Autumn term Year 10

Scheme of Work

(Mapped to Edexcel GCSE (9-1) Computer Science textbook)

GCSE (9-1) Computer Science

Pearson Edexcel Level 1/Level 2 GCSE (9-1) in Computer Science (1CP1)

# Autumn term Year 10

## Assumptions

This scheme of work is based on the assumption that the learners have had previous experience of computer science or programming as part of the National Curriculum for Computing (2014). The first few programming lessons are therefore refresher lessons.

The lessons in the various schemes of work total 150 hours altogether (see course planner); the guided learning hours for GCSE Computer Science are 120. It is expected that centres will modify the schemes of work and the course planner as required to fit their own timetabling.

The programming language used here is Python 3.0. Centres can of course use any of the other languages in the specification, namely Java, Pascal/Object Pascal, Visual Basic.NET, C-derived (C, C++, C#) and use equivalent examples from these languages.

This document is subject to change.

## Mapping to Textbook

**This version of the Autumn term Year 10 scheme of work is mapped to the draft sample Chapter 1 from the Edexcel GCSE (9-1) Computer Science textbook. Chapter 1 covers Problem Solving which includes creating algorithms which can then be programmed in the chosen language. Not all of the content is referenced here; the textbook is designed to be used throughout the 2 years of the course so some content is more comprehensive than is needed, or is not covered in the first term.**

## Explanations

The course has been broken down into 6 terms. Over the course, lessons are split into the 6 topics in the subject content and the colours in the schemes of work mirror the colours of the topics in the course planner.

Weeks are numbered from 1 in each term and there are 2 lessons each week. This scheme of work details the content to be covered in the Autumn term of Year 10. Lesson 1 covers mainly the theoretical principles of computer science. Lesson 2 covers mainly the practical application of problem solving and programming, although it should be remembered that these topics are also examined. Problem solving and programming are usually taught together to increase computational thinking skills, which will also be used in the other subject content topics.

The specification references given represent the specification content that is covered in the lesson. During the problem solving and programming topics, many lessons cover a range of content but only the main specification reference is given. You are strongly advised to use the specification alongside this scheme of work.

The lesson summary gives an overview of the lesson content. The title in bold refers to the subject content headings given for each topic in the specification.

The lesson content gives suggestions on how this material could be taught. Example activities for each lesson are given in the activities files. The activities are numbered with the lesson number, for example Activity 1.2.3 is the third activity for lesson 2 in week 1. These activities are provided to give teachers ideas of activities and programming challenges they may find useful in teaching this specification. They are examples of the types of activities that can be provided and they do not exhaustively cover the specification.

A suggestion for a homework task is given at the end of each lesson. Teachers may wish to refer to the homework in the starter for the following lesson.

A list of useful resources, both online and in print, is given at the end of this document, together with teaching suggestions, starters and plenaries, homework tasks and how to get students to self-assess their programs.

**Disclaimer**: The problem solving and programming lessons are about teaching transferable computational thinking concepts using Python rather than being a Python programming course.

**Autumn term Year 10**

|  |
| --- |
| **Week 1** |

| **Lesson** | **Spec ref** | **Lesson summary** | **Lesson content** | **Lesson resources** | **Chapter 1 Pages** |
| --- | --- | --- | --- | --- | --- |
| 1 | All  4.2.1  4.4  2.1 | **Course introduction and key concepts** | Describe course content: problem solving, programming, data, computers, communication and the internet, the bigger picture in computer science. Explain that different topics will be taught at different times, but that programming and problems solving (key for computational thinking) will be spread throughout the course.  Show a mobile phone, tablet and a laptop running an app and ask students to decompose what they are seeing into input, process and output.  Show motherboard/Raspberry Pi® and show CPU.  Discuss hardware and software components and relate to course content. | Python 3.0 installed  Week 1 Lesson 1 activities | Pages 1-3 and 20-21 as overview |
| **Introduction to Python and programming** | Introduce students to the Python interactive mode and give them the opportunity to write a program that prints text on the screen. Demonstrate how even a simple program can be decomposed into input, process and output. **(Activity 1.1.1)**  Introduce the concept of sequence in a program and give students the opportunity to write simple programs to print lines. **(Activities 1.1.2 and 1.1.3)**  **Homework:** Ask students to find out **five** facts about the Python programming language. |  | Page 5 |
| 2 | 2.2.1 | **Python IDLE and programming** | Discuss ethical impact of using technology.  Share some statements about Python and ask students to decide if they are true or false (and to correct the false statements). **(Activity 1.2.1)**  Discuss why Python is a good language to use to learn to program.  Explain the interactive mode of the Python **I**ntegrated **D**eve**l**opment **E**nvironment (IDLE) and explain some useful keyboard shortcuts. Ask students to make a note of these shortcuts. **(Activity 1.2.2)**  Give students the opportunity to explore features of the print function to display information. **(Activities 1.2.3–1.2.7)** | Week 1 Lesson 2 activities |  |

|  |
| --- |
| **Week 2** |

| **Lesson** | **Spec ref** | **Lesson summary** | **Lesson content** | **Lesson resources** | **Chapter 1 Pages** |
| --- | --- | --- | --- | --- | --- |
| 1 | 1.2.1  1.1.2 | **Problem solving:** ways of describing problems (algorithms, written description, flowchart, pseudo-code) | Explain what is meant by decomposition of a problem.  Explain what an algorithm is and that algorithms can be represented as written descriptions, as pseudo-code, as a flowchart and as program code. Give an example of a specific problem showing all these notations, e.g. calculating the mean. Emphasise that these are different ways of representing the same problem (note: we are not expecting students to understand these different representations yet). **(Activity 2.1.1)**  **Homework:** Ask students to explore ASCII art and write Python commands to display some ASCII art. **(Activity 2.1.2)** | Week 2 Lesson 1 activities | Page 7, 40 |
| 2 | 2.5.1 | **Operators:** arithmetic operators, order of precedence and parenthesis | Demonstrate the Python interactive mode as a calculator to introduce the arithmetic operators.  Give students the opportunity to explore mathematical operators. **(Activities 2.2.1 and 2.2.2)**  Demonstrate how to combine calculations with text. Give students the opportunity to try this for themselves. **(Activity 2.2.3)**  Explain the use of parentheses and the order of precedence. **(Activities 2.2.4 and 2.2.5)**  **Homework:** Ask students to write multiple-choice questions on precedence and to devise their own mnemonic for remembering the order of precedence. **(Activities 2.2.6 and 2.2.7)** | Week 2  Lesson 2 activities | Pages 4-6 |

