, you could use the repeat command to simplify the coding for this by using the built-in repeat command, replacing this code with, for example, **repeat 4 [forward, left]**.

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

Compare:

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

with:

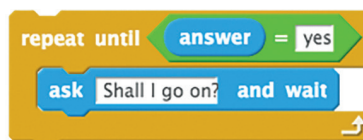
```
REPEAT 3 [
  FORWARD 100
  LEFT 120 ]
```

Both programs draw equilateral triangles. Using repetition reduces the amount of typing and makes the program reflect the underlying algorithm more clearly.

In the examples above, the repeated code is run a fixed number of times, which is the best way to introduce the idea. You can also repeat code forever. This can be useful in real-world systems, such as a control program for a digital thermostat, which would continually check the temperature of a room, sending a signal to turn the heating on when this dropped below a certain value. This is a common technique in game programming. For example, the following Scratch code would make a sprite continually chase another around the screen:



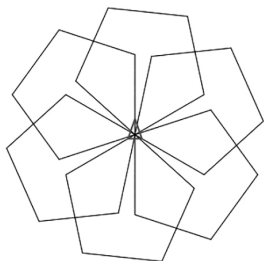
Repetition can be combined with selection, so that a repeating block of code is run as many times as necessary until a certain condition is met, as in this fragment in Scratch:



You can nest one repeating block inside another. The 'crystal flower' programs in Logo use this idea. For example:

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

draws:



## Classroom activity ideas

- Ask pupils to use simple repetition commands to produce a 'fish tank' animation in Scratch, with a number of different sprites each running their own set of repeating motion instructions. This can be made more complex by including some selection commands to change the behaviour of sprites as they touch one another.
- Encourage pupils to experiment with 'crystal flower' programs in Scratch, Logo or other languages that support turtle graphics, and investigate the effect of changing the number of times a loop repeats, as well as the parameters for the commands inside the loop. There are some great opportunities to link computing with spiritual, social and cultural education.



## Further resources

- Barefoot Computing on 'Repetition', available at: <http://barefootcas.org.uk/programme-of-study/use-repetition-programs/repetition> (free, but registration required).
- Digital Schoolhouse dance scripts, available at: [www.resources.digitalschoolhouse.org.uk/key-stage-2-ages-7-10/218-scratch-teaching-dance](http://www.resources.digitalschoolhouse.org.uk/key-stage-2-ages-7-10/218-scratch-teaching-dance).
- Scratch 2.0 Fishtank Game tutorial, available at: [www.youtube.com/watch?v=-qTZ5bFedC8](http://www.youtube.com/watch?v=-qTZ5bFedC8).

## Variables

Unlike the programming structures of sequence, selection and repetition, a variable is an example of a data structure. It is a simple way of storing one piece of information somewhere in the computer's memory whilst the program is running, and getting that information back later. There's a degree of abstraction involved. The actual details of how the programming language, operating system and hardware manage the storing and retrieving of data from the computer's memory chips aren't important to us as programmers. In the same way, these details aren't important when we're using the clipboard for copying and pasting text. One way of thinking of variables is as labelled shoeboxes, with

the difference that the contents don't get removed when they're used.

The concept of a variable is one that many pupils struggle with and it's worth showing them lots of examples to ensure they grasp this. A classic example which pupils are likely to be familiar with, particularly from computer games, is that of score.

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

```
ask Hello! What's your name? and wait
set name to answer
say join Hello name for 2 secs
say join name is a very nice name. for 2 secs
```

Here, name is a variable, in which we store whatever the user types in, and then use it a couple of times in Scratch's response; answer is a special temporary variable used by Scratch to store for the time being whatever the user types in. Notice that variables can store text as well as numbers. Other types of data can be stored in variables too, depending on the particular programming language you're working in.

Variables can also be created by the program, perhaps to store a constant value so that we can refer to it by name (Pi, below), the result of a computation (Circumference in the code below), or random numbers generated by the computer (e.g. Radius below):

```
set Pi to 3.14
set Radius to pick random 1 to 10
set Circumference to 2 * PI * Radius
say join If a circle has a radius of Radius for 2 secs
say join Its circumference would be Circumference for 2 secs
```

The idea that the contents of the 'box' are still there after the variable is used is sometimes a confusing one for those learning to program. Have a look at the following code and decide what will be displayed on the screen:

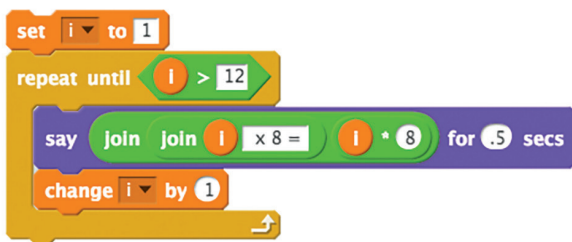
```
set a to 10
set b to 20
set a to b
say join a is a for 2 secs
say join b is b for 2 secs
```

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

In Kodu and other game programming, variables are useful for keeping track of rewards, such as a score, and for introducing some sort of limit, such as a time limit or health points that reduce each time you're hit. Kodu's event-driven approach allows particular actions to be done when variables reach a predetermined level.



One particularly useful example of variables in programming is as an iterator – this is a way of keeping track of how many times you've been round a repeating loop, and of doing something different each time you do. To achieve this, at the beginning of the loop we initialise a counter to zero or one and then add one to it each time we go round the loop. For example, the following script would get Scratch to say its eight times table:



You can also use an iterator like this to work with strings (words and sentences) one letter at a time, or through lists of data one item at a time. Take care with the beginning and end, as it's all too easy to start or end too soon or too late with iterators.



