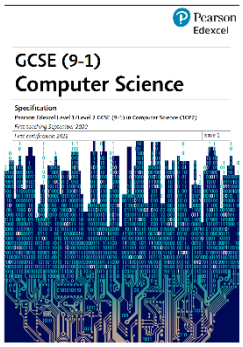
GCSE (9-1) Computer Science New to Pearson

CP2



Learning Objectives

- consider the structure, content and assessment of the qualification, and the support available
- explore possible teaching and delivery strategies
- review the content in the specification and exam techniques
- explore the points-based mark schemes
- explore the levels-based mark schemes.

Subject Content

Topic 1: Computational thinking

Topic 2: Data

Topic 3: Computers

Topic 4: Networks

Topic 5: Issues and impact

Topic 6: Problem solving with programming

The subject content on pages 7 to 14 of the specification details exactly what students have to learn.

Each of the six topics is divided into a number of sections and each section consists of a set of numbered statements. For example (see content on next page):

Section 2.1 is binary. There are six numbered statements specifying what students need to understand and/or be able to do.

Statement 2.1.4 starts with ‘be able to’ – denoting something practical that students have to be able to do – in this case add together two binary numbers and shift the bits in a binary number left and right.

Statement 2.1.5 starts with ‘understand’ – it specifies a piece of theory that students are expected to learn about.

Subject content	Students should:	
2.1 Binary	2.1.1	understand that computers use binary to represent data (numbers, text, sound, 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
	2.1.2	understand how computers represent and manipulate unsigned integers and two's complement signed integers
	2.1.3	be able to convert between denary and 8-bit binary numbers (0 – 255, -127 – 128)
	2.1.4	be able to add together two positive binary patterns and apply logical and arithmetic binary shifts
	2.1.5	understand the concept of overflow in relation to the number of bits available to store a value
	2.1.6	understand why hexadecimal notation is used and be able to convert between hexadecimal and binary
2.2 Data representation	2.2.1	understand how computers encode characters using 7-bit ASCII

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)*

Assessment structure

Paper 1 (1CP2/01)	**Paper 2 (1CP2/02)**
Principles of Computer Science	Application of Computational Thinking
Written examination:	**Practical onscreen examination:**
1.5hrs	2hrs (offline)
50% of the qualification (75 marks)	50% of the qualification (75 marks)

Notes:

Assessment Objectives

Breakdown of Assessment Objectives

Paper	Assessment Objectives			Total % for all Assessment Objectives
	AO1 %	AO2 %	AO3 %	
Paper 1: Principles of Computer Science	30	20	0	50
Paper 2: Application of Computational Thinking	0	20	30	50
Total for GCSE	**30**	**40**	**30**	**100**

Assessment Objectives

Students must:		% in GCSE
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, evaluate and refine solutions.	30
	Total	**100**

Notes:

Content and Assessment – Paper 1 CP2 / 01

Principles of Computer Science

Written examination: 1.5hrs

50% of the qualification (75 marks)

Assessment

- Five compulsory questions
- Each one focused on one topic area

Content

- Computational thinking
- Data
- Computers
- Networks
- Issues and impact

Questions

- MCQs
- Short, medium and extended open response
- Tabular and diagrammatic

Paper 1: Multiple Choice / Response Questions

(ii) Identify the component inside the CPU that stores data.

- ☒ **A** Arithmetic logic unit
- ☒ **B** Clock
- ☒ **C** Main memory
- ☒ **D** Register

(1)

Question Number	Answer	Additional Guidance	Mark
1(a)(ii)	**The only correct answer is D** *A is not correct because the ALU carries out operations, rather than stores data* *B is not correct because the clock is an electrical signal, it does not store data* *C is not correct because main memory is not a component within the CPU*		**(1)**

Notes:

Command word: **Identify**

Paper 1: Short Response Questions

(b) Algorithms can be written in a high-level language.

(i) State what high-level code is translated to.

(1)

.....

Question Number	Answer	Additional Guidance	Mark
1(b)(i)	Award **one** mark for any of the following: • Machine code (1) • Binary (1) • Object code (1) • Executable code / an executable (1)		**(1)**

(ii) State **two** methods of source code translation.

(2)

1 .....

2 .....

Question Number	Answer	Additional Guidance	Mark
1(b)(ii)	Award **one** mark for any of the following up to a maximum of **two** marks: • Compile/Compilation (1) • Interpret/Interpretation (1) • Assemble (1)	Allow compiler/compiling, interpreter/interpreting, assembler/assembling	**(2)**

Notes:

Command word: **Give / State / Name / Define**

Paper 1: Medium Response Questions

(g) One function of an operating system is to manage processes.

(i) Describe **one** way the operating system makes sure each process can use the CPU.

(2)

.....

.....

.....

.....

Question Number	Answer	Additional Guidance	Mark
1(g)(i)	Award **one** mark for any of the following up to a maximum of **two** marks: • A scheduler/scheduling algorithm controls processes (1) • Each process gets a time slice/small amount of (CPU) time (1) • Processes are held in a data structure (1) • Each process may be held in order / priority (1) • Incomplete processes go to the back of the queue (1)	Allow use of 'program' instead of 'process' Award examples e.g. 'round-robin'. For MP3 – award any example of a data structure e.g. array, list, queue.	**(2)**

Notes:

Command word: **Describe / Explain**

Paper 1: Medium Response Questions (cont'd)

(d) A linear search algorithm can be used on both a sorted and an unsorted array.

Describe how a linear search algorithm operates on an **unsorted** array.

(4)

Question Number	Answer	Additional Guidance	Mark
4(d)	Award up to **four** marks for a linked description, such as: • Start at the first position / Iterate/Traverse (through the array) (1), compare the item with the target (1), stop when the target is matched (1), or stop when the end of the list is reached (and the item is not matched) (1)		**(4)**

Notes:

Command word: **Describe / Explain**

Paper 1: Medium Response Questions (cont'd)

(c) Computer programs control some aeroplane landings.

