Pearson Edexcel

GCSE (9-1)

Computer Science

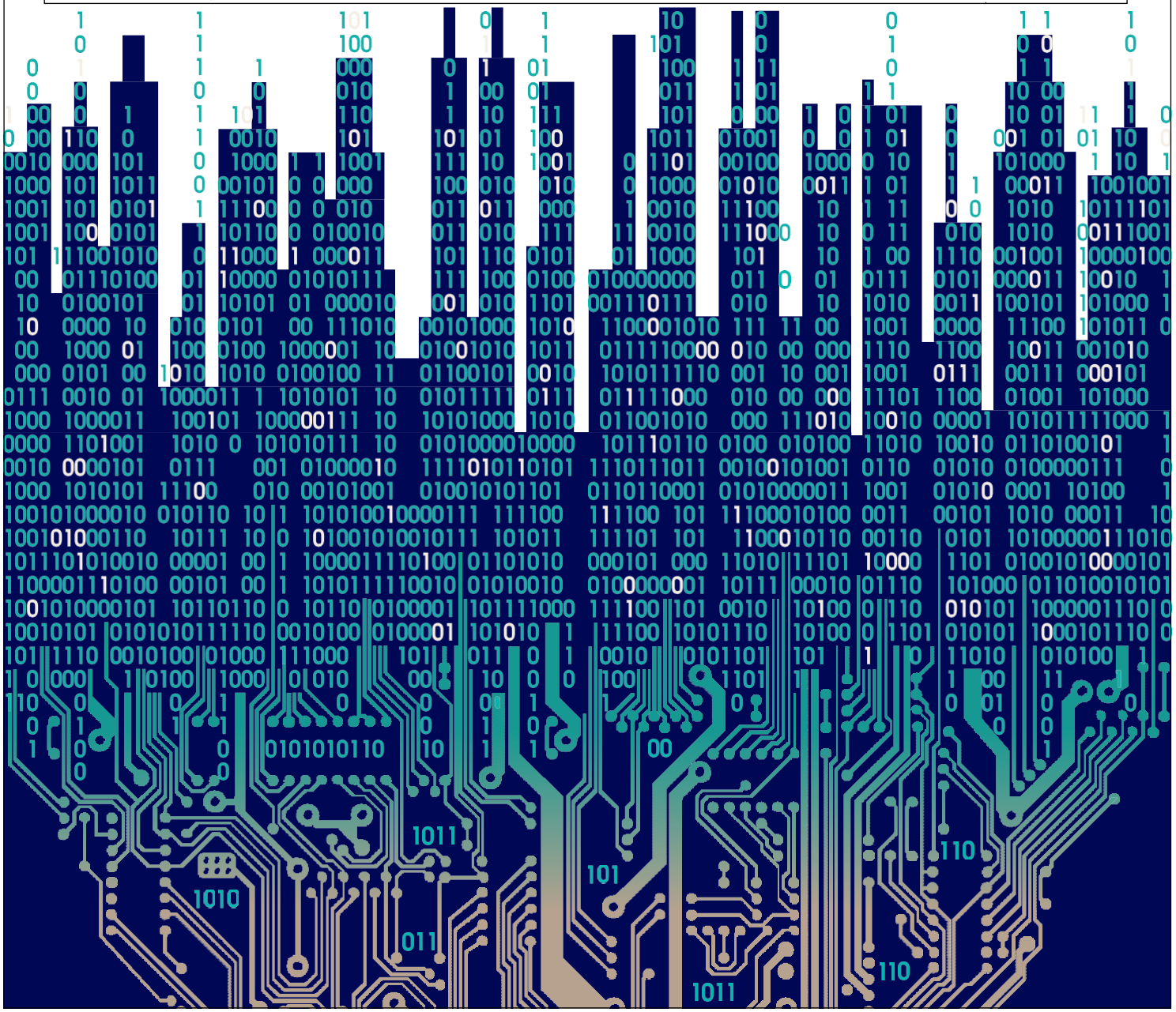
Getting Started Guide

Pearson Edexcel Level 1/Level 2 GCSE (9-1) in Computer Science (1CP2)

First teaching September 2020

First certification 2022

Issue 3



Edexcel, BTEC and LCCI qualifications

Edexcel, BTEC and LCCI qualifications are awarded by Pearson, the UK's largest awarding body offering academic and vocational qualifications that are globally recognised and benchmarked. For further information, please visit our qualifications website at qualifications.pearson.com. Alternatively, you can get in touch with us using the details onour contact us page at qualifications.pearson.com/contactus

About Pearson

Pearson is the world's leading learning company, with 35,000 employees in more than 70 countries working to help people of all ages to make measurable progress in their livesthrough learning. We put the learner at the centre of everything we do, because whereverlearning flourishes, so do people. Find out more about how we can help you and your learners at qualifications.pearson.com

References to third party material made in this document are made in good faith. Pearson does not endorse, approve or accept responsibility for the content of materials, which may be subject to change, or any opinions expressed therein. (Materials may include textbooks,journals, magazines and other publications and websites.)

All information in this specification is correct at the time of going to publication.

Getting Started:
GCSE Computer Science 2020

Contents

Getting Started with GCSE ComputerScience

1. Purpose of this guide

The purpose of this Getting Started Guide is to provide you with additional information about the qualification, and clarify what you need to cover for each of the points listed in the specification.

2. Key features

- ✓ Engaging content derived from the DfE's prescribed subject content – making it simple to switch from another Exam Board's specification.
- ✓ On-screen programming exam in which students use Python the Integrated Development Environment (IDE) of their choice
- ✓ A defined programming language subset (PLS) – specifying exactly what constructs, syntax and libraries students need to be able to use.
- ✓ Focus on teaching programming fundamentals and transferrable skills rather than Python-specific constructs – making it easier for students to learn other programming languages in the Sixth Form and beyond.
- ✓ No need for students to learn a separate pseudocode language – freeing up time for them to concentrate on mastering a real programming language.
- ✓ Comprehensive support – helping you to plan and implement the course successfully.

3. Content overview

The subject content is divided into six topics. Topics 1 – 5 are assessed in Paper 01 and Topic 6 is assessed in Paper 02.

What is included in each topic

1. **Computational thinking**
   - benefits of using decomposition, abstraction and subprograms
   - working with algorithms, expressed as flowcharts, written descriptions and in program code
   - determining the correct output of an algorithm for a given set of data
   - using a trace table to perform a dry-run of an algorithm
   - types of errors found in algorithms and programs
   - correcting logical errors in algorithms
   - how standard sorting and searching algorithms work
   - evaluating the fitness for purpose of an algorithm
   - constructing truth tables

2. **Data**
   - why computers use binary
   - converting between unsigned and signed denary and 8-bit binary numbers
   - adding and shifting binary numbers, overflow
   - converting between hexadecimal and binary, ehy hexadecimal is used
   - binary representation of text, images and sound
   - using binary multiples, and constructing expressions to calculate file sizes and data capacity requirements
   - the need for and different methods of data compression

3. **Computers**

- the stored program concept, and the role of memory, the CPU and buses in the fetch-decode-execute cycle
- the role of secondary storage, and how data is stored on different types of media
- the purpose of an operating system and utility software
- robust software, and methods of identifying software vulnerabilities
- low-level and high-level programming languages
- how interpreters and compilers translate high-level code

4. **Networks**

- reasons for connecting computers in a network, and types of network
- how the internet is structured
- characteristics of wired and wireless connectivity
- constructing expressions involving file size, time, and transmission rate measured in bits per second
- network protocols and the TCP/IP stack
- common network topologies
- network security, and ways of identifying vulnerabilities and protecting networks

5. **Issues and impact**

- environmental issues associated with the use of digital devices
- ethical and legal issues associated with the collection and use of personal data
- ethical and legal issues associated with the use of AI, machine learning and robotics
- methods of intellectual property protection for computer systems and software
- threats to digital systems and data posed by malware and hackers and methods or protection

6. **Problem solving with programming**

- using decomposition and abstraction to understand and solve problems
- identifying the structural components of programs
- converting algorithms into code
- designing, writing and debugging programs
- using techniques to make programs easy to read and maintain
- working with variables and constants, primitive data types, and one-and two-dimensional data structures
- manipulating and formatting strings
- implementing validation and authentication
- responding to user input
- reading from and writing to text files
- using pre-existing and user-defined subprograms
- understanding the difference between and using global and local variables

The Programming Language Subset (PLS)

The PLS is a specific set of Python 3 constructs that students need to understand and be able to use. Python 2 should not be used in the delivery of this qualification.

The constructs contained in the PLS are found in most high-level programming languages and form a good foundation for progression.

While there is nothing to prevent students learning more complex constructs or approaches not included in the PLS, there is no need for them to do so. All problems set in Paper 02 (the onscreen programming exam) can be solved using only the functionalities presented in the PLS. This means that more class time can be used for the teaching of computational thinking and problem solving, rather than the syntax and libraries provided by a programming language.

Students will not be penalised in the assessment for using programming constructs other than those included in the PLS.

The PLS is valid for the lifetime of the qualification and available to download from the Edexcel GCSE Computer Science section of the Pearson website. Should an update be required, the new version will be published no later than 31 January in the year of the examination.

Students will be provided with a hard copy and an electronic version of the PLS to refer to when sitting Paper 02.

The Good Programming Practice Guide (GPPG)

This is a companion guide to the PLS, providing further information about the programming constructs students are expected to understand and use.

4. Assessment overview

The assessment consists of two equally weighted, non-tiered components – one theory paper and one on-screen programming exam.

The design of the papers reflects our commitment to clear wording and structure, helping students to tackle each paper with confidence and demonstrate to us what they have learned.

Gradual ramping of demand across the papers helps students build confidence.

Paper 01: Principles of Computer Science

This is a written exam paper consisting of five questions, and is marked out of 75. Each question consists of multiple parts, and assesses aspects of a single topic of the subject content. The order in which topics appear varies from paper to paper.

Paper 02: Application of Computational Thinking

This is an on-screen programming exam, and is marked out of 75. Students have two hours in which to carry six programming tasks on a computer using Python 3. They may be required to:

- ✓ identify the structural components of a program
- ✓ correct errors in a piece of code
- ✓ choose between alternative lines of code
- ✓ rearrange lines of code
- ✓ follow instructions to complete a program.