## Classroom activity ideas

- In Scratch, get pupils to create a mystery function machine which accepts an **input**, stores this in a variable and then uses mathematical operators to produce an **output** shown on-screen. Setting the display to full screen in Scratch, pupils can challenge one another (and you) to work out what the program does by trying different inputs.
- Pupils can use variables in their games programs, in say Scratch or Kodu, using a score to reward the player for achieving particular objectives (such as collecting apples), and imposing a time limit.



## Further resources

- Bagge, P., 'Text Adventure Game' for Scratch, available at: [http://code-it.co.uk/year4/text\\_adventure\\_game.pdf](http://code-it.co.uk/year4/text_adventure_game.pdf).
- Barefoot Computing on 'Variables', available at: <http://barefootcas.org.uk/programme-of-study/work-variables/variables/> (free, but registration required).
- BBC Bitesize article 'How do computer programs use variables?', available at: [www.bbc.co.uk/guides/zw3dwmn](http://www.bbc.co.uk/guides/zw3dwmn).
- Binary search jigsaw and solution by mgberry, available at: <http://scratch.mit.edu/projects/20255402/> and <http://scratch.mit.edu/projects/28907496/>.
- How to program a Scratch 2.0 times-table test, available at: [www.youtube.com/watch?v=YHGyPfgG1x8](http://www.youtube.com/watch?v=YHGyPfgG1x8).
- Notes and tutorial on variables in Scratch, available at: <http://wiki.scratch.mit.edu/wiki/Variable> and [http://wiki.scratch.mit.edu/wiki/Variables\\_Tutorial](http://wiki.scratch.mit.edu/wiki/Variables_Tutorial).

## Can we fix the code?

Errors in algorithms and code are called 'bugs', and the process of finding and fixing these is called 'debugging'. Debugging can often take much longer than writing the code in the first place. Whilst fixing a program so that it does work can bring a great buzz, staring at code that still won't work can be the cause of great frustration too. This can be tricky to manage in class.

At Level 1, pupils are expected to identify when a program does not do what was intended to be able to correct errors/bugs to evaluate solutions/programs and to suggest improvements.

At Level 2, they should be able to identify any mismatches between the task description and the programmed solution, and indicate how to fix them.

In programming classes, pupils who are focusing on the task of writing a program for a particular goal might want help from you or others to fix their programs. Tempting as this may be, you and they should remember that the objective in class is not to get a working program, but to learn *how* to program; their ability to debug their own code is a big part of that.

## What strategies can you use to support debugging?

One way that you can help is to provide a reasonably robust, general set of debugging strategies which pupils can use for any programming, or indeed more general strategies which they can use when they encounter problems elsewhere.

Debugging should be underpinned by logical reasoning. The Barefoot Computing team suggest a simple sequence of four steps, emphasising the importance of logical reasoning:

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 where there's a problem 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 get started with fixing bugs, they'll need to establish whether it was an issue with 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 in (wait until [space] pressed) blocks 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 activity ideas

- Pupils are likely to make many authentic errors in their own code, which they'll want to fix. You might find that it's worth spending some time giving pupils some bugs to find and fix in other programs, both as a way to help develop their strategies for debugging and to help with assessment of logical reasoning and programming knowledge. Create some programs with deliberate mistakes in, perhaps using a range of logical or semantic errors, and set pupils the challenge of finding and fixing these.
- Encourage pupils to debug one another's code. One approach is for pupils to work on their own program for the first part of the lesson and then to take over their partner's project, completing this and then debugging it for their friend.
- A similar paired activity is for pupils to write code with deliberate mistakes, setting a challenge to their partner to find and then fix the errors in the code.



### Further resources

- Barefoot Computing on 'Debugging', available at: <http://barefootcas.org.uk/barefoot-primary-computing-resources/computational-thinking-approaches/debugging/> (free, but registration required).
- BBC Bitesize 'What is debugging?', available at: [www.bbc.co.uk/guides/ztkx6sg](http://www.bbc.co.uk/guides/ztkx6sg).
- Debugging challenges from Switched on Computing, available via: <http://scratch.mit.edu/studios/306100/>.
- Rubber duck debugging, available at: [http://en.wikipedia.org/wiki/Rubber\\_duck\\_debugging](http://en.wikipedia.org/wiki/Rubber_duck_debugging).

# 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 is 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.

## 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 have been built into robots, so that they can autonomously react to changes in their 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>.
- Barefoot on 'Computer systems', available at: <http://barefootcas.org.uk/barefoot-primary-computing-resources/concepts/computer-systems/> (free, but registration required).
- Barefoot on 'Inputs', available at: <http://barefootcas.org.uk/programme-of-study/work-various-forms-input/inputs/> (free, but registration required).
- Barefoot on 'Outputs', available at: <http://barefootcas.org.uk/programme-of-study/work-various-forms-output/outputs/> (free, but registration required).
- BBC Cracking the Code: Miniature computers, available at: [www.bbc.co.uk/programmes/p01661f7](http://www.bbc.co.uk/programmes/p01661f7).
- BBC Cracking the Code: Robots, available at: [www.bbc.co.uk/programmes/p01661tn](http://www.bbc.co.uk/programmes/p01661tn).

# Computer 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.

- At Level 1, pupils can demonstrate an understanding of the different functions of a browser and search engine, and can recognise what should and shouldn't be searched for on the internet.
- At Level 2, pupils can use search engines to search the internet for specific or relevant information, e.g. using quotation marks to narrow the results<sup>1</sup>.

