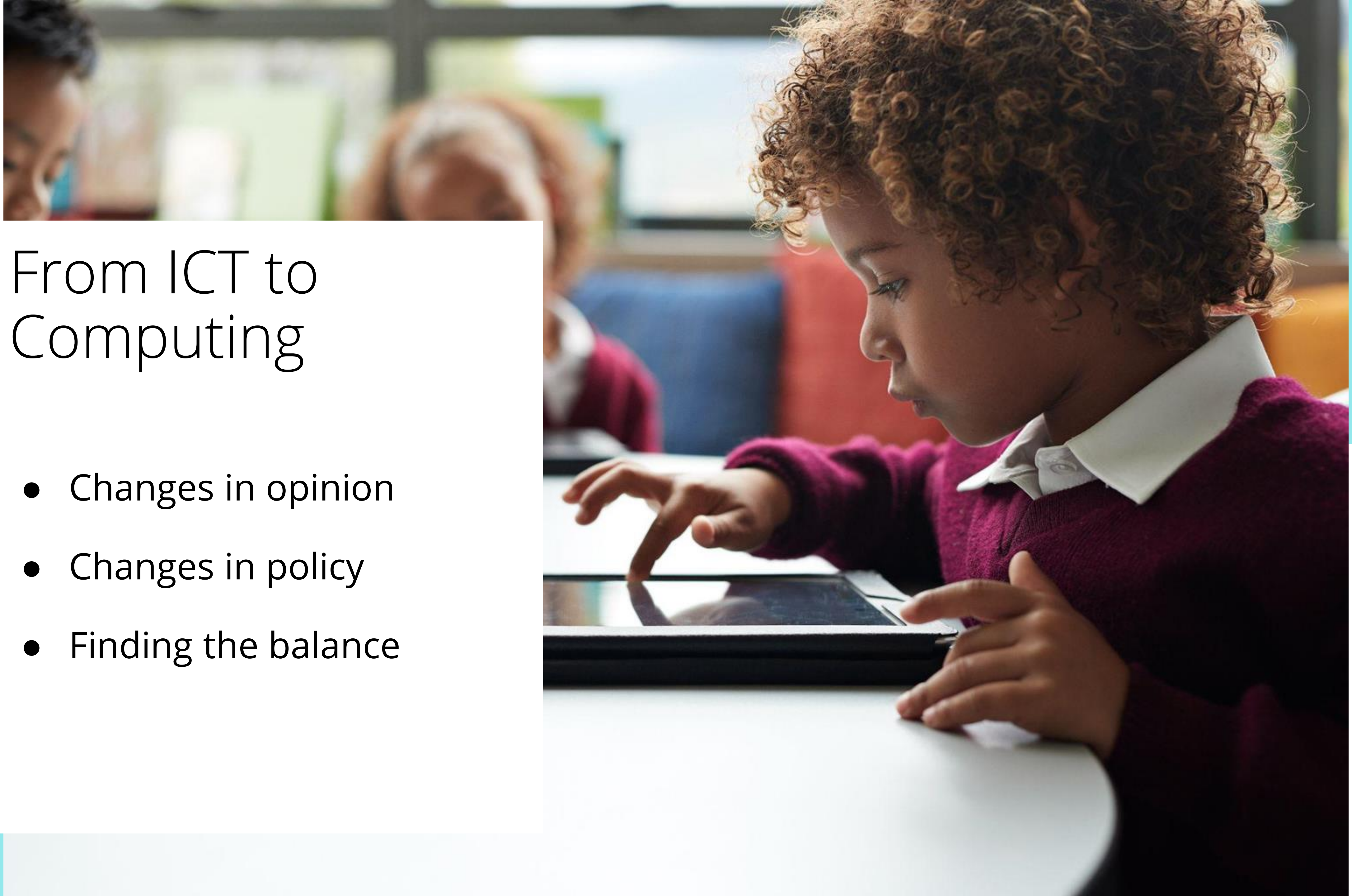
Let’s talk Computing

Paul Clowrey and Sabiha Munshi



From ICT to Computing

- Changes in opinion
- Changes in policy
- Finding the balance

Rebooting the curriculum

- Teaching ICT in 2010, the Key Stage 3 and Key Stage 4 curriculum was built around practical information technology and project management skills, including:

spreadsheets

databases

professional business documents (office)

basic control systems

- Following an English government-led investigation in 2012:

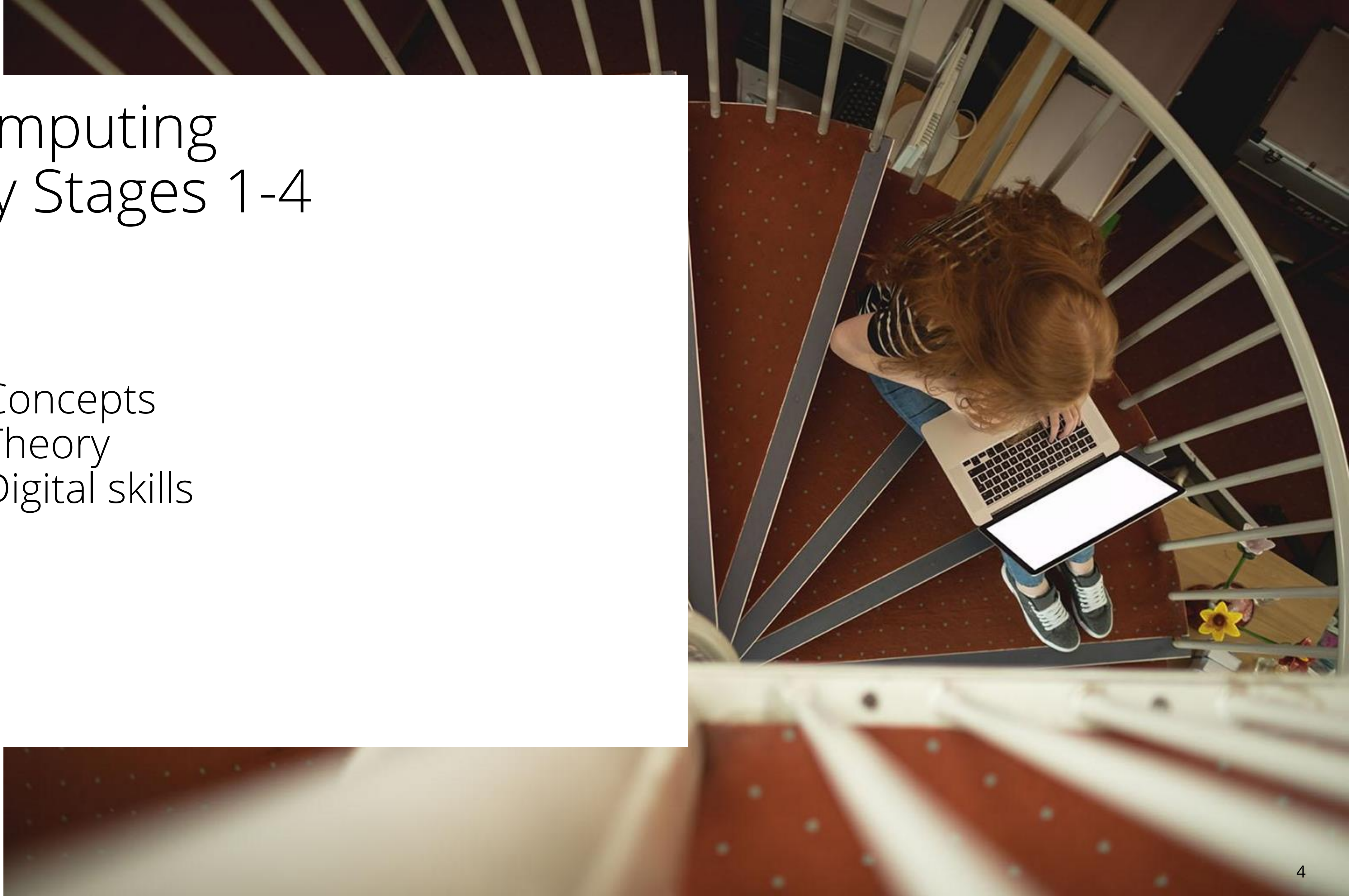
ICT curriculum as it stood was scrapped

replaced with a computer science-based curriculum

formalised in 2013 as the new UK National Curriculum Computing

programmes of study

- Teaching computing in 2022, courses still being developed and teachers embracing the new opportunities of a very flexible curriculum



Computing
Key Stages 1-4

- Concepts
- Theory
- Digital skills

Computing across the curriculum

Key Stage 1

- An introduction to algorithms
- How they become become programs
- The prevalence of computers across the world.

