Pearson Edexcel

GCSE (9-1)

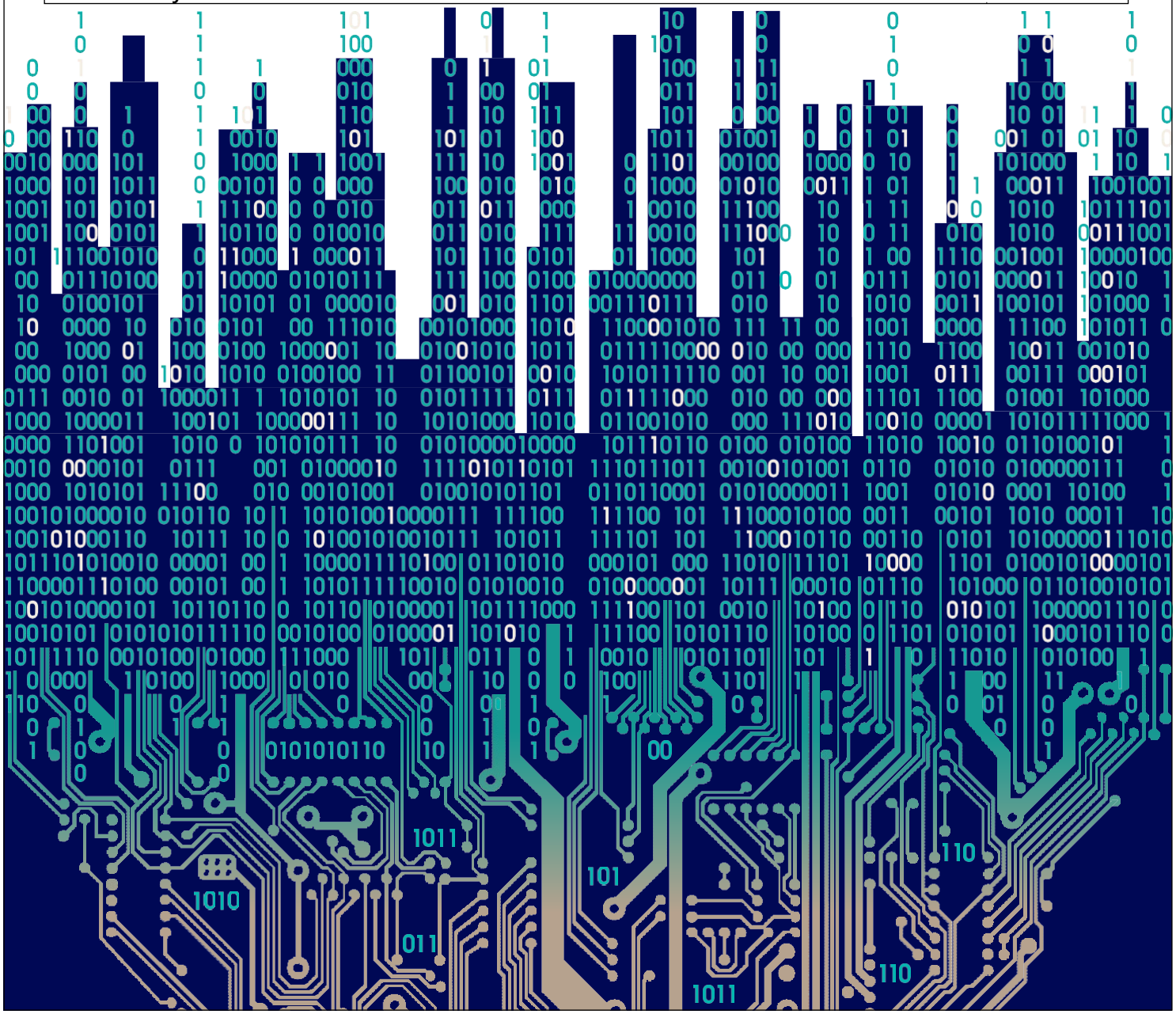
Computer Science

Getting Started Guide

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

First teaching September 2020

First certification 2022 Issue 2



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 on our 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 lives through learning. We put the learner at the centre of everything we do, because wherever learning flourishes, so do people. Find out more about how we can help you and your learners at qualifications.pearson.com

Getting Started: GCSE Computer Science 2020

Contents

Getting Started with GCSE Computer Science

1. Why make changes?

In response to evidence of malpractice in the conduct of the non-examined assessment (NEA) component of GCSE Computer Science, Ofqual decided that – from 2020 onwards – all the prescribed subject content for GCSE Computer Science, including programming skills, will be assessed by formal examination only.

Following this decision by Ofqual, Pearson has been working closely with teachers to decide how best to modify our specification and assessment. We wanted to ensure that students would continue to get a well-rounded experience of computer science, including the practical aspects of the subject, and to incentivise them to learn to program, discovering for themselves the joy – and occasional frustration – associated with problem solving.

Our new 2020 specification provides a stimulating programme of study for students of all ability levels. The assessment consists of just two components – one theory paper and one onscreen practical programming exam.