### 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 connect 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!

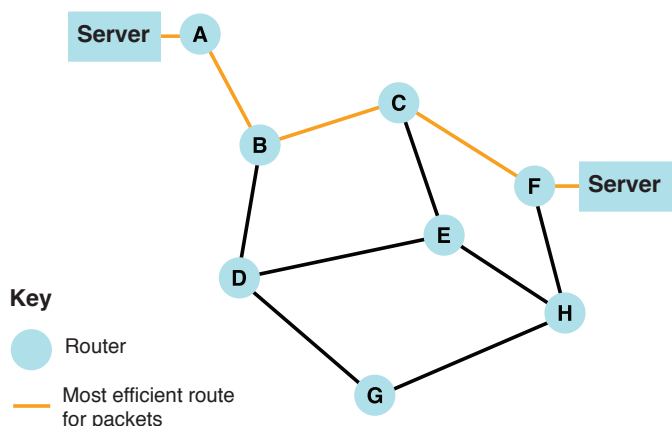
Digitised information needs to be broken down into small chunks by the computer before it can be sent efficiently. These smaller chunks of data are known as 'packets'.

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.

<sup>1</sup> Technologies Benchmarks [Education Scotland 2017]

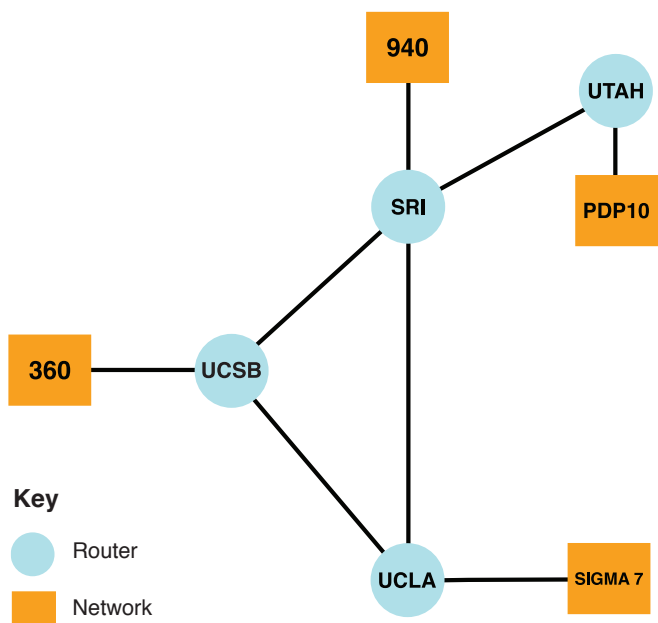


The packets don't all have to travel the same way through the internet: they can take any route from sender to recipient. However, there is generally a most efficient route, which all the packets would take.



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 network.

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:



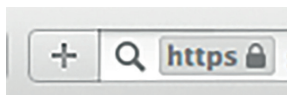
When you type a URL (such as [www.bbc.co.uk](http://www.bbc.co.uk) or [www.computingschool.org.uk](http://www.computingschool.org.uk)) into your browser, you send a packet of data requesting the content of these pages be returned to you. But before this can happen, the domain name first needs to be converted 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, via the internet secretly, 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.



### 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.

Table 1 (6 pupils)

|               |     |     |
|---------------|-----|-----|
| Router<br>1.0 | 1.1 | 1.2 |
| 1.3           | 1.4 | 1.5 |

Table 2 (6 pupils)

|               |     |     |
|---------------|-----|-----|
| Router<br>2.0 | 2.1 | 2.2 |
| 2.3           | 2.4 | 2.5 |

Message slips

To: 2.3  
Sequence: 1 of 3  
Data: What  
From: 1.5

To: 2.3  
Sequence: 2 of 3  
Data: is for  
From: 1.5

To: 2.3  
Sequence: 3 of 3  
Data: tea?  
From: 1.5

Role-playing a computer network in class.

- 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 below.
- Ask your school network manager to talk pupils through how the school network connects their computers to the rest of the internet.



### Further resources

- Bagge, P., 'Year 5 Computer Science Planning', available at: [www.code-it.co.uk/year5/index.htm](http://www.code-it.co.uk/year5/index.htm).
- Barefoot on 'KS2 Modelling the Internet activity', available at: <http://barefootcas.org.uk/programme-of-study/understand-computer-networks-including-internet/ks2-activity-modelling-the-internet/> (free, but registration required).
- Barefoot on 'Internet Services', available at: <http://barefootcas.org.uk/programme-of-study/multiple-services-provided-networks-internet/internet-services/> (free, but registration required).
- BBC Bitesize clip 'Computer networks – LAN and WAN', available at: [www.bbc.co.uk/learningzone/clips/computer-networks-lan-and-wan/4381.html](http://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](http://www.ted.com/talks/andrew_blum_what_is_the_Internet_really).
- Artistic representations of what the internet means, available at: [www.canyoudrawtheinternet.com](http://www.canyoudrawtheinternet.com).
- Mark Dorling and others’ Digital Schoolhouse planning on networks, ‘Networks Unplugged’, available at: [www.digitalschoolhouse.org.uk/documents/networks-unplugged-workshop-pack](http://www.digitalschoolhouse.org.uk/documents/networks-unplugged-workshop-pack).
- Naughton, J., *From Gutenberg to Zuckerberg: What You Really Need to Know About the Internet* (Quercus, 2012).
- Visual traceroute to find the path from your web server to an internet address, available at: [www.yougetsignal.com/tools/visual-traceroute/](http://www.yougetsignal.com/tools/visual-traceroute/).