Tasks increase in complexity, with the final question on the paper requiring students to design and write a program from scratch.

There are no questions that require a written response.

The benefit of this approach is that students have a realistic programming experience and are able to debug their code as they go along to see if it functions correctly. It also allows examiners to test atypical responses for correctness by running them.

We recognise that an onscreen exam may pose some logistical challenges for schools with large cohorts and/or a limited number of computers. However, our experience to date has demonstrated that schools can successfully facilitate this type of assessment.

Further details concerning the administration of the practical exam, including scheduling multiple sessions to accommodate large cohorts, can be found in the Instructions for the Conduct of the Examination (ICE) document, available to download from the Edexcel GCSE Computer Science section of the Pearson website.

The Programming Language Subset and data files will be provided on the morning of the exam. This will be a combination of code samples (*.py) and comma separated value formatted text files (*.txt) where required.

Practical Programming Statement (PPS)

Centres are required to complete a Practical Programming Statement (PPS) to affirm that students studying GCSE Computer Science have been given timetabled and supported opportunities during their course to engage with practical programming.

The PPS must be completed by a member of the senior leadership team at the centre and submitted to the exam board by 31 May in the year of the examination.

Failure by a centre to provide a completed PPS will be considered malpractice and/or maladministration and followed up accordingly.

5. Subject content amplification

The compulsory content of the GCSE in Computer Science combines theoretical knowledge and understanding of the principles of computer science with practical problem solving and programming skills.

Each of the six topics is divided into a number of sections and each section consists of a set of numbered specification points (SPs). Bracketed lists, such as those in SP 1.2.2, define the breadth/depth of coverage required. These are **not** examples. They specify what must be taught.

As a minimum all the numbered SPs in the content must be taught.

Topic 1: Computational thinking

This topic provides the underpinning theory to support the hands-on practical programming that students will undertake during the course and is best taught alongside Topic 6.

Statement Students should:	**Additional information**
1.1 Decomposition and abstraction	
1.1.1 understand the benefit of using decomposition and abstraction to model aspects of the real world and analyse, understand and solve problems	Students should know the meaning of the terms **decomposition** and **abstraction,** and understand how using these computational thinking techniques makes problems easier to understand and solve. They should be able to: • recognise where these technique are being used in a piece of code. See for example *Paper 01 2206, Q5(a)(iii) & (iv)* • give examples of abstraction, such as the use of subprograms to hide implementation details • abstract common attributes of a group of objects so as to create a general model. See for example *Paper 01 Specimen 1, Q3(g)* • discuss the use of decomposition and abstraction in developing software. Students' understanding of decomposition and abstraction is assessed in Paper 01, and their ability to use these techniques when developing code in Paper 02 (see SP 6.1.1).
1.1.2 understand the benefits of using subprograms	Students should know what a **subprogram** is and understand why subprograms are used. See for example *Paper 01 Specimen 1, Q3(d)*. Although Python treats all subprograms as functions, students should know that a function returns one or

Statement Students should:	Additional information
	more results to the calling code, whereas a procedure does not return a result. Students should gain practical experience of using pre-existing (built-in and library) subprograms and writing their own as they develop their programming skills. Their understanding of the benefits of using subprograms is assessed in Paper 01, and their ability to write and use subprograms in program code in Paper 02 (see SPs 6.6.1, 6.6.2 and 6.6.3).
1.2 Algorithms	
1.2.1 be able to follow and write algorithms (flowcharts, pseudocode, program code) that use sequence, selection, repetition (count-controlled, condition-controlled) and iteration (over every item in a data structure), and input, processing and output to solve problems	Students should know what an **algorithm** is and what algorithms are used for. They should be able to follow and write algorithms represented as flowcharts, as written descriptions and in program code. (In this specification, **pseudocode** is defined as an informal written description of an algorithm.) Students should know that a **flowchart** is a diagrammatic representation of an algorithm, and understand the benefits of using flowcharts for this purpose. The six flowchart symbols that students are expected to recognise and be able to use are listed in Appendix 2 of the specification. Students should be able to: • fill in the gaps of a partially completed flowchart. See for example *Paper 01 SAM, Q5(e)* • draw a flowchart using given components. See for example *Paper 01, Specimen 1, Q1(d)* • draw a flowchart from scratch. See for example *Paper 01 2206, Q5(d).* • In Paper 02, students could be given a flowchart to convert into code. See for example *Paper 01 2306, Q4(c).* In Paper 01 students may be asked to answer questions about an algorithm written in **program code**. See for example *Paper 01 2206, Q5(a)(i) & (ii)*. Students should be familiar with the three basic programming constructs used in algorithms

Statement Students should:	Additional information
	(**sequence**, **selection**, and repetition/iteration) and be able to use them in combination to create solutions. Students should know that **repetition** involves repeatedly executing a block of code until some specified condition is met, and understand the difference between **count-controlled** and **condition-controlled** loops. They should know that in this specification the term **iteration** is used to mean repeatedly executing the same block of code on each element of a data structure, e.g. a string, an array, a numeric range or an open file, until every element has been processed. Students should be able to identify the **input**, **processing** and **output** of an algorithm. See for example *Paper 01 Specimen 3, Q1(a)(i)*. The code provided in *Paper 01 SAM, Q5(d)* uses sequence, selection and iteration constructs to input numbers from a file, process them and output the result. Students' understanding of these structural components/programming constructs, and their ability to recognise and use them in algorithms is assessed in Paper 01. Their ability to recognise and use them in program code is assessed in Paper 02 (see SPs 6.2.1 and 6.2.2).
1.2.2 understand the need for and be able to follow and write algorithms that use variables and constants and one- and two-dimensional data structures (strings, records, arrays)	Students should know what **variables** are and what they are used for. Although Python does not explicitly distinguish between variables and constants, students should know that **constants** are used to store values that do not change during program execution and understand why the use of constants in code is good practice. See for example *Paper 01 SAM, Q5(c)*. Students' should understand the concept of a **data structure**, and be able to handle **one-dimensional** and **two-dimensional** data structures. They should understand how an index is used to identify and access individual elements within a data structure. See for example *Paper 01 2206, 5(a)*, which requires students to work with data stored in a one-dimensional array. Students should know that **strings** are arrays of

Statement Students should:	Additional information
	characters, and be familiar with common **string handling techniques** such as slicing and concatenation. They should understand the difference between an **array** – a sequence of items with the same (homogeneous) data type, and a **record** – a sequence of items with different (heterogeneous) data types. They should know that the items that make up a record are referred to as fields. Students' understanding of constants, variables and data structures and their ability to identify and use them in algorithms is assessed in Paper 01. Their ability to identify and use them in program code is assessed in Paper 02 (see SP 6.2.1).
1.2.3 understand the need for and be able to follow and write algorithms that use arithmetic operators (addition, subtraction, division, multiplication, modulus, integer division, exponentiation), relational operators (equal to, less than, greater than, not equal to, less than or equal to, greater than or equal to) and logical operators (AND,OR, NOT)	Students should be familiar with and be able to use arithmetic, relational and logical (Boolean) operators in algorithms. *Paper 01 Specimen 2, Q2(b)* assesses students' ability to recognise operation symbols and categorise them according to operator type. Students should know that **arithmetic operators** are used to perform calculations, and understand order of precedence rules (BIDMAS). They should understand the difference between the **division** operator (/), the **integer division** operator (//) and the **modulus** operator (%), and know which one to use when. See for example *Paper 01 Specimen 1, Q3(f)*. Students should know that **relational operators** are used to compare two values, and that the result of the comparison is either True or False. They should know that **logical (Boolean) operators** are used to evaluate relational expressions, and evaluate to either True or False (see also SP 1.3.1). The code for a linear search algorithm provided in *Paper 01 2206, Q5(b)* uses a logical operator to improve the efficiency of the algorithm. Students' understanding of operators and their ability to use them in algorithms is assessed in Paper 01. Their ability to use them in program code is assessed in Paper 02 (see SPs 6.5.1, 6.5.2 and 6.5.3).

Statement Students should:	Additional information
1.2.4 be able to determine the correct output of an algorithm for a given set of data and use a trace table to determine what value a variable will hold at a given point in an algorithm	Students should be able to determine the correct output of an algorithm for a given input. See for example *Paper 01 Specimen 2, Q2(c)* and *Paper 01 Specimen 3, Q1(f)*. They should know what a **trace table** is used for, and be able to use a trace table to record the value of variables whilst stepping through an algorithm line by line. See for example *Paper 01 SAM, Q5(d)* and *Paper 01 2206, Q5(c)*.
1.2.5 understand types of errors that can occur in programs (syntax, logic, runtime) and be able to identify and correct logic errors in algorithms	Students should know the difference between **syntax**, **logic** and **runtime** errors. *Paper 01 Specimen 2, Q2(d)* asks students to describe the difference between a syntax and a logic error. Students' understanding of types of errors and their ability to identify and correct logic errors in algorithms is assessed in Paper 01. Their ability to identify and correct all three types of error in program code is assessed in Paper 02 (see SP 6.1.5).
1.2.6 understand how standard algorithms (bubble sort, merge sort, linear search, binary search) work	Students should know the difference between **brute force** and **divide and conquer** approaches to sorting and searching. They should know how a **bubble sort** algorithm works and be able to hand trace the operation of a bubble sort on a list of values. See for example *Paper 01 Specimen 1,Q3(c)*. They do not have to be able to code a bubble sort. They should know how a **merge sort** algorithm works and be able to hand trace the operation of a merge sort on a list of values. See for example *Paper 01 Specimen 2, Q2(e)*. Whilst they do not have to be able to code a whole merge sort, students should be able to write code to merge two sorted lists into one sorted list using WHILE loops. They should understand how a simple **linear search** algorithm works on an unsorted list. See for example *Paper 01 2306, Q4(d)*. They should be able to use a loop to iterate through a data structure to perform a linear search. In Paper 02 students may be required to write a linear search

Statement Students should:	Additional information
	algorithm. See for example *Paper 02 2306, Q06* which requires students to write a program that determines if a username and password is stored in a two-dimensional array. They should understand how the **efficiency** of a linear search algorithm can be improved by stopping the search once the value of the item in the current index position is more than the target value. See for example *Paper 01 2206, Q5(b*). Students should know how a **binary search** works and be able to hand trace the operation of a binary search on a list of values. See for example *Paper 01 SAM, Q5(b)*. They do not have to be able to code a binary search. Students should be able to compare and contrast a bubble sort with a merge sort, and a linear search with a binary search.
1.2.7 be able to use logical reasoning and test data to evaluate an algorithm's fitness for purpose and efficiency (number of compares, number of passes through a loop, use of memory)	Students should understand what **fitness for purpose** means in relation to algorithms and use metrics, such as number of compares, number of passes through a loop, or use of memory, to determine the **efficiency** of an algorithm. They are not expected to know about/use Big O notation to measure time or memory efficiency. In *Paper 01 2206, Q5(b)* students are given the code for a linear search algorithm and asked to describe how the use of a Boolean variable makes the algorithm efficient. Students should be able to give the best- and worst-case scenarios for a linear search compared with a binary search, and realise that a merge sort algorithm is an example of an 'out of place sort' and therefore requires more memory than a bubble sort algorithm. Students' ability to use logical reasoning and test data to evaluate the fitness for purpose and efficiency of algorithms is assessed in Paper 01, and of programs in Paper 02 (see SP 6.1.6).