Key Stage 2

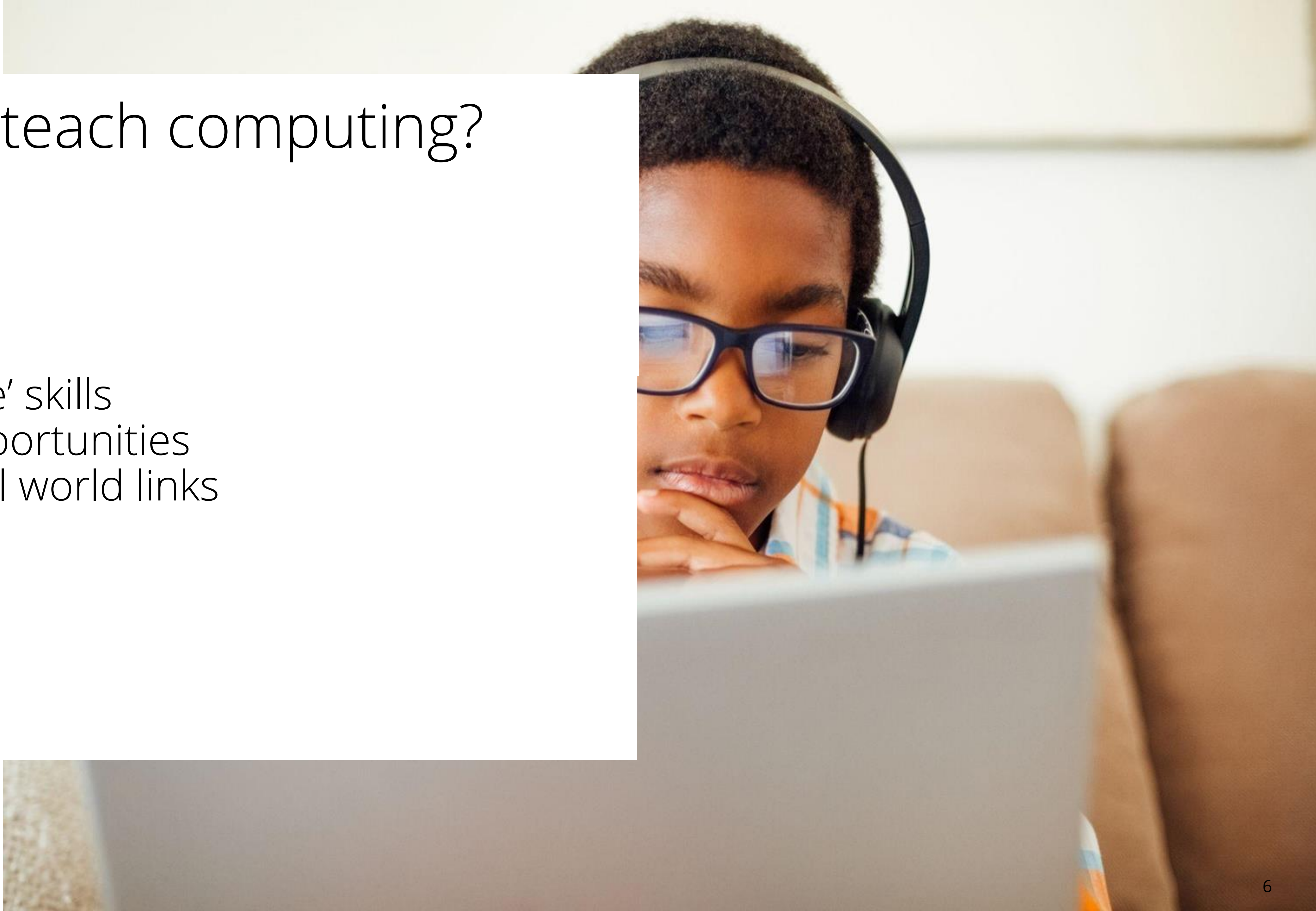
- Being able to create simple programs
- Use technology in a variety of ways
- The importance of internet safety.

Key Stage 3

- Experimenting with programming languages
- Understanding the infrastructure of computer network technology
- Creative projects using digital artifacts
- Using computers responsibly

Key Stage 4

- Develop effective computer science, digital media, and information technology-based products for a range of audiences.
- Understanding the many issues linked to our ever-increasing online lives.