## 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 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 (say, builders.com) 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 addresses 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 about how email works, and about issues with email security. Explain to pupils that email addresses can be 'spoofed' or accounts hacked. So, not all emails are from whom 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.

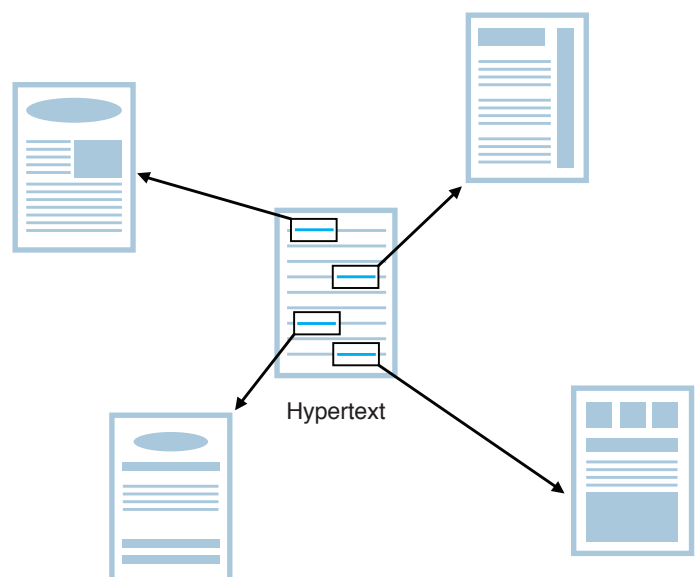


### Further resources

- 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/](http://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](http://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.

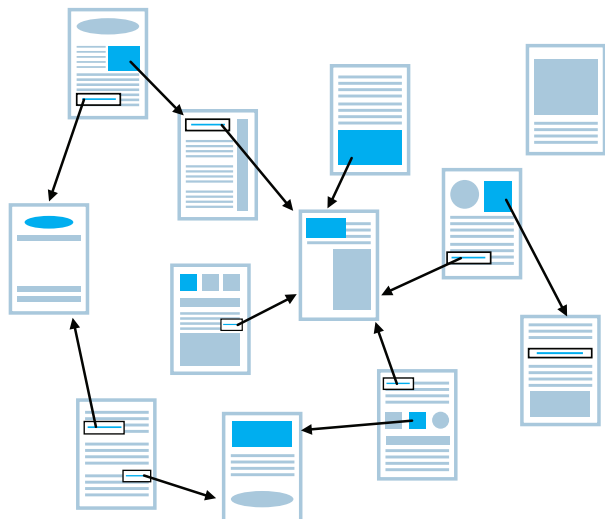


The links in the hypertext take the reader to different documents which extend or support the information in the original document.

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 its 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 sub-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 intended web server (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).

### 3. 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/](http://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 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 benchmarks for Social Studies at level 1 say that pupils should be able to 'Draw comparisons between modern life and life from a time in the past.' Ask pupils to investigate the effects of introduction of the printing press to Europe in the 15th century compared to the creation of the WWW in the 20th century.
- 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](http://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](http://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 which the Scottish curriculum at Level 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>
    
