COMPUTER SCIENCE

A Parent's Guide

BEST CODING + STEM TOY
Tech Advisor
2019

BEST CODING TOY
London Evening Standard
2018

MADE BY BRIGHT LITTLE LABS
bright little labs
MADE BY BRIGHT LITTLE LABS

TOP CODING TOY
BEST BUY
INDY/BEST
INDEPENDENT
2017

BEST STEM BOOK
THE Sun
2019

> “Computer science is no more about computers than astronomy is about telescopes.”

Dear Reader

Comput-ARGHHHHHHH!

Don’t know your algorithms from your elbow? Think the cloud is floating in the sky? Hate printers?* Or do you love all things tech and want to share the joy? Either way, we’re here to help.

Who are Bright Little Labs?

We’re a group of techies and storytellers.

We make interactive stories to promote critical thinking, computer science, toilet humour and equality for ALL kids. We're active in over 30 countries, have Cabinet Office backing, and have won awards for our story-led approach to computer science. Oh, and our mums support us too.

Our founder, Sophie Deen, is a former lawyer, techie and school counsellor. Sophie previously worked at Code Club, alongside Google and the Department For Education, to help introduce the new coding curriculum in primary schools. She was named Campaign Female Frontier Honouree in 2020, and one of Computer Weekly’s ‘Most influential Women in UK IT’ in 2019, 2018 and 2017.

Together, we’re on a mission to increase diversity in the STEM pipeline (that’s Science, Technology, Engineering and Maths. NOT the stalk of a flower). We’re living in a digital age, and everything in the digital age is built on code, so having digital skills is essential to understand the world around us. Businesses think so too. Coding is the #1 sought after skill in employees. We’re proud to help kids understand how the internet works, how data is shared, and how to use technology in a safe, responsible and positive way.

Peace, love + code

Team Bright Little Labs ⚡

** we do too.*

Computer Science 101: the basics

This guide explains computing concepts. Then it links them to the national curriculum. By reading this, you can stay up to date with the kids. You'll be able to explain how to enjoy wholesome memes and not be a milkshake duck.

What is a computer?

It's a thing that can (1) input data (2) process the data using a stored set of instructions and (3) output information. Many household items, even ones that don't have a screen, contain computers; fridges, ovens and washing machines.

From the curriculum:

"Pupils should be taught to: use sequence and repetition in programs; work with variables and various forms of input and output."

Examples:

A MICROWAVE

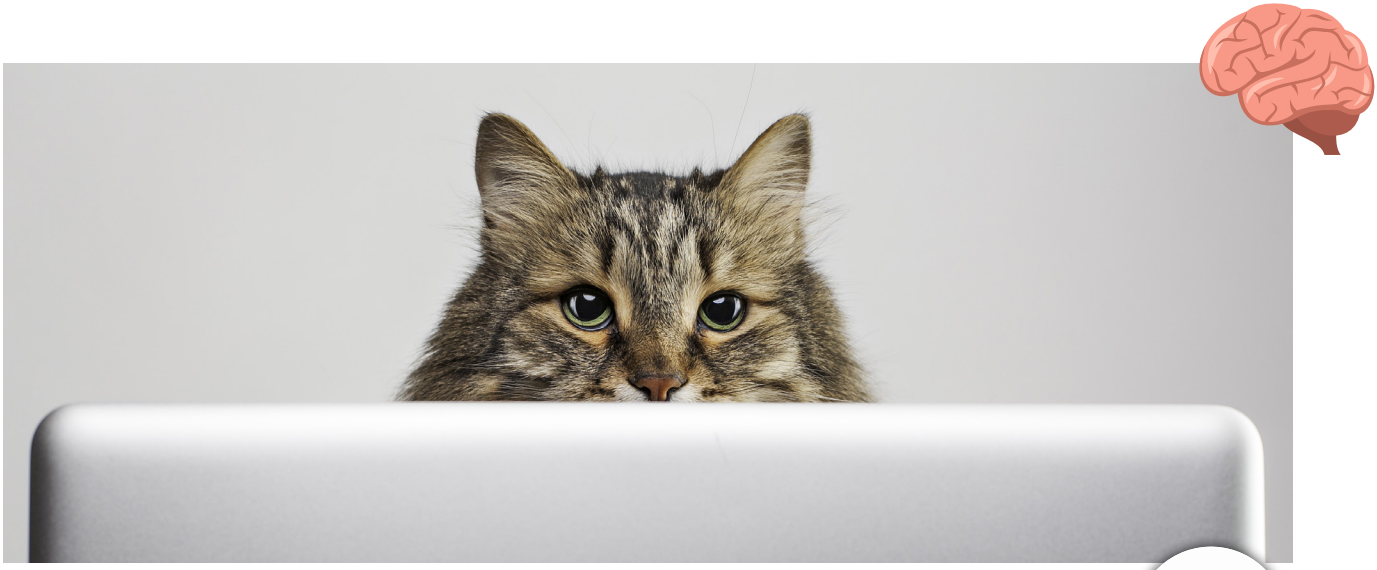
- **Input** - Control buttons, sensor on the oven door.
- **Program** - Embedded as part of the microcontroller.
- **Output** - Microwave generator, display, audio beeps, turntable motor.