|  |
| --- |
| **Week 3** |

| **Lesson** | **Spec ref** | **Lesson summary** | **Lesson content** | **Lesson resources** | **Chapter 1 Pages** |
| --- | --- | --- | --- | --- | --- |
| 1 | 2.1.2  2.1.3  2.1.5 | **Develop code:** error messages, debugging, comment statements, writing, saving and retrieving program files | Explain how to interpret common error messages. Use examples of syntax messages and runtime errors and explain the difference between them. **(Activities 3.1.1 and 3.1.2)**  Explain why it is important for students to be familiar with the various error messages Python produces. Give students the opportunity to explore errors and create some of their own. **(Activity 3.1.3)**  Emphasise to students that when developing code, making mistakes is OK and that debugging programs is a normal part of programming.  Demonstrate how to include comment statements using the # symbol and explain that comments are used to make the code readable. **(Activity 3.1.4)**  Explain how programs can be stored in a file using the file/new window command. Give students the opportunity to practise saving and running program files. Remind students of the need to use meaningful filenames and in which folder to store their files. **(Activities 3.1.5 and 3.1.6)**  **Homework:** Ask students to write a summary sheet on how to run Python programs. **(Activity 3.1.7)** | Week 3  Lesson 1 activities | Pages 6, 10, 24-28 |
| 2 | 2.3.1  2.3.4  2.1.2  2.4.1 | **Data types, variables and input** | Explain and describe the four data types: integer, real, Boolean and char.  Ask students to investigate the origin of the term ‘Boolean’. **(Activity 3.2.1)**  Demonstrate how the type function can be used to inspect the type of a given expression. Give students the opportunity to explore data types. **(Activities 3.2.2 and 3.2.3)**  Introduce the concept of variables using labelled boxes and cards. Explain that a variable is a named space in a computer’s memory where we can store things and that we can change (vary) the things that are stored in that memory.  Explain how to assign values to variables and the conventions for selecting variable names. Give students the opportunity to explore variables, variable names and to write programs that use variables. **(Activities 3.2.4–3.2.6)**  **Homework:** Ask students to complete a table describing Python’s four data types. **(Activity 3.2.7)** | Week 3  Lesson 2 activities | n/a |

|  |
| --- |
| **Week 4** |

| **Lesson** | **Spec ref** | **Lesson summary** | **Lesson content** | **Lesson resources** | **Chapter 1 Pages** |
| --- | --- | --- | --- | --- | --- |
| 1 | 1.2.2  2.4.1 | **Problem solving:** decomposition | Show students this problem: [www.brainboxx.co.uk/a4\_resource/pages/puzzleboxx/puzzles/pizza.htm](http://www.brainboxx.co.uk/a4_resource/pages/puzzleboxx/puzzles/pizza.htm)  Working in small groups, students solve the problem. Then review: how did they breakdown the problem and into what sub-problems? Why? |  | Pages 40-41 |
| **Input/output:** input function, int function | Explain the use of the input function to assign input values to a variable.  Explain the use of the int function to convert a string to an integer and why it is often used in conjunction with the input function when entering numbers.  Give students the opportunity to write programs using the input function, int function and variables. **(Activities 4.1.1–4.1.3)** | Week 4  Lesson 1 activities | Pages 5-6 |
| 2 | 2.4.1 | **Formatting input/output:** .format | Explain that string method .format gives you more control over the formatting of output than using space-separated values in print. Show students how to use the .format method using format fields (placeholders) for variables in strings. Explain that the numbers in the {} brackets are replaced by the objects given in the .format method. They refer to the position of the objects (starting from zero).  Give students the opportunity to experiment with string formatting. **(Activity 4.2.1)**  Show students how the .format method can be used with numbers to format the number of decimal places that are displayed.  Give students the opportunity to experiment with using .format to format numbers. **(Activities 4.2.2 and 4.2.3)**  **Homework:** Set students a programming challenge requiring the use of the input function, the int function, variables and mathematical operators. **(Activity 4.2.4)** | Week 4  Lesson 2 activities |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Week 5** |  | | |
| **Lesson** | | **Spec ref** | **Lesson summary** | | **Lesson content** | **Lesson resources** | **Chapter 1 Pages** |
| 1 | | 1.2.1  1.2.2  1.1.2  1.1.9 | **Problem solving:** analyse a problem and design a solution using decomposition | | Prepare for the lesson by finding out the most recent ‘popular’ passwords (try searching for Splashdata).  The problem: how to remember many passwords? Students work in small groups to decompose the problem and propose a solution. The solution should be a written description of an algorithm (or any other technique they are familiar with at this stage in the course).  Each group should present their algorithm to the rest of the class who should then critically evaluate its fitness for purpose in solving the problem. |  | Pages 20, 40, 41 |
| 2 | | 2.2.1  2.2.2  2.5.2 | **Selection constructs and relational operators:** if, then, else | | Discuss real-life examples of relational operators that can either be true or false such as passwords. Brainstorm other examples. **(Activity 5.2.1)**  Work with the class to complete the table of Python relational operators. **(Activities 5.2.2 and 5.2.3)**  Show students how to write condition statements and explain that they result in being either true or false. **(Activities 5.2.4 and 5.2.5)**  Explain that the if statement tests a condition which will be either true or false (Boolean) and always has two outcomes. It allows branching in the program, depending upon the condition. Describe the syntax of the command including the need for the colon :.  Explain that when an if else command is used one of the blocks of code will be executed depending upon the condition. Show the difference between:  ifcondition:  Block  And:  ifcondition:  Block A  else:  Block B  Demonstrate programs that use if statements and if else. This will be the first time indenting has been used, so emphasise the need to have the correct indentation (four spaces) and how Python uses indentation to indicate the end of a block of code.  Give students the opportunity to write programs using if statements. **(Activities 5.2.6 and 5.2.7)**  **Homework:** Set students a programming challenge requiring the use of an if statement. Programs can be written in structured English or program code. | Week 5  Lesson 2 activities | Pages 9-12 |