  </body>
</html>
```

## Origins of the Web

Tim Berners-Lee started working on the world-wide web project in 1989.

He was working at [CERN](#) in Switzerland at the time.



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 <p>) 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
- <p> to </p> around each paragraph.

Compare the underlined link in the web page with the corresponding code. In the code, <a> to </a> shows where the link should be, and href=“http://home.web.cern.ch/” inside the <a> tag details 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, to the code, source or HTML mode, and to 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/](http://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/](http://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) storing it 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.

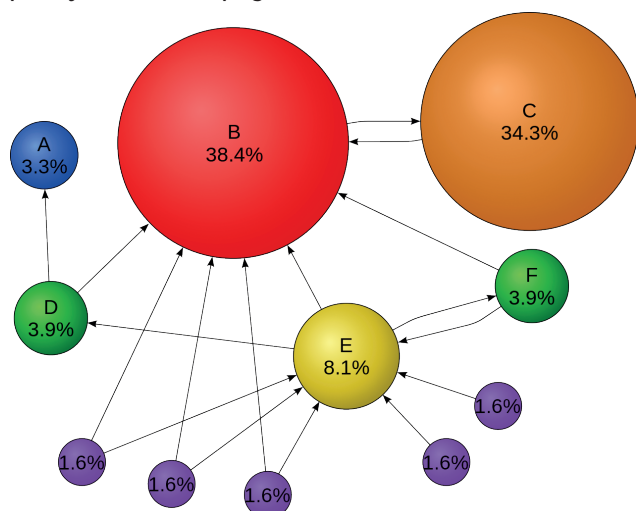
So, when we type a keyword such as 'dog' into a search engine, it consults the index and returns a list of all the web pages on which that keyword appears. Typing in several keywords, e.g. 'dog' and 'bowl', will only return pages with both of these keywords, which helps to narrow down the set of results.

## 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 is, 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 results 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 each 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 2012 CAS Conference, available at: [www.computingatschool.org.uk/index.php?id=aberdeen](http://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](http://www.youtube.com/watch?v=BNHR6IQJGZs).
- Peter Dickman's lecture 'How Google Search Works', available at: [www.youtube.com/watch?v=C8v7AM1o7uM](http://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](http://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.

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*, pages 46–49). 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, or for practising other languages,

or for 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 (as in David Mitchell's QuadBlogging® projects: see Further resources). 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 in 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 remix 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. From the Early Level, pupils should understand appropriate behaviour and language.

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](http://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](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](http://en.wikipedia.org/wiki/Wikipedia:Five_pillars).
- Wikispaces Classroom: creating wikis in school, available at: [www.wikispaces.com/content/classroom](http://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 Digital Literacy curriculum.

The 2017 Technologies benchmarks make a clear distinction between Digital Literacy and Computing Science. As can be seen from the curricular organisers for Digital Literacy:

*Using digital products and services in a variety of contexts to achieve a purposeful outcome; Searching, processing and managing information responsibly; Cyber resilience and internet safety<sup>1</sup>*

encompass the older concept of 'ICT across the curriculum'. The topics which encompassed the broad areas of productivity and creativity will remain an important part of the new curriculum.

### 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<sup>2</sup> 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' 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 Benchmarks are quite careful not to specify particular digital media. Technology currently

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’<sup>3</sup>: 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 the 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 has digitised, and to 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](http://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](http://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](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](http://www.youtube.com/watch?v=uEt8NuqBvPQ); see also Google Trends: [www.google.com/trends/](http://www.google.com/trends/).

<sup>3</sup> Robinson, K., *Out of Our Minds – Learning to Be Creative* (Capstone, 2011).

# Safe and responsible use

## How can we keep children safe online?

Schools have a responsibility to keep pupils safe. The Byron Review,<sup>1</sup> 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 pupils at Level 1 should be able to:

*demonstrate understanding of their rights and responsibilities as a digital citizen, of the potential dangers online, whom to go to for advice and whom to report a concern to.*<sup>2</sup>

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  |
|--|--|------------------------------------|---|---|
| <b>Content</b><br>(child as recipient)   | Adverts;<br>Spam;<br>Sponsorship;<br>Personal info                             | Violent/hateful content            | Pornographic or unwelcome sexual content      | Bias;<br>Racism;<br>Misleading info or advice |
| <b>Contact</b><br>(child as participant) | Tracking;<br>Harvesting personal info  | Being bullied, harassed or stalked | Meeting strangers;<br>Being groomed           | Self-harm;<br>Unwelcome persuasions           |
| <b>Conduct</b><br>(child as actor)       | Illegal downloading;<br>Hacking;<br>Gambling;<br>Financial scams;<br>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](http://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/](http://www.nationalarchives.gov.uk/doc/open-government-licence/version/2/).

<sup>1</sup> Byron, T., *Safer Children in a Digital World: The Report of the Byron Review* (London: DCSF, 2008).

<sup>2</sup> *Technologies Benchmarks* [Education Scotland 2017].

## 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, 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/](http://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 on 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 pupils at Level 2, 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 Level 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 Benchmarks at Level 1 require that pupils develop an understanding of both their rights and their responsibilities as digital citizens. Supporting children’s moral development is a vital part of primary education.

Lawrence Kohlberg's stages of moral development<sup>3</sup> offer 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 when using technology can perhaps be avoided, or their consequences reduced.

### Cyberbullying

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 recognising 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 searches 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 a 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 or with their parents; As this is usually their only way to prove who they are in any online system, 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.

<sup>3</sup> 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 or socialising face-to-face with family and friends. Whilst digital technology is seen by many as transformative in 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 might 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/](http://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](http://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](http://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/](http://www.thinkuknow.co.uk/Teachers/).
- UK Safer Internet Centre, for information and teaching resources, available at: [www.saferinternet.org.uk](http://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](http://www.ohchr.org/en/professionalinterest/pages/crc.aspx).

# Planning learning, teaching and assessment

## The Curriculum Framework in Scotland<sup>1</sup> has at its core the following

The purpose of Curriculum for Excellence is to help children and young people to become:

- successful learners
- confident individuals
- responsible citizens
- effective contributors.

### The guidance states

Teachers should be empowered to use the flexibility that CfE provides, to organise learning for children and young people in ways that best meet learners' needs. Schools should be working in a collegiate way to make key decisions.