Statement Students should:	Additional information
1.3 Truth tables	
1.3.1 be able to apply logical operators (AND, OR, NOT) in truth tables with up to three inputs to solve problems	Students should know how to use the logical operators **AND**, **OR** and **NOT** in expressions and be aware of order of precedence rules (brackets, NOT, AND, OR). They must be able to interpret and construct truth tables to solve problems. See for example *Paper 01 Specimen 1, Q3(e)* and *Paper 01 Specimen 3, Q1(b)*. They do not need to be able to draw logic circuit diagrams.

Topic 2: Data

This is a wide-ranging topic that deals with how numbers, text, images and sounds are represented in binary, and how and why data is sometimes compressed.

Statement Students should:	Guidance
2.1 Binary	
2.1.1 understand that computers use binary to represent data (numbers, text, sounds, graphics) and program instructions and be able to determine the maximum number of states that can be represented by a binary pattern of a given length	Students should understand that a single **binary** digit (bit) can represent either 1 or 0 (corresponding to the electrical states on and off). They should be aware that computers represent, process, store and transmit all **data** as binary patterns and that the meaning of a group of bits depends on its context. They should know how to determine the maximum number of unique **states** that can be represented by a **binary pattern** of a given length (2^n). *Paper 01 SAM, Q1(b)* addresses this requirement. Students should be familiar with decimal (base 10), binary (base 2) and hexadecimal (base 16) numbers. *Paper 01 Specimen 2, Q3(a)* requires students to complete a table giving the number of values per digit of base 2 and base 16 numbers.
2.1.2 understand how computers represent and manipulate unsigned integers and two's complement signed integers	Students should know the difference between an **unsigned** and a **signed integer** and know that a positive integer is a whole number with a value greater or equal to zero and a negative integer is a whole number with a value less than zero. They should know that negating a signed integer involves changing its sign without changing its value. They should recognise circumstances where it is better to use an unsigned rather than a signed integer and vice versa. *Paper 01 SAM, Q1(c)(i)* assesses this understanding. Students should know how signed integers are represented in **two's complement,** using the MSB as a negative value. They should know the range of values that can be represented with 8 bits. They do not need to know about sign and magnitude representation.

Statement Students should:	Guidance
2.1.3 be able to convert between denary and 8-bit binary numbers (0 to 255 and −128 to +127)	Students should be able to **convert** both unsigned and signed denary integers into binary and vice versa. *Paper 01 Specimen 2, Q3(c) and Paper 01 2206, 3(a)(i)* assess students' ability to convert unsigned denary numbers into their binary equivalent. *Paper 01 Specimen 1, Q4(b)* requires students to convert a signed denary number into its two's complement binary equivalent.
2.1.4 be able to add together two positive binary patterns and apply logical and arithmetic binary shifts	Students should be able to **add** together two 8-bit binary numbers. See for example *Paper 01, Specimen 1 Q4(c)* and *Paper 01, Specimen 2 Q3(g)*. They should understand that binary subtraction can be achieved by adding the two's complement of the second number to the first number. i.e. $x - y = x + -y$. See for example *Paper 01 Specimen 3, Q5(b)*. Students should know the difference between and be able to perform **logical** and **arithmetic binary shifts**. They should understand why performing a right shift may result in a lack of precision. They should understand that one use for logical binary shifts is to multiply and divide unsigned binary integers by powers of two. *Paper 01 Specimen 1, Q4(a)* tests this understanding. They should recognise that – whilst arithmetic binary shifts can be used to divide negative numbers – using left arithmetic shifts to implement multiplication of negative numbers does not work because the MSB is not preserved.
2.1.5 understand the concept of overflow in relation to the number of bits available to store a value	Students should know that **overflow** occurs when the result of a calculation is too large to fit into the location assigned to hold it. They should understand the effect of overflow errors on program outcomes. *Paper 01 SAM, Q1(d)* and *Paper 01, 2206, Q3(a)(iv)* assess students' understanding of this concept.

Statement Students should:	Guidance
2.1.6 understand why hexadecimal notation is used and be able to convert between hexadecimal and binary	Students should know that **hexadecimal** is a base 16 number system used as shorthand for binary, and that one hexadecimal digit corresponds to four bits (one nibble) and can represent sixteen unique values (0 – F). *Paper 01 Specimen 2, Q3(e)* and *Paper 01 2206, 3(a)(ii)* assess students' ability to convert from binary to hex and *Paper 01 Specimen 1, Q4(d*) from hex to binary. Students should understand reasons why humans choose to use hex in preference to binary.
2.2 Data representation	
2.2.1 understand how computers encode characters using 7-bit ASCII	Students should know that **ASCII** is a standard for encoding characters (letters, numbers, punctuation marks and control codes), and that ASCII assigns each character its own unique numeric value. *Paper 01 2206, Q3(c)* asks for a description of ASCII. *Paper 01 Specimen 2, Q3(h)* assesses students' understanding of the limitation of standard 7-bit ASCII, i.e. that it is only capable of encoding 128 characters, 32 of which are control codes. Students do not need to learn the details of extended ASCII or Unicode, but they should be aware that alternative coding systems exist that permit a wider range of character sets and non-English characters to be represented. Students should be aware that character codes are grouped and run in sequence and – given the code for one character – be able to derive the code for another.
2.2.2 understand how bitmap images are represented in binary (pixels, resolution, colour depth)	Students should know that a **pixel** (picture element) is the smallest element of a bit-mapped image and that the size of an image is expressed as width x height in pixels. They should know that the **resolution** of an image refers to its physical size when displayed on screen or in print, and is measured in pixels per inch (ppi). The higher the resolution, the more pixels per inch and the better the image quality, e.g. a 200 x 100 pixel bitmap with a resolution of 100 ppi would measure 2" x 1", whereas, with a resolution of 200 ppi, it would measure 1" x 0.5".

Statement Students should:	Guidance
	Students should recognise that an image with a low resolution has fewer pixels per inch and may become pixelated if stretched to fit into a larger space *Paper 01 2206, Q3(d)(i)* assesses students' understanding of the impact of resolution on the visual quality of an image. Students should know that **colour depth** is the number of bits used to represent the colour of a pixel. The greater the number of bits used, the more tones/colours can be represented. *Paper 01 Specimen 1, Q4(e)(ii)* assesses this understanding. They should understand that the size (in pixels) and colour depth of an image determine its file size. The greater the number of pixels and the greater the colour depth, the larger the file size will be. Students should know that metadata is information about the properties of an image and adds to the file size of an image. Students should be able to convert binary data arranged top-down* into a bitmap image and be able to generate the binary code for a bitmap. *Starting with the pixel in the top left corner, moving across from left to right and down row by row, ending with the pixel in the bottom right-hand corner. See for example *Paper 01 Specimen 3, Q5(e)*.
2.2.3 understand how analogue sound is represented in binary (amplitude, sample rate, bit depth, sample interval)	Students should know the difference between a continuous analogue and a discrete digital signal, and understand why an analogue sound is never fully reproducible in a digital format. They should know that: • the **amplitude** of a sound wave determines the sound's loudness – the higher the amplitude, the louder the sound. • a **sample** is a measure of amplitude at a point in time, that the **sample rate** is the number of samples taken per second, measured in hertz, and that **sample interval** is the time between samples. • **bit depth** is the number of bits used to represent each sound sample Students should know that a CD-quality soundtrack

Statement Students should:	Guidance
	uses a sample rate of 44.1KHz, a bit depth of 16 and is recorded in stereo. They should understand how the choice of sampling rate, bit depth and sampling interval affect the accuracy of a digital representation. *Paper 01 Specimen 2, Q3(d)(i)* asks about the effect of decreasing the sample interval, and *Paper 01 Specimen 1, Q4(f)(ii)* asks for a benefit and a drawback of increasing the bit depth. Students should understand how an audio sound is converted from analogue to digital. *Paper 01 SAM, Q1(g)* requires students to label a partially completed diagram showing an analogue sound being sampled. *Paper 01 Specimen 2, Q3(d)(i)* assesses students' ability to plot the position of a sound sample on a graph.
2.2.4 understand the limitations of binary representation of data when constrained by the number of available bits	Students should understand that the number of available bits determines how a character set, an image, a sound, etc., is represented – the more bits there are, the greater the range of unique values. See for example *Paper 01 2023, 5(b)*. They should know how to determine the number of bits required for a specific purpose. See for example *Paper 01 Specimen 2, Q3(f)(ii)* and *Paper 01 Specimen 3, 5(c)*.
2.3 Data storage and compression	
2.3.1 understand that data storage is measured in binary multiples (bit, nibble, byte, kibibyte, mebibyte, gibibyte, tebibyte) and be able to construct expressions to calculate file sizes and data capacity requirements	Students should be familiar with and use base 2 **binary multiples** (IEC units) for constructing expressions to calculate file size and data capacity. Use of denary multiples in this context is not acceptable. *Paper 01 Specimen 2, Q3(b)* is a multiple choice item that assesses if students know why it is preferable to use binary multiples to express data capacity and file sizes. Students should be able to rank the units of measurement in size order. They should be able to convert larger units to smaller one by multiplying by 1024, and smaller to larger units by dividing by 1024. See for example *Paper 01 SAM, Q1(c)(ii)*.