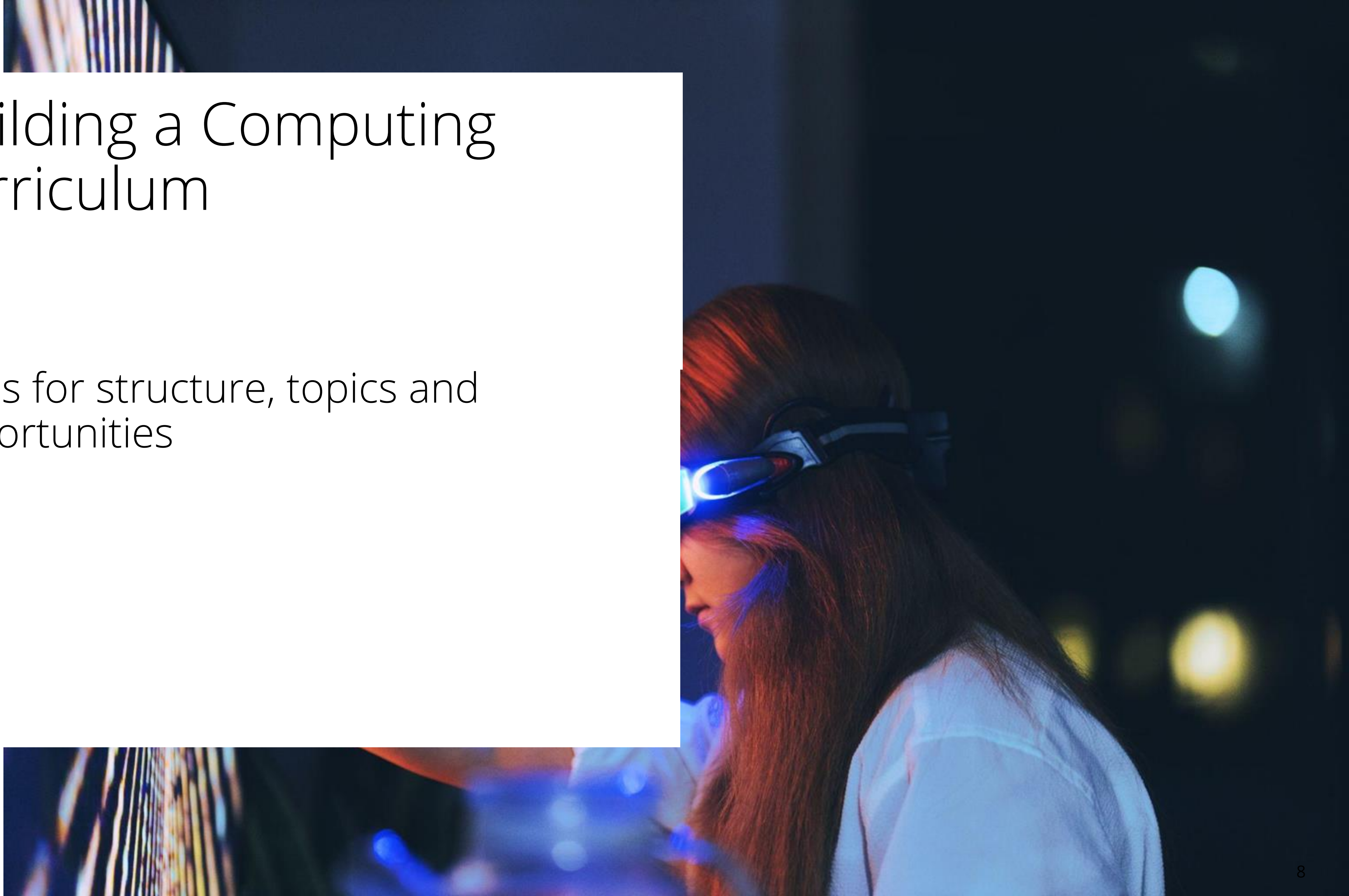


Why teach computing?

- ‘Live’ skills
- Opportunities
- Real world links

Why teach Computing?