AN IPHONE

- **Input** - LOTS. Touch screen, cameras, GPS, microphone, accelerometer, proximity sensors etc.
- **Program** - Built in or downloaded from the app store - the iOS operating system.
- **Output** - Screen, speakers, vibration motor.

Can a computer think?

No. Computers don't think. They just follow instructions.



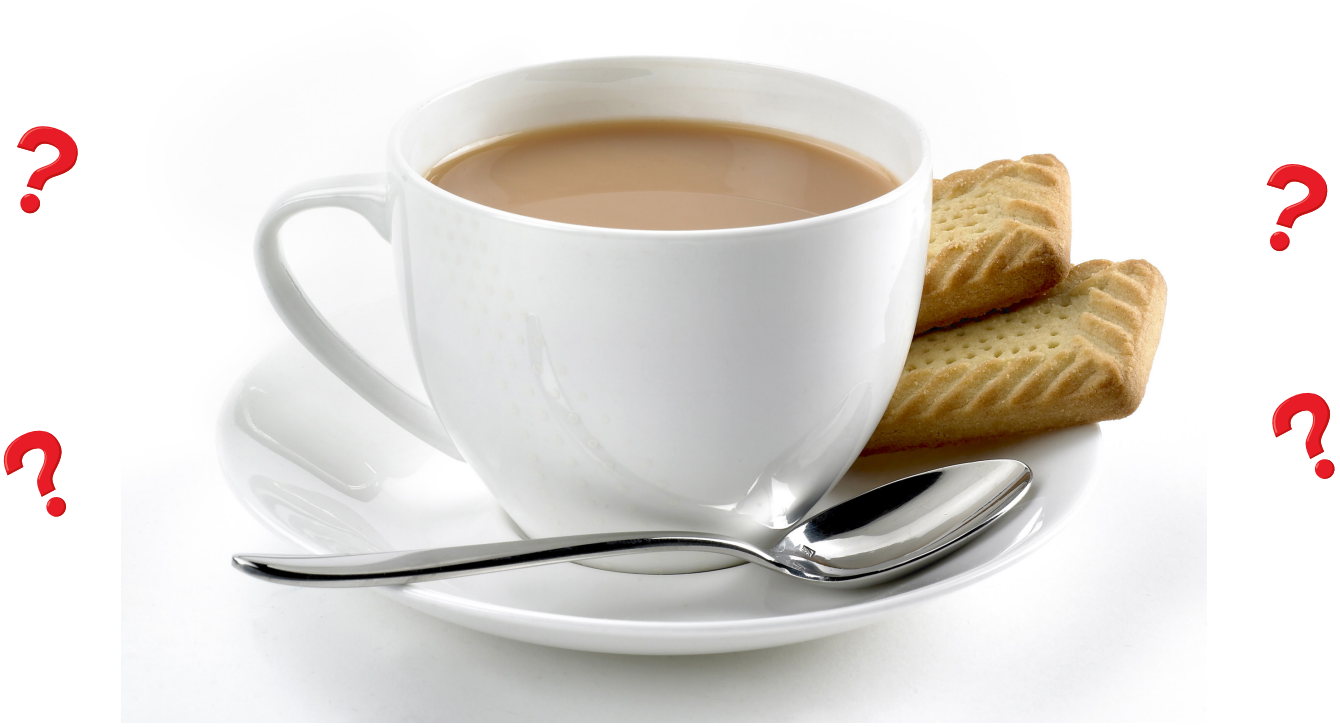
What is code? What is an algorithm? HELP ME.