Statement Students should:	Guidance
	They should be able to construct expressions to calculate: • the file size of an image (width x height x colour depth), or – given the file size and the values of any two of the variables – calculate the value of the missing one. See for example *Paper 01 SAM, Q1(h)*, *Paper 01 Specimen 2, Q3(f)(i)* and *Paper 01 2206, Q3(d)(ii)*. • the file size of an audio recording (sample rate * bit depth * duration), or – given the file size and the values of any two of the variables – calculate the value of the missing one. See for example *Paper 01 Specimen 3, Q5(d)*. There is no need for them to perform the calculation.
2.3.2 understand the need for data compression and methods of compressing data (lossless, lossy)	Students should know that **compression** is a technique for reducing file size and understand why reducing the size of a file is sometimes necessary or desirable. They should understand that using a **lossy** algorithm to compress a file results in data being permanently lost, whereas using a **lossless** algorithm allows the original file to be exactly reconstructed from the compressed data. They should understand that different types of data lend themselves to different compression methods. See for example *Paper 01 Specimen 1, Q4(g)* and *Paper 01 Specimen 3, Q5(a)*. There is no need for them to know about any particular compression algorithms or compressed file formats. See also SP 3.2.2 which deals with compression tools.

Topic 3: Computers

This topic is concerned with the role of hardware and software components in a computer system.

Statement Students should:	Guidance
3.1 Hardware	
3.1.1 understand the von Neumann stored program concept and the role of main memory (RAM), CPU (control unit, arithmetic logic unit, registers), clock, address bus, data bus, control bus in the fetch-decode-execute cycle	Students should know what is meant by the **stored program concept** and **the fetch-decode-execute cycle**. They should know the characteristics of **main memory**. Students should know that the central processing unit (**CPU)** sequentially **fetches, decodes and executes** instructions stored in memory. They do not need to know how modern computers use pipelining to overlap the three stages. *Paper 01 Specimen 1, Q2(a)(i*) requires students' to complete a simple diagram of the fetch-decode-execute cycle. Students should understand the role of the **control unit** (CU) and the **arithmetic logic unit** (ALU) in the fetch-decode-execute cycle. They should know that **registers** are small memory cells within the CPU that are used for temporary storage of data and interim results. They do not need to be able to name, identify or describe the function of specific registers, such as the program counter or accumulator. Students should know that the **clock** synchronises the actions of the CPU, with each tick of the clock triggering an operation, and that the speed of the clock is measured in hertz. They should understand how the speed of the clock affects the performance of the CPU. See for example *Paper 01 Specimen 1, Q2(a)(iv*), which assesses students understanding of why a higher clock speed is desirable. Students should know that a **bus** is a set of parallel wires through which data/signals are transmitted from one component to another. They should be aware that the width of the bus is the number of parallel wires it has, and determines how many addressable memory locations there are. Students should know the function of the **address**

Statement Students should:	Guidance
	bus, **data bus** and **control bus** in the fetch-decode-execute cycle. They should know which buses are bi-directional and which unidirectional, and why this is the case. *Paper 01 Specimen 1, Q2(a)(v*) asks students to name a unidirectional bus. *Paper 01 SAM, Q4(c)* asks for a description of the role of the control unit and buses when fetching an instruction from memory. *Paper 01 Specimen 2, Q5(e)* asks students to draw a flowchart to show the process required the read the contents of a memory location into the CPU.
3.1.2 understand the role of secondary storage and the ways in which data is stored on devices (magnetic, optical, solid state)	Students should understand that **secondary storage** provides long-term, non-volatile storage for programs and data. They should understand how data is stored on **magnetic** media. See for example *Paper 01 Specimen 1, Q2(b)*. They should know that a magnetic hard drive is a mechanical device with moving parts and understand the implications of this for speed of access, robustness and durability. Students should understand how data is stored on **optical** media. See for example *Paper 01 2206 Q2(c)(i)*. They should understand that **solid state** devices have no moving parts and use electrical circuits to persistently store data. They do not need to know the physics of how semiconductors and floating gate transistors work. *Paper 01 Specimen 3 Q3(d)(i)* asks why solid state drives are not affected by fragmentation. Students should know the advantages and disadvantages of each type of storage and recognise when one is more suitable than another for a particular purpose. *Paper 01 Specimen 2, Q5(d)(ii)* asks students to select a suitable storage device for an IoT device. Students do not need to know how cloud storage works.

Statement Students should:	Guidance
3.1.3 understand the concept of an embedded system and what embedded systems are used for	Students should understand how an **embedded system** differs from a general-purpose computer. See for example *Paper 01 2206, Q2(d)*. They should know about common hardware components of embedded systems. *Paper 01 Specimen 3, Q3(a)* requires students to complete a diagram of an embedded system by adding labelled boxes and arrows to show a sensor and an actuator that controls a motor. Students should understand the role of the microcontroller in an embedded system and recognise that power consumption is a major consideration for the design of an embedded system. They should be able to describe uses of embedded systems and outline how an embedded system could carry out a particular task. *Paper 01 Specimen 1, Q2(g*), for example, asks students to describe how an embedded system could be used to control the windscreen wipers of a car. *Paper 01 Specimen 2, Q5(d*)(i) assesses students' ability to apply what they know about embedded systems in general to the specific case of a battery-powered IoT lawn mower. Students should be familiar with the concept of the Internet of Things and be aware of privacy and security concerns associated with it.
3.2 Software	
3.2.1 understand the purpose and functionality of an operating system (file management, process management, peripheral management, user management)	Students should know that the purpose of an **operating system** (OS) is to control the computer's hardware and software They should know that **file management** is the process of organising, storing and retrieving files. They should understand that: • files are organised in directories, folders and sub-folders • be familiar with common file handling functions such as save, open, rename and delete Students should know that when a program is loaded into memory in order to be executed it becomes a **process**, and that at any single time multiple processes will be active. They should understand that the goal of **process management** is to share

Statement Students should:	Guidance
	out finite resources (CPU and main memory) between competing processes. Whilst there is no need for students to know the details of any particular scheduling algorithm, they should understand that: • only one process at a time can have exclusive use of the CPU • the OS uses a scheduling algorithm to prioritise processes • processes are held in a queue whilst waiting their turn and go to the back of the queue once their timeslot ends if not yet complete. *Paper 01 SAM, Q4(d)* asks students to describe how the OS uses scheduling to allocate processor time. Students should understand how the OS shares main memory between processes, allocating each its own section of RAM, and uses a paging algorithm to swap processes between RAM and virtual memory. See for example *Paper 01 Specimen 1, Q2(f).* Students should know that **peripheral management** involves controlling peripherals, such as the keyboard, monitor and printer. They should understand that the OS uses device drivers to enable it to communicate with peripherals, and that peripheral use interrupts to signal to the OS that they need immediate attention. Students should know that **user management** includes: • adding and deleting users • granting access rights to applications, data and systems • authenticating users' identity • using permissions to control what users are allowed to do. *Paper 01 Specimen 2, Q5(c)* asks students to describe the purpose of user management and *Paper 01 2206 1, Q2(g)* asks for an explanation of how an OS enables an administrator to manage users. Students should know that the OS provides a user interface, and be aware of features of CLI, GUI and natural language interfaces.

Statement Students should:	Guidance
3.2.2 understand the purpose and functionality of utility software (file repair, backup, data compression, disc defragmentation, anti-malware)	Students should know that **utility software** is a collection of tools that help maintain the functionality of a computer. *Paper 01 SAM, Q4(a)* requires students to give two examples. Students should know that: • **file repair** software is used to repair or recover data from damaged or corrupted files. • **backup software** automates the process of backing up files so that data can be recovered if something goes wrong. They should understand the difference between full and incremental backup. • **data compression** software reduces the size of files so that they take up less storage space and need less time to transfer. (See also SP 2.3.2 which covers the two types of compression – lossy and lossless.) *Paper 01 2206, Q2(e)* asks students to identify two reasons for using data compression. • **defragmentation** software is used to reorganise the data stored on a magnetic hard drive when it has become fragmented so that related pieces of data are stored contiguously. *Paper 01 Specimen 2, Q5(a)* asks for a description of the process of defragmentation. *Paper 01 Specimen 3 Q3(d)(i)* requires students to show the state of three files stored on a hard drive after running a defragmentation utility. • **anti-malware** software detects, quarantines and removes malware from a computer. Students should know the difference between signature-based and behaviour-based detection. (See also SPs 5.3.1 and 5.3.2.) Students should be able to select appropriate utility tools for a specific task.

Statement Students should:	Guidance
3.2.3 understand the importance of developing robust software and methods of identifying vulnerabilities (audit trails, code reviews)	Students should know what constitutes **robust software**. They should be familiar with **vulnerabilities**, such as weak authentication or lack of encryption, and understand how they are exploited by criminals. They should understand the purpose of **code reviews** and **audit trails**. *Paper 01 SAM, Q4(b)* asks about the role of code reviews in helping to produce robust software. *Paper 01 Specimen 2, Q5(b)* asks for two advantages of keeping an audit trail.
3.3 Programming languages	
3.3.1 understand the characteristics and purposes of low-level and high-level programming languages	Students should know that machine code and assembly language are **low-level languages** and be able to describe characteristics of low-level languages, such use of mnemonics and machine-specific. See for example *Paper 01 2206, Q2(a)(i)&(ii)*. Students should be aware that assembly language programs need to be translated into machine code before they can be executed, and that an assembler translates each line of assembly language into a single machine code instruction. They should be able to identify tasks for which low-level languages are used, e.g. writing control software for embedded systems with limited processing power and memory. There is no need for students to have any practical experience of writing programs written in machine code or assembly language. Students should know that most computer programs are written in a **high-level language** and understand why this is the case. They should be familiar with characteristics of high-level languages, such as availability of ready-made subprograms and components. *Paper 01 Specimen 3, Q3(g)* asks students to describe two ways in which a high-level language differs from a low-level language. *Paper 01 SAM, Q4(e)* is a 6-mark essay question that requires students to discuss the merits of using a high-level as compared with a low-level language for writing the code for an alarm system.