## PLANNING LEARNING, TEACHING AND ASSESSMENT USING THE BENCHMARKS

The purpose of the Benchmarks is to set out very clear statements about what children and young people need to learn to achieve each level of the curriculum. The Benchmarks streamline and embed a wide range of existing assessment guidance (significant aspects of learning, progression frameworks and annotated exemplification) into one key resource to support teachers' professional judgement.

---

<sup>1</sup>Education Scotland Curriculum For Excellence  
HMIE 2016

## WHAT TO DO: KEY MESSAGES

- ✓ Periodically (from time to time) use assessments to sample and pull together learning in a joined-up way.
- ✓ Plan an appropriate balance between ongoing and periodic assessment – this will vary from stage to stage.
- ✓ Periodically use assessments to sample and pull together learning in a joined-up way.
- ✓ Plan an appropriate balance between ongoing and periodic assessment – this will vary from stage to stage.
- ✓ Moderate assessment judgements by taking account of a sample of evidence from different sources, to discuss standards and the progress of learners.
- ✓ As a school, develop simple and effective approaches to monitoring and tracking learners' progress. Tracking needs to be as easy to use as possible.
- ✓ Regularly discuss tracking information with colleagues, to plan additional support and interventions to help improve learners' progress.
- ✓ Evaluate learners' progress on an ongoing basis and keep short concise notes to help planning for next steps in learning. This will include identifying where additional support and challenge may be needed.
- ✓ Use the benchmarks to help monitor progress and support overall professional judgement of when a learner has achieved a curriculum level.

## WHAT TO AVOID: KEY MESSAGES

- ✗ Avoid spending time on assessment activities which do not help to identify children's and young people's next steps in learning.
- ✗ Do not over-assess learners or assess the same content repeatedly in different ways. Do not create large portfolios of evidence.
- ✗ Avoid duplication and keeping evidence of every detail within the Benchmark.
- ✗ Avoid waiting until learners have demonstrated evidence of every aspect of learning within the Benchmarks before moving on to the next level.
- ✗ Avoid spending too much time collecting a wide range of evidence for moderation purposes.
- ✗ Do not track and record progress against individual Es and Os.
- ✗ Do not track progress and achievement using the terms 'developing, consolidating, secure'.
- ✗ Do not spend time writing long reports for parents which describe lots of classwork or use professional jargon.

At this time, remember the guidance:<sup>2</sup>

*There is no need to provide curriculum level judgements in all curriculum areas – stick to literacy and numeracy.*

---

<sup>2</sup>Guidance on using Benchmarks for Assessment  
Education Scotland March 2017

## EFFECTIVE PRACTICE

The overarching statements from How Good is our School<sup>3</sup> for schools displaying effective practice, are as follows:

### **Digital innovation**

Children and young people work individually and in teams creating both digital and non-digital solutions. As their digital literacy becomes more sophisticated, they embed computation to solve problems. Increasingly they apply the core principles underpinning digital technologies to develop their own ideas. Their skills are up-to-date with technological advances informed by a range of sources including the expertise of the young people themselves.

### **Digital literacy**

Children and young people are innovative, confident and responsible in the use of technologies and staying safe online. They critically examine and make informed choices about the use of digital technology to enhance and personalise learning in school and, where appropriate, beyond the school day. They anticipate and respond to new opportunities and threats caused by developments now and in the future.

## Examples of good practice

Many examples of good practice in the delivery of Digital Learning and Computing Science exist in schools at present. Some examples identified by HMIE<sup>4</sup> are given overleaf.

---

<sup>3</sup>How Good is our School  
Education Scotland 2015

<sup>4</sup>Building Society  
Young people's experiences and outcomes in the technologies  
Education Scotland 2014

*‘Children program the wolf to follow a route to the three pigs’ houses, motivated well to stick to the task, and enjoy reading the story as it unfolds. A related activity involves the children in problem solving, finding their way through a maze displayed on an interactive whiteboard. The children are skilled in using colours to record blind alleys, and show determination in completing the task.’*

*‘The children clearly benefit from the structured programme in ICT, and are able to describe out-of-school activities such as popular computer games where they are able to recognise and apply the skills they have learned. The children enjoy creative opportunities in coding, preparing short animations, and have a high level of awareness of career opportunities relating to the digital technologies. The principal teacher has built considerable expertise in using Glow, an enthusiasm which is reflected in the children’s motivation. All classes post Glow blogs – ‘Being creative is a good skill because you can even use it when you are writing!’*

*‘Children use cameras very well to make their own photograph album. They independently use computers to play games, watch DVDs and listen to music.’*

*‘The school’s programme in technologies has a substantial impact on children’s overall learning experience. Staff have prepared a framework of technologies experiences, including a specific programme to support progression in ICT. These experiences provide an important structure, whilst retaining the capacity to respond to current technologies themes such as those featuring in the media. At P1, children extend their skills in programming toys, often in the context of problem-solving challenges. Children in P1 are confident in using tablet computers for activities to enhance their learning in numeracy.’*

*‘In P5, children learn vital aspects of personal safety relating to use of the internet, in the context of well-structured development of search engine skills. Provision at P7 includes some very strong aspects, including a programme specifically in computer science. Children follow a modular programme over eight weeks, developing their skills in coding.’*



## Further resources

- Barefoot Computing teaching resources, available at: <http://barefootcas.org.uk/> (free, but registration required).
- Computing At School community website resources, available at: <http://community.computingatschool.org.uk/resources>.
- Computing At School (CAS) Include: ComputingScience for All, available at: <http://casinclude.org.uk/>.
- CS Unplugged, free activities and resources, available at: <http://csunplugged.org/>.
- Code Club network of after-school coding clubs, available at: [www.codeclub.org.uk/](http://www.codeclub.org.uk/).
- CoderDojo network of free computing programming clubs, available at: <https://coderdojo.com/>.
- 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/](http://www.yrs.io/).

# Resources

## Websites

Computing At School (CAS) hosts a large resource bank of plans, resources and activities. CAS is free to join: <http://community.computingschool.org.uk/door>.

The BCS Barefoot Computing project is developing concept guides and exemplar activities. It's free, but registration is required: <http://barefootcas.org.uk/>.

Naace (the ICT association) and CAS have developed joint guidance on the new computing curriculum: [www.computingschool.org.uk/index.php?id=primary-national-curriculum-guidance](http://www.computingschool.org.uk/index.php?id=primary-national-curriculum-guidance).

New Zealand-based CS Unplugged produces an excellent collection of resources exploring computer science ideas through classroom- rather than computer-based activities: <http://csunplugged.org>.