Code is the language that computers speak in. They don't speak English (annoyingly) and can only understand code.

This means that when we want a computer to do something, we need to translate what we're saying into code. Each line of code is an instruction, and lots of lines of code = an algorithm: a very precise sequence of instructions, or a set of rules, for performing a specific task.

A computer does stuff in the order you tell it to. It executes each line of code, then the next, and so on. Like following a recipe.

Tea breaks are essential.
That's when a lot of thinking happens.



Is there an easy way to translate human language into computer code?

Yes.

First, we figure out the instructions in human language (also called pseudocode). Here are the steps you'd take to create the perfect cup of tea.

1. Collect teapot, tea bag and mug
2. Fill kettle with water
3. Plug in kettle
4. Place tea bag in teapot
5. Boil kettle
6. Add boiling water to teapot
7. Wait until tea is brewed
8. Pour tea into mugs
9. Add milk/sugar*
10. Serve

**This depends on what sort of person you are. We've heard some people add milk before water. Weird.*

Now, we can rewrite those instructions as something a bit like computer code.

```
1. collect: kettle,teapot,teabag,mug;
2. take: kettle, input: water (250 ml);
3. take: kettle, input: plug;
4. take teapot, input: teabag;
5. boil_kettle ()
6. take: teapot, input kettle_water;
7. wait (180 seconds)
8. take cup; input tea;
9. take: cup, input sugar, milk;
10. serve ()
```

Choices  

What about the fact that some people want milk and sugar, and others don't want either? If we want a computer to be able to make a choice, and do different things depending on that choice, we can write our instructions like this:

```
1.    if (sugar wanted)
2.          then add sugar
3.    else
4.          don't add sugar
```

Generally, the code for choices looks like this:

```
1.    if (condition)
2.          then statements
3.    else
4.          #else statements
```

Loops

Loops are not just a silly word. When we add the boiling water to the teapot, we only want to do that until it's full. Not forever, or there would be a huge mess. We can do this by introducing a 'loop' – an instruction to repeat something WHILE something else is happening or UNTIL something else happens. We'd write it like this:

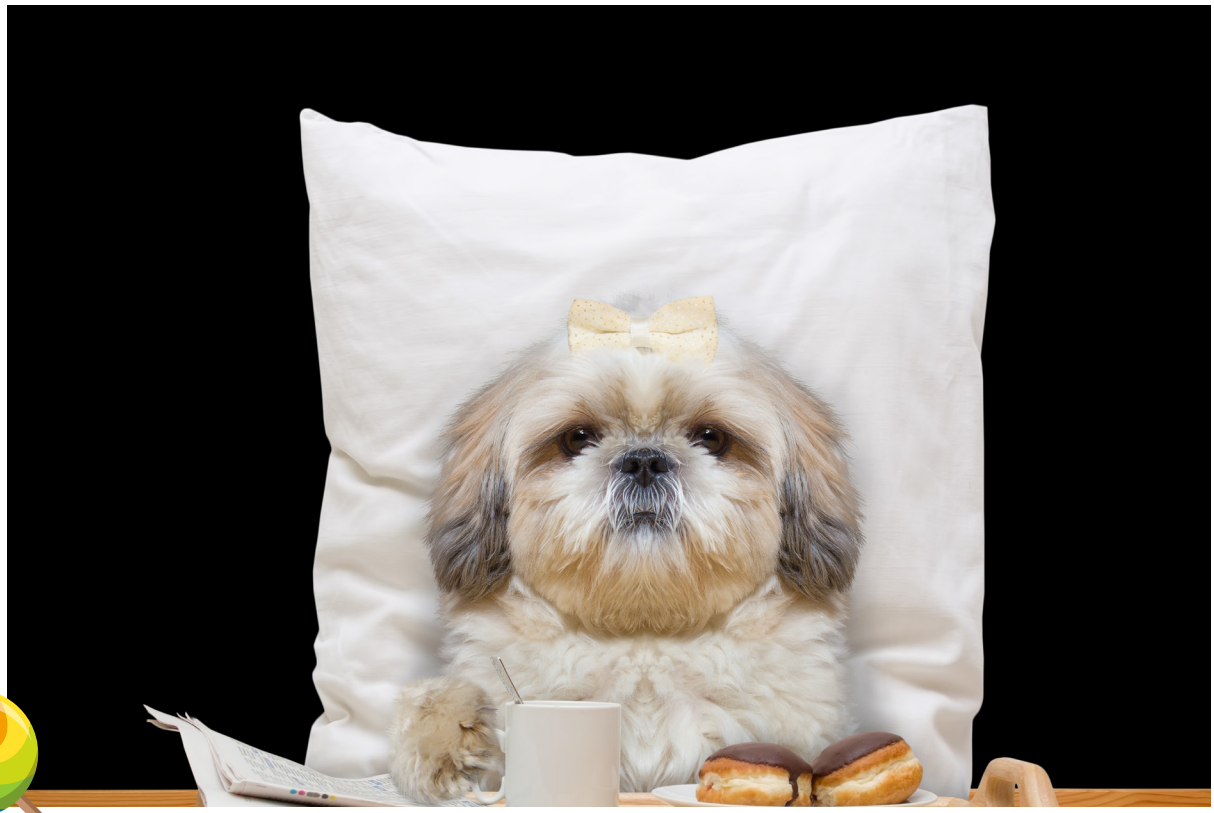
```
1.    while (kettle is not full)
2.          add water to the kettle
```

Generally, the code for loops looks like this:

```
1.    while (condition)
2.          do something
```

Alternatively, we might want a loop to only last for a particular amount of time (clock time), or for a certain number of times (repetitions).

```
1. for a thing / time / repetition (e.g. a mug in a set of mugs / 30
seconds / 10 times)
2.    do something
```



Here, we want the computer to pour the tea into each mug separately:

```
1. for mug in a set_of_mugs
2.    pour tea into each mug
```

Finally, using choices and loops, we have a better algorithm for making tea:

```
collect: kettle,teapot,teabag,mug;
while kettle_full = false:
   take: kettle, input: water (250 ml);
take: kettle, input: plug;4. take teapot, input: teabag;5. boil_kettle
()
take: teapot, input kettle_water;
wait(180)
while empty_cup = true
   take empty_cup; input tea;
   if (sugar_wanted):
      take: cup, input: sugar (5 g);
 if (milk_wanted):
    take: cup; input: milk (5 ml);
  serve ()
```