Statement Students should:	Guidance
3.3.2 understand how an interpreter differs from a compiler in the way it translates high-level code into machine code	Students should understand that every processor has its own set of machine code instructions and that a program written in a high-level programming language must be translated before it can be executed. They should know how an **interpreter** and a **compiler** handle the translation process. See for example *Paper 01 2206, Q2(iv)* and *Paper 01 Specimen 1, Q2(e)* which asks students to describe two ways a compiler differs from an interpreter. *Paper 01 Specimen 3, Q3(e*) asks for three features of a compiler. Students should be aware of the benefits and drawbacks of each method of translation and what each is best suited for.

Topic 4: Networks

This topic is concerned with different types of network, including the internet, and deals with the role of protocols in enabling communication across a network and network security

Statement Students should:	Guidance
4.1 Networks	
4.1.1 understand why computers are connected in a network	Students should know that a computer **network** consists of two or more connected devices that exchange data and share resources. They should understand the reasons why devices are connected to networks, such as to support collaborative working and to enable rapid deployment of new software or updates. See for example *Paper 01 2206, Q1(a)(i)*. Students should be aware that networks have their disadvantages, e.g. they can be vulnerable to malware (see SP 4.2.1), and network outages have an organisation-wide impact.
4.1.2 understand different types of networks (LAN, WAN)	Students should know the characteristics of a **LAN** and a **WAN** and how a LAN differs from a WAN. See for example *Paper 01 2206, Q1(a)(ii)*. *Paper 01 SAM, Q2(a)* assesses students' understanding of the type of network used to connect geographically-disperse sites.
4.1.3 understand how the internet is structured (IP addressing, routers)	Students should recognise that the **internet** is a global network of networks. They should know what **IP addresses** are used for. *Paper 01 2206, Q1(b)(i)* asks students to name the unique identifier used when sending or receiving data packets over the internet. Students should know the difference between IPv4 and IPv6 addresses and why the latter are needed. They should understand the process that take place when a web browser on a user's machine requests a web page from a web server, including the role of a DNS server in matching IP addresses to domain names. *Paper 01 Specimen 1, Q5(e)* asks students to complete a diagram illustrating this process. Students should know that the IP addresses of the sender and receiver are two of the items included in a packet header and understand why they are

Statement Students should:	Guidance
	needed (see also SP 4.1.7). Students should know that the principal high-speed data transmission routes between interconnected networks is called the internet backbone. They should understand the role of **routers** in forwarding data packets between networks and selecting the best route for them to take.
4.1.4 understand how the characteristics of wired and wireless connectivity impact on performance (speed, range, latency, bandwidth)	Students should know the characteristics of different types of **wired** (copper and fibre-optic cable) and **wireless** (Wi-Fi, Bluetooth, Zigbee, RFID and NFC) transmission media. They should be able to compare the performance of networks, and know that: • **bandwidth** is a measure of the capacity of a network, i.e. the theoretical amount of data that can be transferred in a given time, measured in bits per second. • the **speed** of a network is the actual rate of data transfer in a given time, measured in bits per second. • the **range** of a network is the maximum distance a signal is able to reach. • network **latency** is the delay between a signal being sent and received measured in milliseconds. *Paper 01 2206, Q1(a)(iv)* asks for a definition of latency. *Paper 01 Specimen 3, Q4(b)* asks students to explain why the performance of a network can be effected by its environment. Students should be able to compare the suitability of wired and wireless methods of connection, and understand why many networks use a combination of both.
4.1.5 understand that network speeds are measured in bits per second (kilobit, megabit, gigabit) and be able to construct expressions involving file size,	Students should know that **network speeds** are measured in bits per second, using base-10 denary multiples (SI units). They should be able to rank these units of measurement in size order, and convert between units. See for example *Paper 01 2206, Q1(a)(iii)*. Students should be able to construct expressions involving **file size**, **transmission rate** and **time**.

Statement Students should:		Guidance
	transmission rate and time	*Paper 01 SAM, Q2(e)* requires students to construct an expression to show how many seconds it will take to transmit 20 MiB of data using a transmission rate of 2Mbps. *Paper 01 Specimen 1, Q5(d)* asks for the expression to calculate the minimum transmission rate required to transmit a 250 MiB file in exactly one hour.
4.1.6	understand the role and need for network protocols (Ethernet, Wi-Fi, TCP/IP, HTTP, HTTPS, FTP) and email protocols (POP3, SMTP, IMAP)	Students should understand that a communication **protocol** is a set of rules governing data transmission between devices. They should know that **HTTP**, **HTTPS**, **FTP, POP3, SMTP** and **IMAP** are network protocols operating at the application layer of the TCP/IP stack (see SP 4.1.7) and be able to outline what each does**.** Students should know that POP3 and IMAP are email protocols and understand how each of them handles emails. See for example *Paper 01 Specimen 3, 4(a)*. They should know that **TCP/IP** operates at the transport layer of the TCP/IP stack and understand its role in splitting data into packets. They should know that **Ethernet** and **Wi-Fi** operate at the link layer of the TCP/IP stack and understand their role in data transmission. There is no need for students to know all the individual protocols that make up the Ethernet and Wi-Fi 'families' of protocols.
4.1.7	understand how the 4-layer (application, transport, Internet, link) TCP/IP model handles data transmission over a network	Students should understand that: • the **TCP/IP model** is a four-layer stack that specifies how data is exchanged over the internet. • each layer of the stack is responsible for a specific part of the process. • different protocols operate in each layer of the stack. See *Paper 01 Specimen 2, Q1(a)*. • data passes down through the layers of the stack when it is being sent and up through the layers when it is received. Students should be able to name the four layers of the model, and put them in the correct order. They should be able to describe the function performed by each layer. *Paper 01 SAM, Q2(d)* asks about the transport

Statement Students should:	Guidance
	layer's role in checking that incoming packets are received correctly. *Paper 01 Specimen 3, Q4(c)* asks about the use of checksums in the error-checking process. Student should know that data is split up into packets for transmission across the internet, and that a packet consists of a header, a payload, and a footer. *Paper 01 SAM, Q2(c)* asks about the contents of the data packets.
4.1.8 understand characteristics of network topologies (bus, star, mesh)	Students should know that a network **topology** describes the arrangement of devices in a network. They should know that: • in a **bus** topology all the devices are connected to a single cable. • in a **star topology** all the devices are physically connected to a central node (switch/hub). *Paper 01 Specimen 2, Q1(b)(i)* asks students to complete a diagram of a star network. • a **mesh network** can be fully connected (each device connected to all other devices) or partially connected (only some devices connected directly to one another). *Paper 01 SAM, Q2(b)* requires students to complete a diagram of a fully connected mesh network. Students should understand how the characteristics of a topology affect its performance, scalability, reliability and security, and be able to select the most appropriate topology for a given scenario. *Paper 01 Specimen 1, Q5(b)* asks students to describe a disadvantage of the bus topology and *Paper 01 2206, Q1(e)* a disadvantage of the star topology. *Paper 01 Specimen 2, Q1(b)(i)* asks students why a star topology is preferable to a bus topology.
4.2 Network security	
4.2.1 understand the importance of network security, ways of identifying network vulnerabilities	Students should know that **network security** is any activity designed to protect a network and its data from internal and external threats. They should know that a network **vulnerability** is a weakness that can be exploited by a criminal to gain

Statement Students should:	Guidance
(penetration testing, ethical hacking) and methods of protecting networks (access control, physical security, firewalls)	unauthorised access to information or resources, and be able to give examples of vulnerabilities, such as weak passwords or poorly configured firewalls. Students should be able to explain how internal (white box) and external (black box) **penetration testing** is used to identify network vulnerabilities. *Paper 01 Specimen 2, Q1(e)* asks about the use of penetration testing to identify vulnerabilities that could be exploited by people who work for an organisation. Students should know the difference between ethical (white-hat) and criminal (black-hat) hackers and be able to explain how **ethical hacking** helps to identify and fix vulnerabilities. They should know that the objective of **access control** is to keep unauthorised users and devices from gaining access to a network, and restricting what authorised users are allowed to do. Students should know what authentication is and why multi-factor authentication provides greater security. They should know how file permissions (read, write, delete and execute) are used to control what data network users can access and what they are allowed to do with it. (See also SP 3.2.1.) *Paper 01 Specimen 1, Q5(c)* asks students to explain what type of access someone on work experience should have to confidential data. Students should know that **physical security** methods are designed to prevent unauthorised people from gaining physical access to network components such as servers and desktop computers. They should be able to give examples of physical security measures. Students should know how a **firewall** helps to protect networks by monitoring incoming and outgoing traffic and using a set of rule to determine which traffic to allow in and out and which to bar. See for example *Paper 01 2206, Q1(c)*.

Topic 5: Issues and impact

Topic 5 is concerned with issues and impact associated with the use of computing technology.

Statement Students should:	Guidance
5.1 Environmental	
5.1.1 understand environmental issues associated with the use of digital devices (energy consumption, manufacture, replacement cycle, disposal)	Students should know how activities related to the **manufacture** of digital devices, such as mining operations, and component production, damage to the environment. *Paper 01 2023, Q3(d)* asks about the environmental impact of the large volume of water used in the manufacture of digital devices. Students should be aware of the environmental issues associated with the **energy consumption** of digital devices, including use of fossil fuels to generate electricity. They should understand factors that contribute to the short **replacement cycle** of digital devices, such as smartphones, and be aware of initiatives to address this issue. *Paper 01 Specimen 2, Q2(a*) ask students for two ways in which the useful life of a smartphone can be extended. Students should understand the environmental issues associated with the **disposal** of e-waste. See for example *Paper 01 SAM, Q3(a)*. They should be able to explain ways of reducing the impact of digital devices on the environment, such as recycling of electronic waste and environmental monitoring. See for example *Paper 01 Specimen 1, Q1(c).*
5.2 Ethical and legal	
5.2.1 understand ethical and legal issues associated with the collection and use of personal data (privacy, ownership, consent, misuse, data protection)	Students should know that an action is **legal** if it is in accordance with the law, and **ethical** if it is considered to be the right thing to do, and that an action may be legal but nevertheless unethical. They should know that **personal data** relates to an identified or an identifiable individual. They should understand the value of personal data to organisations and ways in which personal data is obtained, e.g. through cookies on website, location-based apps and social media posts. They should be aware of **privacy** concerns arising from the collection and use of personal data.