Former CAS CPD Coordinator, Mark Dorling, has made available a large collection of lesson plans and other resources through the Digital Schoolhouse project for London schools: [www.digitalschoolhouse.org.uk/](http://www.digitalschoolhouse.org.uk/).

CAS Primary Master Teacher, Phil Bagge, has shared detailed lesson plans for many computer science and digital literacy topics: <http://code-it.co.uk/philbagge.html>.

A group of teachers and teacher trainers convened by the NCTL, and chaired by Toshiba's Bob Harrison, has worked together to curate resources for initial teacher training for the computing curriculum: [bit.ly/ittcomp](http://bit.ly/ittcomp).

There is a large collection of resources for teaching all aspects of computing on the TES website, for Levels 1 and 2. There's also a discussion forum online. Free, but registration is required: [https://community.tes.co.uk/search/?SB=postcount\\_i%20desc&tc=1%2Fsubject%2FInformationTechnology](https://community.tes.co.uk/search/?SB=postcount_i%20desc&tc=1%2Fsubject%2FInformationTechnology).

Code Club provides detailed plans and resources for extra-curricular clubs, which might be adapted for use within the school curriculum. Free, but registration required: [www.codeclub.org.uk/](http://www.codeclub.org.uk/).

Code Club Pro provides training for teachers on the computing curriculum: [www.codeclubpro.org/](http://www.codeclubpro.org/).

In the US, code.org make available a range of high-quality curriculum materials and activities linked to programming and computational thinking: <http://code.org/>.

The BBC has an extensive set of resources for pupils, linked to the new computing curriculum: [www.bbc.co.uk/schools/0/computing/](http://www.bbc.co.uk/schools/0/computing/).

BBC Two's Cracking the Code: [www.bbc.co.uk/programmes/b01r9tww/clips](http://www.bbc.co.uk/programmes/b01r9tww/clips).

The Raspberry Pi foundation has a good collection of high-quality resources which are relevant to other platforms as well as the Pi: [www.raspberrypi.org/](http://www.raspberrypi.org/).

Resources for teaching the safe, respectful and responsible use of technology are widely available. Childnet International and CEOP's Thinkuknow are both good starting points for exploring these topics: [www.childnet.com/](http://www.childnet.com/) and [www.thinkuknow.co.uk/](http://www.thinkuknow.co.uk/).

SWGfL provides free access to digital-literacy materials: [www.digital-literacy.org.uk/Home.aspx](http://www.digital-literacy.org.uk/Home.aspx).

Futurelab is an independent charitable organisation commissioning research into, and providing funding for, cutting-edge applications of technology in education. Their archive is also a useful resource: [www.futurelab.org.uk/](http://www.futurelab.org.uk/).

## Publications

Armoni, M. and Ben-ari, M., *Computer Science Concepts in Scratch* (Rehovot: Weizmann Institute of Science, 2013).

Barr, V. and Stephenson, C., 'Bringing computational thinking to K-12', *ACM Inroads* 2:1 (2011).

Berry, M., *Computing in the Primary Curriculum: A Guide for Primary Teachers* (Cambridge: Computing At School, 2013).

Bird, J., Caldwell, H. and Mayne, P., *Lessons in Teaching Computing in Primary Schools* (Exeter: Learning Matters, 2014).

Brennan, K. and Resnick, M., 'New frameworks for studying and assessing the development of computational thinking', AERA 2012 conference paper (2012).

Byron, T., *Do We Have Safer Children in a Digital World? A Review of Progress Since the 2008 Byron Review* (Nottingham: DfE, 2010).

Department for Education, *National Curriculum in England, Key Stages 1 and 2 Framework Document* (London: DfE, 2013).

Hammersley, B., *Now For Then: How to Face the Digital Future Without Fear* (London: Hodder, 2012).

Hey, T., *The Computational Universe, A Journey Through A Revolution* (Cambridge: CUP, 2014).

Mozilla, *Why Mozilla Cares About Web Literacy* (2014).

Naughton, J., *From Gutenberg to Zuckerberg: What You Really Need to Know About the Internet* (Quercus, 2011).

Ofsted, *The Safe Use of New Technologies* (London: Ofsted, 2010).

Ofsted, *ICT in schools 2008–11* (London: Ofsted, 2011).

Papert, S., *Mindstorms: Children, Computers, and Powerful Ideas, 2nd ed.* (New York, NY: Basic Books, 1993).

Petzold, C., *Code: The Hidden Language of Computer Hardware and Software* (Microsoft Press, 2009).

Wing, J M., 'Computational thinking and thinking about computing', *Philosophical transactions of the Royal Society A*, 366, 3717–3725 (2008).

# Knowledge and skills audit form

Use this form to audit your computing knowledge and skills.

The statements are taken from Level 2 benchmarks organised by the relevant Experiences and Outcomes.

|  | I am very confident I can do this | I am confident I can do this | I am not confident I can do this | Page reference |
|--|-----------------------------------|------------------------------|----------------------------------|----------------|
| <b>Digital Literacy</b>  |                                   |                              |                                  |                |
| <b>Using digital products and services in a variety of contexts to achieve a purposeful outcome</b><br>TCH 2-01a |                                   |                              |                                  |                |
| I can save in a range of standard file formats   |                                   |                              |                                  |                |
| I can save files using an organized filing system  |                                   |                              |                                  |                |
| I can use an online 'cloud'- based service   |                                   |                              |                                  | 42 - 43        |
| I can identify the key features of input, output and storage devices   |                                   |                              |                                  | 31             |
| I can select and use applications and software to capture, create and modify text, images, sound and video       |                                   |                              |                                  | 44 - 45        |
| I can select the most appropriate digital software to perform a task   |                                   |                              |                                  | 44 - 45        |
| <b>Searching, processing and managing information responsibly</b><br>TCH 2-02a                                   |                                   |                              |                                  |                |
| I can use search engines to search the internet for specific or relevant information                             |                                   |                              |                                  | 40 - 41        |
| I can access websites and use navigation skills to retrieve information for a specific task                      |                                   |                              |                                  | 36 - 38        |
| I can demonstrate an understanding of usage rights and can apply these within a search                           |                                   |                              |                                  | 48             |
| <b>Cyber resilience and internet safety</b><br>TCH 2-03a   |                                   |                              |                                  |                |
| I can demonstrate an understanding of the content I should include in an online profile.                         |                                   |                              |                                  | 46 - 48        |
| I can discuss the importance of being a responsible digital citizen  |                                   |                              |                                  | 46 - 48        |
| I can identify appropriate ways to report concerns   |                                   |                              |                                  | 46 - 48        |
| I can use strong passwords   |                                   |                              |                                  | 48             |