- Computing is a 'live' subject
- The content changes as the world changes
- Linked directly to our homes, schools, places of work and social interaction
- An opportunity for classroom innovation for anyone interested in:
  - Embracing the new
  - Online learning platforms
  - How our society is changing, for better or worse
  - Linking with schools and learners around the world
  - Programming real industry standard computing languages



Building a Computing curriculum

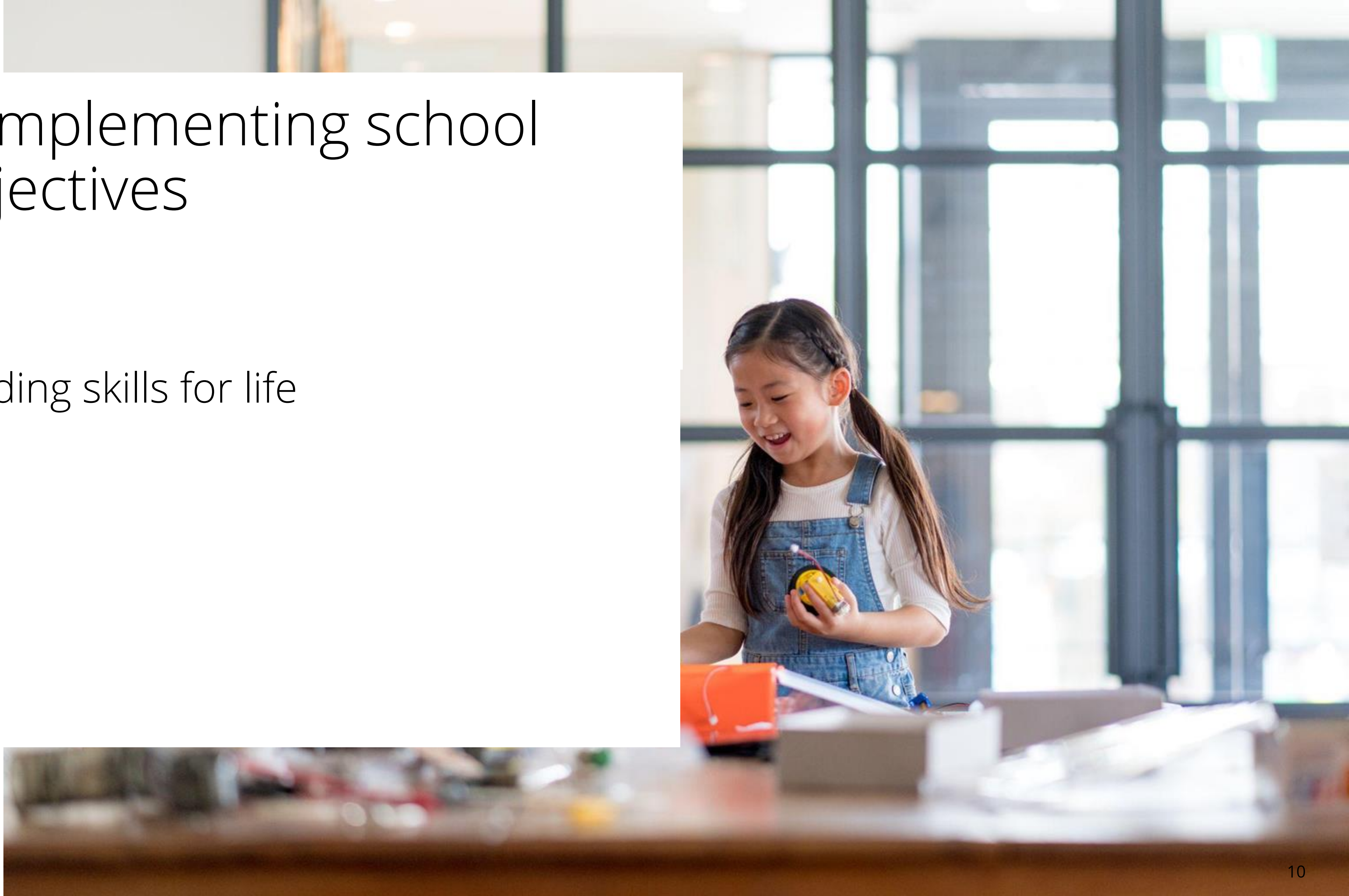
Ideas for structure, topics and opportunities

Building a Computing curriculum

- Primary and lower secondary is where the most benefit can be gained
- Computing lessons should be designed to fit your curriculum and your students.
- With a weekly lesson, topics of 6-10 weeks are ideal
- Computing includes computer science and information technology skills
- Look for those innovation opportunities
- Keep pulling it back to students' own experiences and their hopes for the future.

Complementing school objectives

Building skills for life



How does Computing complement school objectives?