Explain **one** reason computers are **not** capable of safely controlling all aeroplane landings.

(2)

Question Number	Answer	Additional Guidance	Mark
3(c)	Award up to **two** marks for a linked explanation, such as: • The landing program may crash / display incorrect data / not deal with extreme scenarios (weather / wind / animal / visual / emergency) (1) because: • the system could be out of date / have errors / bugs / not been tested enough • the scenario is beyond the range of what it has been programmed to respond to (1) • A computer's input may be wrong/compromised (1) because of a sensor/hardware/mechanical error (1) • A computer cannot make moral/ethical judgements (1) because it is only capable of following instructions (1)	For both marks, the expansion must associate with the statement.	**(2)**

Notes:

Command word: **Describe / Explain**

Paper 1: Extended Response Questions – levels based

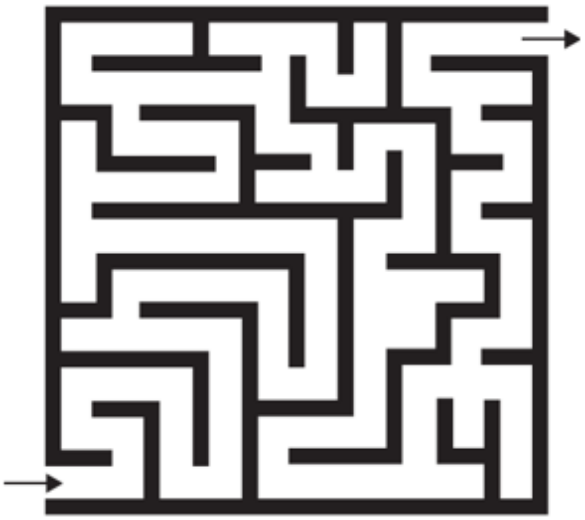
(f) A group of students are working together on a single maze game. The arrow keys control the character. When the character reaches the end of the maze without touching a wall, a happy sound is played. The game also displays a score.

Discuss the use of decomposition and abstraction in developing this game.

Your answer should include:

- a definition of each term
- the benefits each brings to the group of students
- an example of where each could appear in the program code.

(6)



Notes:

Command word: **Discuss**

Question Number	Answer
4(f)	Indicative content: **Definition** • Decomposition is breaking down into smaller parts. Problems, solutions, and algorithms can be decomposed. • Abstraction is the process of removing or hiding unnecessary detail. **Benefits** • It is usually easier to solve several smaller problems, such as checking if touching a wall or updating the score display, than solve one big problem, such as making a game. • Different parts of the program can be shared between the class members to speed up development, for example, one group could work on the code to control the character, while another works on creating and playing the sounds. • Once all the pieces, like sounds, movement, and score are working correctly, the smaller solutions can be combined to make a larger solution, with fewer errors. • The individual parts of the program, such as updating the score can be ignored by the group of students writing the code for moving the character with the arrows / allowing each group of students to focus only on the small problem they have been given means time is not wasted on analysis not relevant to the solution. **Appear in program code** • Different blocks of code logic show decomposition, such as moving the character and updating the score. These blocks could be shown separated by white space. • Subprograms are decompositions because they're blocks of code logic, separated from the main program. Subprograms could be used for updating the score and resetting the character back to the starting position. • Abstraction is used each time a subprogram, either built-in, in a library, or in the code file is called. Library routines in the game would include one to play the sounds and to get the keyboard input. • Subprograms are abstractions because their names hide how they work internally, even if their name is not descriptive. A subprogram to update the score should have a descriptive name, such as updateScore, but could just be called X. Either way, how it works internally is still hidden from the caller.

Level	Mark	Descriptor
	0	No rewardable content.
Level 1	1–2	Basic, independent points are made, showing elements of understanding of key concepts/principles of computer science. (AO1) The discussion will contain basic information with little linkage between points made or application to the context. (AO2)
Level 2	3–4	Demonstrates adequate understanding of key concepts/principles of computer science. (AO1) The discussion shows some linkages and lines of reasoning with some structure and application to the context. (AO2)
Level 3	5–6	Demonstrates comprehensive understanding of key concepts/principles of computer science to support the discussion being presented. (AO1) The discussion is well developed, with sustained lines of reasoning that are coherent and logically structured, and which clearly apply to the context. (AO2)

Notes:

Command word: **Discuss**

(f) An analogue sound is represented in digital form.

The sound is one second long and is sampled at 10 Hz.

The digital representation has a bit depth of 5 and is stored in two's complement.

Sound data:

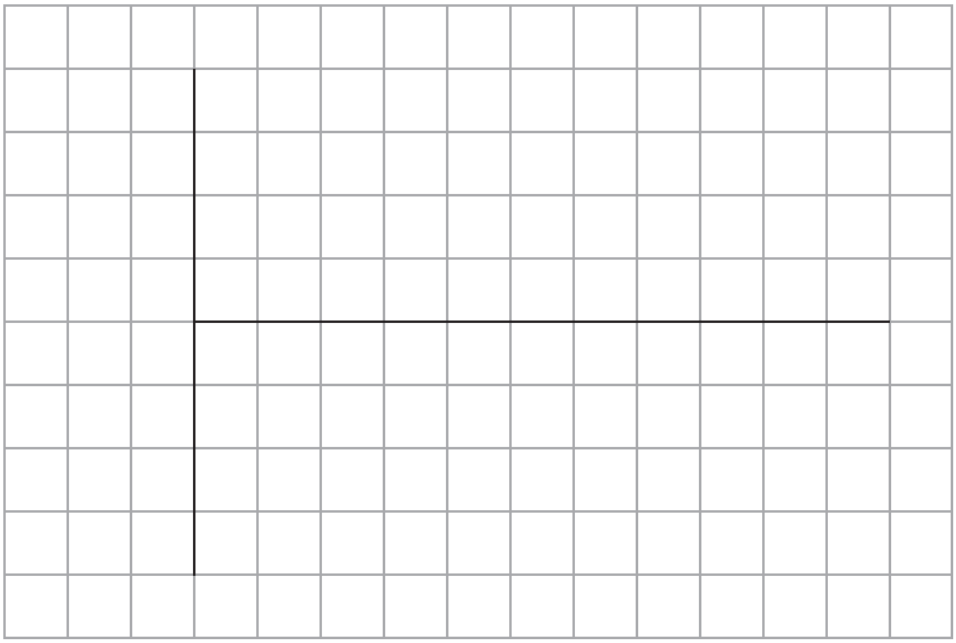
00000 11111 11111 11111 11111

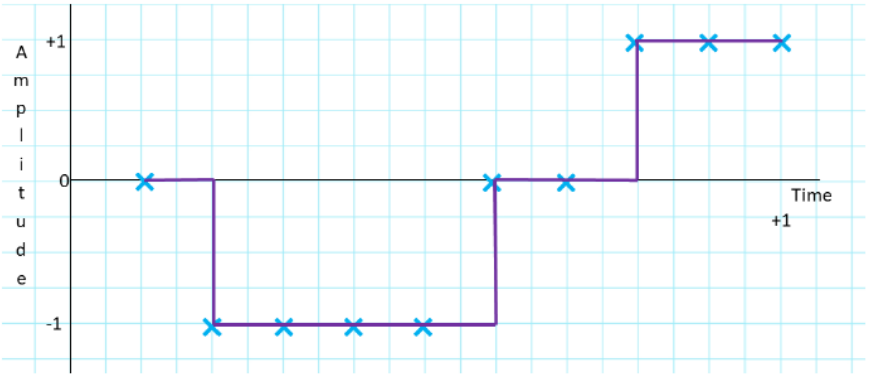
00000 00000 00001 00001 00001

Draw a graph to represent the data sampled.

You must include:

- labels for the x and y axes
- values for the x and y axes
- each sample plotted as an ×
- samples joined up to show the digital form.



Question Number	Answer	Additional Guidance
5(f)	Award **one** mark for any of the following up to a maximum of **six** marks: • x-axis labelled correctly as time/seconds (1) • y-axis labelled correctly as amplitude/value/sample (1) • value labels on x-axis as 0 and 1 (1) • value labels y-axis as -1 and 1 (1) • all 10 values plotted to correct points (1) • points joined to form a square wave, even if not all points are there or some are plotted inaccurately (1) 	Allow first sample at time 0 Award inverse (vertical flip) if y-axis labels also flipped, i.e. must recognise 11111 as -1. Ignore any other values on x and y axes Accept binary representation for value labels on y axis.

Notes:

Command word: **Draw**

Content and Assessment – Paper 2 CP2 / 02

Application of Computational Thinking

Practical onscreen examination: 2hrs

50% of the qualification (75 marks)

Content

- Understanding algorithms, problem solving and reading, writing and refining programs in **Python 3**
- Use of an Integrated Development Environment (IDE)

Only available in Python 3

Assessment

- Modify, design, write, test, refine and evaluate programs in Python 3
- Six compulsory questions onscreen

Questions

- Students are provided with 6 code files and a Program Language Subset.
- Two command words which are **amend** and **write**

Notes:

	Paper 2
Ramped demand within the paper	✓
Command words linked to particular skills and mark tariffs Only 'amend' and 'write' for Paper 2	✓
Clear and consistent levels-based mark schemes	✓
Application of key principles and concepts; being able to analyse problems computationally	✓
Onscreen, with the use of your favoured Integrated Development Environment (IDE) to write and amend programs in response to the questions.	✓

Content and assessment – Paper 2 CP2 / 02

Paper 2 addresses Topic 6: Problem solving with programming. There are 6 sub-topics:

6.1	Develop code	'Be able to use, write, refine……
6.2	Constructs	'Be able to identify, write……..
6.3	Data types and structures	'Be able to write……..
6.4	Input / output	'Be able to write……..
6.5	Operators	'Be able to write……..
6.6	Subprograms	'Be able to write……..

Approach – Paper 2 CP2 / 02

- Offline IDE of choice which should be the normal way of working
- Liaise with your IT support team

Students will be provided with:

- 6 coding files in Python 3
- .txt files as necessary
- Hard copy of question paper
- A paper and electronic copy of The Programming Language Subset (PLS)

Notes:

```
File  Edit  Format  Run  Options  Window  Help
# --------------------------------
# Import libraries
# --------------------------------

# --------------------------------
# Constants
# --------------------------------

# --------------------------------
# Global variables
# --------------------------------

# --------------------------------
# Subprograms
# --------------------------------

# --------------------------------
# Main program
# --------------------------------

```

Selection

`if <expression>:` `  <command>`	If <expression> is true, then command is executed.

`if <expression>:`
`  <command>`
`else:`
`  <command>`
	If <expression> is true, then first <command> is executed, otherwise second <command> is executed.

`if <expression>:`
`  <command>`
`elif <expression>:`
`  <command>`
`else:`
`  <command>`
	If <expression> is true, then first <command> is executed, otherwise the second <expression> test is checked. If true, then second <command> is executed, otherwise third <command> is executed.
	Supports multiple instances of 'elif'.
	The 'else' is optional with the 'elif'.

Notes:

Approach – Paper 2 CP2 / 02 (cont'd)

Repetition

`while <condition>:` `  <command>`	Pre-conditioned loop. This executes <command> while <condition> is true.

Iteration

`for <id> in <structure>:` `  <command>`	Executes <command> for each element of a data structure, in one dimension
`for <id> in range (<start>, <stop>):` `  <command>`	Count-controlled loop. Executes <command> a fixed number of times, based on the numbers generated by the range function

`for <id> in range (<start>, <stop>,`
`  <step>):`
`  <command>`
	Same as above, except that <step> influences the numbers generated by the range function

Notes:

Paper 2: Question 1

Suggested time: 10 minutes

1 A program is being developed to show the average daily temperature and add up the costs of buying ice cream.

- It displays each temperature stored in an array of temperatures.
- It adds up all the ice cream costs entered by the user, until the user enters 0.
- It then calculates a discount. When the total cost is over 100.00, the discount is 10%. Otherwise, the discount is 5%.

