

# Pearson Edexcel Level 1/Level 2 GCSE (9–1)

**Specimen Assessment Material for first teaching  
September 2020**

Time: 2 hours

Paper Reference **1CP2/02**

## **Computer Science**

### **Paper 2: Application of Computational Thinking**

#### **You must have:**

- a computer workstation with appropriate programming language code editing software and tools, including an IDE that you are familiar with which shows line numbers
- a 'STUDENT CODING' folder containing code and data files
- printed and electronic copies of the Programming Language Subset (PLS) document.

#### **Instructions**

- Answer all the questions on your computer.
- Save new or amended code using the file name provided and place it in the 'COMPLETED CODING' folder.
- You must **not** use the internet at any time during the examination.

#### **Information**

- The 'STUDENT CODING' folder in your user area includes all the code and data files you need.
- The total mark for this paper is 75.
- The marks for each question are shown in brackets.

#### **Advice**

- Read each question carefully before you start to answer it.
- Save your work regularly.
- Check your answers and work if you have time at the end.

Turn over ►

**S68826A**

©2020 Pearson Education Ltd.

1/1/1/1/1/



  
**Pearson**

**Answer ALL questions.**

**Suggested time: 15 minutes**

- 1** A programmer is learning about subprograms. A program includes subprograms to manipulate numbers and text. It prints out the contents of a data structure and locates a substring in a quote. It does not function correctly.

Open file **Q01.py**

Amend the code to:

- complete original line 16 to create a one-dimensional data structure, implemented as a list, and initialise it with nine integers between 0 and 100, inclusive. The integers can be in any order.  
`myList =`
- complete original line 19 to initialise a string variable to the quote:  
*Better a witty fool than a foolish wit*  
`mySentence =`
- fix the indentation error on original line 38  
`return (location)`
- complete original line 48 with a built-in function to find the length of the string variable 'mySentence'  
`end = mySentence)`
- fix the syntax error on original line 51  
`while (location != -1)`
- complete original line 55 with the keyword for selection  
`(location != -1):`
- fix the syntax error on original line 57  
`print (FOOL + " found at location: " + str location)`
- fix the NameError on original line 44  
`shwList (myList)`
- fix the TypeError on original line 29  
`outString = outString + item + " "`
- fix the logic error on original line 59 that causes an infinite loop.  
`start = location - 1`

Do **not** add any additional functionality.

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

**(Total for Question 1 = 10 marks)**

DO NOT WRITE IN THIS AREA

DO NOT WRITE IN THIS AREA

DO NOT WRITE IN THIS AREA

DO NOT WRITE IN THIS AREA





DO NOT WRITE IN THIS AREA

DO NOT WRITE IN THIS AREA



**Suggested time: 15 minutes**

**2** A program is needed to simulate the pedestrian crossing lights that tell pedestrians when to stop and when to walk. The program repeats this sequence three times.

The hand signal is displayed for 15 seconds.	
The walk signal is displayed for 10 seconds.	
The walk signal, along with a countdown, is displayed for 20 seconds.	 

(Source: © Pearson Asset Library)

Open file **Q02.py**

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

You will need to choose between alternative lines of code given on original lines:

- 5 and 6
- 29 and 30
- 33 and 34
- 37 and 38
- 40 and 41
- 44 and 45
- 47 and 48

Make a choice by removing the # at the beginning of the line you choose to execute.

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 = 10 marks)**



S 6 8 8 2 6 A 0 3 1 2

**Suggested time: 15 minutes**

- 3** The flowchart on the facing page is for an algorithm that performs a presence check validation and length check validations on a name entered by the user. No other validation is required.

The program has these requirements:

- add a comment to identify the code for the presence check
- add a comment to identify the code for a length check
- test the functionality of the program using the data in this table.

Input	Expected output
Empty string	Name cannot be blank
Ab	Name is too short
Rebecca	All checks passed
abcdefghijklmnopqrstuvwxyz	Name is too long

Open file **Q03.py**

Write the code to implement the algorithm in the flowchart.

Do **not** add any additional functionality.

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

**(Total for Question 3 = 10 marks)**

DO NOT WRITE IN THIS AREA

DO NOT WRITE IN THIS AREA

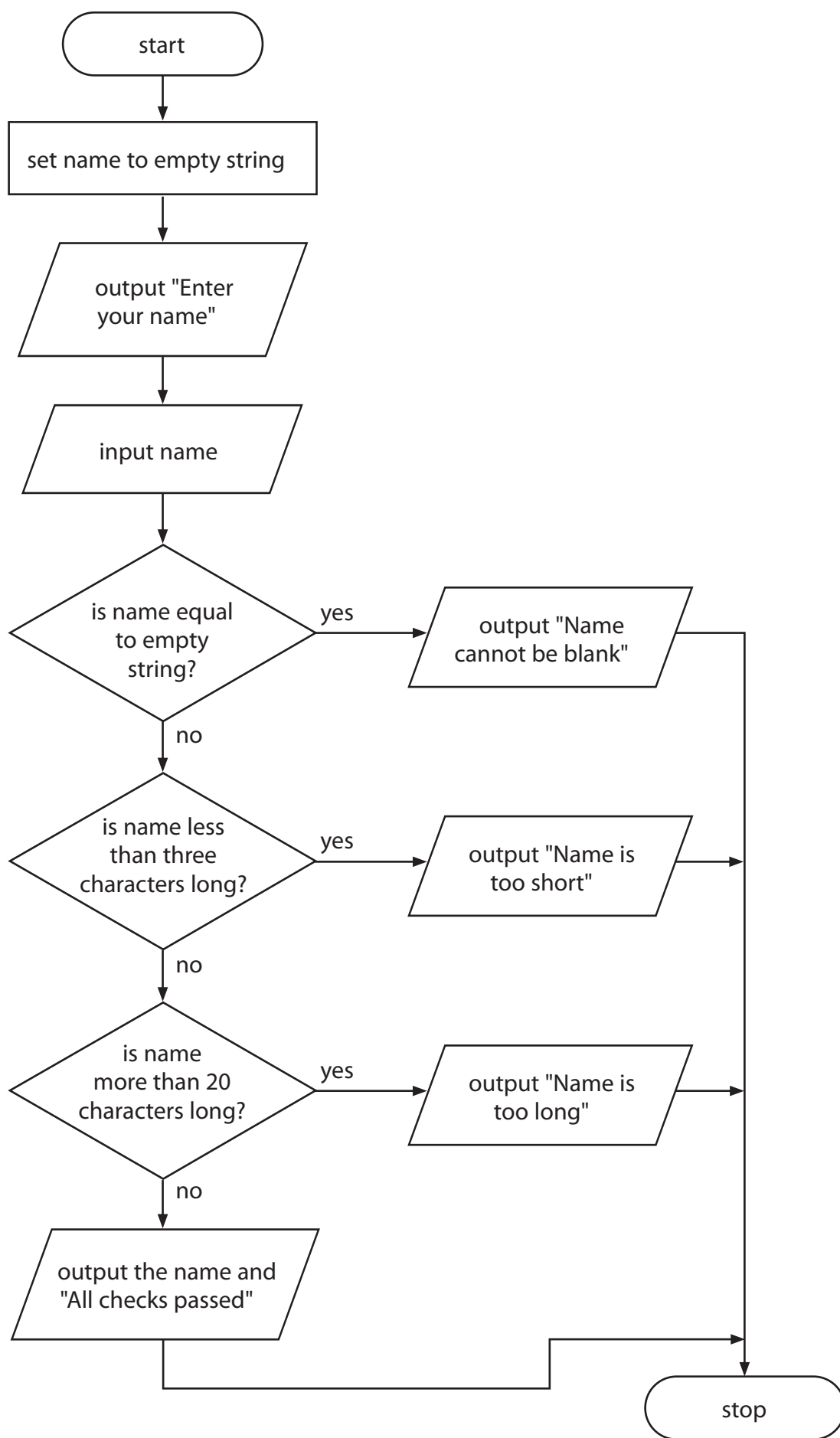
DO NOT WRITE IN THIS AREA

DO NOT WRITE IN THIS AREA

DO NOT WRITE IN THIS AREA

DO NOT WRITE IN THIS AREA





S 6 8 8 2 6 A 0 5 1 2

**Suggested time: 20 minutes**

**4** A program is needed to calculate the number of fabric widths required to make curtains.

The curtain poles are 120, 180, or 240 centimetres long.

The selected fabric is 147 centimetres wide.

The usable fabric width is calculated by the normal fabric width less the width of the selvedge (the unprinted edges) which is 8 centimetres.

The fullness of the curtains can be 2, 2.5, or 3 times the width of the fabric.

The number of fabric widths is calculated by:

$(\text{pole length} \times \text{fullness ratio}) \div \text{usable fabric width}$ , rounded up to the next integer using the ceiling function

The number of fabric widths required for each combination of inputs is shown in this table.

Pole length	Fullness ratio	Fabric widths required
120	2.0	2
120	2.5	3
120	3.0	3
180	2.0	3
180	2.5	4
180	3.0	4
240	2.0	4
240	2.5	5
240	3.0	6

The program must meet the following requirements:

- accept the pole length, no validation required
- accept the fullness ratio, validate as 2, 2.5, or 3
- calculate the usable fabric width
- calculate the number of fabric widths required
- report the number of fabric widths required.

DO NOT WRITE IN THIS AREA

DO NOT WRITE IN THIS AREA

DO NOT WRITE IN THIS AREA

DO NOT WRITE IN THIS AREA

DO NOT WRITE IN THIS AREA

DO NOT WRITE IN THIS AREA



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)



S 6 8 8 2 6 A 0 7 1 2



**Suggested time: 25 minutes**

- 5 A program is needed to track the results from an election. The data includes the party name, the number of votes, and the percentage of votes. The data is stored as structured data. These are implemented as lists.

The program needs to be amended to meet these requirements:

- use the most appropriate list or lists
- initialise two variables
- create formats for displaying the data
- create totals for each numeric column.

This is what the output must look like. Column widths may vary, but alignment and number formats should match this table.

Party	Votes	Percentage
Conservative Party	13830975	43.7
Labour Party	10181243	32.3
Liberal Democrat Party	3564231	11.6
Scottish National Party	1131279	3.9
Green Party	853632	2.8
=====		
Total	29561360	94.3

Open file **Q05.py**

Amend the code to:

- use the most appropriate list or lists
- process and display the data.

Do **not** add any additional functionality.

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

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

**(Total for Question 5 = 15 marks)**





**Suggested time: 30 minutes**

- 6 A program controls a sorting machine for packages based on weight. Gates are opened to direct packages down different paths.

Weight in grams	Path
1 to 100	Green
101 to 750	Yellow
751 to 1000	Red

The maximum number of packages that can be processed in any one hour is 100. At the end of each hour, the program reports sorting information.

Open file **Q06.py**

Write a program to meet the following requirements:

**Inputs**

- Prompt for and accept the number of packages, 1 to 100 inclusive.
  - When invalid input is received, the number of packages should be set to 100

**Process**

- Create a data structure for each coloured path.
- Fill each data structure with random weights, between 1 and 1000, inclusive. The random weights are to be generated by the program.
- Find the number of weights in each data structure.
- Find the maximum weight in each data structure.

**Outputs**

- Display an output line for each path with the colour, the number of weights, and the maximum weight.
- Outputs should be formatted as English sentences with appropriate spacing between words and punctuation.

Decompose the solution, using one or more subprograms.

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)**

**TOTAL FOR PAPER = 75 MARKS**



S 6 8 8 2 6 A 0 9 1 2



DO NOT WRITE IN THIS AREA

DO NOT WRITE IN THIS AREA

DO NOT WRITE IN THIS AREA

DO NOT WRITE IN THIS AREA

DO NOT WRITE IN THIS AREA

DO NOT WRITE IN THIS AREA

BLANK PAGE



DO NOT WRITE IN THIS AREA

DO NOT WRITE IN THIS AREA

DO NOT WRITE IN THIS AREA

DO NOT WRITE IN THIS AREA

DO NOT WRITE IN THIS AREA

DO NOT WRITE IN THIS AREA

BLANK PAGE



S 6 8 8 2 6 A 0 1 1 1 2

DO NOT WRITE IN THIS AREA

DO NOT WRITE IN THIS AREA

DO NOT WRITE IN THIS AREA

BLANK PAGE