- Developing the ‘whole child’, Computing helps build:

  Self confidence – seeing instant outcomes
  Social skills – how the Internet makes us citizens of the Earth
  Skills for further education
  Skills for employment
  Research, revision and project management
  Problem solving and independent thinking
  Acceptable behavior

Key Computing terminology

- What is the key terminology in computing and how can we explain these to students?

What is the key computing terminology in computing and how can we explain these to students?

Algorithm Simply a set of step-by-step instructions to carry out a function. This can be as verbal, pictorial or simple everyday steps. Students are encouraged to create this as ‘unplugged’ (or non-device activities) to help them write programs on Scratch or other.

Programming language

A computer programming language using specific terms and syntax that humans use to write computer instructions. This has to be precise and a good way to think of this is **‘will the computer understand what I’m inputting?’** In the 4-11 age bracket, it is mainly block programming such as Scratch, however 11-14 and beyond may use programming language such as python.

Variable

A variable is something that can be changed and in computer programming. We use variables to store information that can be used later in our program. For example, when designing a game, we may have a variable that adds a point each time the player scores.

Debug

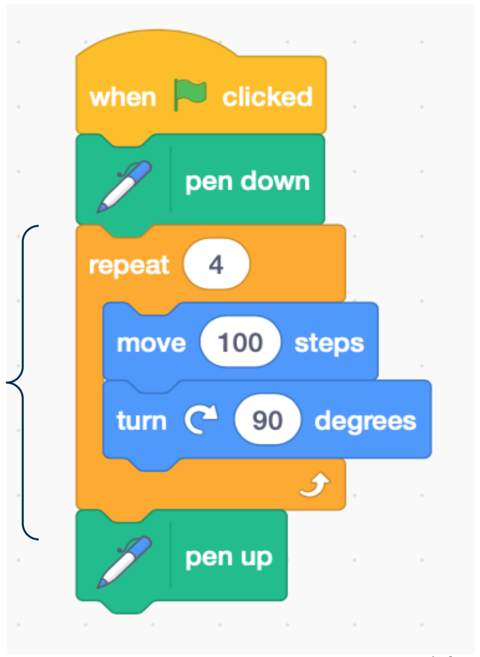
This is the process of looking for and ***removing errors*** from a computer program. An example may be where an extra step has been added, which means that the program isn't functioning as it should. A recommendation for all students is to 'role play' the program or work with another student who can help them spot the error.

Take away the fear of failure in students by ensuring that they understand that in 'real life,' programmers are constantly debugging and that sometimes even they need others to spot the errors to help them debug.

Iteration

This is sometimes called ***repetition*** and is the process of repeating a task over and over again.

- **Count controlled iteration** is when a loop is carried out for a set amount of times
- **Condition controlled iteration** is when the loop could be carried out indefinitely as we do not know how many times it will be repeated

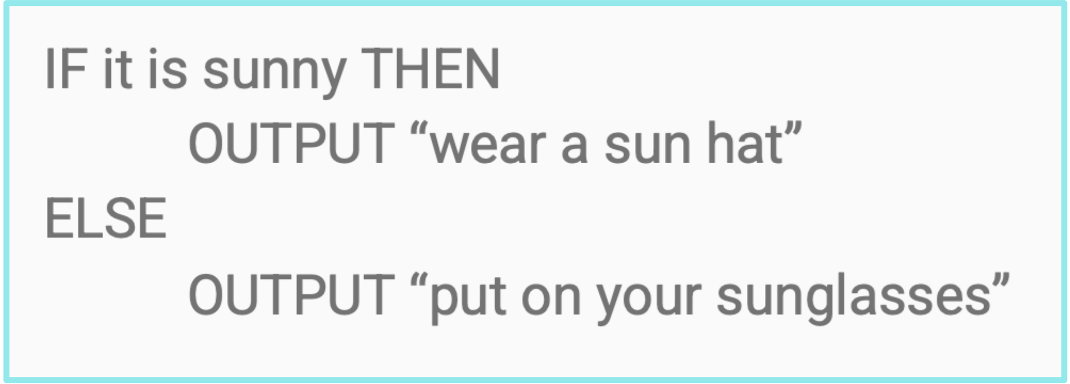


```
when 🏳 clicked
pen down
repeat 4
    move 100 steps
    turn ↻ 90 degrees
pen up
```