Open file **Q01.py**

Amend the lines at the bottom of the code to give the:

- name of a constant used in the program
- name of an array used in the program
- line number of an initialisation of a variable with a real number
- line numbers for a selection construct
- line numbers for a repetition construct
- line numbers for an iteration construct
- line number for an instruction that outputs information to the screen.

Do **not** add any additional functionality.

Save your amended code file as **Q01FINISHED.py**

(Total for Question 1 = 7 marks)

Notes:

Paper 2 Question 1 - Student Code File

```
# ---------------------------------------------------------------------
# Constants
# ---------------------------------------------------------------------
DISCOUNT_5 = 0.05           # 5% discount
DISCOUNT_10 = 0.10          # 10% discount

# ---------------------------------------------------------------------
# Global variables
# ---------------------------------------------------------------------
theTemperatures = [27.7, 29.8, 33.0, 31.6, 28.4, 28.0, 27.9]
aCost = 0.0
total = 0.0

# ---------------------------------------------------------------------
# Main program
# ---------------------------------------------------------------------
for temp in theTemperatures:        # Display temperatures
    print (temp)

# Add up costs until the user enters zero
aCost = float (input ("Enter a cost: "))
while (aCost != 0.0):
    total = total + aCost
    aCost = float (input ("Enter a cost : "))

# Calculate discount based on the total of numbers entered by the user
if (total > 100.00):
    total = (1 - DISCOUNT_10) * total       # Get 10% discount
else:
    total = (1 - DISCOUNT_5) * total        # Get 5% discount

print (total)
# ---------------------------------------------------------------------
# =====> Write your answers here in the right-hand column
# Left                                          # Right
# ---------------------------------------------------------------------
# Name of a constant used in the program        #
# Name of an array used in the program          #
# Line number of an initialisation of a
#      variable with a real number              #
# Line numbers for a selection construct        #
# Line number(s) for a repetition construct     #
# Line number(s) for an iteration construct     #
# Line number for an instruction that outputs
#      information to the screen                #
```

Notes:

Paper 2: Question 1 - Mark Scheme

Question number	MP	Appx. Line	Answer	Additional guidance	Mark
1			Award marks as shown.		
	1.1	37	DISCOUNT_5 / DISCOUNT_10 (1)	Alternative line numbers should be awarded, in the event that students change the code layout by inserting/deleting lines. Award first response only. Do not skip over incorrect response to get to a correct response. Allow spelling/transcription errors. Do not award where a line number is required and a text response is provided. Do not award repetition for iteration or vice versa. Do not award assignments, e.g. line 21, for initialisation, as it is not the first time the variable is used.	
	1.2	38	theTemperatures (1)		
	1.3	40	10 / 11 / 12 (1)		
	1.4	41	27 / 27, 29 / 27-30 (1)		
	1.5	42	22 / 22-24 (1)		
	1.6	43	17 / 17-18 (1)		
	1.7	44	18 / 21 / 24 / 32 (1)		**(7)**

Notes:

Paper 2: Question 2

Suggested time: 15 minutes

2 A program is written for an exercise routine. It displays the names of warm-up exercises, stored in an array. The user enters a number. That number of exercises is selected randomly from the array and displayed.

Open file **Q02.py**

Amend the code to:

- fix the syntax error on original line 4
```
import randum
```
- fix the syntax error on original line 16
```
for exercise in exerciseTable
```
- complete original line 20 to generate a random number between 0 and 4
```
index = random.
```
- complete original line 8 to make the exercise names be string data types
```
exerciseTable = ["squats", "planks", pushups,
                 "lunges", "burpees"]
```
- fix the IndexError on original line 21
```
name = exerciseTable[index + 1]
```
- fix the NameError on original line 22
```
print (naime)
```
- fix the logic error on original line 19 that causes one less exercise to be printed than is asked for
```
for count in range (numExercises - 1):
```
- use white space to improve the readability of the code.

Do **not** change the functionality of the given lines of code.

Do **not** add any additional functionality.

Save your amended code file as **Q02FINISHED.py**

(Total for Question 2 = 8 marks)

Notes:

Paper 2: Question 2 – Student Code File

```
# -----------------------------------------------------------------
# Import libraries
# -----------------------------------------------------------------
import randum
# -----------------------------------------------------------------
# Global variables
# -----------------------------------------------------------------
exerciseTable = ["squats", "planks", pushups, "lunges", "burpees"]
index = 0
name = ""
numExercises = 0
# -----------------------------------------------------------------
# Main program
# -----------------------------------------------------------------
print ("Here is the exercise table")
for exercise in exerciseTable
    print (exercise)
numExercises = int (input ("How many exercises do you need (1-5)? "))
for count in range (numExercises - 1):
    index = random.
    name = exerciseTable[index + 1]
    print (naime)
```

Notes:

Exemplar response:

```
# -----------------------------------------------------------------
# Import libraries
# -----------------------------------------------------------------
import random

# -----------------------------------------------------------------
# Global variables
# -----------------------------------------------------------------
exerciseTable = ["squats", "planks", "pushups", "lunges", "burpees"]
index = 0
name = ""
numExercises = 0

# -----------------------------------------------------------------
# Main program
# -----------------------------------------------------------------
print ("Here is the exercise table")
for exercise in exerciseTable:
    print (exercise)

numExercises = int (input ("How many exercises do you need (1-5)? "))
for count in range (numExercises):
    index = random.randint (0, 4)
    name = exerciseTable[index]
    print (name)
```

