Computational Thinking is all about solving problems effectively – with or without a computer

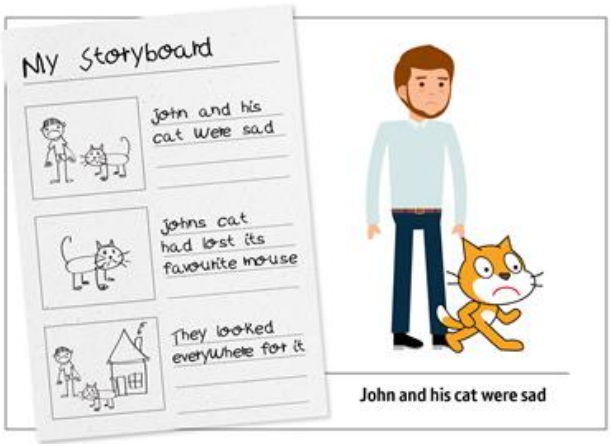
Computational thinking

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

For a computer animation, for example, you'll first plan the story and how it will be shot. Then, you'll use the computer hardware and software to create the animation.

Computational thinking is not thinking about computers or like computers: computers don't think for themselves – not yet, at least!



When creating an animation of a story, you first think about the sequence of events.

Concepts and Approaches

Computational thinking involves six different concepts and five approaches to working.

Click on the icons below to find out more about computational thinking concepts.

Logic Algorithms Decomposition Patterns Abstraction Evaluation

Click on the icons below to find out more about computational thinking approaches.

Tinkering Creating Debugging Persevering Collaborating

Download resource

Why is computational thinking important?
Computational thinking is the building blocks of our digital world, with the concepts forming the basis of much computer science. Computer scientists are interested in finding the most-efficient ways to solve problems, maximising accuracy and minimising resources (e.g. time / space). They look for solutions which can be applied elsewhere to save resources in the future.

Plenty of other people benefit from computational thinking, and not just when using computers themselves. A team of software engineers creating a new game is not really so different from teachers working together on a school play: in each case, it's necessary to identify the key steps or rules for getting a complex task done, thereby breaking it down (decomposing it) into smaller, more-manageable parts. It can also be helpful to consider the ways in which previous projects were successfully accomplished.

What does computational thinking look like in the Primary curriculum?
England's national curriculum for computing states that:
"a high-quality computing education equips pupils to use computational thinking and creativity to understand and change the world."

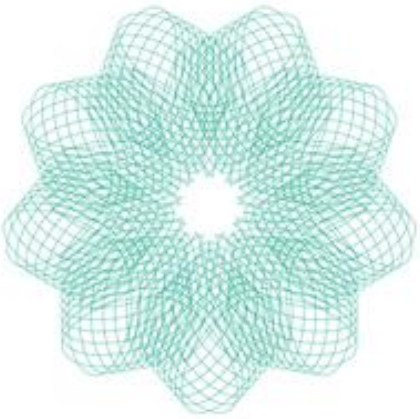
In addition, there are many ways to develop computational thinking outside of the computing curriculum. In maths, pupils must identify the key information in a problem before they can go on to solve it. Before they write a story, encourage pupils to first plan the main events, settings and characters. In art, music or D&T, ask them to think about what they're going to create and how they can break the creative process down into discrete, planned phases.

Pupils consider the best way to build a toy robot.

3-5 years
There are many opportunities to introduce the building blocks of computational thinking. For example, pupils can think about the steps involved in getting dressed for winter, decomposing the overall task into constituent tasks and then sequencing the instructions (writing an algorithm).

5-7 years
Pupils can decompose simple problems and create simple sequences of instructions (algorithms) – perhaps explaining the steps to grow a plant from seed. They may be able to label the parts of a flower diagram (also decomposition), and to check with a partner if their work is correct (collaborating, debugging and evaluation).

7-11 years
Pupils demonstrate increasing levels of computational thinking as their cognitive ability develops. They can decompose to a more detailed level, design more-complex algorithms, spot patterns and more easily use abstraction, for example in creating crystal flowers and simulations. Computational-thinking approaches become more familiar as pupils persevere in debugging problems and collaborate with each other.



Fun projects such as creating crystal flowers or a Viking animation can help to develop computational thinking.