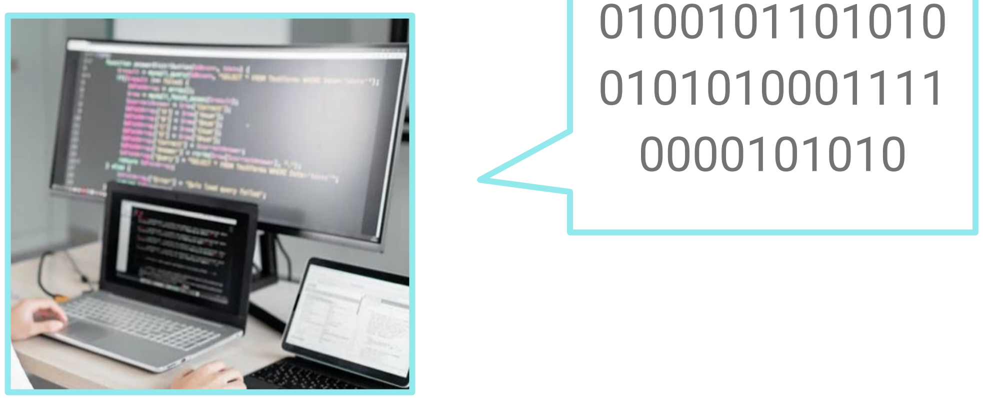
Selection

Selection adds the ability to allow more complex interaction from the user with the program. It allows for the user to enter (input) and the program will then be able to decide how to respond (output), depending on the input given.

Selection also allows programs to become more complex in design, and taking it a step further, we can have nested selection which is where there is ‘selection within selection.’



```
IF it is sunny THEN
        OUTPUT “wear a sun hat”
ELSE
        OUTPUT “put on your sunglasses”
```



Binary

A computer only understands machine code known as binary. A computer is built up of switches – they can be on (represented by a 1) and off (represented by a 0). Every time you do anything on a computer, it is converted into machine code for the computer to execute.

It is useful to explain to students that any command that is entered into the computer, has to be converted into binary for the computer to be able to understand it. However, It is carried out so quickly, that we (as the user) do not notice it.

Computational thinking

- What is computational thinking?
- What are the core constructs in computational thinking?
- How can we develop computational thinking with students?

What is computational thinking?

The curriculum places **computational thinking and creativity** at the heart of computing, but what exactly is it?

Is it **'thinking like a computer?'**

No. A computer does not have a brain or thought processes so we cannot 'think' like a computer. It is an inanimate object that only executes the functions (output) that we, as humans input into it.

Is it thinking about **'problems'** in a way that allows a computer to solve them?

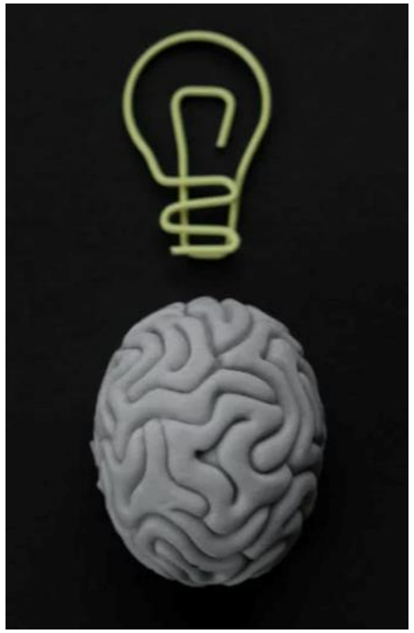
Computational thinking is something that people do, not computers. It includes logical thinking, the ability to recognise patterns, think in algorithms, decompose a problem and abstract a problem.

Simply put, computational thinkers are like **problem solvers!**

pattern recognition

decomposition

abstraction



logical reasoning

algorithmic design

How do we develop computational thinking skills in students?

Why is this so important?

Research by Richard Sheldrake (2018) showed that exposure to and building confidence in computational and critical thinking skills can often be a predictor in students following careers in STEM fields.

Some ideas of how to make computational thinking fun are:

1. **Abstraction** Make treasure maps (or maze grids) in small groups and ask the students to mark where the treasure is with an X. Swap the map with another group and ask them to plan the quickest/ fastest/ danger-free route to the treasure!