| I can understand the law as it relates to inappropriate or illegal online behaviours                             |                                   |                              |                                  | 46 - 48        |

|   | I am very confident I can do this | I am confident I can do this | I am not confident I can do this | Page reference |
|---|-----------------------------------|------------------------------|----------------------------------|----------------|
| <b>Computing Science</b>  |                                   |                              |                                  |                |
| <b>Understanding the world through computational thinking</b><br>TCH 2-13a  |                                   |                              |                                  |                |
| I can compare activities consisting of a single sequence of steps with those consisting of multiple parallel steps                              |                                   |                              |                                  |                |
| I can identify algorithms that include repeated groups of instructions a fixed number of times until a condition is met                         |                                   |                              |                                  | 10 - 12        |
| I can identify when a process is not predictable because it has a random element  |                                   |                              |                                  |                |
| I can use a recognised set of instructions (an algorithm) to sort real-world objects  |                                   |                              |                                  | 10 - 12        |
| <b>I can explain core programming language concepts in appropriate technical language</b><br>TCH 2-14a  |                                   |                              |                                  |                |
| I can explain the meaning of individual instructions (including variables and conditional repetition) in a visual programming language          |                                   |                              |                                  | 24 - 28        |
| I can predict what a complete program in a visual programming language will do when it runs,  |                                   |                              |                                  | 24 - 28        |
| I can explain and predict how parallel activities interact  |                                   |                              |                                  |                |
| <b>I understand how information is stored and how key components of computing technology connect and interact through networks</b><br>TCH 2-14b |                                   |                              |                                  |                |
| I can demonstrate an understanding that all computer data is represented in binary, for example, numbers, text, black-and-white graphics        |                                   |                              |                                  | 30 - 31        |
| I can describe the purpose of a computer's processor, memory and storage and the relationship between them                                      |                                   |                              |                                  | 30 - 31        |
| I can demonstrate an understanding of how networks are connected and used to communicate and share information, for example the internet        |                                   |                              |                                  | 32 - 41        |
| <b>I can create, develop and evaluate computing solutions in response to a design challenge</b><br>TCH 2-15a                                    |                                   |                              |                                  |                |
| I can create programs in a visual programming language, including variables and conditional repetition  |                                   |                              |                                  | 24-28          |
| I can identify patterns in problem-solving and reuse aspects of previous solutions appropriately  |                                   |                              |                                  | 15 - 16        |
| I can identify any mismatches between the task description and the programmed solution, and indicate how to fix them                            |                                   |                              |                                  | 17             |

# Glossary

**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').

**Creative Commons:** A licensing scheme where the creator of an original work allows others to use it without having to seek further permission, subject to a number of agreed conditions: [www.creativecommons.org](http://www.creativecommons.org).

**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.

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

**Domain Name Service (DNS):** The distributed automatic system that converts domain names into the IP addresses which are used for routing packets of data via the internet.

**Encrypt:** To securely encode information so that it can only be read by those knowing both the system used and a secret, private key.

**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.

**Hypertext Mark-up Language (HTML):** HTML is the computer language in which web pages are composed.

**Hypertext Transfer Protocol (HTTP):** HTTP is the standard protocol for the request and transmission of HTML web pages between browser and web server.

**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.

**Internet Protocol (IP) addresses:** Numeric addresses uniquely specifying computers directly connected to the internet, also used on private networks to uniquely identify computers on that network.

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

**Network server:** A computer connected to a local area network (LAN) providing services – such as file storage, printing, authentication, web access or email – automatically to other computers on the network.

**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 set of programs on a computer which deal with the 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.

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

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

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

**Repetition:** The execution of a section of computer code over a number of times, as part of a program.

**Router:** Network hardware which forwards packets of data onwards to the most appropriate hardware to which it is connected.

**Screencast:** A recording of on-screen action that is often accompanied by an audio narration.

**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 aspect of most computer games.

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

**Uniform Resource Locator (URL):** A standard for specifying the location on the internet of certain files.

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

**Web server:** A service running on a computer (or sometimes for the computer itself) that returns HTML data for a web page when it receives an HTTP request via the local network or the internet.

**World Wide Web:** A service provided by computers connected to the internet (web servers), in which pages of hypertext (web pages) are transmitted to users.



# QuickStart Computing Scotland

## Subject Knowledge for Primary teachers

Inside and online you will find all the support you and your colleagues need to plan, teach and assess your computing curriculum with confidence and success. This toolkit includes:

- Computing subject knowledge information and videos
- Guidelines and PowerPoint presentations to help you run tailored CPD sessions in your school or cluster
- Support for planning a scheme of work
- Tried and tested ideas for successful computing lessons
- Assessment and progression support

All this guidance is also available online at:  
<http://computingatschool.org.uk/scotland/quickstart>

