FACTFILE:
GCSE DIGITAL TECHNOLOGY
Unit 4
DIGITAL AUTHORING CONCEPTS

Rewarding Learning

Digital Design Principles

Learning Outcomes

Students should be able to:

- explain in simple terms the underlying concepts of computational thinking – abstraction and decomposition.

Computational Thinking

If you understand the topics above you will have some of the building blocks needed to create computer programs in Python. Moreover, the programming constructs that have been discussed are common to other high level programming languages. Java and C#, for example, also have integer, string and Boolean data types.

There is more to creating useful software, however, than just understanding a few of the building blocks. Software development normally begins with a conversation between the *software developer* and the *client*. A client is just a person or organisation who has approached a software developer for help with a problem – typically with the expectation of having some software developed.

During this initial conversation the developer aims to understand the client's problem(s). When the problem is understood, a *computational solution* – i.e. one capable of being automated on a computer – is designed, programmed, tested and deployed. In order to do this the software developer needs a whole range of skills – not just a knowledge of coding. These skills are sometimes described collectively as *computational thinking*.

Computational Thinking

"*Computational thinking* allows us to take a complex problem, understand what the problem is and develop possible solutions. We can then present these solutions in a way that a computer, a human, or both, can understand."

BBC Bitesize: Introduction to Computational Thinking

Computational Thinking

"*Computational thinking* is a problem solving process that includes a number of characteristics, such as logically ordering and analyzing data and creating solutions using a series of ordered steps (or algorithms), and dispositions, such as the ability to confidently deal with complexity and open-ended problems."

Google for Education: Exploring Computational Thinking

https://edu.google.com/resources/programs/exploring-computational-thinking/

At the very least Computational thinking requires skills in the following four important processes.

- Problem Decomposition
- Pattern Recognition
- Abstraction
- Algorithm Design

We will illustrate these four processes in the context of an example. Imagine that a school teacher (the client) presents a software developer with the following problem description.

Pupil Marks Problem

I have a class of thirty pupils studying biology – both boys and girls. Approximately once a week each pupil takes a short class test, and at the end of the year there is a final examination. Half way through the year we have a parents' evening, at which I meet with each pupil's parent(s) to discuss their progress.

I need help in recording and organising all of these test and exam marks. I need to be able to see the marks of the whole class, including the class average, at any time during the year. At the parents' meeting I need to have a detailed performance record for each pupil: their mark on each class test, their average mark, and the class average.

I have tried creating a spreadsheet to help with this but there is too much data – it becomes cumbersome and data entry is error prone. I definitely do not want to use a spreadsheet.

Problem Decomposition

Problem decomposition involves breaking a large, complex problem into a number of smaller, simpler ones.

- The smaller problems should be easier to solve than the large one.
- It should be possible to combine the solutions to the smaller problems, to create a solution to the larger problem.

Problem Decomposition

"[Problem decomposition is] the breaking down of a system into smaller parts that are easier to understand, program and maintain."

BBC Bitesize: Decomposition

http://www.bbc.co.uk/education/guides/zqqfyrd/revision

For the *Pupil Marks Problem*, decomposition might identify (at least) the following sub-problems.

- Record class test mark for an individual pupil:
  - enter mark;
  - store mark.
- Record examination mark for an individual pupil:
  - enter mark;
  - store mark.

Pattern Recognition

Pattern recognition involves identifying similarities among problems. If two problems are similar then perhaps their solutions will also be similar.

Pattern Recognition

"*Pattern recognition* [...] involves finding the similarities or patterns among small, decomposed problems that can help us solve more complex problems more efficiently."

BBC Bitesize: Pattern Recognition

http://www.bbc.co.uk/education/guides/zxxbgk7/revision

For the *Pupil Marks Problem*, pattern recognition might notice that the storing and retrieval of data are common tasks, which have been solved many times before. Because of this there are a range of existing software application packages – called database management systems – one of which might be of use in solving the current problem.

Pattern recognition might also notice that the task of recording a class test mark for an individual pupil is essentially the same as the task of recording an exam mark. A solution to either one of these can be adapted to work for the other.

Abstraction

Abstraction is about differentiating between the important aspects of a problem – those ones that are essential to finding a solution – and the less important aspects.

Abstraction

"*Abstraction* is the process of filtering out – ignoring – the characteristics of patterns that we don't need in order to concentrate on those that we do. It is also the filtering out of specific details. From this we create a representation (idea) of what we are trying to solve."

BBC Bitesize: Abstraction

http://www.bbc.co.uk/education/guides/zttrcdm/revision

For the *Pupil Marks Problem*, abstraction might produce the following simplification.

- A group of pupils take a series of assessments and a single mark is stored for each pupil in each assessment.
- Calculations to be carried out:
  – For each assessment: class average = the average of all pupil marks;
  – For each pupil: assessment average = the average (so far) of all assessment marks.
- Assessment Report to be produced:
  – List all students;
  – For each student, show their mark in the given assessment;
  – Show the average mark gained in the given assessment.
- Student Report to be produced:
  – List all assessments that have been completed;
  – For each assessment, show the given student's mark and the class average;
  – Show the given student's average mark across all assessments that they have completed.

Other aspects are considered unimportant. For example, the distinction between class tests and final exams is unimportant – each just requires a single assessment mark to be stored for each pupil.

Question: What other aspects of the problem description are unimportant to the solution?

Algorithm Design

An algorithm is a sequence of steps designed to solve a particular problem. An algorithm might be expressed:

- in a stylised form of English called *pseudocode*;
- In a graphical form called a *flow diagram*;
- directly in a programming language such as Python, Java or C#.

Algorithm Design

"An *algorithm* is a plan, a set of step-by-step instructions to solve a problem. If you can tie shoelaces, make a cup of tea, get dressed or prepare a meal then you already know how to follow an algorithm."

BBC Bitesize: Algorithm Design

http://www.bbc.co.uk/education/guides/zpp49j6/revision/1

Whatever notation is chosen to express an algorithm, there will normally be constructs that are used to achieve: *sequencing*, *selection* and*iteration*.

- Sequencing is the ability to say that steps must be carried out in a given order – step 1, then step 2, then step 3, etc.
- Selection is the ability to specify different paths in an algorithm so that certain steps are only carried out if specified conditions are met. The conditional statements (if/else) in some of the Python examples above were used to achieve selection.
- Iteration is the ability to say that certain steps in the algorithm are to be repeated a given number of times, or until a given condition has been met. A repeated sequence of steps in an algorithm is often called a *loop*.

For the *Pupil Marks Problem*, we might develop an algorithm to calculate the average of a collection of marks. This might be expressed in pseudocode like this:

```
TotalSoFar = 0
NumberSoFar = 0
Repeat
    Get next mark (M)
    Increase NumberSoFar by 1
    Increase TotalSoFar by M
    Until all of the marks have
been processed
If NumberSoFar > 0
    Then Average = TotalSoFar /
NumberSoFar
    Else Average = 0
```

Question: Can you identify instances of sequencing, selection and iteration in this algorithm?

Problem decomposition, pattern recognition, abstraction and algorithm design are not always distinct – it is sometimes not clear where one ends and another begins. They are, however, powerful ways of thinking that can work together to produce computational solutions to real world problems.

Resources

Computational Thinking

- Computational Thinking for Educators: What is Computational Thinking? https://computationalthinkingcourse.withgoogle.com/unit
- Centre for Computational Thinking, https://www.cs.cmu.edu/~CompThink/
- Google for Education: Exploring Computational Thinking,

https://edu.google.com/resources/programs/exploring-computational-thinking/

- Open University: Introduction to Computational Thinking, http://www.open.edu/openlearn/science-maths-technology/computing-and-ict/introduction-computational-thinking/content-section-0#