Statement Students should:	Guidance
	Paper 01 SAM, Q3(c) addresses the data privacy concerns associated with the use of digital assistant technologies. Students should know that **data protection** legislation assigns individuals **ownership** of their own personal data and requires organisations to elicit **consent** before collecting data. They should know the six data protection principles set out in the legislation. *Paper 01 Specimen 3, Q2(c)* for two of these principles. *Paper 01 2206, Q4(b)* asks for two pieces of information an organisation must tell people when requesting consent to use their personal data. Students should know that the use of cookies on websites for collecting and storing personal data is governed by legislation. *Paper 01 Specimen 2, Q4(c)* asks students to discuss the legal and ethical issues associated with a company's collection and use of data. Students should know how computer **misuse** legislation helps protect individuals' personal data by deterring hackers from accessing it.
5.2.2 understand ethical and legal issues associated with the use of artificial intelligence, machine learning and robotics (accountability, safety, algorithmic bias, legal liability)	Students should understand that **artificial intelligence** (AI) is an umbrella term referring to computer systems able to perform tasks normally requiring human intelligence, such as speech recognition and decision-making. They should know that: • **robotics** involves creating autonomous systems to perform specific tasks without human intervention, and should be able to give examples. • **machine learning** involves training an algorithm to learn from its inputs without being specifically programmed to perform a task. Students should understand the causes and impact on individuals and communities of **algorithmic bias**, and know about methods of reducing the risk. See for example *Paper 01 2206, Q4(c)*. *Paper 01 Specimen 3, Q2(e)* asks for two ways that the data used to train a machine learning algorithm could cause algorithmic bias.

Statement Students should:	Guidance
	Students should be aware of **accountability** and **safety** concerns associated with the growing use of these technologies. They should recognise that there are currently few legal provisions for these technologies, leaving issues of **liability** and bias largely unregulated.
5.2.3 understand methods of intellectual property protection for computer systems and software (copyright, patents, trademarks, licensing)	Students should know that **intellectual property** (IP) refers to creations of the human intellect and how **copyright**, **patents** and **trademarks** protect IP. Whilst students should be aware of the existence of legislation designed to protect IP, they do not need to know the details of any particular laws. Students should know that **licensing** allows the creator of a software application to specify how it can be used and distributed. Whilst students do not need a detailed knowledge of different types of licences, they should be aware of the difference between open-source and proprietary licences.
5.3 Cybersecurity	
5.3.1 understand the threat to digital systems posed by malware (viruses, worms, Trojans, ransomware, key loggers) and how hackers exploit technical vulnerabilities (unpatched software, out-of-date anti-malware) and use social engineering to carry out cyberattacks	Students should understand that **malware** is malicious software intentionally designed to cause harm. They should be know the characteristics of different types of malware including: • how **viruses** and **worms** replicate and spread, and what harm they do. See for example *Paper 01 Specimen 2, Q4(a)*. • how **Trojans** get themselves installed on a computer, and – once in situ – what they can be used for. • how **ransomware** attacks are used by criminals to extort money. *Paper 01 SAM, Q3(b)(i)* requires students to identify a ransomware attack from a supplied screenshot. • that a **key logger** records key strokes and malicious uses for key loggers. Students should know that **hackers** are criminals who exploit **technical vulnerabilities** to break into computer systems and networks and how they use **unpatched software** and **out-of-date anti-malware** to launch cyberattacks.

Statement Students should:	Guidance
	Paper 01 SAM, Q3(b)(ii) asks why unpatched software makes digital systems vulnerable to cyberattacks. *Paper 01 Specimen 2, Q4(b*) asks why software should be patched regularly. Students should know how **social engineering** works and be familiar with pretexting, phishing, baiting and quid pro quo techniques. *Specimen 1, Q1(d)* asks why clicking on a link in an email from an unknown source is risky.
5.3.2 understand methods of protecting digital systems and data (anti-malware, encryption, acceptable use policies, backup and recovery procedures)	Students should know about **methods of protecting** digital systems and data. They should know the purpose of **anti-malware** software and understand the difference between signature-based and behaviour-based detection. See also SP 3.2.2. Students should know that **encryption** is the process of encoding data so that it cannot be read by anyone not in possession of the decryption key. Student should know what is covered by and how **acceptable use policies** help protect systems and data. See for example *Paper 01 SAM, Q3(d).* Students should be aware of the need for data **backup and recovery** procedures and be able to describe elements of these, e.g. RAID, off-site storage and stand-by equipment/premises. They should know the difference between incremental and full backup, and understand why files should be backed up regularly. See for example *Paper 01 Specimen 3, Q2(b)* and *Paper 01 2206, Q4(a).* (See also SP 3.2.2.)

Topic 6: Problem solving with programming

Topic 6 focuses on programming. Students should be competent at reading, writing and debugging programs. They must use Python 3 to write programs.

Statement Students should:	Guidance
6.1 Develop code	
6.1.1 be able to use decomposition and abstraction to analyse, understand and solve problems	Students should be able to use **decomposition** and **abstraction** to help them understand problems and design effective solutions. The levels-based mark scheme for Solution Design allocates marks for use of these computational thinking techniques. See for example *Paper 02 Specimen 2, Q05*. Students' understanding of decomposition and abstraction is assessed in Paper 01 (see SP 1.1.1), and their ability to use them when developing code in Paper 02.
6.1.2 be able to read, write, analyse and refine programs written in a high-level programming language	The PLS specifies which part of Python 3 students should be able to use when working with code. With the exception of the final question, which requires students to design and **write** a program from scratch, questions on Paper 02 require them to **analyse** and **refine** given code by correcting errors, adding or rearranging lines, selecting the correct line, improving readability, etc. See for example *Paper 02 2206, Q02, Q03* and *Q04*.
6.1.3 be able to convert algorithms (flowcharts, pseudocode) into programs	Students should be able to: • Convert **flowcharts** into programs. See for example *Paper 02 Specimen 1, Q04* and *Paper 02 Specimen 2, Q03*. • Convert **written descriptions** given in questions and in starter code into programs. See for example *Paper 02 SAM, Q06*. Students' ability to follow and write algorithms is assessed in Paper 01 (see SPs 1.2.1, 1.2.2 and 1.2.3), and their ability to convert algorithms into code in Paper 02.

Statement Students should:	Guidance
6.1.4 be able to use techniques (layout, indentation, comments, meaningful identifiers, white space) to make programs easier to read, understand and maintain	Students should use a range of **techniques** to make their programs easy to read. They should be consistent in their use of style conventions, such as bracketing conditional statements and leaving a space between the name of a function and its parameter list. *Paper 02 SAM, Q02* requires students to improve the readability of given code by choosing a more meaningful name for a variable, inserting white space and adding a comment. The levels-based mark scheme for Good Programming Practices awards marks for layout, meaningful variable names, comments, and white space. See for example *Paper 02 Specimen 2, Q06*. Further information can be found in the *Readability* section of the GPPG.
6.1.5 be able to identify, locate and correct program errors (logic, syntax, run-time)	Students should know how to locate and fix **syntax** errors in code. See for example *Paper 02 Specimen 2, Q01* and *Paper 02 2206, Q02*. They should know how to use test data to identify **run-time** and **logic** errors which – if undetected – would prevent code from functioning correctly. See for example *Paper 02 Specimen 2, Q04* and *Paper 02 2206, Q03*. The levels-based mark scheme for Functionality awards marks for error-free, robust solutions. See for example *Paper 02 Specimen 1, Q04*. Students' understanding of types of errors is assessed in Paper 01 (see SP 1.2.5). Their ability to locate and correct errors in code in Paper 02.
6.1.6 be able to use logical reasoning and test data to evaluate a program's fitness for purpose and efficiency (number of compares, number of passes through a loop, use of memory)	Students should use **logical reasoning** to correctly rearrange lines of code or select which line of code is correct. See for example. *Paper 02 Specimen 1, Q03*. The levels-based mark scheme for Functionality allocates marks for **fitness for purpose**. See for example *Paper 02 2206, Q04*. Students' ability to use logical reasoning and test data to evaluate algorithms is assessed in Paper 01 (see SP 1.2.7), and to evaluate programs in Paper 02.

Statement Students should:	Guidance
6.2 Constructs	
6.2.1 understand the function of and be able to identify the structural components of programs (constants, variables, initialisation and assignment statements, command sequences, selection, repetition,iteration, data structures, subprograms, parameters, input/output)	The PLS specifies the **structural components** and programming constructs students should recognise Further information can be found in the *Supported Data types and conversion*, *Structured data types* and *Programming constructs* sections of the GPPG. In Paper 02, students may be required to identify structural components used in given code. See for example *Paper 02 Specimen 1, Q01*. Students' understanding of these structural components and their ability to recognise and use them in algorithms is assessed in Paper 01 (see SPs 1.2.1 and 1.2.2). Their ability to use them in program code is assessed in Paper 02.
6.2.2 be able to write programs that make appropriate use of sequencing, selection, repetition (count-controlled, condition-controlled), iteration (over every item in a data structure) and single entry/exit points from code blocks and subprograms	The PLS specifies the programming constructs students should be able to use when writing code. Further information can be found in the *Programming constructs* sections of the GPPG. Students should be able to create code using combinations of these constructs. In *Paper 02 Specimen 1, Q04*, marks are awarded for use of selection and repetition, and in *Paper 02 Specimen 3, Q03* for creating a two-dimensional data structure. Students' understanding of these programming constructs and their ability to recognise and use them in algorithms is assessed in Paper 01 (see SPs 1.2.1 and 1.2.2). Their ability to use them in program code is assessed in Paper 02.
6.3 Data types and structures	
6.3.1 be able to write programs that make appropriate use of primitive data types (integer, real, Boolean, char)	The alternative names for **primitive data types** used by Python are provided in the PLS. Students should know that Python's list data structure is used to store sequences of items with the same data type (arrays) and sequences of items