Teaching of the new specification starts in September 2020 (with the first examination taking place in 2022).

2. Changes to the subject content

We've taken this opportunity to slim down the subject content, so that teachers will have sufficient curriculum time to deliver it. We hope that this will give students a more in-depth understanding of the core principles and concepts of computer science, while still equipping them with the computational thinking skills necessary to progress to A-Level and beyond.

As before, the subject content is divided into six topics. However, some of the topic titles have changed and we've reorganised and renumbered the topics according to which exam they are assessed in.

The revised structure

1. Computational thinking (previously Topic 1 – Problem solving)
2. Data (previously Topic 3)
3. Computers (previously Topic 4)
4. Networks (previously Topic 5 – Communication and the internet)
5. Issues and impact (previously Topic 6 – The bigger picture)
6. Problem solving with programming (previously Topic 2 – Programming)

Where we thought it would be helpful, we've included additional guidance to clarify exactly what needs to be covered.

Summary of the main changes to subject content

Topic	Main changes
1. Computational thinking	• The benefits of using decomposition and abstraction to help solve problems is still covered in this topic. However, the practical application of these techniques has moved into Topic 6, since it is assessed in the programming exam (Paper 02). • Similarly, the need for variables, constants, data structures and operators is covered in this topic and will be assessed in the theory exam (Paper 01). Their practical application is covered in Topic 6 and assessed in Paper 02. • Students no longer need to learn a semi-formal pseudocode language as well as a true high-level programming language. However, they must be able to produce informal written descriptions of algorithms using imprecise English language statements. • Trace tables and truth tables have been relocated here, since they are assessed in Paper 01.

Topic	Main changes
2. Data	• There is no longer any need for students to learn two different ways of representing signed integers. They only need to understand two's complement representation. • The requirement to know how a run-length encoding (RLE) algorithm works has been removed, as have the sections on encryption and databases. • The confusion around which units of measurement to use has been removed. Students will use binary units – kibibytes, mebibytes etc. – to express file sizes and data capacity.
3. Computers	• Students will no longer be tested on the input-process-output model or the function of input/output devices such as printers or screens. • Nor do they need to learn about the concept of storing data in the cloud or on other contemporary secondary storage. • The importance of developing robust software and methods of identifying code vulnerabilities, which used to be in the Communication and the internet topic, have moved into this topic. • The section on logic (truth tables) that used to be in this topic has moved into Topic 1.
4. Networks	• Students no longer need to learn about client-server and peer-to-peer network usage models, but they are still expected to understand types of networks. • They only need to understand the characteristics of three topologies – bus, star and mesh; they are not required to know about the ring topology. • They need to understand that the internet is a global system of interconnected networks that uses the TCP/IP model to link devices worldwide and that routers direct traffic on the Internet. There's no need for them to study the worldwide web. • The section on network security has been slimmed down and some of the content moved elsewhere, notably to the new section on cybersecurity in Topic 5. Students need to understand how penetration testing and ethical hacking help to identify network vulnerabilities and how access control, physical security and firewalls help to protect networks.

Topic	Main changes
5. Issues and impact	• We know that students are keen to explore issues relating to cybersecurity, so we've added a new section to this topic covering cyber threats and methods of protection. • We've also fleshed out the section on ethical and legal matters, focusing on issues associated with the collection and use of personal data, AI, machine learning and robotics.
6. Problem solving with programming	• The practical application of abstraction and decomposition has moved from Topic 1 into this topic. • There is no longer any need to teach students how to produce detailed test plans, more suited to large-scale software development projects. • The Programming Language Subset (PLS) document (see below) provides additional clarification of the programming constructs students need to understand. • Students no longer have a choice of which high-level programming language they can use. Only Python 3 is approved for use.

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.

3. How the assessment has changed

The three assessment objectives set by Ofqual and their weightings remain unchanged. They are:

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%

This has allowed us to keep the same balance of theory and practical application, split between two equally weighted, non-tiered exam papers.

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

There is no longer a requirement for students to undertake a 20-hour programming project, although they should still be given the opportunity to do practical programming (see below).

Paper 01: Principles of Computer Science

This paper has been restructured, so that each question now focuses on just one topic of the subject content, rather than aspects of several different topics. We think that this will be better for students, as they will no longer have to mentally switch from one topic to another within a single question.

Apart from that, the question styles and the subject content covered by the paper remain largely unchanged.

The paper is now marked out of 75, rather than 80, and is 10 minutes shorter than it used to be.

Paper 02: Application of Computational Thinking

We know that students really enjoy learning to program and want their ability to design, write and test code to count towards their qualification grade. So we've replaced the previous paper-based exam with an onscreen version, which we think will give students a more authentic and engaging assessment experience.

The duration of the paper remains unchanged. Students have 2 hours in which to carry out a variety of programming tasks on a computer using Python 3. The tasks increase in complexity – ranging from spotting and correcting errors, filling in gaps, improving readability, rearranging lines of code, to designing and writing a program from scratch.