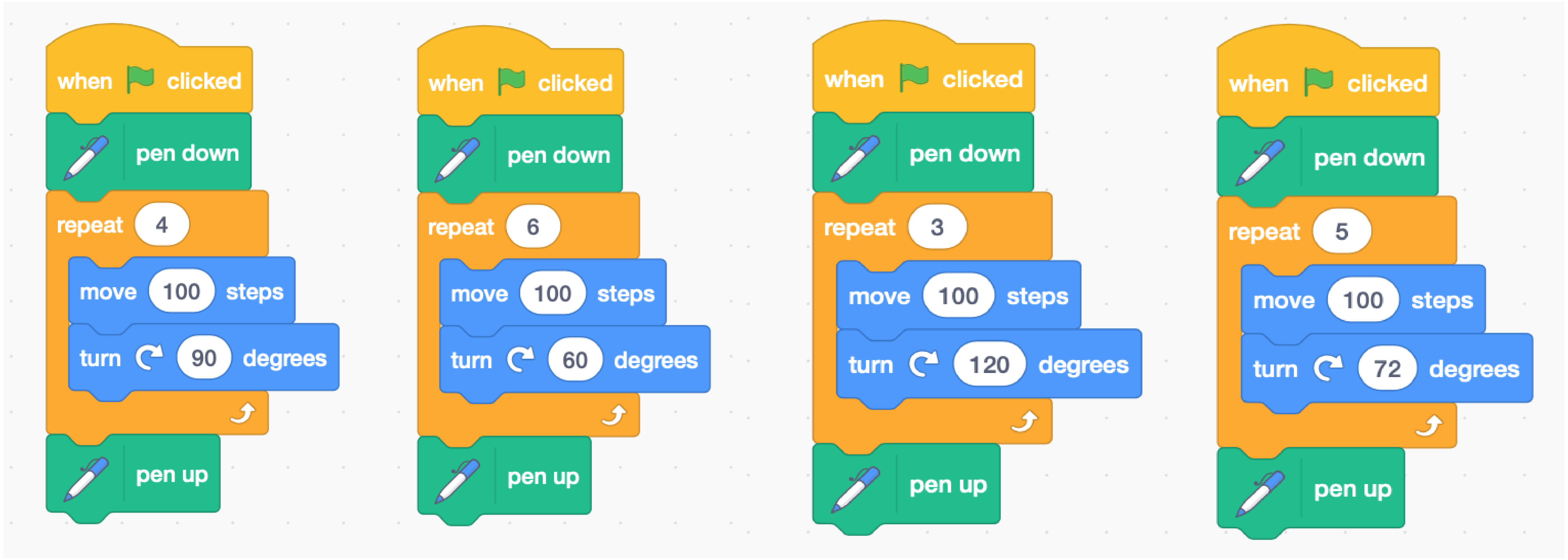
1. **Algorithm design** Creating recipes is always such fun in school. Students will not even realise that they are using algorithms in the sequence of step by step instructions and add a trick up your sleeve by missing out steps so that students then need to debug. For example, making a banana smoothie but with the milk missing from the algorithm. Or the teacher role playing making a sandwich but with the algorithm stating that you need to spread butter, but without mentioning using a butter knife to spread it. This is where **logical thinking** will develop too!

1. **Decomposition** Break up a complex problem into simpler parts before solving each part individually. A fun way of understanding decomposition is breaking down a dance sequence, creating a game, or when creating a character based on an algorithm with multiple steps.

4. **Pattern recognition**

Students could do this when using Scratch to draw shapes.

- Can students work out which shape is drawn from each of these?
- **Challenge:** Can students write the program for an octagon, decagon or heptagon? Can they explain using their logical reasoning as to why and how they know?



```
when 🏁 clicked
pen down
repeat 4
    move 100 steps
    turn ↻ 90 degrees
pen up
```

```
when 🏁 clicked
pen down
repeat 6
    move 100 steps
    turn ↻ 60 degrees
pen up
```

```
when 🏁 clicked
pen down
repeat 3
    move 100 steps
    turn ↻ 120 degrees
pen up
```

```
when 🏁 clicked
pen down
repeat 5
    move 100 steps
    turn ↻ 72 degrees
pen up
```

Successful Computing lessons

- What can successful computing lessons involve?

What can successful computing lessons involve?

a curriculum that includes all 3 areas of computing
- Digital Literacy
- Computer Science
- Information Technology

Cross curricular and thematic learning links

a mixture of both 'screen' and 'unplugged' activities

Add complexity with adequate 'challenge' opportunities and address misconceptions

a learning environment that allows computational thinking constructs to be explored and reflected upon

importance of 'role play' in computing

Q&A

Pearson