Statement Students should:		Guidance
	and one- and two-dimensional structured data types (string, array, record)	with mixed data types (records). They should be able to use indexing to access items in one- and two-dimensional lists. See for example *Paper 02 2023, Q06*. The PLS provides details of loop constructs used to iterate through data structures, and the list and string subprograms students should be able to use. Further information can be found in the *Data types and conversion*, *Structured data types*, *Programming constructs* and *Supported subprograms* sections of the GPPG. Students' understanding of primitive and structured data types is assessed in Paper 01 (see SP 1.2.2). Their ability to use them in program code is assessed in Paper 02.
6.3.2	be able to write programs that make appropriate use of variables and constants	Students should know the difference between variables and constants. Although Python does not support **constants** in the way most other high-level languages do, students are expected to adhere to the convention of using upper case characters to name variables that hold non-changing values, i.e. are behaving as if they were constants. Further information about variables and constants can be found in the *Data types and conversion* section of the GPPG. *Paper 02 SAM, Q01* requires students to create an integer variable named roll, initialise it to 0 and subsequently assign the result of a library call to it. They must also create a constant named SIDES and give it the value 6. *Paper 02 2206, Q04* awards a mark for use of a given constant. Students' understanding of variables and constants is assessed in Paper 01 (see SP 1.2.2), and their ability to use of them in Paper 02.
6.3.3	be able to write programs that manipulate strings (length, position, substrings, case conversion)	Students should know how to slice and concatenate strings, and should be able to use all of the built-in string subprograms listed in the PLS. They should know how to add a new line control character to a string, and how to remove control characters from, and split a comma-separated value

Statement Students should:	Guidance
	string read in from a file. *Paper 02 Specimen 1, Q05* awards a mark for use of concatenation to build output strings. *Paper 02 Specimen 3, Q05* awards marks for stripping off the newline control character at the end of lines read in from a text file and splitting the lines by comma. Further information about string manipulation can be found in the *Structured data types and Supported subprograms* sections of the GPPG.
6.4 Input/output	
6.4.1 be able to write programs that accept and respond appropriately to user input	Eliciting **user input** and displaying output are basic operations that all students should master. Input prompts should be clearly worded. Students should be aware that in some circumstances string input will need to be converted to a different data type. *Paper 02 2206, Q01* requires students to use input with a prompt, and convert string input to integer. Output messages should be fit for audience and purpose, with attention paid to spelling and punctuation. *Paper 02 2206, Q01* requires students to use string concatenation to join parts of string output. The formatting strings section of the PLS provides details of the string.format() method that students should know how to use to customise output. Unless specifically asked to use string.format(), students are free to use the alternative f-string method of formatting text strings. *Paper 02 Specimen 2, Q05* awards marks for use of layout string for column headings and footers, and formatting percentages to two decimal places. Further information can be found in the *Inputs and outputs* section of the GPPG.
6.4.2 be able to write programs that read from and write to comma-separated value text files	The PLS provides details of the file operations students should be able to use. Students should know the difference between writing and appending items to a file and should run a program more than once to ensure that it produces

Statement Students should:	Guidance
	the correct file outcome. *Paper 02 2206, Q05* requires students to write data to a file, and *Paper 02 Specimen 3, Q05* to read data from a file. Further information about working with files can be found in the *Inputs and outputs* section of the GPPG.
6.4.3 understand the need for and be able to write programs that implement validation (length check, presence check, range check, pattern check)	Students should understand the need for **validation**, and be able to write basic validation routines to check the validity of user input, e.g. checking that an entered string has a minimum length or checking that a value entered lies within a given range. Students are only expected to use string manipulation functions to check for simple patterns. *Paper 02 SAM, Q3* requires students to add validation to a program so that it only accepts numbers from 1 to 20. *Paper 02 Specimen 2, Q03* requires students to write code to perform a presence check and a length check validation. Students should be aware that a while loop that terminates only when a correct input is received is an effective way of validating user input.
6.4.4 understand the need for and be able to write programs that implement authentication (ID and password, lookup)	Students should understand the need for **authentication**, and be able to write basic authentication routines, e.g. checking that a correct username and/or password has been entered by comparing the input with a value stored in a list. *Paper 02 SAM, Q6* requires students to write a program to authenticate system logins.
6.5 Operators	
6.5.1 be able to write programs that use arithmetic operators (addition, subtraction, division, multiplication, modulus, integer division,	Students should be able to use **arithmetic operators** in calculations. A table of operators is provided in the PLS. Further information can be found in the *Programming constructs* sections of the GPPG. In *Paper 02 Specimen 1, Q04* the accurate use of arithmetic operators is a consideration for the award of functionality marks in the levels-based mark

Statement Students should:	Guidance
exponentiation)	scheme. Students' ability to use arithmetic operators in algorithms is assessed in Paper 01 (see SP 1.2.3), and in code in Paper 02.
6.5.2 be able to write programs that use relational operators (equal to, less than, greater than, not equal to, less than or equal to, greater than or equal to)	Students should be able to use **relational operators** in comparisons. A table of operators is provided in the PLS. Further information can be found in the *Programming constructs* sections of the GPPG. In *Paper 02 Specimen 1, Q04* the accurate use of relational operators is a consideration for the award of functionality marks in the levels-based mark scheme. Students' ability to use relational operators in algorithms is assessed in Paper 01 (see SP 1.2.3), and in program code in Paper 02.
6.5.3 be able to write programs that use logical operators (AND, OR, NOT)	Students should be able to use **logical operators** to combine multiple expressions that produce a single true or false outcome. A table of operators is provided in the PLS. Further information can be found in the *Programming constructs* sections of the GPPG. *Paper 02 Specimen 2, Q04* requires students to use a logical operator to test the validity of the value entered by the user. Students' ability to use these operators in algorithms is assessed in Paper 01 (see SPs 1.2.3 and 1.3.1), and in program code in Paper 02.
6.6 Subprograms	
6.6.1 be able to write programs that use pre-existing (built-in, library) and user devised subprograms (procedures, functions)	The PLS specifies the **built-in** and **library** modules students should be able to use. Further information can be found in the *Supported subprograms* sections of the GPPG. *Paper 02 SAM, Q1* requires students to import the random library and use the random.randint() subprogram to simulate the roll of a dice. In *Paper 02 Specimen 2, Q04* students must write or complete the definition of two subprograms. Students' understanding of the benefits of using

Statement Students should:	Guidance
	subprograms is assessed in Paper 01 (see SP 1.1.2). Their ability to use pre-existing and user-defined subprograms in program code is assessed in Paper 02.
6.6.2 be able to write functions that may or may not take parameters but must return values and write procedures that may or may not take parameters but do not return values	Students should know how to write subprograms that take **parameters** and ones that return one or more **values** to the calling code. In *Paper 02 2023, Q05* students are required to write a procedure to display a welcome message and call the procedure in the main program. Further information can be found in the Subprograms section of the GPPG. *Paper 02 Specimen 2, Q04* awards marks for correctly completing a subprogram definition by adding parameters and calling the subprogram with the correct arguments in the correct order. In *Paper 02 2023, Q05* students are required to change the names of a subprogram's parameters and then use these local variables rather than global variables in the body of the subprogram.
6.6.3 understand the difference between and be able to write programs that make appropriate use of global and local variables	Students should understand the difference between **local variables** and **global variables.** They should understand the concept of scope and know that the same identifier can mean different things in different scopes. They should understand why it is good practice to minimise the use of global variables. Further information can be found in the *Subprograms* section of the GPPG. *Paper 02 Specimen 3, Q05* awards a mark for using only local variables. The Python keyword 'global' isn't included in the PLS. If a variable is needed everywhere it should be declared and initialised at the level of the main program.

6. Assessment guidance

Paper 01: Principles of Computer Science

Paper code: 1CP2/01

What is assessed?

Assesses Topics 1–5 of the subject content

Key features

- A 1 hour 30 minutes written paper
- Marked out of 75
- Worth 50% of the qualification
- Five compulsory questions, each with multiple parts
- Each question assesses aspects of a single topic from the subject content
- The order in which topics appear in the paper will vary from series to series
- Students write their responses in the answer book provided
- Use of a calculator is not permitted
- First assessment May/June 2022

Question types

- A mix of multiple choice, short, medium and extended open response, tabular and diagrammatic items
- One essay-style question that is marked using a levels-based mark scheme and is worth 6 marks
- One 6-mark practical question, such as designing an algorithm for a particular purpose, that is marked using a points-based mark scheme
- Includes some questions that target computer-related mathematics

Paper 02: Application of Computational Thinking

Paper code: 1CP2/02

What is assessed?

Assesses Topic 6 of the subject content

Key features

- A 2 hour practical, computer-based, onscreen exam
- Marked out of 75
- Worth 50% of the qualification
- Six compulsory practical programming questions that require students to design, write, test and refine programs, using a subset of Python 3 (the PLS) and an integrated development environment (IDE) with which they are familiar
- Students are provided with code files, a hard copy of the question paper and the PLS document
- Includes some questions that target computer-related mathematics
- Use of USB memory sticks of other storage devices is not permitted
- First assessment May/June 2022

Question types

- There are just two question types – students will either be required to amend a given piece of code or to write code from scratch.
- Some questions have a points-based mark scheme; some use a combination of points-based and levels-based.
- The levels-based mark schemes reward students for the functionalityof their programs, the quality of their designs and evidence of good programming practice.

Conducting the exam

- Centres must set up a designated secure user area for each candidate in advance of the exam taking place.
- Students must not be able to save any files that they produce during the exam anywhere other than in their designated secure user area and must not be able to access this area at any time other than during the exam.
- On the morning of the exam, centres must download, unzip and place the secure files in each candidate's secure user area.

- The secure files will include the Programming Language Subset (PLS), code samples (*.py), and comma-separated value formatted text files (*.txt) where required.
- There is no need for students to have access to a printer during the exam, nor should they be provided with printed copies of the secure data files.
- Students are not allowed to refer to textbooks, centre-prepared manuals or access shared folders during the examination, but may use theoffline help facilities provided by the IDE.
- At the end of the exam, centres must create a zipped folder for each candidate containing all files that they created during the exam and upload them to Edexcel to be marked.
- The Instructions for the Conduct of the Examination (ICE) document provides further information.

The sample assessment materials (SAMs), specimen and past exam papers available on the GCSE Computer Science section of the Pearson website, are a valuable source of reference. They illustrate the format and style the two exams and the types of questions studentsare likely to encounter.

Assessment objectives

Ofqual has set three assessment objectives (AOs) for GCSE Computer Science, which all specifications must adhere to.