There are no questions that require a written response.

The benefit of this approach is that students are able to check their solutions work during the exam and examiners can 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 with iGCSE Computer Science, which has a similar onscreen exam, 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.

Reduced set of command words

We have reduced the number of command words that can be used in questions – 15 for Paper 01 and just 2 for Paper 02. This will make it easier for students to know what type of response is expected and reduces the likelihood of them losing marks by not providing an appropriate answer.

A list of these command words, their definitions and mark tariffs, is provided in Appendix 1 of the specification.

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.

4. Content guidance

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 statements. Bracketed lists, such as those in Statement 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 statements 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:	**Guidance**
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 are expected to understand that decomposition involves breaking a complex task down into a number of simpler sub-tasks, each of which performs a specific function, and that abstraction is the process of removing any unnecessary detail from a problem in order to reduce its complexity and make it easier to solve. The emphasis here is on understanding the benefits of using these techniques. Students will gain practical experience of using them when designing and coding their own solutions to problems (see Statement 6.1.1). Questions in Paper 01 may require students to abstract common attributes of a group of objects in order to create a general model. For example, attributes common to all playing cards include suit, rank and colour. Other details such as size and shape can be ignored by a programmer who is creating a general model for a card.
1.1.2 understand the benefits of using subprograms	Students should know that a subprogram carries out a specific task within a larger program. The emphasis here is on understanding the benefits of using subprograms, such as improving the readability of code, making the logic clearer, reducing the time needed to develop a solution, facilitating testing/debugging and being able to reuse the same code many times.

Statement Students should:	Guidance
	Students will get practical experience of using pre-existing (built-in and library) subprograms and of writing their own when they are developing their programming skills (see Section 6.6). Python treats all subprograms as functions, nevertheless students will need to know the difference between functions and procedures.
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 must be able to identify the inputs, processing and outputs used in an algorithm; fill in the gaps of a partially completed algorithm; determine how a given algorithm works; and design algorithms themselves from scratch. Their ability to do so will be assessed in the written exam (Paper 01). Their ability to translate algorithms into code will be assessed in the practical exam (Paper 02). They must be able to interpret and create algorithms represented as flowcharts, pseudocode and program code. The term 'pseudocode' is used to denote an informal written description of a solution that uses imprecise English language statements and does not require the use of any specific command words or syntax. The six flowchart symbols that students are expected to recognise and be able to use are listed in Appendix 2 of the specification. Python is the only approved high-level programming language. The Programming Language Subset (PLS) document specifies the subset of Python 3 commands and constructs students must be familiar with and be able to use. Students should know that repetition involves repeatedly executing a block of code until some specified condition is met, whereas iteration involves repeatedly executing the same block of code on a set of items (such as a list, a string, a numeric range or an open file) until every item has been processed. The program provided in *SAM Paper 01, Q5(d)* uses sequence, selection and iteration constructs to input numbers from a file, process them and output the result. *SAM Paper 01, Q5(e)* is a gap-filling exercise. The flowchart makes use of selection to process an input and produce an appropriate output.

Statement Students should:	Guidance
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)	Although Python does not explicitly distinguish between variables and constants, students are expected to know the difference between them and understand how the use of constants aids program readability and maintainability. *SAM Paper 01, Q5(c)* reinforces this point. It asks for an explanation of why it is good practice to use constants, with the code extract illustrating the convention of using capital letters to denote constants. Students are also expected to understand the concept of a data structure and the characteristics of three different structured data types – strings, records and arrays. In Python, records and arrays are both represented as lists. However, students 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.
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)	This statement lists the arithmetic, relational and logical operators that students are expected to be familiar with and be able to use in algorithms and in program code. The same list appears in Section 6.5. Students must understand the difference between the division operator (/) that divides the first value by the second value and returns the result, including the decimal part; the integer division operator (//) that divides the first value by the second value and returns only the whole number (integer) part of the result; and the modulus operator (%) that divides the first value by the second value and returns the remainder. They must understand the rules of precedence (BIDMAS).

Statement Students should:	Guidance
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	It's not sufficient to know what a trace table is and what trace tables are used for; students are expected to be able to hand trace the values stored in variables as an algorithm executes. *SAM Paper 01, Q5(d)* illustrates how this statement is likely to be assessed.
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 are expected to be familiar with all three error types and know which ones can occur in algorithms and which ones only come to light when a program is being written or executed. Although *SAM Paper 01, Q5(a)* appears straightforward, it does require students to realise that runtime and syntax errors only occur in programs, while logic errors will prevent both algorithms and programs working as intended. In Paper 01 students may be asked to find and correct logic errors in an algorithm.
1.2.6 understand how standard algorithms (bubble sort, merge sort, linear search, binary search) work	Students are expected to understand how each of these algorithms works. Except for the linear search, they do not have to be able to code them. Students should be able to use a while loop to carry out a linear search. The loop condition should test if the search item has been found or the end of the list has been reached. They should understand that a linear search on a sorted list can be made more efficient by identifying when the position the target should be at is passed. The merging of two sorted lists forms part of the merge sort. Students should be able to code the merging of two sorted lists into one sorted list using while loops. They should be able to compare and contrast a linear search with a binary search and a bubble sort with a merge sort. *SAM Paper 01, Q5(b)* illustrates one way that this statement may be assessed.

Statement Students should:	Guidance
	Alternatively, they could be given a practical task to do, such as showing what an unsorted list of values would look like at the end of the first pass through a bubble sort or identifying which values in a sorted list would be compared with the target item during the execution of 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 are expected to be able to evaluate the fitness for purpose of algorithms. They should be able to follow the code and make inferences about its efficiency. They should recognise, for example, that a search algorithm that continues to compare each item in a list with the target value, even after the target value has been located, uses more compares and is less efficient than one that uses a Boolean flag to terminate the search as soon as the target is found. They 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 to carry out its operation. Students are not expected to know about/use Big O notation to measure time or memory efficiency.
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 understand the operation of the AND, OR and NOT logical operators and be aware of the order of precedence rules for logic (brackets, NOT, AND, OR), so as to be able to interpret and construct truth tables to solve problems. 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 just two values, 1 and 0 (corresponding to the electrical states on and off). They should be aware that computers represent, process, store and transmit everything as binary patterns and that the meaning of a group of bits depends on its context – the same binary pattern can represent a number, a character, a pixel or a program instruction. They should know how to determine the number of unique values that can be represented by a binary word of a given length (2^n). *SAM Paper 01, Q1(b)* addresses this requirement.
2.1.2 understand how computers represent and manipulate unsigned integers and two's complement signed integers	Students are expected to understand the difference between signed and unsigned integers and recognise circumstances in which one of them is more appropriate to use than the other. *SAM Paper 01, Q1(c)(i)* illustrates this. They are expected to understand two's complement representation of signed integers. They do not need to know about sign and magnitude representation or how real numbers (fractions) are represented.
2.1.3 be able to convert between denary and 8-bit binary numbers (0 to 255 and –128 to +127)	Conversion of both unsigned and signed denary integers into binary and vice versa is required. *SAM Paper 01, Q1(e)* is mapped to this statement.

Statement Students should:	Guidance
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 positive binary numbers, using a maximum of 8 bits per number. They should be able to perform logical and arithmetic binary shifts and understand why 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. And 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 are expected to know what overflow is and that it occurs when the result of a calculation cannot fit into the size of the location assigned to hold it. They should understand the impact that overflow errors can have on program outcomes. *SAM Paper 01, Q1(d)* is mapped to this statement.
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 that is used as a shorthand for binary and that one hexadecimal digit corresponds to four bits (a nibble) and can represent sixteen unique values (0 – F). They should understand that it is easier for humans to read hexadecimal and write it with fewer mistakes than to deal with long strings of binary numbers. *SAM Paper 01, Q1(f)* illustrates the type of question that could be asked. In (i) candidates are asked to convert binary into hexadecimal. In (ii) they are asked for an explanation of why hexadecimal notation is used. There is no need for them to be able to convert from hex directly to denary.
2.2 Data representation	
2.2.1 understand how computers encode characters using 7-bit ASCII	Students are expected to understand how characters are represented in 7-bit ASCII. They should know that the main limitation of this representation is that it is only capable of encoding 128 characters, 32 of which are control codes.

Statement Students should:	Guidance
	Students do not need to learn about extended ASCII or Unicode, but should be aware that these alternative coding systems permit a wider range of character sets and non-English characters to be represented. They 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 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 resolution is expressed in pixels per inch (ppi) and that it determines the size of an image when displayed, 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". They should be able to explain how resolution impacts on image quality – the greater the resolution, the better the image quality, but the larger the file size. 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. They should be able to construct an expression 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. *SAM Paper 01, Q1(h)* addresses this requirement. Students should be able to convert binary data arranged top-down* using a colour depth of 2^0 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.

Statement Students should:	Guidance
2.2.3 understand how analogue sound is represented in binary (amplitude, sample rate, bit depth, sample interval)	Students should know the difference between an analogue and a digital signal and that the amplitude of an analogue signal is sampled and stored as binary data during the process of analogue to digital conversion. They should know that bit depth is the number of bits used to represent each sound sample and understand the difference between sample rate (the number of samples taken per second, measured in hertz) and sample interval (the time between samples). They should understand why increasing the sampling rate and/or bit depth makes the resulting digital representation more accurately match the original audio and the effect this will have on file size. *SAM Paper 01, Q1(g)* illustrates one way in which this statement may be assessed. Candidates are asked to label a partially completed diagram showing an analogue sound being sampled. Students should be able to construct an expression to calculate 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.
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 bits available determines how a character set, an image, a sound etc., is represented – the more bits there are, the greater the range of unique values.

Statement Students should:	Guidance
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 able to rank the units of measurement in size order and convert from a larger unit to a smaller one and from a smaller unit to a larger one. *SAM, Paper 01, Q1(c)(ii)* illustrates this. They should be able to construct expressions to calculate file sizes or data capacity requirements, expressed as binary multiples. Use of denary multiples, e.g. 1000 (kilobyte) rather than 1024 (kibibyte), is not acceptable. Marks are awarded for demonstrating how a value can be calculated rather than for simply providing the result of carrying out the calculation.
2.3.2 understand the need for data compression and methods of compressing data (lossless, lossy)	Students are expected to understand what data compression is and why it may be desirable/necessary. They should understand the difference between lossless and lossy compression algorithms, the pros and cons of each and why – in some circumstances – the data loss caused by applying a lossy compression algorithm is acceptable, e.g. using a lossy algorithm to compress music files that are streamed online and a lossless one for distributing high quality sound files on a CD. There is no need for them to study specific lossy formats, such as JPEG or MP3.

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 understand that both data and instructions are stored in memory and are fetched, decoded, and executed in a sequence by the CPU. They should be able to explain the role of the main hardware components in the processor. While they do need to know that registers within the CPU provide quickly accessible storage for data, intermediate results, etc., they do not need to be able to name, identify or describe the function of specific registers. They should be familiar with the terms 'uni-directional' and 'bi-directional' and know that, the data and control buses are bi-directional, whereas the address bus is uni-directional. They should know that clock speed is measured in hertz, and that the speed of the clock determines how many instructions the CPU can execute per second. *SAM, Paper 01, Q4(c)* maps to this statement.
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 how the role of secondary storage differs from that of main memory and why both are needed. They should understand how data is physically stored on magnetic, optical and solid-state storage devices and recognise why one type of storage may be more suitable than another for a particular purpose.
3.1.3 understand the concept of an embedded system and what embedded systems are used for	Students should know how an embedded system differs from a general-purpose computer. They should understand the role of a microcontroller in an embedded system.

Statement Students should:	Guidance
	They should be able to describe applications of embedded systems and outline how an embedded system could carry out a particular task, such as controlling the headlights on a car or switching lights off in an unoccupied room. They 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 are expected to understand the four key management functions of an operating system (OS) and be able to describe how an OS carries out each function. *SAM, Paper 01, Q4(d)* hones in on the process management function of the OS. Students should know how files are organised in directories, folders and sub-folders; be familiar with common file management functions such as save, open, rename and delete; and understand how the OS uses permissions to control access to files. They should know how the OS uses scheduling and virtual memory to share out finite resources between competing processes. They should know that the OS uses device drivers to control the operation of peripheral devices. They should know that an OS provides a user interface that enables users to easily interact with a computer and be familiar with authentication techniques, such as passwords and biometrics, used to control user access.
3.2.2 understand the purpose and functionality of utility software (file repair, backup, data compression, disc defragmentation, anti-malware)	Students are expected to understand the purpose of utility software and be able to select appropriate tools for a specific task. *SAM, Paper 01, Q4(a)* is a very straightforward question that maps to this statement. It requires candidates to give two examples of utility software. Section 5.3 also covers anti-malware in the context of protecting digital systems from cyberattacks.

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 that robust software can handle unexpected actions without crashing or producing incorrect output and understand why this is important. They should recognise that criminals can exploit weaknesses present in software to cause damage to data and/or computer systems or to steal sensitive information. They are expected to understand that the purpose of a code review is to identify bad programming practices, inefficient code, security vulnerabilities, adherence to requirements, etc. and that keeping an audit trail helps improve accountability and identify the point at which an error was introduced into the code, enabling it to be rolled back if necessary. *SAM, Paper 01, Q4(b)* asks about the role of code reviews in helping to produce robust software.
3.3 Programming languages	
3.3.1 understand the characteristics and purposes of low-level and high-level programming languages	Students should understand that every processor has its own set of machine code instructions and that a program written in any programming language, other than machine code, must be translated before it can be executed. They should be able to explain how a low-level language, such as assembly language, differs from a high-level language, such as Python, and what each is used for. *SAM, Paper 01, Q4(e)* is a 6-mark essay question that maps to this statement. Candidates must demonstrate that they have a comprehensive understanding of the characteristics of the two languages and their appropriateness for creating the code for an alarm system. Their answer must be relevant to the context, be coherent, logically structured and show sustained lines of reasoning.
3.3.2 understand how an interpreter differs from a compiler in the way it translates high-level code into machine code	Students should know that an interpreter translates high-level program code line by line as it is being executed, whereas a compiler converts the whole program upfront into executable machine code. There is no need for them to have any practical experience of writing/interpreting programs written in machine code or assembly language.

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 must be able to explain what a computer network is and reasons why devices are connected to networks, such as to allow data and peripherals to be shared, to support collaborative working and to enable rapid deployment of new software or updates.
4.1.2 understand different types of networks (LAN, WAN)	Students should understand that a local area network (LAN) is differentiated from a wide area network (WAN) by the size of the area it covers. *SAM Paper 01, Q2(a)* maps to this statement.
4.1.3 understand how the internet is structured (IP addressing, routers)	Students should be aware that the internet is the biggest example of a WAN and be able to explain the role of IP addressing and routers in directing data on the Internet. They should know the processes that take place when a web browser on a user's machine requests a web page from a web server. They should understand that data is split up into packets for transmission across a network and how the contents of a packet header help to ensure reliable and efficient data transfer. *SAM Paper 01, Q2(c)* asks about the contents of the data packets that are routed around the internet.
4.1.4 understand how the characteristics of wired and wireless connectivity impact on performance (speed, range, latency, bandwidth)	Students should understand 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 suitability of wired and wireless methods of connecting devices on a network and be able to explain why most networks use a combination of both.

Statement Students should:	Guidance
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, transmission rate and time	Students should know that network data speeds are measured in bits per second; be familiar with the terms kilobit, megabit and gigabit; and be able to construct expressions involving file size, transfer rate and time. For example, given the file size and the time taken to transmit a file, they should be able to calculate the transmission rate. They should be aware that data speeds are measured in base-10 units whereas file sizes and data capacity use base-2 units. *SAM Paper 01, Q2(e)* is an example. It requires candidates to show how many seconds it will take to transmit 20 MiB of data using a transmission rate of 2 Mbps.
4.1.6 understand the role of and need for network protocols (Ethernet, Wi-Fi, TCP/IP, HTTP, HTTPS, FTP) and email protocols (POP3, SMTP, IMAP)	Students should understand that the purpose of protocols is to establish the rules of communication between devices. They should be able to explain what each protocol is used for. There is no need for them 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 specifies how data is exchanged over the internet by specifying how it should be broken into packets, addressed, transmitted, routed and received at its destination. They should be able to name the four layers of the model, put them in the correct order and describe the main function of each. *SAM Paper 01, Q2(d)* asks about the transport layer. They should know which protocols operate in which layer. • Application: HTTP, HTTPS, SMTP, POP3, IMAP, FTP • Transport: TCP • Internet (or network layer): IP • Link (or network interface): Ethernet Wi-Fi.

Statement Students should:	Guidance
4.1.8 understand characteristics of network topologies (bus, star, mesh)	Students should be able to draw a labelled diagram of each network topology. They should understand how the topology of a network affects its performance, scalability, reliability and security, and be able to select the most appropriate topology for a given scenario. *SAM Paper 01, Q2(b)* requires candidates to complete a diagram of a fully connected mesh network.
4.2 Network security	
4.2.1 understand the importance of network security, ways of identifying network vulnerabilities (penetration testing, ethical hacking) and methods of protecting networks (access control, physical security, firewalls)	Students must understand the importance of network security and be able to explain how internal (white box) and external (black box) penetration testing are used to identify network vulnerabilities. They should know the difference between ethical (white-hat) hackers and criminal (black-hat) hackers and how ethical hacking helps to identify and fix vulnerabilities. They should know how access control, file permissions, physical security and firewalls help to protect networks.

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 be aware of the sustainability issues associated with the manufacture, use and disposal of digital devices, including the amount of energy they consume. They should understand the negative impact of the short replacement cycle of digital devices, particularly mobile phones, and how this can be mitigated.

Statement Students should:	Guidance
	SAM, Paper 01, Q3(a) is straightforward. It asks for two environmental issues associated with the disposal of redundant technology. Students should be able to explain some of the ways in which the environmental impact of digital devices can be reduced.
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 understand the ethical and legal issues associated with the collection and use of personal data. They should understand how the UK Data Protection Act – the UK's implementation of the General Data Protection Regulation (GDPR) – enshrines in law a set of ethical principles that protect individuals against misuse of their personal data, and it confers on them the 'right to be forgotten'. Students should know about the rights of data subjects and the obligations of organisations that hold personal data. They should know about the legislation that governs the use of cookies for collecting personal data. They should know how computer misuse legislation helps protect individuals' personal data by deterring hackers from access it. *SAM, Paper 01, Q3(c)* addresses the data privacy concerns associated with the use of digital assistant digital technologies.
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 some of the ethical concerns and societal implications associated with the growing use of self-learning, autonomous machines, such as surgical robots, driverless cars, automated weapons and assistive robotics. They should realise that, at the present time, there are few legal provisions for these technologies, leaving issues of liability, bias, etc. largely unregulated.

Statement Students should:	Guidance
5.2.3 understand methods of intellectual property protection for computer systems and software (copyright, patents, trademarks, licensing)	Students should understand the difference between copyrights, patents, trademarks and licences and the role of each in helping to protect intellectual property (IP). Whilst they should be aware of the existence of legislation designed to protect IP, they do not need to know the details of any particular laws. They should know that a software licence is a legally binding contract specifying how software can be used and distributed. They do not need a detailed knowledge of different types of licences but should be aware of the two main categories – open source and proprietary.
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 are expected to understand the threat by malware and recognise the tell-tale signs of a malware attack. *SAM, Paper 01, Q3(b)(i)* requires candidates to recognise a ransomware attack and *(b)(ii)* asks them to explain one way in which digital systems may be vulnerable to attack. The mark scheme provides a long list of possible responses. They should be able to give examples of technical vulnerabilities and explain how these can be used to gain unauthorised access to computer systems and networks. They should know the purpose of software patches and why it is important to install them promptly. Students should know how social engineering works and be familiar with pretexting, phishing, baiting and quid pro quo techniques.
5.3.2 understand methods of protecting digital systems and data (anti-malware, encryption, acceptable use policies, backup and recovery procedures)	*SAM, Paper 01, Q3(d)* requires an explanation of how an acceptable use policy helps to protect data. 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 do not need to know the details of particular methods of backup.

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 taught how to use decomposition and abstraction alongside learning how these techniques help to solve problems (see Statement 1.1.1).
6.1.2 be able to read, write, analyse and refine programs written in a high-level programming language	As well as writing their own programs, students must be able to read, analyse and refine programs written by others. *SAM Paper 02, Q1, Q2 and Q3* all require candidates to refine programs by adding or completing lines, correcting errors, improving readability, etc. In *SAM Paper 02, Q4* candidates must rearrange the lines of code in a program in order to get it to work and produce the correct output. *SAM Paper 02, Q6* requires candidates to write and refine a program of their own design.
6.1.3 be able to convert algorithms (flowcharts, pseudocode) into programs	The algorithms may be ones that candidates have created themselves or ones provided for them.
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 realise that, by using these techniques, they help others read their program code and improve its maintainability. Meaningful identifiers should be used to name variables, constants and subprograms. Any sequence of letters, digits and underscores, starting with a letter, can be used as an identifier.

Statement Students should:	Guidance
	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. *SAM Paper 02, Q2* shows one way in which this statement may be assessed. Candidates are required to improve the readability of the code by choosing a more meaningful name for a variable, inserting white space and adding a comment. Students should realise that adding comments always helps the reader, providing they are pertinent and not excessive. *SAM Paper 02, Q4* is less specific about what steps candidates should take, but awards marks for features that aid readability.
6.1.5 be able to identify, locate and correct program errors (logic, syntax, run-time)	Students should understand when and where different types of error occur and how to detect and correct them in programs (see also Statement 1.2.5). *SAM Paper 02, Q2* requires candidates to correct three syntax errors in a program and *SAM Paper 02, Q3* to fix a run-time error and two logic errors.
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)	Normal, boundary and erroneous data should be used to check that programs work correctly. Students should be able to identify ways of improving the efficiency of a program, for example, by reducing the number of comparisons that are made in order to find a value in a list. They should recognise that – from a design perspective – it is more efficient to use a single two-dimensional data structure to store a set of related values than multiple, parallel one-dimensional structures. In tackling *SAM Paper 02, Q4* candidates must use logical reasoning to rearrange the lines of code so that the program produces correct output.

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 document provides more information about the structural components that students are expected to recognise. They should be able to identify occurrences of these structural components in programs. They should be able to use condition-controlled and count-controlled 'WHILE' loops to repeatedly execute a block of code until a specified condition is met. They should be able to use the Python 'FOR…IN' command to iterate through a collection of items, such as a list, a string, a numeric range or an open file, executing the same block of code on each of them.
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 provides more information about the structural components that students should be familiar with and be able to use when writing programs. The use of flow control statements, such as break, continue or exit, not recommended, since they make code more difficult to read and debug and may cause incorrect behaviours elsewhere in a program. In some instances, where readability and/or functionality are affected by the use of these flow control statements, marks could be lost. The use of try/except for exception handling is not mentioned in the PLS. However, some students may want to learn this approach to error handling.

Statement Students should:	Guidance
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) and one- and two-dimensional structured data types (string, array, record)	The *Data types and conversion* section of the PLS includes a table that lists the alternative names used by Python for these primitive data types. In Python, the list data structure is used to store sequences of items with the same data type (arrays) and sequences of items with mixed data types (records). Students should know that the elements of a record are referred to as fields. Students should be able to use indexing to access items in a list and a FOR IN loop to process every item in a list. They are expected to be able to create, append to, and delete from a list. The list methods they should be familiar with can be found in the *Library modules* section of the PLS.
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 and understand how the use of constants aids program readability and maintainability. Python does not support constants in the way most other high-level languages do. However, 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. *SAM Paper 02, Q1* requires candidates 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 set it to 6.
6.3.3 be able to write programs that manipulate strings (length, position, substrings, case conversion)	The string modules students are expected to use are listed in the *Library modules* section of the PLS.

Statement Students should:	Guidance
6.4 Input/output	
6.4.1 be able to write programs that accept and respond appropriately to user input	Input and display output are basic operations that all students should master. They must be able to customise output, formatting strings using the string.format() method with positional parameters (see the *Library modules* section of the PLS for further details). In *SAM Paper 02, Q5* candidates are asked to format the numerical output (volume) to three decimal places. Output messages should be fit for audience and purpose, with attention paid to spelling and punctuation.
6.4.2 be able to write programs that read from and write to comma-separated value text files	The *Inputs and outputs* section of the PLS provides details of the file handling commands students should be able to use. They should know the difference between writing and appending items to a file. They should know how to add a new line and/or carriage return character to a string, and how to remove control characters from and split a comma-separated string read in from a file, so that it can be stored as a set of values in a list. It is important that students are familiar with this aspect of file handling so that they are proficient and confident reading in and using the content of files in their solutions. In some instances, questions in the exam may require candidates to read in data from a file. Where this is the case, the data file will be provided.
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 data validation and be able to incorporate simple data validation checks into their own programs, e.g. checking that an entered string, such as a password, has a minimum length or checking that a value entered lies within a given range. *SAM Paper 02, Q3* requires candidates to add validation to a program so that it only accepts numbers from 1 to 20. Candidates are only expected to use string manipulation functions to check for simple patterns. 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.

Statement Students should:	Guidance
6.4.4 understand the need for and be able to write programs that implement authentication (ID and password, lookup)	Students should be able to write basic authentication routines, such as checking that a correct username and/or password has been entered by comparing the input with a value stored in a list.
6.5 Operators	
6.5.1 be able to write programs that use arithmetic operators (addition, subtraction, division, multiplication, modulus, integer division, exponentiation)	Students should be able to write expressions using arithmetic operators to perform various arithmetic calculations. They must understand the difference between the division operator (/) that divides the first value by the second value and returns the result, including the decimal part; the integer division operator (//) that divides the first value by the second value and returns only the whole number (integer) part of the result; and the modulus operator (%) that divides the first value by the second value and returns the remainder. They must understand the rules of precedence (BIDMAS).
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 write expressions for selection statements and loop terminating conditions using relational operators to compare values and determine the relation between them.
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 in conditional statements for loops and IF statements.

Statement Students should:	Guidance
6.6 Subprograms	
6.6.1 be able to write programs that use pre-existing (built-in, library) and user devised subprograms (procedures, functions)	Python refers to subprograms as functions or methods. The *Library modules* section of the PLS specifies a limited set of library modules and an even smaller set of actual subprograms found in them that students are expected to be familiar with and be able to use. They include four imported modules: random, math, time and turtle. *SAM Paper 02, Q1* requires candidates to import the random library so that the program can simulate the roll of a dice. For *SAM Paper 02, Q5* candidates must complete a user-devised function to calculate the volume of paper cones.
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	Python does not differentiate between procedures and functions. Both may or may not take parameters, but only functions return values to the calling program.
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, global variables, and parameters as local variables They should understand the concept of scope and know that the same identifier can mean different things in different scopes. 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.

5. 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
- Candidates 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 candidates 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
- Candidates 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 – candidates 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 candidates for the functionality of 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.
- Candidates 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 candidates to have access to a printer during the exam, nor should they be provided with printed copies of the secure data files.
- Candidates are not allowed to refer to textbooks, centre-prepared manuals or access shared folders during the examination, but may use the offline 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) 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 students are 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

Questions targeting AO1 require candidates 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. Candidates are asked for two environmental issues associated with the disposal of digital technology. The command word is 'state'.

SAM Paper 01, Q3(b)(ii) is more demanding, requiring candidates 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 because it is unpatched.

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

Questions targeting AO2 require candidates 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 candidates 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, Q2 illustrates how application of knowledge is assessed in the practical paper. Candidates 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, Q2* is to change the identifier of a variable to a more meaningful name. In doing so, candidates, 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. Candidates 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 are for exercising reasoned judgement and logic in order to rearrange lines of code so that the correct output is 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, Q6 assesses all three. Candidates 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 candidates 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 candidates 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 candidates 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

Command words are used consistently in the papers to indicate the type of response expected.

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 candidates 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, Q1 and Q2) and some questions that use a combination of points-based and levels-based (see for example SAM Paper 02, Q5, which has one LBMS, Q3 which has two and Q6 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 focuses on a different aspect of a candidate's response – solution design, good programming practices and functionality – all of which students should be doing routinely when programming. A maximum of three marks can be 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.

6. 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.

7. 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 – Download and customise our sample Scheme of Work to help you plan your course.

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.

We are developing a range of teaching and learning materials to help you prepare for the onscreen assessment, including a suit of exam style questions and potential solutions supported with commentary, similar to those that students will tackle in the exam, with both commentaries and solutions.

Analysing student performance – Our online ResultsPlus service provides detailed analysis of your students' exam performance, helping you to identify topics and skills where students would benefit from further learning.

Help and support – Our subject advisory service, led by Tim Brady, is a direct and personal source of help and support. Sign up to receive Tim's regular e-newsletter, which will keep you up-to-date with qualification and service news. TeachingICT@pearson.com

Training events – We host a series of online training events each year to help teachers deepen their understanding of our qualifications.

Textbooks – new updated versions of the Edexcel Student Book, Revision Guide and Revision Workbook to support the new specification.

March 2020

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

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