Paper 2: Question 2 – Mark Scheme

Question number	MP	Appx. Line	Answer	Additional guidance	Mark
2			Award marks as shown.	Award equivalent expressions, if accurate and fix the error	
			Import libraries		
	2.1	4	Misspelling of randum changed to random (1) Original: `import randum` Amended: `import random`		
			Global variables		
	2.2	8	Single/double quotes added to 'pushups' in array (1) Original: `exerciseTable = ["squats", "planks", pushups, "lunges", "burpees"]` Amended: `exerciseTable = ["squats", "planks", "pushups", "lunges", "burpees"]`		
			Main program – printing akk exercises		
	2.3	16	Missing colon added to for loop (1) Original: `for exercise in exerciseTable` Amended: `for exercise in exerciseTable:`		
			Main program – choosing exercises		
	2.4	19	-1 removed (1) Original: `for count in range (numExercises - 1):` Amended: `for count in range (numExercises):`		
	2.5	20	Call to random completed with randint (0,4) (1) Original: `index = random.` Amended: `index = random.randint (0, 4)`		
	2.6	21	+1 removed (1)		**(8)**

			Original: `name = exerciseTable[index + 1]` Amended: `name = exerciseTable[index]`		
	2.7	22	Misspelling of naime changed to name (1) Original: `print (naime)` Amended: `print (name)`		
			Whole code file		
	2.8	-	At least one additional use of white space, in a correct location that improves readability (1)	Ignore excessive white space	

Notes:

Paper 2: Question 3

Suggested time: 20 minutes

3 A program is used in a shop that sells building materials.

The program reads in data about screws from a file. The data file is provided.

The program counts the number of copper screws.

The program stores the names of 12 bricks in an array. It writes the names of the bricks to a different file, one name per line. Brick names must be in uppercase.

The program displays this output on the screen.

```
Total screws: 26 Copper screws: 5
Wrote 12 brick names to file
```

The output shows 26 screws were read from the file, and five are made from copper. It also shows 12 brick names were written to the file.

Open file **Q03.py**

Amend the code to make the program work and produce the correct output.

You will need to:

- amend some lines of code
- choose between alternative lines of code. Make a choice by removing the # at the beginning of the line you choose to execute
- run the program at least twice and check the output file each time to make sure it meets the requirements.

Do **not** change the functionality of the given lines of code.

Do **not** add any additional functionality.

Save your amended code as **Q03FINISHED.py**

(Total for Question 3 = 15 marks)

Notes:

Paper 2: Question 3 – Student Code File

```
44  # Process the screws
45
46  # =====> Choose the correct line to open the file
47  #inFile = open ("Screws", "r")
48  #inFile = open ("Screws", "a")
49  #inFile = open (INPUT_FILE, "a")
50  #inFile = open (INPUT_FILE, "r")
51
52  for line in inFile:
53      # =====> Choose the correct line to locate the
54      #        substring in the line
55      #if (line.find (SPECIFIED_MATERIAL) == -1):
56      #if (line.find (SPECIFIED_MATERIAL) != -1):
57      #if (line.find (SPECIFIED_MATERIAL) == False):
58      #if (line.find (SPECIFIED_MATERIAL) == True):
59          foundCount = foundCount + 1
60  # =====> Complete the line to increment total
61      total =
62
63  # =====> Choose the correct line to close the file
64  #inFile.close ()
65  #Screws.close ()
66  #INPUT_FILE.close ()
67  #outFile.close ()
```

Notes:

Paper 2: Question 3 – Mark Scheme

Question number	MP	Appx. Line	Answer	Additional guidance	Mark
3			Award marks as shown.	Only one response allowed for MCQ. Do not award if more than one line uncommented.	**(15)**
			Constants		
	3.1	7	Add "Screws.txt", including quotes `INPUT_FILE = "Screws.txt"` (1)	• Must be name of supplied file, which is "Screws.txt"	
	3.2	10	Add .txt to Bricks file name `OUTPUT_FILE = "Bricks.txt"` (1)	• Allow .txt after the quotes • Allow .csv	
			Global variables		
	3.3	16	Add brickTable as name of array before assignment symbol `brickTable = ["Rustic", …` (1)		
	3.4	31	Choose integer initialisation `total = 0` (1)		
	3.5	36	Choose string initialisation: `outLine = ""` (1)		
			Processing copper screws		
	3.6	50	Choose constant file name and open for read only: `inFile = open (INPUT_FILE, "r")` (1)		
	3.7	56	Choose result of find() != -1 `if (line.find (SPECIFIED_MATERIAL) != -1):` (1)		
	3.8	61	Add code to increment total by one `total + 1` (1)	• Allow `total += 1`	
	3.9	64	Choose closing that matches the correct opening on line 48		
			`inFile.close ()` (1)		
	3.10	73	Choose output that will create the one given in the question paper `print ("Total screws: " + str(total) + " " + SPECIFIED_MATERIAL +" screws: " + str(foundCount))` (1)		
			Processing bricks		
	3.11	79	Choose opening the file for writing only `outFile = open (OUTPUT_FILE, "w")` (1)	• As file does not exist on first run, the "a" will create the file. If run again, the program will append bricks, resulting in an incorrect output file.	
	3.12	85	Choose the line to convert the brick name to uppercase `brick = brick.upper ()` (1)		
	3.13	93	Choose the line to add a line feed so each brick name is on a separate line `outLine = brick + "\n"` (1)		
	3.14	98	Choose the line to write the correct variable to the output file `outFile.write (outLine)` (1)		
	3.15	105	Choose output that will create the one given in the question paper `print ("Wrote", len (brickTable), "brick names to file")` (1)		

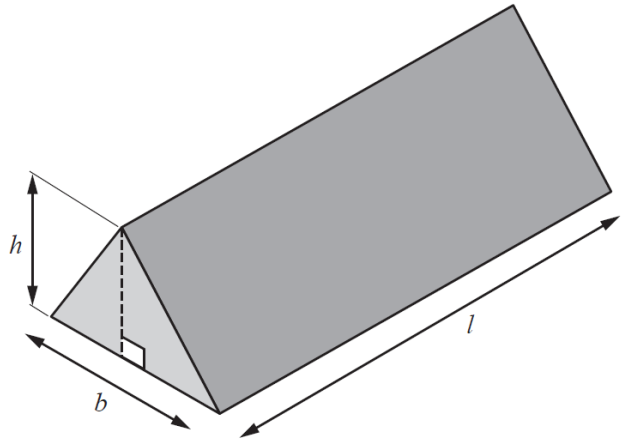
Notes:

Paper 2: Question 4

Suggested time: 25 minutes

4 A program is required to calculate the volume of a prism. All dimensions are entered by the user. The dimensions are decimal numbers greater than 0

The volume of this prism is the area of the triangle multiplied by the length of the prism.