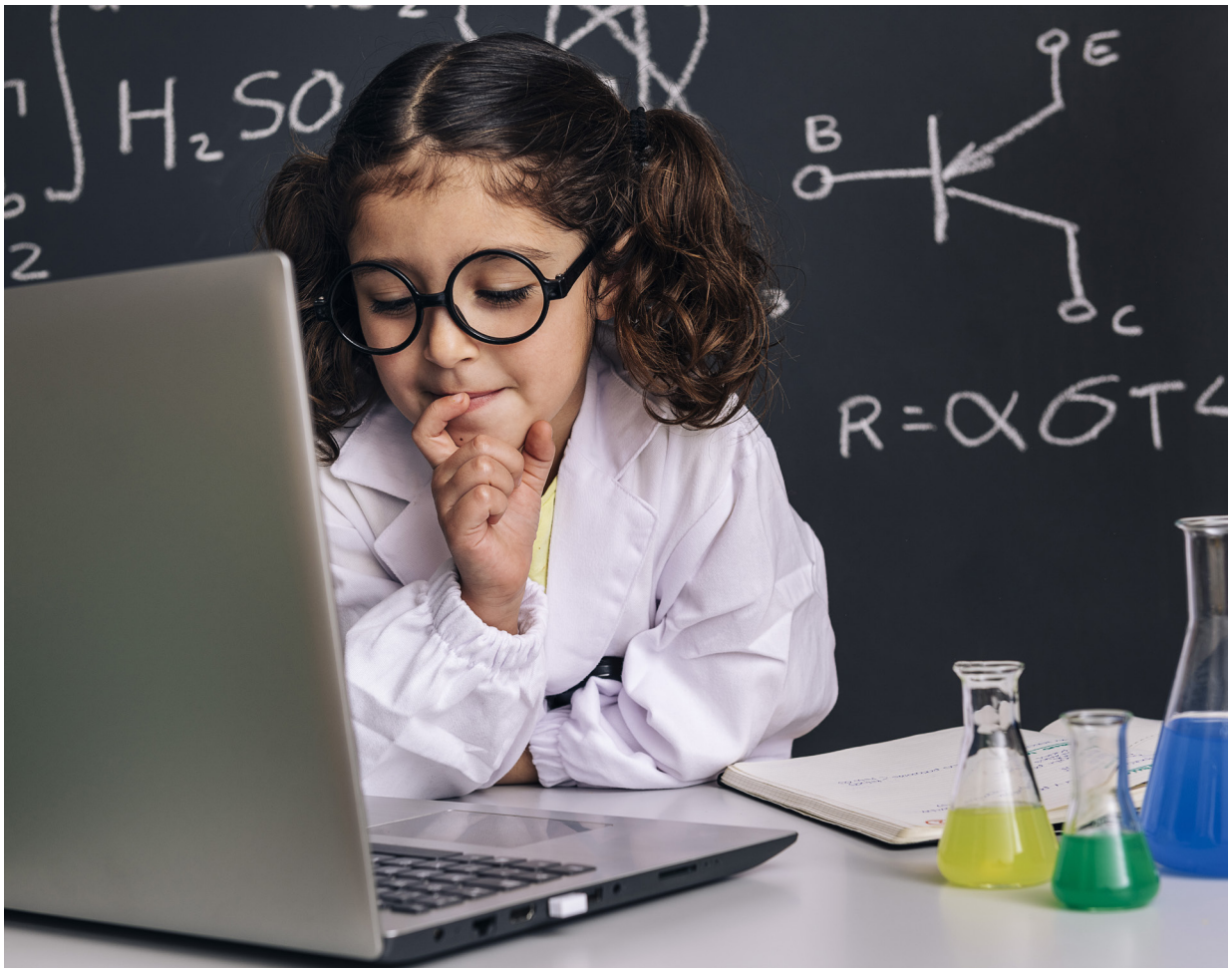
Crystal?

You can ask kids to create an algorithm for other things, like going to bed at night, packing their bag for school, making a peanut-butter sandwich... or for something extra complicated - they could create an algorithm which lets them know when a grown-up or sibling is approaching their bedroom! This will help them hide any secret spy plans just in time.

From the curriculum:

"Pupils should be taught to: understand what algorithms are; how they are implemented as programs on digital devices; and that programs execute by following precise and unambiguous instructions."

How to think like a computer scientist



Now you've covered the basics, you're going to learn how to think like Tim Berners-Lee. He invented the internet!* Here are a few more computer science concepts for you:

Abstraction – this is when you remove unnecessary detail to help analyse a problem or create a system. For example, the London Tube map doesn't show distances, but it does show the relationships between stations and train lines

From the curriculum:

"Pupils should be taught to: understand and apply the fundamental principles and concepts of computer science, including abstraction, logic, algorithms and data representation."