|  |
| --- |
| **Week 6** |

| **Lesson** | **Spec ref** | **Lesson summary** | **Lesson content** | **Lesson resources** | **Chapter 1 Pages** |
| --- | --- | --- | --- | --- | --- |
| 1 | 4.2  4.4 | **What is a computer?** Hardware vs. software | As a class, brainstorm the question ‘What is a digital computer?’  In pairs, ask students to list as many different types of digital computers as possible. Suggest they look for computers both in the past as well as the present, and note that there are such things as embedded computers (more later in the course). **(Activity 6.1.1)**  Ask students to contribute the digital computers on their list to a class list. Ask whether any of the devices on the class list have made them think again about what a digital computer is.  Have a range of digital computers for students to look at such as mobile phone, tablet, Raspberry Pi®, PC, laptop, embedded computer, games console, etc. If you do not have the physical devices, show images of different types of computers. You will find a good range on Tumblr: [www.physical-computing.tumblr.com](http://physical-computing.tumblr.com/)  Define a digital computer as an automatic, programmable data processor. As a class, discuss each part of the definition.  What does digital mean? Digital means that the computer only understands the binary quantities that take two values, 1 or 0. Mention that other types of computer do exist, including the quantum computer (which students will learn more about later in the course) and show a picture of a quantum computer: [www.en.wikipedia.org/wiki/File:DWave\_128chip.jpg](http://en.wikipedia.org/wiki/File:DWave_128chip.jpg)  What does automatic mean? Automatic means that it operates without human intervention, except where human intervention is planned.  What does programmable mean? Programmable means that the computer follows instructions that are stored as a program.  What does data processor mean? Data processor means that the computer does things with data.  Explain that computers are made up of two components: hardware and software. Ask students if they can define these components. Hardware is the physical components that make up a computer. Software is the programs that can be run on the computer.  Ask students to sort the list into hardware and software devices **(Activity 6.1.2)** and then, as a class, brainstorm why both are needed.  Explain that some hardware such as a memory card may contain software but the device itself is hardware. | Week 6  Lesson 1 activities | n/a |
|  | 4.1.1 | Input-process-output model | Explain that we can think about how computers work using the input-process-output model. Draw and label the parts of the model (see **Activity 6.2.1** for model).  Explain that the input-process-output model can be used to model computer systems. Display an empty input-process-output model on the board and brainstorm other examples.  **Homework:** Ask students to summarise what they have learnt in this lesson. **(Activity 6.1.3)** |  |  |
| 2 | 2.4.1 | **Programming:** input-process-output model | Consolidate input-process-output model: write model on the board again and refresh.  Ask students to write a Python program that asks the user to enter three numbers, adds up the numbers and displays the answer on the screen. Ask students to explain the program using the input-process-output model. **(Activity 6.2.1)** | Week 6  Lesson 2 activities | n/a |
| 2.1.1  2.2.1  2.2.2 | **Selection constructs and writing readable code:** elif, randint() | Explain the use of the elif command (else if) to test multiple conditions. This is like a case statement in other programming languages. This activity also introduces random number generation. Remind students about input-process-output in this program.  if condition:  Block A  elif condition:  Block B  elif condition:  Block C  else:  Block D  Show students how to write programs using if elif commands. **(Activity 6.2.2)**  **Homework:** Ask students to apply the input-process-output model to actions they complete regularly. **(Activity 6.2.3)** | Week 6  Lesson 2 activities | Linked to Page 9 |

|  |
| --- |
| **Week 7** |

| **Lesson** | **Spec ref** | **Lesson summary** | **Lesson content** | **Lesson resources** | **Chapter 1 Pages** |
| --- | --- | --- | --- | --- | --- |
| 1 | 3.1.1  3.1.2  3.1.3 | **Binary:** introduction and representation of unsigned integers | Explain to students that they will now be learning how different types of data are represented in a computer and the limitations of these representations. Remind students of the different data types they encountered when programming in Python, i.e. ints, floats, bools and strings. Ask them to identify other types of data that are handled by computers, e.g. images, video, sound, light, temperature, program instructions. Emphasise that in a computer all these different types of data are represented as a series of numbers. (Note: some of this may be familiar to students, emphasise going into significant more detail for GCSE.)  Use the YouTube video ‘What are binary numbers?’ to introduce the concept of binary numbers: [www.youtube.com/watch?v=kcTwu6TFZ08](http://www.youtube.com/watch?v=kcTwu6TFZ08).  Explain that a computer’s electronic circuits consist of hundreds of thousands of tiny switches, called transistors. Like a light bulb, a transistor can be either on or off. When a transistor is on it produces a ‘1’ and when it is switched off it produces a ‘0’. This is why computers represent all data as strings of 0s and 1s. Transistors can switch on and off thousands of times per second, which enables computers to process huge amounts of data very quickly.  Explain that a single binary digit is known as a bit and can have one of two values (0 or 1). This means it can represent two options, e.g. true/false, yes/no, black/white. By using groups of bits a computer is able to represent a wider range of values, e.g. with 3 bits a computer can represent eight different pets (000 = cat, 001 = dog, 010 = rabbit, 011 = goldfish, 100 = parrot, 101 = hamster, 110 = guinea pig, 111 = tortoise). If someone has a more unusual pet, such as a snake, another bit would be needed. With 4 bits the computer could represent 16 different pets. A group of four bits is known as a nibble and a group of 8 bits is known as a byte.  Display the binary code 01101011 on the board and explain that this code represents the letter k, the number 107, a shade of grey and potentially lots of other things besides. Emphasise that the same binary code is used to represent many different things. Ask students how they think computers know which type of data a particular binary code represents – make sure they understand that it all comes down to what the computer has been programmed to do. **(Activity 7.1.1)**  Use **Autumn Year 10 Binary slides** to explain that the binary number system that computers use has just two digits, 0 and 1. Explain that, just as with our denary number system, the position of a digit in a binary number is significant. The further to the left it is the more it is worth, and the most significant bit is the furthest left. Introduce and practise binary to denary conversion and vice versa. **(Activities 7.1.2 and 7.1.3)**  **Homework:** Ask students to watch the YouTube video ‘Binary numbers in 60 seconds’: [www.youtube.com/watch?v=qdFmSlFojIw](http://www.youtube.com/watch?v=qdFmSlFojIw), and to play the Cisco Binary Number Game: [www.forums.cisco.com/CertCom/game/binary\_game\_page.htm](http://forums.cisco.com/CertCom/game/binary_game_page.htm) to practise converting between binary and denary and vice versa. | Week 7  Lesson 1 activities  YouTube video: ‘What are binary numbers?’  Autumn Year 10 slides  YouTube video: ‘Binary numbers in 60 seconds’  Cisco Binary Number Game | n/a |
| 2 | 2.1.2 | **Techniques to improve code readability and understanding** | Share with students the style conventions for producing readable code. Show them how to improve the readability of their code. **(Activity 7.2.1)**  Give students the opportunity to improve the readability of some of the programs they have already written. **(Activities 7.2.2 and 7.2.3)**  **Homework:** Ask students to design a poster to illustrate how to use the selection constructs with hints and tips for writing Python programs. | Week 7  Lesson 2 activities | n/a |