The formula to calculate the area of the triangle is:

$$A = \frac{1}{2} \times b \times h$$

- A is the area of the triangle
- b is the width of the base of the triangle
- h is the height of the triangle

The formula to calculate the volume of this prism is:

$$V = A \times l$$

- V is the volume of the prism
- A is the area of the triangle
- l is the length of the prism

Notes:

Paper 2: Question 4 (cont'd)

The program must meet these requirements:

- take three decimal inputs from the user
  - all inputs must be greater than zero
- check for invalid inputs, using relational and logical operators
- display an error message if an input is invalid. Invalid input should not be processed
- process all valid inputs
- calculate the area of the triangle
- display the area of the triangle, rounded to two decimal places
- calculate the volume of the prism
- display the volume of the prism using the <string>.format() function in eight columns with two decimal places. Include the words 'cubic units' after the volume
- in all cases, display a goodbye message just before terminating.

Test the functionality of the program using the data in this table.

	b	***h***	***l***	**A**	**V**
Prism 1	4.567	1.23	89.01	2.81	250.00
Prism 2	2.74	6.01	5.55	8.23	45.70

Open file **Q04.py**

Amend the code to meet the requirements.

Do **not** add any additional functionality.

Save your amended code as **Q04FINISHED.py**

(Total for Question 4 = 15 marks)

Notes:

Paper 2: Question 4 – Student Code File

```
# ------------------------------------------------------------------------
# Global variables
# ------------------------------------------------------------------------
# =====> Write your code here

# ------------------------------------------------------------------------
# Main program
# ------------------------------------------------------------------------
# =====> Write your code here
# Take three decimal inputs from the user

# Check for invalid inputs, using relational and logical operators

# Display an error message if any input is invalid
# Invalid input should not be processed

# Process all valid inputs
# Calculate the area of the triangle

# Display the area of the triangle, rounded to two decimal places

# Calculate the volume of the prism

# Display the volume of the prism using the <string>.format() function
# in eight columns with two decimal places

# In all cases, display a goodbye message just before terminating

```

Notes:

Paper 2: Question 4 – Levels Based Mark Scheme

			Levels-based mark scheme to a maximum of 3 from:		
	4.13 4.14 4.15		Functionality (3) Execute with test data given in question paper. Execute with negative numbers to check validation.	Considerations for levels-based mark scheme: • Functionality - Translates without syntax and runtime errors • Functionality - Calculations are accurate, regardless of output • Functionality - Output messages are accurate and fit for purpose • Functionality - Fully meets requirements	

Functionality (levels-based mark scheme)

0	1	2	3	Max.
No rewardable material	**Functionality (when the code is run)** • The component parts of the program are incorrect or incomplete, providing a program of limited functionality that meets some of the given requirements. • Program outputs are of limited accuracy and/or provide limited information. • Program responds predictably to some of the anticipated input. • Solution is not robust and may crash on anticipated or provided input.	**Functionality (when the code is run)** • The component parts of the program are complete, providing a functional program that meets most of the stated requirements. • Program outputs are mostly accurate and informative. • Program responds predictably to most of the anticipated input. • Solution may not be robust within the constraints of the problem.	**Functionality (when the code is run)** • The component parts of the program are complete, providing a functional program that fully meets the given requirements. • Program outputs are accurate, informative, and suitable for the user. • Program responds predictably to anticipated input. • Solution is robust within the constraints of the problem.	3

Notes:

Paper 2: Question 4 – Exemplar Answer

Q04 Example 1

```
# ----------------------------------------------------------------
# Global variables
# ----------------------------------------------------------------
# -----> Write your code here

# ----------------------------------------------------------------
# Main program
# ----------------------------------------------------------------
# =====> Write your code here
# Take three decimal inputs from the user
base_width = float(input("What is the width of the base of the triangle? "))
height = float(input("What is the height of the triangle? "))
length = float(input("What is the length of the triangle? "))

# Check for invalid inputs, using relational and logical operators
# Invalid input should not be processed
while base_width <= 0 or height <= 0 or length <= 0:
    # Display an error message if any input is invalid
    print("Invalid input")
    base_width = float(input("What is the width of the base of the triangle? "))
    height = float(input("What is the height of the triangle? "))
    length = float(input("What is the length of the triangle? "))
    # invalid inputs prevented from being processed by a while loop

# Process all valid inputs
# Calculate the area of the triangle
area = height * base_width * 0.5

# Display the area of the triangle, rounded to two decimal places
print("{:2.2f}".format(area))

# Calculate the volume of the prism
volume = area * length

# Display the volume of the prism using the <string>.format() function
# in eight columns with two decimal places
print("{:2.2f}".format(volume))
# In all cases, display a goodbye message just before terminating
print("Goodbye.")
```

Notes:

Paper 2: Question 5

Suggested time: 25 minutes

5 A program is being developed to generate a user identification string.

The letter part of the identification string is made up of the last name joined with the first letter of the first name. All letters must be in lowercase.

The number part of the identification string is the sum of the ASCII values for each of the digits in the date of birth (ddmmyyyy).

The identification string for the user Viola Bassir, born 15th June 2005, is bassirv403, all in lowercase.