Decomposition - this is when you break down problems (or systems) into smaller parts. Most code is divided into smaller units, as this helps a lot in development and testing. Decomposition is a useful way of thinking about any project, like putting on a school play.

From the curriculum:

"Pupils should be taught to: design, write and debug programs that accomplish specific goals, including controlling or simulating physical systems; solve problems by decomposing them into smaller parts."

Logical reasoning - computers are predictable: if they are in the same state, given the same input and use the same programs, they should reliably produce the same output.

From the curriculum:

"Pupils should be taught to: use logical reasoning to predict the behaviour of simple programs."

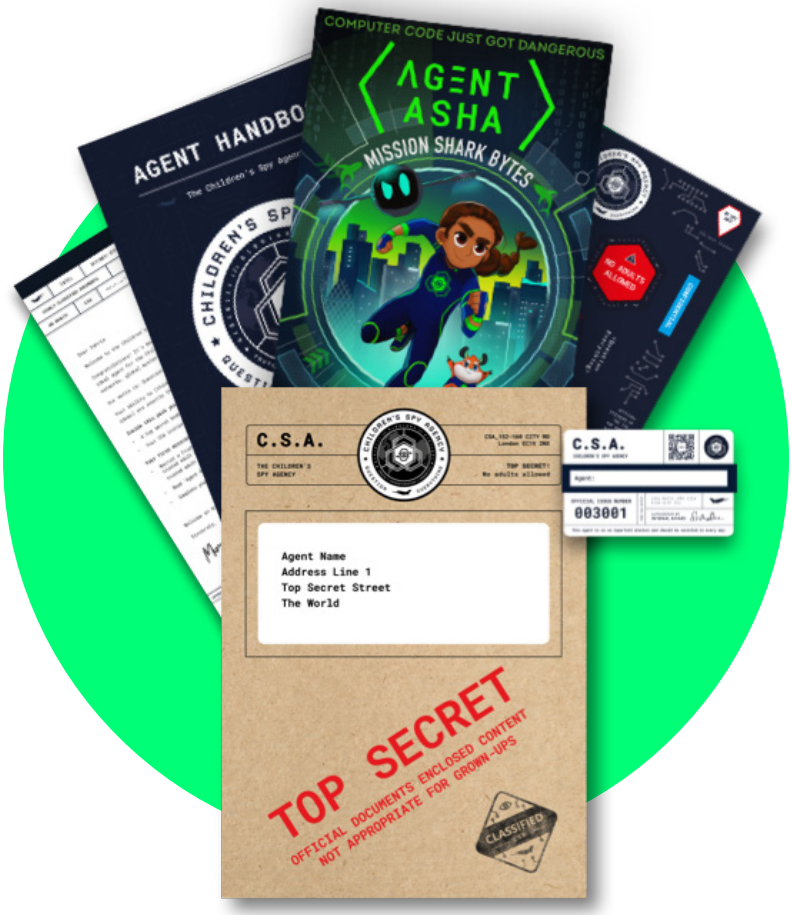
Patterns or generalisation - basically, this means you should be as lazy as poss. Hurray! A good coder will build on the work of others, looking to re-use someone else's code rather than writing lots of new stuff. Sort of like copying someone else's homework, but tweaking a few words so the teacher doesn't notice.

***Hear Sir Tim's talk: '30 years on, what's next #ForTheWeb?' here**

Learn digital literacy with the Agent Asha Gift Pack

A personalised gift to inspire the next generation of coders, creators and critical thinkers.

Agent Asha teaches kids how to go undercover *(internet safety)*, design gadgets *(computer programming)*, avoid data-hungry baddies *(digital literacy)*, analyse intelligence *(fake news)* and more.



Use code CATS for **15% off**

https://shop.brightlittlelabs.com/