AO1	Demonstrate knowledge and understanding of the key concepts and principles of computer science	30%
AO2	Apply knowledge and understanding of key concepts and principles of computer science	40%
AO3	Analyse problems in computational terms: • to make reasoned judgements • to design, program, test, evaluate and refine solutions	30%

AO1 – Demonstrate knowledge and understanding of the key concepts and principles of computer science

Questions targeting AO1 require students to demonstrate what they have learned about the subject content of the specification.

30% of the total marks for the qualification is allocated to this AO, of which no more than half can be for recall of facts. AO1 is assessed in Paper 01 only.

SAM Paper 01, Q3(a) uses the command word 'state', indicating that straightforward knowledge recall is required. Students are asked for two environmental issues associated with the disposal of digital technology.

SAM Paper 01, Q3(b)(ii) is more demanding, requiring students to demonstrate their understanding of why digital systems may be vulnerable to cyberattacks. The command word 'explain' signals that an element of justification must be provided in the response, e.g. 'Software may contain security bugs (1), because it is unpatched (1).'

Explain questions are worth two marks.

Paper 01, 2022, Q1(c) asks students to describe how a firewall protects a LAN. Students must provide two linked statements in their response, e.g. 'A firewall uses a set of rules (1), to determine which data to allow into or out of the network.'.

Describe questions are worth between two and four marks and are assess understanding

AO2 – Apply knowledge and understanding of the key concepts and principles of computer science

Questions targeting AO2 require students to apply what they have learned about an aspect of the subject content to a particular context or contexts.

40% of the total marks for the qualification is allocated to this AO – split evenly across the two papers.

SAM Paper 01, Q3(b)(i) is an example of a straightforward question requiring students to apply their knowledge to a given context – in this case a notification of a ransomware attack that appears on a computer screen.

SAM Paper 02, Q02 illustrates how application of knowledge is assessed in the practical paper. Students are told which three lines of code contain syntax errors, so that finding and fixing the errors is relatively straightforward.

Another requirement of *SAM Paper 02, Q02* is to change the identifier of a variableto a more meaningful name. In doing so, students, must apply their understanding of what constitutes a meaningful name in the given context.

SAM Paper 01,Q1(g) illustrates how the requirement to apply understanding may be assessed in Paper 01. Students must calculate the file size of a given bitmap image.

AO3 – Analyse problems in computational terms to make reasoned judgements, and to design, program, test, evaluate and refine solutions

30% of the total marks for the qualification is allocated to AO3, which is assessed in Paper 02 only.

There are two strands to this assessment objective:

3.1. make reasoned judgements

3.2. design, program, evaluate and refine solutions.

'Reasoned judgements' are those based on a logical chain of thinking.

Of the 15 marks allocated to *SAM Paper 02, Q4*, 9 marks are for exercising reasoned judgement and logic in order to rearrange lines of code so that the

correct outputis generated for each value in a given set of test data.

Strand 3.2 is further subdivided into:

3.2a: design solutions

3.2b: program solutions

3.2c: evaluate and refine solutions.

SAM Paper 02, Q06 assesses all three. Students must decompose the problem and design a logical solution, using appropriate programming constructs, variables and data structures (3.2a). Their solution must demonstrate good programming practices, such as use of meaningful identifiers, effective commenting and good use of white space/layout (3.2b). They must produce a functioning program that meets requirements and is fit for audience and purpose, indicating that effective testing/refinement has been carried out (3.2c).

Computer-related mathematics

The two papers each have a minimum of 8% of the marks allocated to the assessment of computer-related mathematics. These are questions that require students to use mathematics for activities, such as:

- Constructing a general expression that uses one or more arithmetic, relational and/or logical operators, such as *SAM Paper 02, Q3,* which requires students to use relational and logical operators to validate input.
- Interpreting conditional statements, such as those used in selection statements or loop terminating conditions, that contain operators, e.g. *SAM Paper 01, Q5(d)*, which requires students to complete a trace table to show the execution of a program.
- Constructing an expression for calculating a file size, e.g. *SAM Paper 01, Q1(h),* or how long it would take to transmit a 20 MiB data file across a network, e.g. *SAM Paper 01, Q2(e)*.
- Constructing a formula to carry out a calculation, e.g. *SAM Paper 02, Q5* that involves calculating the volume of a cone.

Command words

A defined set of command words are used across the papers to indicate the type of response expected – 13 in Paper 01 and just 2 in Paper 02. This means that students know what type of response is expected and can provide an appropriate answer

A list of these command words, their definitions and mark tariffs is provided in Appendix 1 of the specification. These are fixed for the life of the qualification and will be applied by examiners consistently year-on-year.

Students should be encouraged to read questions carefully and should be taught the meaning of the command words used in examination questions and the significance of the number of marks allocated to a question.

Mark schemes

Questions that require relatively short responses – a few words, a couple of sentences, a diagram, a flowchart, or a piece of code – are marked using a pre-determined, points-based mark scheme (PBMS). Those that require longer answers and have a higher mark tariff, often have a levels-based mark scheme (LBMS), since there are multiple valid approaches students can take when answering them.

A LBMS describes three levels of response. Each level is associated with a band of one or more marks. Examiners apply the principle of 'best fit' when deciding what mark to award.

Most items in Paper 01 are marked using a PBMS. However, there is one 6-mark 'essay' question which has a LBMS (see for example SAM Paper 01, Q4(e)).

Paper 02 has some exclusively points-based questions (see for example SAM Paper 02, Q01 and Q02) and some questions that use a combination of points-based and levels-based (see for example SAM Paper 02, Q05, which has one LBMS, Q03 which has two and Q06 which has three). Students receive a mark for each accurate piece of code in their response (points-based) and a mark for the holistic quality of their entire response (levels-based).

Each LBMS in Paper 02 focuses on a different aspect of a student's response – solution design, good programming practices and functionality – all of which students should be doing routinely when programming. A maximum of three marks are awarded for each.

The bulleted descriptors for each level are designed to help examiners decide on the 'best fit'. Not all bullets are relevant to every question.

You can see the LBMS grids in the Sample Assessment Material (SAM). They do not change over time.

7. Planning

GCSE Computer Science is intended to be taught in 120–140 Guided Learning Hours. This equates roughly to two 1-hour lessons a week over two years (or equivalent).

It is important to bear in mind that both exams have the same weighting, so equal curriculum time should be given to cover the subject content for each. One popular approach is to have one 'theory' lesson (focusing on Topics 2–5) and one 'practical' lesson (focusing on Topics 1 and 6) each week.

Computational thinking and problem-solving with programming (Topics 1 and 6) are at the core of the qualification and should be taught in tandem throughout the course.

Opportunities to link theory and practical work should be identified and utilised wherever possible.

Time should be set aside at regular intervals, e.g. each half-term/term, for some form of interim assessment.

A variety of tactics should be employed to prevent students forgetting what they have already learned and avoid cramming it all in at the last minute. This could be by incorporating short reviews of previously taught concepts into ongoing lessons, setting homework tasks that require past learning to be revisited, or by using mixed-topic quizzes that require students to retrieve knowledge spanning the entire GCSE content.

We recommend using the PRIMM approach to develop students' programming skills. PRIMM stands for

- Predict
- Run
- Investigate
- Modify
- Make

Using this approach, students start by reading and understanding code before they progress to writing programs.

8. Support

Pearson is committed to supporting great computer science teaching. We have put together a comprehensive package of **free** support to help you plan and implement the GCSE in Computer Science. It includes help with:

Planning – Our interactive Scheme of Work has a comprehensive set of lesson plans, class and homework activities and solutions that cover all the subject the whole content and is fully customisable.

Understanding the assessment requirements – The sample assessment material will help you familiarise yourself with the format and level of demand of the two exams and understand how your students' papers will be marked. In addition, three sets of specimen papers provide further practice questions and can be used as mocks.

Examiner marked student exemplars – Examiner commentary on student's responses, demonstrating application of the mark scheme and showcasing answers.

Free access to our online ResultsPlus service – provides detailed analysis of your students' exam performance, allowing you to track and analyse and your students' progress.

Free access to marked exam scripts for Paper 01 – so you can easily review your students' performance.

Expert subject advice – Tim Brady, your dedicated Computer Science Subject Advisor, is a direct and personal source of help and support. You can sign up to receive Tim's regular updates, call him on 0333 016 4160 or email him at TeachingICT@pearson.com.

Facebook support group –https://www.facebook.com//groups/140885586105397/

Training events – When it comes to tracking progress and preparing for assessment, we'll provide support and resources to help you and your students throughout the course. Check out are CPD courses on the Pearson Professional Development Academy.

- Planning and delivering the new GCSE Computer Science specification from 2020.
- Pearson Edexcel GCSE (9-1) Computer Science: Introduction to Onscreen Assessment followed by training for each question on paper 2 (6 additional recordings).
- Pearson Edexcel GCSE (9-1) Computer Science: Introduction to Onscreen Assessment Specimen set 1 followed by training for each question on paper 2 (6 additional recordings).
- Introduction to Onscreen Assessment Specimen set 2 followed by training for each question on paper 2 (6 additional recordings).
- Pearson Edexcel GCSE (9-1) Computer Science: Introduction to Onscreen Assessment Specimen (Set 3) followed by training for each question on paper 2 (6 additional recordings).
- Pearson Edexcel GCSE (9-1) Computer Science: Feedback on Summer exams - Components 1 and 2 (one for each summer series)

Wide range of free and paid for resources – new updated versions of the Pearson Edexcel Student Book, an Active book (eBook) version of the Student Book, Revision Guideand Revision Workbook to support the new specification.

Video resources – for those new to the onscreen paper 2 – these videos presented by Tim Brady, subject advisor, are designed to walk through each of the 6 questions in paper 2 SAMs. These can be found on our website under teaching and learning resources: https://qualifications.pearson.com/en/qualifications/edexcel-gcses/computer-science-2020.coursematerials.html#%2FfilterQuery=category:Pearson-UK:Category%2FTeaching-and-learning-materials. Also available are videos on working with text files.

Book an appointment with your subject advisor

March 2020

For information about Edexcel, BTEC and LCCI qualificationsvisit qualifications.pearson.com

Pearson Edexcel is a registered trademark of Pearson Education Limited

Pearson Education Limited. Registered in England and Wales No. 872828
Registered Office: 80 Strand, London WC2R 0RL.
VAT Reg No GB 278 537121