Open file **Q05.py**

Amend the code to:

- Ensure local and global variables with the same names are not confused
  - change the names of the local variables to distinguish them from the global variables with the same name
- Welcome the user
  - add a procedure, with no parameters, to display a welcome message for the user
  - call the welcome procedure before taking input from the user
- Ensure the last name and first name are all lowercase
  - convert last name and first name to lowercase after they are inputted by the user
- Validate the date of birth in the main program using the built-in string manipulation subprograms
  - check that only the digits 0 to 9 appear in the date of birth
  - call the makeID() function if the date of birth is valid
  - tell the user if the date of birth is invalid. Invalid input should not be processed
- Generate the correct number part of the identification string in the makeID() function
  - correct the logic error caused by using the int() function in the number part calculation rather than using a function that returns the ASCII value of the character

Do **not** add any additional functionality.

Save your amended code file as **Q05FINISHED.py**

(Total for Question 5 = 15 marks)

Notes:

Paper 2: Question 5 – Student Code File

```
# ----------------------------------------------------------------------
# Global variables
# ----------------------------------------------------------------------
lastName = ""
firstName = ""
dob = ""
myID = ""

# ----------------------------------------------------------------------
# Subprograms
# ----------------------------------------------------------------------
# =====> Change the names of the local variables to distinguish them
#        from the global variables with the same name
def makeID (lastName, firstName, dob):
    namePart = ""
    numberPart = 0

    namePart = lastName + firstName[0]  # Letter part

    # =====> Correct the logic error caused by using the int() function
    #        in the number part calculation rather than using a function
    #        that returns the ASCII value of the character
    for character in dob:
        numberPart = numberPart + int (character)

    yourID = namePart + str (numberPart)

    return (yourID)

# =====> Add a procedure, with no parameters, to display a
#        welcome message for the user

# ----------------------------------------------------------------------
# Main program
# ----------------------------------------------------------------------
# =====> Call the welcome procedure before taking input from the user

# Get last name and first name from the user
lastName = input ("Enter your last name: ")
firstName = input ("Enter your first name: ")

# =====> Convert last name and first name to lowercase after they
#        are inputted by the user

# Get date of birth from the user
dob = input ("Enter your date of birth (ddmmyyyy): ")

# =====> Check that only the digits 0 to 9 appear in the date of birth

# =====> Call the makeID() function, if the date of birth is valid
myID = makeID (lastName, firstName, dob)
print (myID)

# =====> Tell the user, if the date of birth is invalid
```

Notes:

Paper 2: Question 5 – Levels Based Mark Scheme

	5.10 5.11 5.12		Solution design (3)	• Variables in makeID subprogram changed to match new header variable names • Welcome subprogram must have a body (indented line) and no return() statement • Uses [<string>.isdigit()] rather than loop over all characters

Solution design (levels-based mark scheme)

0	1	2	3	Max.
No rewardable material	• There has been little attempt to decompose the problem. • Some of the component parts of the problem can be seen in the solution, although this will not be complete. • Some parts of the logic are clear and appropriate to the problem. • The use of variables and data structures, appropriate to the problem, is limited. • The choice of programming constructs, appropriate to the problem, is limited.	• There has been some attempt to decompose the problem. • Most of the component parts of the problem can be seen in the solution. • Most parts of the logic are clear and appropriate to the problem. • The use of variables and data structures is mostly appropriate. • The choice of programming constructs is mostly appropriate to the problem.	• The problem has been decomposed clearly into component parts. • The component parts of the problem can be seen clearly in the solution. • The logic is clear and appropriate to the problem. • The choice of variables and data structures is appropriate to the problem. • The choice of programming constructs is accurate and appropriate to the problem.	3

Notes:

Paper 2: Question 5 – Levels Based Mark Scheme (cont'd)

	5.13 5.14 5.15		Functionality (3)	• Program translates • Program runs without runtime errors • A welcome message is displayed • Displays an error message if date of birth is invalid • Fully meets requirements

Functionality (levels-based mark scheme)

0	1	2	3	Max.
No rewardable material	**Functionality (when the code is run)** • The component parts of the program are incorrect or incomplete, providing a program of limited functionality that meets some of the given requirements. • Program outputs are of limited accuracy and/or provide limited information. • Program responds predictably to some of the anticipated input. • Solution is not robust and may crash on anticipated or provided input.	**Functionality (when the code is run)** • The component parts of the program are complete, providing a functional program that meets most of the stated requirements. • Program outputs are mostly accurate and informative. • Program responds predictably to most of the anticipated input. • Solution may not be robust within the constraints of the problem.	**Functionality (when the code is run)** • The component parts of the program are complete, providing a functional program that fully meets the given requirements. • Program outputs are accurate, informative, and suitable for the user. • Program responds predictably to anticipated input. • Solution is robust within the constraints of the problem.	3

Notes:

Paper 2: Question 6

Suggested time: 25 minutes

6 A program is required to determine if a user can access a database. The names and passwords of users are stored in a two-dimensional array.

Open file **Q06.py**

Write a program to meet these requirements.

Inputs

- Prompt for and accept a user name and a password
  - neither should be blank

Process

- Implement authentication by searching the array for the user's name and password
  - ensure the search works for any length of array

Output

- Display a suitable message when the correct combination of name and password is found
- Display a suitable message when the user's name is found but the password does not match
- Display a suitable message when the user's name is not found

Use comments, white space and layout to make the program easier to read and understand.

Do **not** add any additional functionality.