|  |
| --- |
| **Week 8** |

| **Lesson** | **Spec ref** | **Lesson summary** | **Lesson content** | **Lesson resources** | **Chapter 1 Pages** |
| --- | --- | --- | --- | --- | --- |
| 1 | 3.1.1  3.1.2  3.1.3  1.2.1 | **Binary:** consolidate previous lesson, computational thinking | Start off the lesson by playing Binary Bingo: [www.community.computingatschool.org.uk/resources/1392](http://community.computingatschool.org.uk/resources/1392)  Recap on what students have learnt so far about binary. **(Activities 8.1.1–8.1.3)**  Remind students that they have to be able to convert values from 0-255.  Working in pairs, set students the challenge of writing a program that converts 4-bit binary numbers into denary. **(Activity 8.1.4)**  Computational thinking activity: get students to plan how they would create an animated PowerPoint presentation that shows binary counting from 0 to 1111.  **Homework:** Ask students to create the animated PowerPoint presentation they planned above. | Binary Bingo Activity  Week 8  Lesson 1 activities | n/a |
| 2 | 1.1.1  1.1.2  2.5.3 | **Flowcharts and Boolean operators** | Introduce flowcharts and the flowchart symbols. **(Activity 8.2.1)**  Show the YouTube video ‘*The Big Bang Theory* – The Friendship Algorithm’: [www.youtube.com/watch?v=k0xgjUhEG3U](http://www.youtube.com/watch?v=k0xgjUhEG3U)  Demonstrate how flowcharts can represent if (decision) problems. **(Activity 8.2.2)**  Show students how to write program code from a flowchart. **(Activities 8.2.3 and 8.2.4)**  Explain the concept of AND, OR and NOT (Boolean operators) to combine the results of two condition statements.  condition\_one **and** condition\_two  This condition is true only if condition\_one and condition\_two are both true.  condition\_one **or** condition\_two  This condition is true if condition\_one or condition\_two is true.  **not** condition  This creates a new condition, which is the reverse of the original, setting true to false and false to true.  **Homework:** Ask students to draw flowchart for their activities that evening. How much iteration is involved? (Explain the term iteration.) | Week 8  Lesson 2 activities  *The Big Bang Theory* YouTube video | Pages 4-5, 16-17 |

|  |
| --- |
| **Week 9** |

| **Lesson** | **Spec ref** | **Lesson summary** | **Lesson content** | **Lesson resources** | **Chapter 1 Pages** |
| --- | --- | --- | --- | --- | --- |
| 1 | 3.1.2  3.1.3  3.1.4 | **Binary:** representation of unsigned integers, binary arithmetic (addition) | Use **Autumn Year 10 Binary Arithmetic slides** to explain how to add up in binary and to introduce the problem of overflow, which occurs when the result of a calculation is too big to fit into the number of bits available (there will be more on overflow later). Give students the opportunity to practise binary addition. **(Activity 9.1.1)**  Go through arithmetic and logical shifts in the slides. Students should then complete **Activity 9.1.2.**  **Homework:** Students should complete **(Activity 9.1.3)** | Week 9  Lesson 1 activities  Autumn Year 10 slides | n/a |
| 2 | 2.5.3  4.3.1 | **Boolean operators, logic, and truth tables** | Give students the opportunity to explore Boolean operators. **(Activities 9.2.1–9.2.3)**  Show students how to write programs that use AND, OR, NOT conditions.  Introduce truth tables as a way of listing all the combinations of true and false. Work with the class to complete the truth tables for AND, OR and NOT. **(Activity 9.2.4)**  Give students the opportunity to write programs that use logical operators in condition statements. **(Activity 9.2.5)**  Demonstrate how IDLE’s colour coding can be useful in making sure the syntax is correct. **(Activity 9.2.6)**  **Homework:** Ask students to write a logical operator question to test other students. They must be able to explain the answer. | Week 9  Lesson 2 activities | Pages 16-17 |

|  |
| --- |
| **Week 10** |

| **Lesson** | **Spec ref** | **Lesson summary** | **Lesson content** | **Lesson resources** | **Chapter 1 Pages** |
| --- | --- | --- | --- | --- | --- |
| 1 | 3.1.2  3.1.3  3.2.4 (binary numbers) | **Binary:** representation ofsigned and unsigned integers | Point out to students that they already know how unsigned integers are represented in binary and how to do simple binary arithmetic. Explain that in this lesson they will be learning how binary can represent other types of numbers, i.e. negative numbers.  Use **Autumn Year 10 Negative Numbers** **slides** to explain the two methods of representing negative numbers, i.e. sign and magnitude (where the most significant bit indicates whether the number is negative [0] or positive [1]) and two’s complement (where the most significant bit is a negative number). Explain the problems associated with sign and magnitude representation and how two’s complement overcomes them. **(Activities 10.1.1 and 10.1.2)**  Go through the answers, then students should complete **Activities 10.1.3 and 10.1.4**.  **Homework:** Students should write their own explanation of how to represent negative numbers. | Week 10  Lesson 1 activities  Autumn Year 10 slides | n/a |
| 2 | 1.1.1  1.1.2  2.1.1 | **Pseudo-code** | Show a program written in pseudo-code and explain that pseudo-code means ‘mock’ or ‘pretend’ code and that it is way of describing/writing a program without using a specific programming language.  Explain the reasons why flowcharts and pseudo-code are used to write algorithms.  Show them the list of the pseudo-code commands in *Appendix 1* of the specification. Point out that pseudo-code uses control structures and keywords similar to those found in programming languages.  Demonstrate how to translate programs written in pseudo-code into Python code. **(Activities 10.2.1 and 10.2.2).**  Talk through **Activity 10.2.3** and ask students to complete this task in the given order.  **Homework:** Students should create a table and note the advantages and disadvantages of using flowcharts and pseudo-code. | Week 10  Lesson 2 activities  Specification *Appendix 1* | Pages 8-19 |

|  |
| --- |
| **Week 11** |

| **Lesson** | **Spec ref** | **Lesson summary** | **Lesson content** | **Lesson resources** | **Chapter 1 Pages** |
| --- | --- | --- | --- | --- | --- |
| 1 | 3.1.1  3.1.5 | **Hexadecimal:** what is it and why is it used? Converting hex-binary-hex | Refresh data representation – that 0s and 1s can represent numbers, text, graphics, sound (more later). Write a long series of 0s and 1s on the board – what is it? Cannot know without context.  Explain that early programmers had to program in just 0s and 1s. Discuss the problems with that (errors, time to code, etc.).  Using the **Autumn Year 10 Hexadecimal Number slides**, introduce hexadecimal code. Then after each explanation on how to convert hex-binary-hex complete appropriate activity **(Activities 11.1.1 and 11.1.2)**  Explain the use of hexadecimal code to represent binary.  Students should now work through **Activity 11.1.3**.  **Homework:** Find the meanings of the words binary, denary and hexadecimal. | Week 11 Lesson 1 Activities  Autumn Year 10 slides | n/a |
| 2 | 2.3.3 | **Data types:** string manipulation and string methods | Review the data type string and remind students that a string can be assigned to a variable. Review how quotes are used to delimit strings.  Explain that a string is a data structure that stores a set of characters. Discuss how much data comes in the form of strings (e.g. emails, Twitter).  Explain how the position of each character in a string is called the index, which starts from 0. **(Activity 11.2.1)**  Review how strings can be assigned to variables and demonstrate how they can be concatenated. **(Activity 11.2.2)**  Demonstrate how to slice strings using square brackets []. **(Activity 11.2.3)**  Give students the opportunity to experiment with slicing string. **(Activity 11.2.4)**  Demonstrate how the len() function can be used to find the length of a string.  Demonstrate how to write programs that manipulate strings. Explain that strings are immutable, i.e. items in a string list cannot be changed (unlike lists).  Review string methods including the .format method to produce fancier formatting output. **(Activities 11.2.5 and 11.2.6)**  **Homework:** Set students a programming challenge requiring string manipulation. Programs can be written in pseudo-code or program code. | Week 11  Lesson 2 activities | n/a |

|  |
| --- |
| **Week 12** |

| **Lesson** | **Spec ref** | **Lesson summary** | **Lesson content** | **Lesson resources** | **Chapter 1 Pages** |
| --- | --- | --- | --- | --- | --- |
| 1 | 4.5.1 | **High and low level programming languages** | Explain to students that in the next two lessons they will be learning about how Programming languages run on computers by being translated into 0s and 1s. Use slides from low and high level languages slides.  Show students images of early computers, e.g. Colossus, Manchester Baby, ENIAC. Explain how they were programmed in machine code.  Go through the **Autumn Year 10 Low and High level Languages slides**. Students complete **Activities 12.1.1 and 12.1.2.**  **Homework:** Complete **Activity 12.1.3** to gather data about the popularity of current programming languages. | Week 12 Lesson 1 Activities  Autumn Year 10 slides | n/a |
| 2 | 2.3.2 | **Data structures:** lists(arrays) using Python docs | Explain that a list (known as an array in some languages) is another data structure. It lets you store a list of things. Show how lists use square brackets [].  Demonstrate how to create a list and how lists are indexed. Demonstrate how to display individual elements and a range of elements in a list.  Demonstrate how to add items to a list, delete items from a list and how to check that an item is in a list. **(Activities 12.2.1 and 12.2.2)**  Explain that all data types (strings, integers, floats, Boolean and other lists) can be stored in a list.  Give students the opportunity to write programs that use lists. **(Activity 12.2.3)**  Explain that lists are mutable, i.e. the items in them can be changed. Demonstrate the commands to change values in a list. (Tuples are like lists but are immutable; they cannot be changed).  Demonstrate to students how to use the Python docs tutorial to find out more about handling lists.  **Homework:** Ask students to use the Python docs tutorial to find out **three** more facts about lists to share with the class. **(Activity 12.2.4)** | Week 12  Lesson 2 activities | Pages 29-30 |

|  |
| --- |
| **Week 13** |

| **Lesson** | **Spec ref** | **Lesson summary** | **Lesson content** | **Lesson resources** | **Chapter 1 Pages** |
| --- | --- | --- | --- | --- | --- |
| 1 | 4.5.2  2.1.1  2.1.2 | **Translating programming languages** | Ask the class if they know why they don’t have to learn to program in machine code. Explain to students that it is thanks to assemblers and compilers that they can program in a high level language like Python and yet the CPU can still run the program in machine code.  Using the **Autumn Year 10 Translating Programming Languages slides**, explain that machine code is a sequence of binary digits that represent the program in such a way as the processor can run it.  An assembler is a program that takes a program in assembly language and translates it into machine code so that the processor can run it.  A compiler is a program that takes a program written in a high level language such a Python or C and translates it into assembly language or into machine code that a particular processor can understand and run.  Students should summarise this information by labelling a diagram showing how high level code is translated into binary (machine) code. **(Activity 13.1.1)**  Go through any issues identified in the homework set for Week 12 Lesson 2.  Students should now complete the program in **Activity 13.1.2,** which is a simple translator.  **Homework:** Complete the extension to **Activity 13.1.2**. | Week 13 Lesson 1 Activities  Autumn Year 10 slides | n/a |
| 2 | 2.2.2 | **Repetition construct:** for loops, range function | Explain that loops are an important programming construct that lets a program repeat parts of the program code.  Introduce the concept of repetition as a programming construct by giving the example of iterating through a list and also through a range of values.  Explain how a for loop repeats (or iterates) over the items in a list or a string. Describe the syntax of the for loop. Explain the significance of the indent, i.e. that the loop ends when the block of code is no longer indented. **(Activities 13.2.1 and 13.2.2)**  Demonstrate how the built in range() function allow a range of values to be generated.  Give students the opportunity to write more programs that use for loops. **(Activities 13.2.3 and 13.2.4)**  **Homework:** Congratulate the students on making a good start with loops/iteration. Explain that they will do more of these types of program in the next lesson, and to think carefully about what they have learned and make notes on anything they are still unsure about. | Week 13 Lesson 2 Activities | Pages 15-19 |

|  |
| --- |
| **Week 14** |

| **Lesson** | **Spec ref** | **Lesson summary** | **Lesson content** | **Lesson resources** | **Chapter 1 Pages** |
| --- | --- | --- | --- | --- | --- |
| 1 | 4.2.1 | **Hardware:** the function of internal components | Start the lesson by recapping the definition of hardware. Explain that this lesson will look at the hardware components used in a digital computer. Explain that we will focus on the hardware components in a desktop computer because they are larger and easier to see but that similar components exist in all other digital computers, such as mobile phones.  Explain that computers are made up of millions of electronic components, changing state hundreds of millions of times a second. These components are reliably processing data at phenomenal speeds. We going to explore how the digital computer works and the first step is to identify the hardware components that make up a digital computer.  As a class watch these videos which identify the parts of a computer and describe their functions: [www.gcflearnfree.org/computerbasics/7](http://www.gcflearnfree.org/computerbasics/7) and [www.youtube.com/watch?v=yRmPTbGBqVI](http://www.youtube.com/watch?v=yRmPTbGBqVI)  Demonstrate an actual desktop computer. Take the case off and ask students to identify the processor (CPU), RAM memory, secondary storage (hard disk drive), CD and DVD drives, graphics cards and network card. Explain that the components in a digital computer need electricity, which is supplied by batteries or from a mains supply.  Ask students to name the hardware components from the functions given. **(Activity 14.1.1)**  Explain that the ways in which these hardware components relate to each other in a computer is called the computer architecture. The choice of hardware components affects the computer’s speed and power, as well as how much the computer costs and how it can be used.  Explain that the CPU is connected to the other components by wiring known as buses. There are three types of buses:   |  |  | | --- | --- | | Data bus | Transfers binary data from place to place | | Control bus | Sends and receives signals that control the CPU and other components | | Address bus | Carries the binary address of memory locations |   Show how the data, address and control buses send electronic signals between the CPU and memory, and also to other devices such as the hard disk drive: [www.en.wikipedia.org/wiki/File:Computer\_system\_bus.svg](http://en.wikipedia.org/wiki/File:Computer_system_bus.svg). Show the students a bus either as cable or tracks in a printed circuit board. Get students to complete a diagram representing how the hardware components are connected using buses and to summarise what the different buses do. **(Activities 14.1.2 and 14.1.3)**  If you have time, introduce the students to the Raspberry Pi® as another example of a digital computer: [www.youtube.com/watch?v=Jj4pjfU\_-jo](http://www.youtube.com/watch?v=Jj4pjfU_-jo)  Explain that the Raspberry Pi® is an example of a cheap digital computer and includes some of the same hardware components we have just identified in the PC. Show the students a Raspberry Pi® (or a picture of a Raspberry Pi®). Explain that the CPU is an ARM processor and is a ‘System on a Chip’ where the memory is placed physically above the CPU. Get students to label the parts of the Raspberry Pi®. **(Activity 14.1.4)**  If you have access to Raspberry Pi®s, you could get students to insert the SD card and power cable.  **Homework:** Search for an image that shows the components that make up a different type of digital computer (e.g. a tablet computer or a smart phone). Label the main components that make up that computer including CPU (processor), memory, power supply, secondary storage, input and output devices.  Compare your labelled image with someone else in your class who chose a different digital computer. What are the similarities and differences? | Week 14 Lesson 1 Activities    ‘Inside a desktop computer’ GCF LearnFree.org video  ‘How to identify the components inside your computer’ YouTube video  A desktop computer and a bus (either cable or PCB)  ‘Computer System Bus’: Wikipedia  ‘Raspberry Pi – Tutorial 1 – An introduction’ YouTube video | n/a |
| 2 | 2.2.2 | **Repetition construct:** for loops, continued | To consolidate programming in for loops, students should complete **Activities 14.2.1–14.2.3**.  **Homework:** Set students a programming challenge requiring the use of a for loop. Programs can be written in structured English, program code or flowcharts. | Week 14 Lesson 2 Activities | Pages 15-19 | |

|  |
| --- |
| **Week 15** |

| **Lesson** | **Spec ref** | **Lesson summary** | **Lesson content** | **Lesson resources** | **Chapter 1 Pages** | |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 4.4.1 | **Operating systems:** files, hardware and the user interface. | Begin the lesson by asking students to list as many operating systems as they can think of **(Activity 15.1.1)** then ask students to contribute to a class list. As a class, brainstorm what these operating systems might have in common.  As a class, work on a definition of an operating system: a collection of programs that manages the hardware and software of a computer system, including sharing the computer’s time, memory, hardware and other resources.  Ask students to think about what it might be like to use a computer without an operating system. What would happen? Explain that, without an operating system, the hardware cannot function and the user cannot use the computer (even though the user rarely interacts with the operating system itself because there are applications between the operating system and the user). The user is removed from the computer’s hardware by the operating system and application software.  Ask students to complete the diagram to show how the operating system provides an interface between the computer’s hardware and the application software used by the user. **(Activity 15.1.2)**  Demonstrate different operating systems running on laptops, tablets and PCs. Get the class to get out their mobile phones and identify the operating systems they are running. Make a class list of the different operating systems that are currently running in the classroom on the various digital devices present.  Explain that in many computers the programs that run the operating system are stored on the hard disk. When computer starts up, the operating system is loaded from the hard disk into memory.  Note: It may be necessary to demonstrate some of these functions on computers that are not connected to the school network as some of these functions (such as control/alt/delete on Windows operating systems, see [www.en.wikipedia.org/wiki/Windows\_Task\_Manager](http://en.wikipedia.org/wiki/Windows_Task_Manager)) are restricted on school networks to maintain the integrity of the network. This makes the teaching point that operating systems can be used to manage the users.  All operating systems have the following main functions:   1. **File management**   The operating system manages the storage of files and directories (folders). The operating system stores files in a hierarchy of directories. Demonstrate a hierarchy of directories using an operating system of your choice (e.g. Raspbian on the Raspberry Pi®, Windows), then give students a chance to experiment. (Note: students may need a non-networked machine for this.) **(Activity 15.1.3)**   1. **Hardware management**    1. Input /output management   The operating system manages data transfers between the CPU and the input and output devices, such as the keyboard and printers. The software that allows this data transfer may already be part of the operating system, or the user might need to install a driver to handle input and output from a particular device.   * 1. Resource management   A resource is any part of the computer system. Examples include a printer, memory, hard disk drive or processor time. The operating system allocates the computer system’s available hardware to the programs running (processes) at any one time. While a program is running, the data and instructions are stored in the main memory and it is the operating system that allocates appropriate memory to each program. The operating system also ensures that instructions and data do not interfere with each other.   1. **User management**   The operating system also manages the user(s) of the computer system, providing users with an interface to the computer system, privileges to perform certain tasks and security features to protect the users and the system. Give students a chance to see how the operating system keeps track of various users. (Note: students may need a non-networked machine for this.) **(Activity 15.1.4)**  The operating system provides the interface that allows users to use the computer. This can be either a Graphic User Interface (GUI) such as Windows or Android or a command line interpreter (CLI). Demo both and briefly point out some of the main differences. Explain that early computers used a CLI interface but many now also include a GUI interface, which includes the use of icons and menus. Demonstrate how to display the Windows CLI by using the command prompt (start/command prompt in Windows 7). Use ‘help’ to display a list of the commands available.  Students should understand that some operating systems, such as those found on mobile phones, are single user operating systems, while other operating systems are multi-user which allows more than one user to ‘log on’ using a username and a password.  Many operating systems have an administrator or ‘superuser’ who has full privileges to alter any aspects of the computer system including adding new user accounts. Users can be granted different privileges, such as the right to access particular files or directories. Restricting users’ access is one of the ways in which system administrators protect the integrity of the computers and network.   1. **Process management**   Tell students this is a very important function in an operating lesson and will be covered in the next lesson.  Also note that networks need to be accommodated in a network operating system and this will be covered in the Networks topic.  **Homework:** Ask students to do some additional research into the functions of operating systems and then to design a poster or write a summary sheet explaining the six functions. This article would be a useful starting point: [www.computer.howstuffworks.com/operating-system.htm](http://computer.howstuffworks.com/operating-system.htm) | Week 15 Lesson 1 Activities | n/a | |
| 2 | 1.1.2  2.2.2 | **Repetition construct**: while loop, flowcharts | Explain that the while loop is another repetition construct that tests a condition and executes the commands while that condition is true.  Explain that the condition needs to change at some point or else the program will run forever and that while loops are pre-conditioned loops, i.e. they test the condition at the beginning of the loop. (It may be useful to recap on conditions and Boolean [true/false] values.)  Describe how while loops often have a ‘sentry variable’ which is tested in the condition. These sentry variables are usually initialised before the loop starts to make sure there is at least one ‘pass’ through the loop. Explain the syntax of the while loop. **(Activities 15.2.1–15.2.3)**  Give students the opportunity to write programs that use while loops.  Explain how the break command can be used to ‘breakout’ of a loop.  Explain how a while loop would be represented in a flowchart. **(Activity 15.2.4)**  **Homework:** Set students a programming challenge requiring the use of a while loop. Programs must be written in program code. | Week 15 Lesson 2 Activities | | Pages 14, 20, 24, 26, 27 |

|  |
| --- |
| **Week 16** |

| **Lesson** | **Spec ref** | **Lesson summary** | **Lesson content** | **Lesson resources** | **Chapter 1 Pages** |
| --- | --- | --- | --- | --- | --- |
| 1 | 4.4.1 | **Operating Systems:** processes | Refresh the functions of the previous lesson on operating systems. Now start the lesson by demonstrating the task manager on a Windows machine (or something similar) and ask pupils to look at ‘processes’. Discuss what is meant by a process. Explain that a computer program is a collection of instructions. A process is the execution of those instructions.  Get pupils to open up a few applications (web browser, word processor, spreadsheet, graphics software) and to see how each application runs as a process in the task manager. **(Activity 16.1.1)**  Explain that all application software such as a web browser, word processor, spreadsheet or apps are just computer programs and run as processes on a computer. Find the executable files for the application software on your computers. On a Windows operating system these will have a file type of .exe and are often located in the Program Files folder.  The operating system:   * gives each process a unique process ID * allocates space in memory to hold the instructions and data for a program * assigns priority which determines the order in which processes will run * links processes with a user.   Show how the operating system allows you to list the processes that are running and also to end (or kill) processes.  Display the processes tab in the task manager window and ask pupils how one CPU can run more than one process at the same time.  Explain that multi-processing operating systems allow many processes to be run at the same. The CPU can only execute one task at a time. To perform multi-processing it switches quickly between the processes. It is the operating system that schedules which process to run. A range of different algorithms can be used to schedule the processes to run.  Run the unplugged activity to simulate processes being scheduled to run. **(Activity 16.1.2)** You can discuss the idea of ‘First In First Out’ (FIFO), which is being used for the queue. They will come across this concept again in a later block.  **Homework:** Take a screenshot of the processes running on a computer and annotate it to show the name of the process, the process id, the priority (if shown) and the user. Now find out how to list the processes running on your mobile phone. | Week 16 Lesson 1 Activities | n/a |
| 2 | 1.1.1  2.6.1 | **Pseudo-code and built-in subprograms:** random function | Explain that Python has modules that contain pre-written functions that we can use. Demonstrate how to import and run modules.  Explain that random numbers are useful and used in many programs especially for writing games. Demonstrate the use of the randint() function to generate random numbers.  Give students the opportunity to write programs that uses the randint() function. **(Activities 16.2.1–16.2.3)**  **Homework:** Ask students to write a program that ‘flips a coin’ and randomly produces the answer ‘heads’ or ‘tails’.  **Extension:** Modify the program so that it prints the number of ‘heads’ and ‘tails’ thrown after ten goes. | Week 16 Lesson 2 Activities | Pages 17, 41-43 |

# Suggestions for teaching this content

* Provide code as images (print screen) to force the students to type in the program code rather than just use copy and paste. They will learn more by making mistakes and having to correct them.
* Use two projectors on two screens. This allows you to display code on one screen and information or questions on the other. Use a plain wall or card on wall to make the screens.
* Divide the class into groups who work together to solve problems. Explain that they must discuss problems within the group before they ask for help. Introduce the concept of coaching other students within their groups to ensure everyone finishes within the set time. Have competitions between groups.
* Get students to ‘think, pair, share’ with problems (before asking the teacher a question) as well as using this to answer questions. Think about the problem by themselves, discuss the problem in a pair, then share it with a group around them.

## Starters and plenaries

Ideas for starters or plenaries which give students the opportunity to review computational thinking skills, reflect on their learning and build up specialist technical vocabulary:

* Give students some code and ask them what it does.
* Give students a flowchart and ask them to produce the code.
* Give students the pseudo-code and ask them to produce the code.
* Give students some code and ask them to depict as a flowchart.
* Give students some pseudo-code and ask them to depict as a flowchart.
* Give students a written description and ask them to depict as a flowchart.
* Give students a written description and ask them to produce the code.
* Give students a list of commands jumbled up and ask them to put them in the right order.
* Give students some code and ask them to find the syntax errors.
* Give students some code and ask them to produce a test plan.
* Give students some input data and ask for appropriate test data (valid, invalid, extreme).
* Ask students to summarise the lesson in 140 characters.
* Ask students to identify three ways in which a program could be improved.
* Who can find out about a topic first using Python help? (Do as a group.)
* Play videos introducing a particular aspect of Python programming.
* Ask students to produce five questions and answers on a topic.
* Online polls for formative assessment of topics.
* Peer-review of a program – how to give and receive constructive feedback.
* Self-assessment of a program.
* Invite students to ‘show and tell’ a program – what are the good features and the bad features? How could it be improved?
* What would happen if I made this change to the program?
* ‘Role play’ being a program, e.g. students act as manual debuggers and give values of variables at certain break points in a program.
* Use games, e.g. hangman with technical terms.
* Give a definition of a technical term and ask students to say what the term is.

## Suggestions for additional homework questions or extension exercises

* Ask students to summarise a topic in 140 characters or less.
* Ask students to produce a program in code/pseudo-code/flowchart given as a written description (or any combination of this).
* Ask students to write a written description of a program given as program code.
* Ask students to make a ‘topic in plain English’ video (for an example see: [www.youtube.com/watch?v=VumBNb6gcBk](http://www.youtube.com/watch?v=VumBNb6gcBk))
* Ask students to write a quiz (with answers) on a topic.
* Ask students to find the definitions of technical terms.
* Ask students to create a digital or classroom poster to explain a technical term or concept.
* Challenge students to find the five best websites to help with a particular topic.
* Ask students to search online to find the answers to some Python programming problem.
* Ask students to assess a program using an assessment checklist (give all the class the same program).

## Self-assessment of programming

By the end of the term, students should be able to assess their programs by asking the following questions:

* Does the program work? Are there any errors? How successfully does the program meet the requirements?
* How does this program work? What programming constructs have been used?
* Is the program readable? Have comments been included to explain the program? Would someone else understand the program?
* What are the good features and bad features of the program? How could it be improved?
* Did you learn anything new about programming from writing this program?
* Did you coach another student to help them solve this problem? If so, what did you learn from helping others?
* Encourage students to use the correct technical terms when discussing their programs.

General resources

* *BCS Glossary of Computing and ICT*, 13th edition (ISBN 9781780171500)
* A range of articles on teaching coding: [www.edsurge.com/guide/teaching-kids-to-code](https://www.edsurge.com/guide/teaching-kids-to-code)

Software

* Audacity (free open source, cross-platform software for recording and editing sounds) available to download from [www.audacity.sourceforge.net](http://audacity.sourceforge.net)
* Python (free open source software) available to download from [www.python.org/getit/](http://www.python.org/getit/)

Python resources

* Official Python documentation (also available through help in IDLE): [www.docs.python.org/3/](http://docs.python.org/3/)
* Python summer school from Anglia Ruskin University is an excellent resource with videos and programming challenges: <http://www.pythonschool.net/>
* Python code for kids is a clearly written summary of the Python language written in accessible language: [www.pythondictionary.code-it.co.uk/](http://pythondictionary.code-it.co.uk/)
* ‘Python in 10 minutes’ is a quick run through of the basic concepts: [www.korokithakis.net/tutorials/python/](http://www.korokithakis.net/tutorials/python/)
* ‘Quintin Cutts – Too much doing, not enough understanding’ is a 20-minute video containing useful ideas and concepts on how to teach programming: [www.youtube.com/watch?v=Pim4aYfiZiY](http://www.youtube.com/watch?v=Pim4aYfiZiY)

Free online books

* *Think Python How to Think Like a Computer Scientist* is a free online book with programming challenges at the end of each chapter: [www.greenteapress.com/thinkpython/thinkpython.pdf](http://www.greenteapress.com/thinkpython/thinkpython.pdf)
* *A Byte of Python* is an excellent online book, though it does not use IDLE as the editor: [www.swaroopch.com/notes/Python/](http://swaroopch.com/notes/Python/)

Books

* *Python Programming for the Absolute Beginner*, M. Dawson (published Course Technology 2010) (ISBN 9781435455009) is an excellent book with clear explanations of each bit of code together with free downloads and example games built into each chapter.
* *Python for Kids: A Playful Introduction to Programming*, J. Briggs (published No Starch Press 2013) (ISBN 9781593274078)