Save your amended code as **Q06FINISHED.py**

(Total for Question 6 = 15 marks)

Notes:

Paper 2: Question 6 – Student Code File

```
# ----------------------------------------------------------------------
# Global variables
# ----------------------------------------------------------------------
userTable = [["LArmstrong3", "RedChair"],
             ["SBarrett7", "PurpleDesk"],
             ["EChisholm4", "YellowStool"],
             ["VDunn1", "OrangeFuton"],
             ["DElms5", "GreenCouch"],
             ["EFirsova13", "PinkMattress"],
             ["JGolland6", "GreenTable"],
             ["FHartley13", "BrownMirror"],
             ["DJohnstone12", "GoldBed"],
             ["GKirkhope8", "WhiteNights"],
             ["LLemon8", "BeigeDresser"],
             ["HMacCunn6", "GreyOttoman"],
             ["PNewland10", "BlackWardrobe"],
             ["AOldham5", "OrangeFuton"],
             ["JPook8", "YellowStool"]]

# =====> Write your code here


# ----------------------------------------------------------------------
# Main program
# ----------------------------------------------------------------------

# =====> Write your code here
```

Notes:

Paper 2: Question 6 – Levels Based Mark Scheme

6.10 6.11 6.12		Good programming practice (3)	• Meaningful variable names • Layout with white space improves readability • Commenting is sufficient to completely follow the logic, without being excessive

Good programming practices (levels-based mark scheme)

0	1	2	3	Max.
No rewardable material	• There has been little attempt to lay out the code into identifiable sections to aid readability. • Some use of meaningful variable names. • Limited or excessive commenting. • Parts of the code are clear, with limited use of appropriate spacing and indentation.	• There has been some attempt to lay out the code to aid readability, although sections may still be mixed. • Uses mostly meaningful variable names. • Some use of appropriate commenting, although may be excessive. • Code is mostly clear, with some use of appropriate white space to aid readability.	• Layout of code is effective in separating sections, e.g. putting all variables together, putting all subprograms together as appropriate. • Meaningful variable names and subprogram interfaces are used where appropriate. • Effective commenting is used to explain logic of code blocks. • Code is clear, with good use of white space to aid readability.	3

6.7 6.8 6.9		Solution design (3)	• Solution decomposed into component parts • Stops loop when name and password match found • Stops loop when name match found, but not password • Minimum number of passes through the data (i.e. visits each record only once)

6.13 6.14 6.15		Functionality (3)	• User messages are informative and fit for purpose • Robust, does not crash with syntax or runtime errors • Fully meets requirements, including working with any length of array

Notes:

Summary: Levels Based Mark Scheme by Question

Question 4 – Solution Design

Question 5 – Solution Design, Functionality

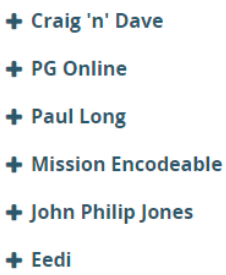
Question 6 – Solution Design, Functionality, Good Programming Practice

Useful resources for planning and delivery

- **Mapping document** – provides side by side comparison between exam boards
  https://qualifications.pearson.com/content/dam/pdf/GCSE/Computer%20Science/2020/teaching-and-learning-materials/evaluation-of-new-gcses-in-computer-science.pdf

- **Scheme of Work** – a superb lesson by lesson schedule which is fully resourced (also as interactive version)
  https://qualifications.pearson.com/content/dam/pdf/GCSE/Computer%20Science/2020/teaching-and-learning-materials/gcse-cs-sow-collation.zip (complete download)

- **Other resources** – free / fee payable
  https://qualifications.pearson.com/en/qualifications/edexcel-gcses/computer-science-2020.resources.html?filterQuery=category:Pearson-UK:Publisher%2FPearson



- **Effective delivery** – important to run practical alongside theory over the duration of the course

- **Video Resources**
  https://qualifications.pearson.com/en/qualifications/edexcel-gcses/computer-science-2020.coursematerials.html#filterQuery=Pearson-UK:Category%2FTeaching-and-learning-materials

Notes:

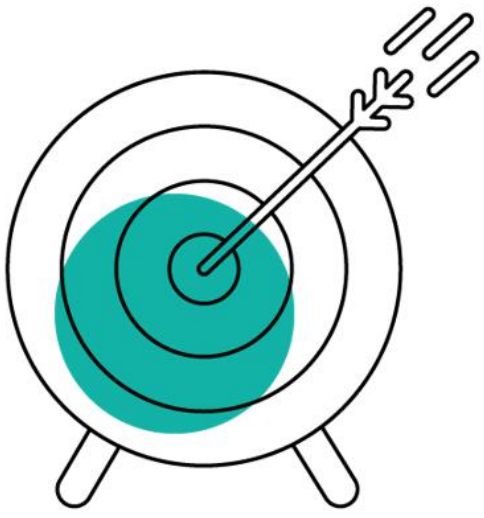
Useful resources for assessment

- SAMs, exam materials (QP, MS and ER) and exemplars https://qualifications.pearson.com/en/qualifications/edexcel-gcses/computer-science-2020.coursematerials.html#%2FfilterQuery=category:Pearson-UK:Category%2FTeaching-and-learning-materials
- Eedi https://diagnosticquestions.com/Questions (account set-up required)

Notes:

You now:

- Know the structure of the course and its assessment
- Have detailed knowledge of both papers 1 (written) and 2 (practical)
- Know methods of delivery to prepare your students to be successful
- Know available tools for planning, delivery and assessment.



Subject Adviser Support

Your subject advisor

Tim Brady

Computer Science and ICT Subject Advisor

Email : TeachingComputerScience@pearson.com
TeachingICT@pearson.com

Facebook : BTEC Tech Award in DIT Facebook group
BTEC Firsts in I&CT Facebook group
T-Level Digital Facebook group
GCSE Computer Science Facebook group

Phone : +44 (0) 344 463 2535
(Teaching Services team | Mon-Fri, 9am-5pm GMT)

- **Book an appointment with your subject advisor**
- **Access the customer support portal**
- **Access the Pearson community**
- **Sign up to receive subject advisor updates**

https://qualifications.pearson.com/en/support/support-for-you/subject-advisors.html

Qualification News and Support

https://qualifications.pearson.com/en/forms/subject-advisor-updates-for-teachers-and-tutors.html

Notes: