Delegate Booklet

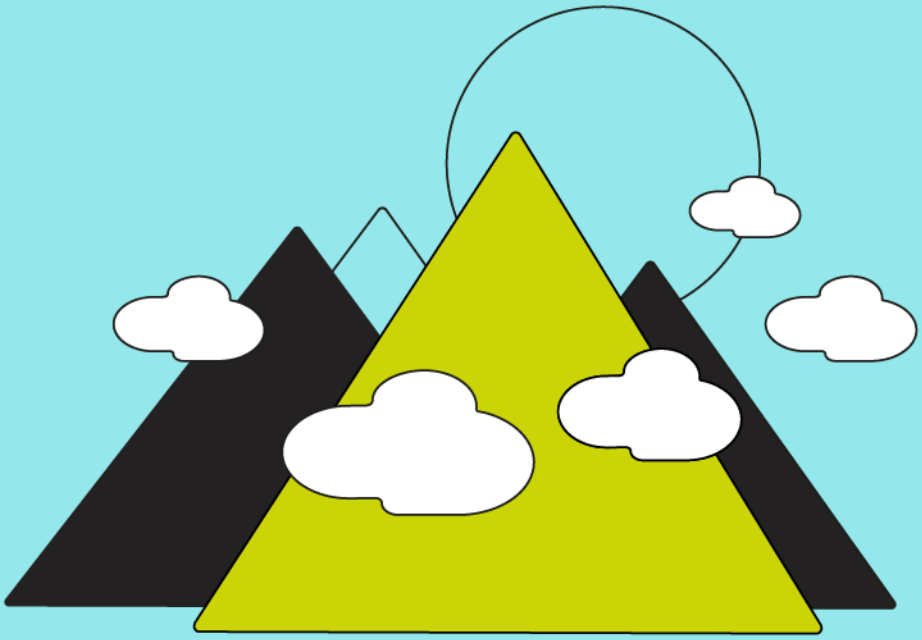
Course Title:

Pearson Edexcel

GCSE Computer Science: Preparing for the summer exam

Course Code:

1CP2-23O2



Pearson

Table of contents

Table of contents 2
Section 2 – Paper 1 3
Two-mark Explain 3
Two-mark Describe 4
Six-mark Discuss 5
Six-mark non-essay 8
Section 3 – Paper 2 10
Name the part 10
Fix the errors 13
Select the lines 19
Translate a flowchart 28
Complete the code 31
Comment-first coding 35
Parson's problem 39
Open 45
Key Documents 49

Section 2 – Paper 1

Two-mark Explain

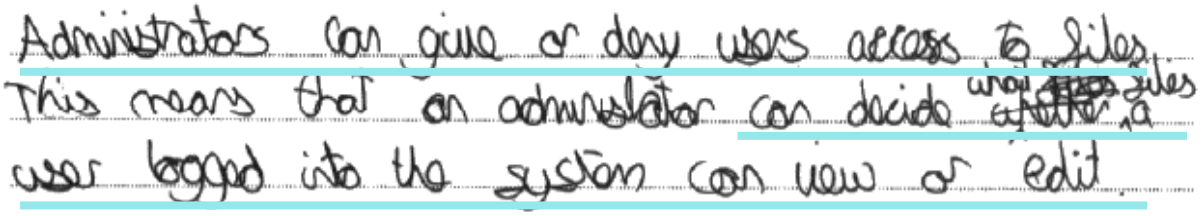
(g) Some users are given administrator privileges. 2022 Q02g

Explain **one** way an operating system allows an administrator to manage users.

(2)

Question Number	Answer
2(g)	A linked explanation such as: • Users can be added/deleted (1) so multiple people can use the same computer (1) • Edit user permissions (1) so only specific users can securely access their storage space (1) • Control the amount of resources/storage each user can access (1) so the limited storage on the machine can be shared (1) • Enforce user permissions (1) so only certain users are allowed to install programs / access certain files (1)

2 marks



Administrators can decide what files a user logged into the system can view or edit (1) **because** they can give or deny users access to files (1)

Model Answer

Two-mark Describe

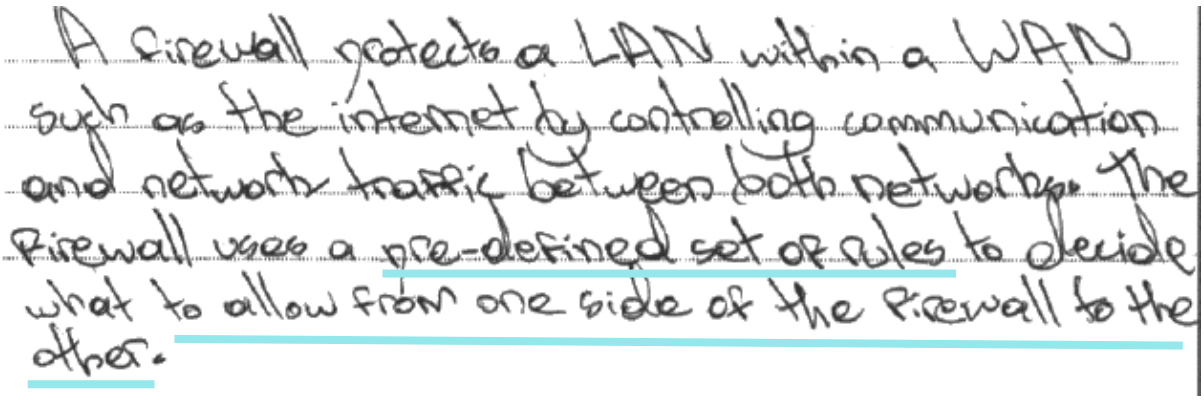
2022 Q01c

(c) Describe how a firewall protects a local area network (LAN).

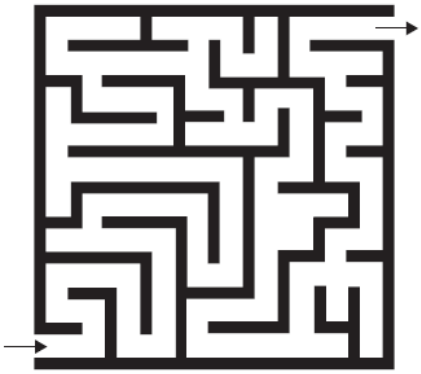
(2)

Question Number	Answer	A
1(c)	A description to include two from: • Monitors/checks traffic (1) using a set of rules / list of authorised/unauthorised computers/addresses/protocols (1) to decide if data is allowed into or out of the network (1)	

2 marks



(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:

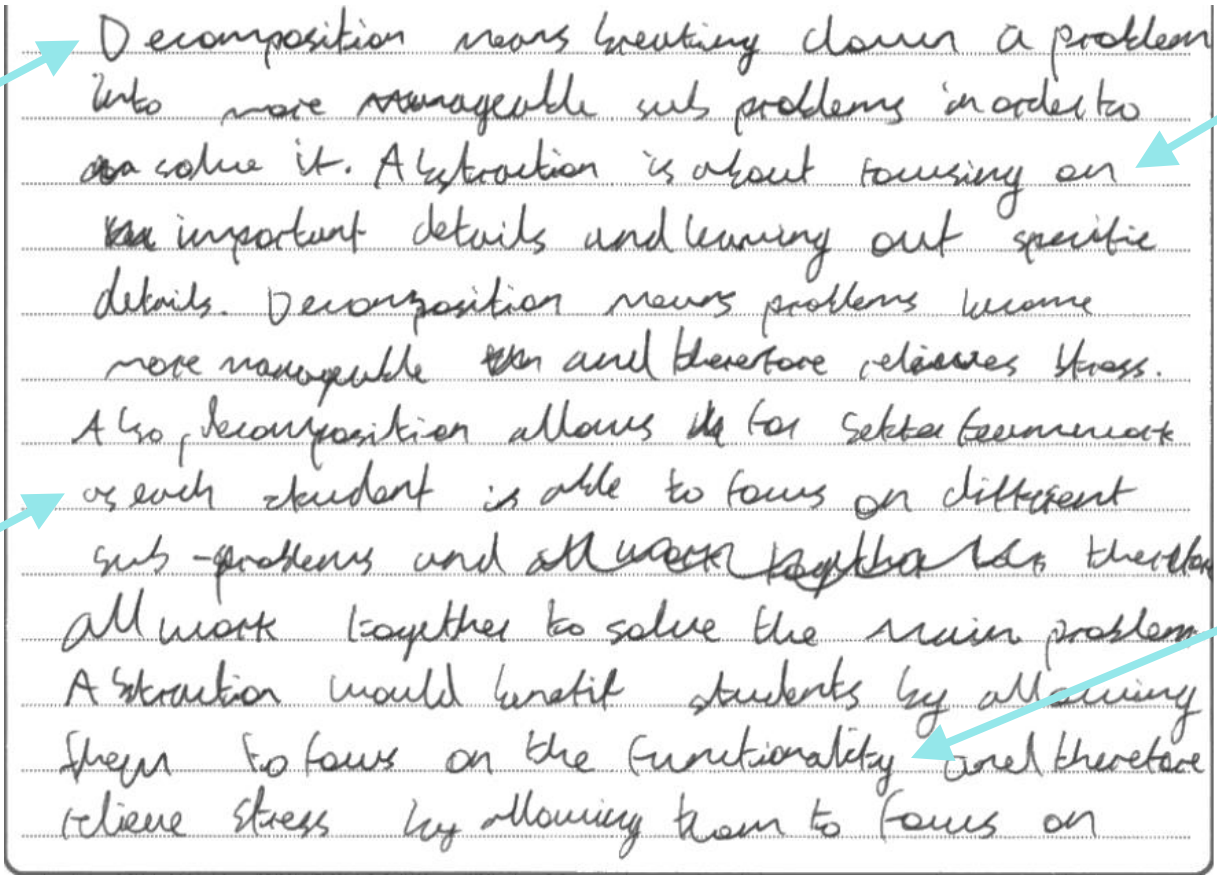
- a definition of each term
- the benefits each brings to the group of students
- an example of where each could appear in the program code.

(6)

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)

6 marks - Comprehensive understanding of the key concepts provided. Well-developed discussion that demonstrates coherent reasoning. There is a logical structure which applies to the context.



Decomposition means breaking down a problem into more manageable sub problems in order to solve it. Abstraction is about focusing on important details and leaving out specific details. Decomposition means problems become more manageable and therefore reduces stress. Also, decomposition allows us for better teamwork as each student is able to focus on different sub-problems and all work together to then all work together to solve the main problem. Abstraction would benefit students by allowing them to focus on the functionality and therefore relieve stress by allowing them to focus on one thing at a time. An example where decomposition could appear in the ~~source code~~ program code is having functions that control each movement of the character. Another example is having one section to keep track of score and another to store it. An example where abstraction could appear in the program code is leaving out a design for the character or having it seperate from the functionality of the character. Another is leaving out animations for movement or adding it after the coding of it to make it easier when coding.

(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

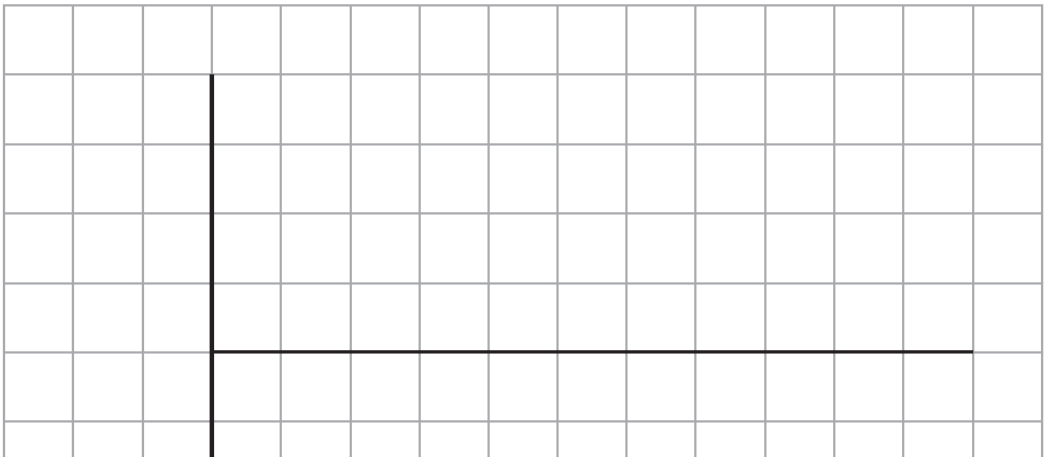
00000 00000 00001 00001 00001

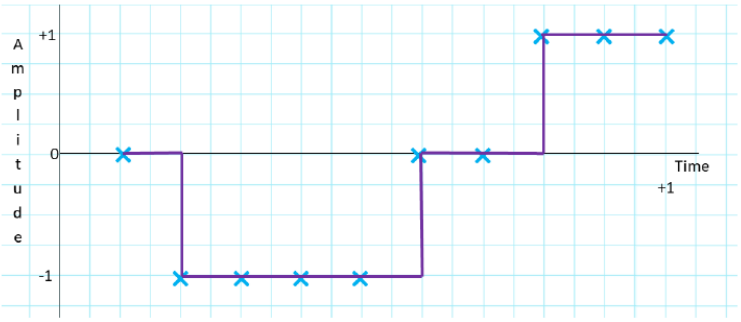
Draw a graph to represent the data sampled.

You must include:

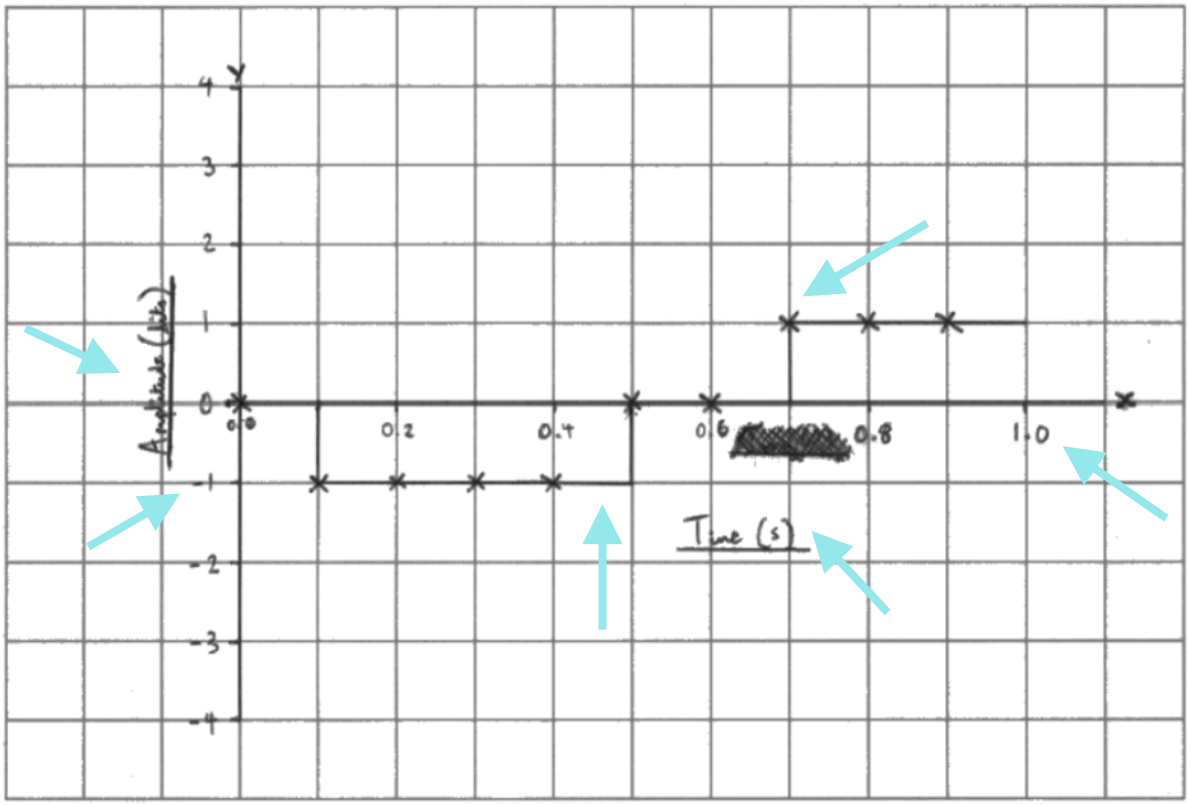
- labels for the x and y axes
- values for the x and y axes
- each sample plotted as an ×
- samples joined up to show the digital form.

(6)



Question Number	Answer	Additional Guidance	Mark
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.	**(6)**

6 marks



Section 3 – Paper 2

Name the part

2023 Q01

Question paper

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)

Student code

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.	**(7)**
	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)		

Student response

```
# ----------------------------------------------------------------
# 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        #DISCOUNT_5
# Name of an array used in the program          #theTemperatures
# Line number of an initialisation of a
# variable with a real number             #21
# Line numbers for a selection construct        #27,29
# Line number(s) for a repetition construct     #22
# Line number(s) for an iteration construct     #17
# Line number for an instruction that outputs
#      information to the screen                #32,18,21,24
```

6 marks

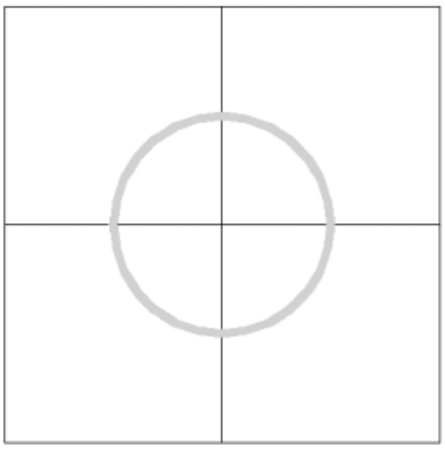
See additional guidance

Fix the errors

Question paper

2022 Q02

Suggested time: 15 minutes



2 A program uses turtle graphics to draw a simple image. This is the image that must be produced.

Both the circle and the outside square are centred on the horizontal and vertical grid lines. The outside square is 400 × 400. The circle is 200 across. The circle outline is coloured gold. All other lines are black.

The program has errors and does not work correctly.

Open file **Q02.py**

Amend the code to:

- add a comment to identify the data type of the argument to the turtle.mode () subprogram call on original line 19
  ```
  turtle.mode ("standard")
  ```
- fix the NameError on original line 23
  ```
  screen.setup (WIDTH, HIGHT)
  ```
- fix the AttributeError on original line 28
  ```
  theTurtle = turtle.turtle ()
  ```
- fix the TypeError on original line 36
  ```
  theTurtle.pendown (200)
  ```
- fix the logic error that causes the vertical axis to be too far right on original line 42
  ```
  theTurtle.setpos (100, 200)
  ```
- fix the logic error that causes the vertical axis to be drawn too short on original line 48
  ```
  theTurtle.forward (100)
  ```
- fix the logic error that causes the outside square to tilt left of the vertical axis on original line 56
  ```
  theTurtle.setheading (95)
  ```
- add a line to set the size of the pen to the constant BIG on original line 68
- add a line to set the colour of the pen to gold on original line 71
- add a line to hide the turtle on original line 78.

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)

Student code

```
# -----------------------------------------------------------------------
# Import libraries
# -----------------------------------------------------------------------
import turtle

# -----------------------------------------------------------------------
# Constants
# -----------------------------------------------------------------------
WIDTH = 800
HEIGHT = 600
BIG = 8

# -----------------------------------------------------------------------
# Main program
# -----------------------------------------------------------------------
# Setup the turtle environment
# =====> Add a comment to identify the data type of the argument
#        to the turtle.mode () subprogram
turtle.mode ("standard")
screen = turtle.Screen ()

# =====> Fix the NameError
screen.setup (WIDTH, HIGHT)
turtle.screensize (WIDTH, HEIGHT)

# Prepare the turtle
# =====> Fix the AttributeError
theTurtle = turtle.turtle ()                # Create a turtle
theTurtle.penup ()

# Draw grid lines
theTurtle.setpos (-200, 0)
theTurtle.setheading (0)

# =====> Fix the TypeError
theTurtle.pendown (200)
theTurtle.forward (400)
theTurtle.penup ()

```

```
# ====> Fix the logic error that causes the vertical axis to be
#       too far right
theTurtle.setpos (100, 200)
theTurtle.setheading (270)
theTurtle.pendown ()

# =====> Fix the logic error that causes the vertical axis
#        to be drawn too short
theTurtle.forward (100)
theTurtle.penup ()

# Draw a square
theTurtle.setpos (-200, -200)               # Lower left

# =====> Fix the logic error that makes the outside square
#        tilt left of the vertical axis
theTurtle.setheading (95)                   # Point north
theTurtle.pendown ()
for count in range (4):
    theTurtle.forward (400)                 # Side
    theTurtle.right (90)                    # Turn
theTurtle.penup ()

# Draw a circle
theTurtle.setpos (100, 0)                   # Right side of circle
theTurtle.setheading (90)                   # Point north

# ====> Add a line to set the size of the pen to the constant BIG


# =====> Add a line to set the colour of the pen to gold


theTurtle.pendown ()
theTurtle.circle (100)                      # Radius of 100
theTurtle.penup ()

# =====> Add a line to hide the turtle


print ("Be sure to close the turtle window.")
turtle.done ()
```

Mark scheme

Question number	MP	Appx. Line	Answer	Additional guidance	Mark
2			Award marks as shown.		
	2.1	19	Any comment with the word "string" in it near the turtle.mode () call (1)		
	2.2	23	Name error – correct spelling of constant HEIGHT (1)		
	2.3	28	Attribute error – Requires a capital letter <turtle>.Turtle () (1)		
	2.4	36	Type error – Remove argument to <turtle>.pendown () (1)		
	2.5	42	Logic error – Move vertical grid line back to origin (1) `theTurtle.setpos (0, 200)`		
	2.6	48	Logic error – Correct length of vertical grid line (1) `theTurtle.forward (400)`		
	2.7	56	Logic error – Correct heading for starting point of square (1) `theTurtle.setheading (90)`		
	2.8	68	Control pen size with a constant (1) `theTurtle.pensize (BIG)`	• Do not allow mark for first added line that uses the default 'turtle' rather than 'theTurtle', even if the rest of the line is correct. Allow follow through.	
	2.9	71	Set the pen colour to "gold" (1) `theTurtle.pencolor ("gold")`		
	2.10	78	Hide the turtle (1) `theTurtle.hideturtle ()`		**(10)**

8 marks

Student response

```
1   # -----------------------------------------------------------------
2   # Import libraries
3   # -----------------------------------------------------------------
4   import turtle
5
6   # -----------------------------------------------------------------
7   # Constants
8   # -----------------------------------------------------------------
9   WIDTH = 800
10  HEIGHT = 600
11  BIG = 8
12
13  # -----------------------------------------------------------------
14  # Main program
15  # -----------------------------------------------------------------
16  # Setup the turtle environment
17  # =====> Add a comment to identify the data type of the argument
18  #        to the turtle.mode () subprogram
19  turtle.mode ("standard")  #
20  screen = turtle.Screen ()
21
22  # =====> Fix the NameError
23  screen.setup (WIDTH, HEIGHT)
24  turtle.screensize (WIDTH, HEIGHT)
25
26  # Prepare the turtle
27  # =====> Fix the AttributeError
28  theTurtle = turtle.Turtle ()                # Create a turtle
29  theTurtle.penup ()
30
31  # Draw grid lines
32  theTurtle.setpos (-200, 0)
33  theTurtle.setheading (0)
34
35  # =====> Fix the TypeError
36  theTurtle.pendown ()
37  theTurtle.forward (400)
38  theTurtle.penup ()
39
```

```
# ====> Fix the logic error that causes the vertical axis to be
#       too far right
theTurtle.setpos (0, 200)
theTurtle.setheading (270)
theTurtle.pendown ()

# =====> Fix the logic error that causes the vertical axis
#        to be drawn too short
theTurtle.forward (400)
theTurtle.penup ()

# Draw a square
theTurtle.setpos (-200, -200)               # Lower left

# =====> Fix the logic error that makes the outside square
#        tilt left of the vertical axis
theTurtle.setheading (90)                   # Point north
theTurtle.pendown ()
for count in range (4):
    theTurtle.forward (400)                 # Side
    theTurtle.right (90)                    # Turn
theTurtle.penup ()

# Draw a circle
theTurtle.setpos (100, 0)                   # Right side of circle
theTurtle.setheading (90)                   # Point north

# ====> Add a line to set the size of the pen to the constant BIG
theTurtle.pensize(BIG)

# =====> Add a line to set the colour of the pen to gold
#theTurtle.pencolor()

theTurtle.pendown ()
theTurtle.circle (100)                      # Radius of 100
theTurtle.penup ()

# =====> Add a line to hide the turtle
theTurtle.hideturtle()

print ("Be sure to close the turtle window.")
turtle.done ()
```

Select the lines

Question paper

2023 Q03

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)

Student code

```
# ----------------------------------------------------------------
# Constants
# ----------------------------------------------------------------
SPECIFIED_MATERIAL = "Copper"

# =====> Add the correct input file name
INPUT_FILE =

# =====> Add the correct extension to this file name
OUTPUT_FILE = "Bricks"

# ----------------------------------------------------------------
# Global variables
# ----------------------------------------------------------------
# =====> Complete the line with the correct variable name for the array of
bricks
 = ["Rustic", "Heather",
                "Staffordshire", "Tudor", "Hampton",
                "Norman", "Northcote",
                "Tuscan", "Regency",
                "Concrete Common",
                "Old English",
                "Hadrian Gold"]

inFile = ""
outFile = ""
foundCount = 0

# =====> Choose the correct value to initialise the variable
#total = 0.0
#total - ""
#total = 0
#total = True

# =====> Choose the correct value to initialise the variable
#outLine = False
#outLine = ""
#outLine = 0.0
#outLine = 0

```

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

# Process the screws

# =====> Choose the correct line to open the file
#inFile = open ("Screws", "r")
#inFile = open ("Screws", "a")
#inFile = open (INPUT_FILE, "a")
#inFile = open (INPUT_FILE, "r")

for line in inFile:
    # =====> Choose the correct line to locate the
    #        substring in the line
    #if (line.find (SPECIFIED_MATERIAL) == -1):
    #if (line.find (SPECIFIED_MATERIAL) != -1):
    #if (line.find (SPECIFIED_MATERIAL) == False):
    #if (line.find (SPECIFIED_MATERIAL) == True):
        foundCount = foundCount + 1
# =====> Complete the line to increment total
    total =

# =====> Choose the correct line to close the file
#inFile.close ()
#Screws.close ()
#INPUT_FILE.close ()
#outFile.close ()

# =====> Choose the correct line to display the output
#print ("Total screws: ", foundCount, "SPECIFIED_MATERIAL", total)
#print ("Total screws: ", total)
#print ("Total screws: " + str (foundCount) + "Copper" + str (total))
#print ("Total screws: " + str (total) + " " + SPECIFIED_MATERIAL + "
screws: " + str (foundCount))

```

```
# Process the bricks

# =====> Choose the correct line to open the bricks file
#outFile = open (OUTPUT_FILE, "r")
#outFile = open (OUTPUT_FILE, "w")
#outFile = open ("Bricks", "r")
#outFile = open (OUTPUT_FILE, "a")

for brick in brickTable:
    # =====> Choose the correct line to convert the case
    #brick = brick.upper ()
    #brick = brick.isalpha ()
    #brick = brick.format ("{}")
    #brick = brick.isupper ()

    # =====> Choose the correct line to complete the output line
    #outLine = brick
    #outLine = brick + "\r"
    #outLine = brick + "\n"
    #outLine = brick + ","

    # =====> Choose the correct line to write the line to the file
    #outFile.writelines (brickTable)
    #outFile.write (outLine)
    #outFile.writelines (brick)
    #outFile.write (brick)

outFile.close ()

# =====> Choose the correct line to display the output
#print ("Wrote", len (brickTable), "brick names to file")
#print ("Wrote", total, "brick names to file")
#print ("Wrote {:^5.2f} brick names to file".format (len (brickTable)))
#print ("Wrote {:^5.2f} brick names to file".format (total))
```

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)		

Display output:

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

Bricks.txt file contents:

```
RUSTIC
HEATHER
STAFFORDSHIRE
TUDOR
HAMPTON
NORMAN
NORTHCOTE
TUSCAN
REGENCY
CONCRETE COMMON
OLD ENGLISH
HADRIAN GOLD
```

Student response

13 marks

```
# ------------------------------------------------------------
# Constants
# ------------------------------------------------------------
SPECIFIED_MATERIAL = "Copper"

# =====> Add the correct input file name
INPUT_FILE = "Screws.txt"

# =====> Add the correct extension to this file name
OUTPUT_FILE = "Bricks.txt"

# ------------------------------------------------------------
# Global variables
# ------------------------------------------------------------
# =====> Complete the line with the correct variable name for the arr
brickTable = ["Rustic", "Heather",
              "Staffordshire", "Tudor", "Hampton",
              "Norman", "Northcote",
              "Tuscan", "Regency",
              "Concrete Common",
              "Old English",
              "Hadrian Gold"]

inFile = ""
outFile = ""
foundCount = 0

# =====> Choose the correct value to initialise the variable
#total = 0.0
#total - ""
total = 0
#total = True

# =====> Choose the correct value to initialise the variable
outLine = False
#outLine = ""
#outLine = 0.0
#outLine = 0

```

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

# Process the screws

# =====> Choose the correct line to open the file
#inFile = open ("Screws", "r")
#inFile = open ("Screws", "a")
#inFile = open (INPUT_FILE, "a")
inFile = open (INPUT_FILE, "r")

for line in inFile:
    # =====> Choose the correct line to locate the
    #        substring in the line
    #if (line.find (SPECIFIED_MATERIAL) == -1):
    if (line.find (SPECIFIED_MATERIAL) != -1):
    #if (line.find (SPECIFIED_MATERIAL) == False):
    #if (line.find (SPECIFIED_MATERIAL) == True):
        foundCount = foundCount + 1
# =====> Complete the line to increment total
    total = total + 1

# =====> Choose the correct line to close the file
inFile.close ()
#Screws.close ()
#INPUT_FILE.close ()
#outFile.close ()

# =====> Choose the correct line to display the output
#print ("Total screws: ", foundCount, "SPECIFIED_MATERIAL", total)
#print ("Total screws: ", total)
#print ("Total screws: " + str (foundCount) + "Copper" + str (total))
print ("Total screws: " + str (total) + " " + SPECIFIED_MATERIAL + "
```

```
# Process the bricks

# =====> Choose the correct line to open the bricks file
#outFile = open (OUTPUT_FILE, "r")
outFile = open (OUTPUT_FILE, "w")
#outFile = open ("Bricks", "r")
#outFile = open (OUTPUT_FILE, "a")

for brick in brickTable:
    # =====> Choose the correct line to convert the case
    brick = brick.upper ()
    #brick = brick.isalpha ()
    #brick = brick.format ("{}")
    #brick = brick.isupper ()

    # =====> Choose the correct line to complete the output line
    #outLine = brick
    #outLine = brick + "\r"
    outLine = brick + "\n"
    #outLine = brick + ","

    # =====> Choose the correct line to write the line to the file
    #outFile.writelines (brickTable)
    #outFile.write (outLine)
    outFile.writelines (brick)
    #outFile.write (brick)

outFile.close ()

# =====> Choose the correct line to display the output
print ("Wrote", len (brickTable), "brick names to file")
#print ("Wrote", total, "brick names to file")
#print ("Wrote {:^5.2f} brick names to file".format (len (brickTable)))
#print ("Wrote {:^5.2f} brick names to file".format (total))

```

Translate a flowchart

Specimen 2 Q03

Question paper

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

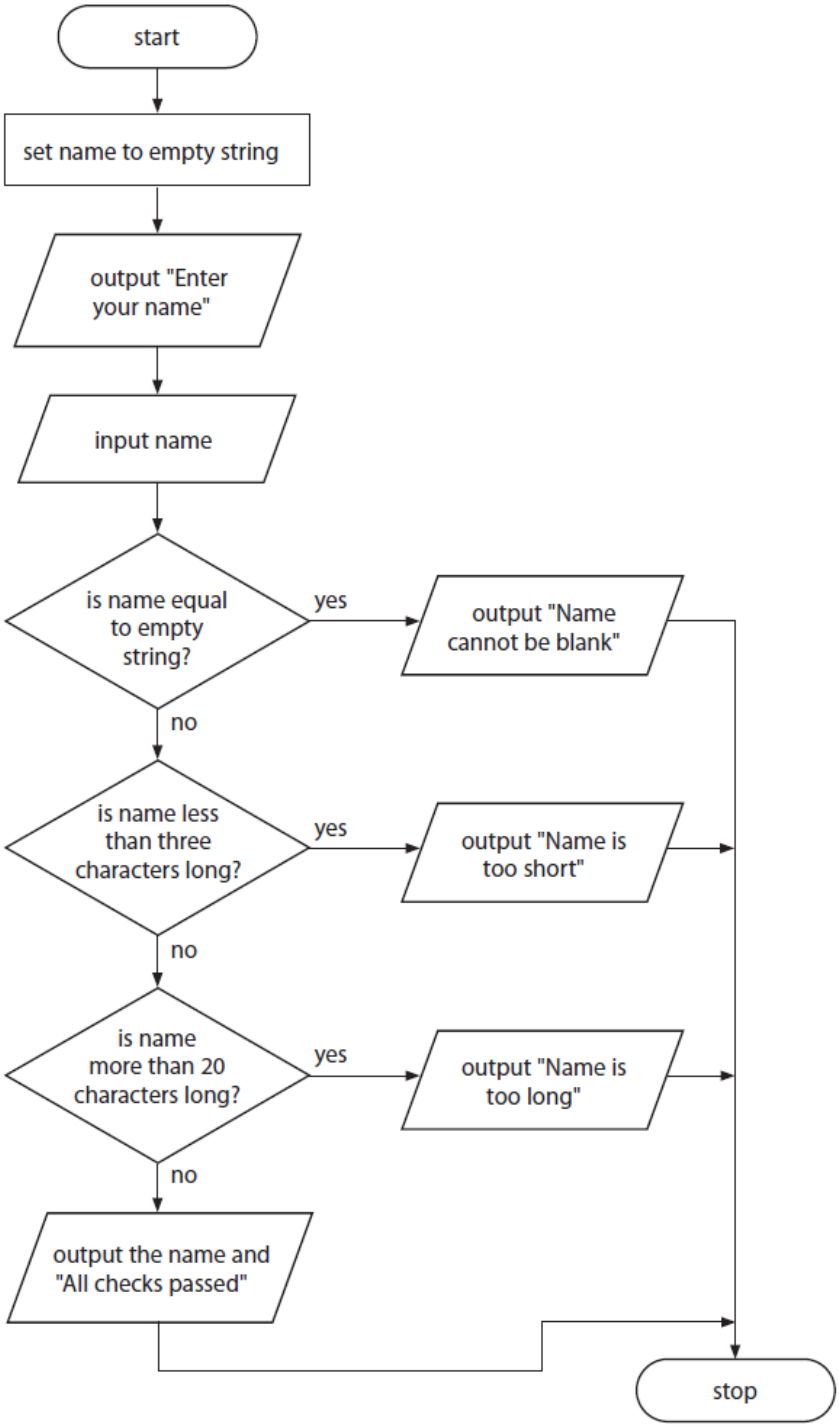
Open file **Q03.py**

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)


start
set name to empty string
output "Enter your name"
input name
is name equal to empty string?
yes
output "Name cannot be blank"
no
is name less than three characters long?
yes
output "Name is too short"
no
is name more than 20 characters long?
yes
output "Name is too long"
no
output the name and "All checks passed"
stop

Student code

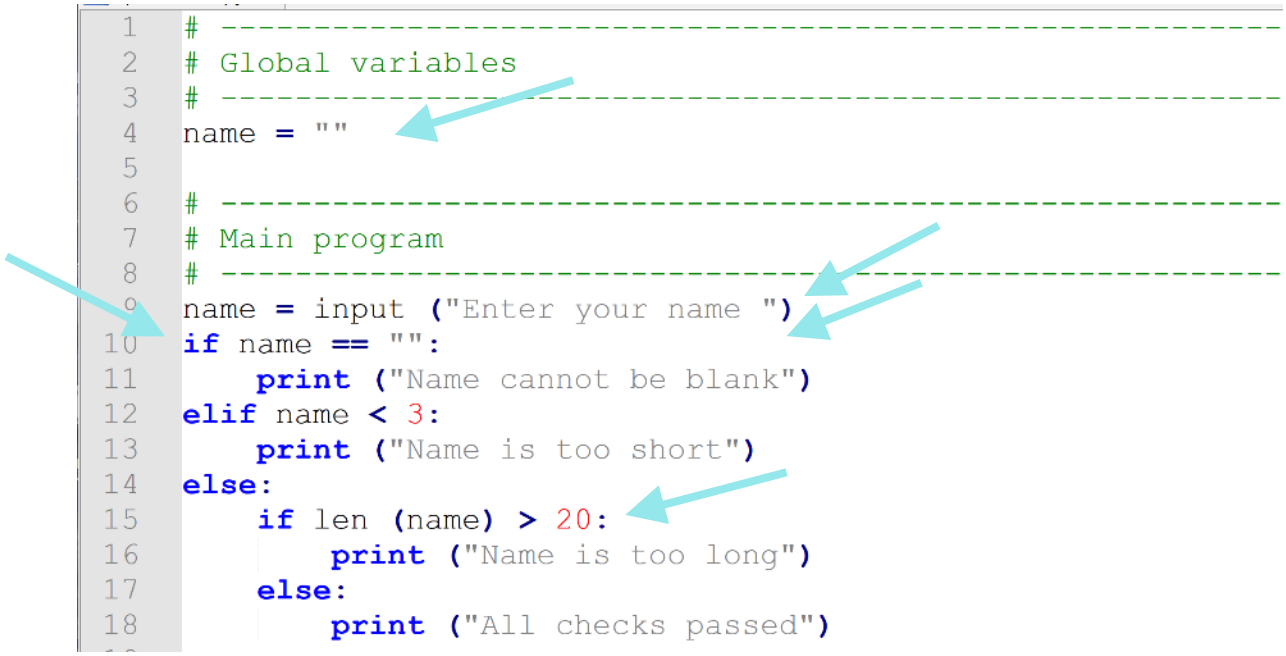
```
# ----------------------------------------------------------------
# Global variables
# ----------------------------------------------------------------

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

Mark scheme

Question number	Answer	Additional guidance	Mark
3	Award marks as shown. • Variable 'name' set to "" (1) • Get value of 'name' from user using 'input' (1) • At least one 'if' statement used (1) • Presence check using "" / len (name) == 0 (1) • Length check using '< 3' (1) • Length check using '> 20' (1) • Use of 'if...elif...elif...else' rather than separate 'if' (1) • Presence check identified in comments and at least one length check identified in comments (1) • Both value of 'name' and 'All checks passed' printed on same line (1) • Functions correctly for normal and erroneous test data, i.e. empty string, name of 2 characters, name of 21 characters, name of 10 characters. (1)		**(10)**

Student response

5 marks



```
# ----------------------------------------------------------------
# Global variables
# ----------------------------------------------------------------
name = ""

# ----------------------------------------------------------------
# Main program
# ----------------------------------------------------------------
name = input ("Enter your name ")
if name == "":
    print ("Name cannot be blank")
elif name < 3:
    print ("Name is too short")
else:
    if len (name) > 20:
        print ("Name is too long")
    else:
        print ("All checks passed")
```

Complete the code

Question paper

Specimen 3 Q01

1 A program must calculate the circumference of a circle. The user enters the radius of the circle. A radius of zero or less is invalid.

The formula to calculate the circumference of a circle is:

$$circumference = 2\pi r$$

- π is the constant Pi
- r is the radius.

Open file **Q01.py**

Amend the code to add or complete lines to:

- import the math library
- create two variables
- take input from the user
- check for an invalid input of zero or less
- display a message to tell the user the input is invalid
- calculate the circumference
- round the circumference to three decimal places using the round() function.

Do **not** add any additional functionality.

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

(Total for Question 1 = 10 marks)

Student code

```
# ----------------------------------------------------------------------
# Import libraries
# ----------------------------------------------------------------------
# =====> Import the math library


# ----------------------------------------------------------------------
# Global variables
# ----------------------------------------------------------------------
# =====> Create an integer variable named radius and set it to 0


# =====> Create a real variable named circumference and set it to 0.0


# ----------------------------------------------------------------------
# Main program
# ----------------------------------------------------------------------
# =====> Complete the line to assign an integer, input by
#        the user, to the variable radius
radius =

# =====> Complete the code in the brackets to check for
#        an invalid radius of zero or less input by the user
if (radius           ):
    # =====> Add a line to tell the user the entry is invalid

else:
    # =====> Complete the calculation of the circumference
    circumference =

    # =====> Complete the line to round circumference to three
    #        decimal places using the round() function
    circumference =

    print ("The circumference is", circumference)
```

Mark scheme

Question number	Answer	Additional guidance	Mark
1	Award marks as shown. • Add a line to import the math library (1) Original: <Blank> Amended: `import math` • Create and set an integer variable (1) Original: <Blank> Amended: `radius = 0` • Create and set a real variable (1) Original: <Blank> Amended: `circumference = 0.0` • Complete the line to take input from the user Original: `radius =` Amended: `radius = int (input ("Enter the radius of a circle: "))` Call to input(), with a prompt (1) Call to int() to convert string to integer (1) • Complete the line to validate for negative radii (1) Original: `radius` Amended: `radius <= 0` • Add a line to print an invalid input message (1) Original: <Blank> Amended: `print ("Invalid radius")` • Complete the calculation of the circumference (1) Original: `circumference =` Amended: `circumference = 2 * math.pi * radius` • Complete the line to round circumference to three decimal places Original: `circumference =` Amended: `circumference = round (circumference, 3)` Call to round() (1) Correct parameters to round() (1)	• Bullet 4: Accept any appropriate string prompt. • Bullet 6: Accept any appropriate message. • Bullet 7: Accept any order for calculation. Accept approximate numeric value of Pi, if math library not used	**(10)**

Student response

9 marks

```
# ----------------------------------------------------------------
# Import libraries
# ----------------------------------------------------------------
# =====> Import the math library
import math

# ----------------------------------------------------------------
# Global variables
# ----------------------------------------------------------------
# =====> Create an integer variable named radius and set it to 0
radius = 0

# =====> Create a real variable named circumference and set it to 0.0
circumference = 0

# ----------------------------------------------------------------
# Main program
# ----------------------------------------------------------------
# =====> Complete the line to assign an integer, input by
#        the user, to the variable radius
radius = int(input("Please enter the radius of the circle: "))

# =====> Complete the code in the brackets to check for
#        an invalid radius of zero or less input by the user
if (radius <= 0):
    # =====> Add a line to tell the user the entry is invalid
    print("Invalid")
else:
    # =====> Complete the calculation of the circumference
    circumference = 2 * (math.pi * radius)

    # =====> Complete the line to round circumference to three
    #        decimal places using the round() function
    circumference = round(circumference, 3)

    print ("The circumference is", circumference)
```

Comment-first coding

Specimen 1 Q05

Question paper

Suggested time: 25 minutes

5 A program is needed to track the number and health scores of native British trees. The tree data is stored as structured data. These are implemented as three lists.

The program needs to be amended to meet these requirements:

- construct a record for each type of tree consisting of fields for name, number and health score
- use commas to separate fields
- write each record to a separate line in an output file.

This is what your final output file should look like. There are exactly 7 lines in the output file.

```
1 Maple,1357,0.15
2 Ash,8421,0.18
3 Sycamore,9287,0.73
4 Birch,1043,0.38
5 Hazel,3743,0.92
6 Willow,2948,0.24
7 Oak,10826,0.78
```

Open file **Q05.py**

Amend the code to:

- process the lists to generate the output file.

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 = 13 marks)

Student code

```
# ------------------------------------------------------------------
# Constants
# ------------------------------------------------------------------
OUTFILE = "Trees.txt"

# ------------------------------------------------------------------
# Global variables
# ------------------------------------------------------------------
treeNames = ["Maple", "Ash", "Sycamore", "Birch", "Hazel", "Willow", "Oak"]
treeCounts = [1357, 8421, 9287, 1043, 3743, 2948, 10826]
treeScore = [0.15, 0.18, 0.73, 0.38, 0.92, 0.24, 0.78]
# ===> Write your code here

# ------------------------------------------------------------------
# Main program
# ------------------------------------------------------------------
# ===> Open the output file for writing


# ===> Use iteration to process each item in treeNames

    # ===> Create a comma separated value output line

    # ===> Write the line to the file

# ===> Close the output file

```

Mark scheme

Question number	Answer	Additional guidance	Mark
5	Award marks as shown. • Use of 'for' or 'while' loop to process every tree (1) • Use of string concatenation to build output strings (1) • Output string contains 3 items in order (name, count, score) (1) • Output string fields separated by commas (1) • Line feed added to end of output string (1) • Use of same or equivalent index variable for tree names, count, and score. (1) • Use of white space and comments aids readability (1) Levels-based mark scheme to a maximum of 6, from: • Solution design (3) • Functionality (3)	• Fixing error with odd numbers can be done in several different ways (see examples) • Award any accurate tests for validation range Considerations for levels-based mark scheme: • [6.4.2] Each tree's information is on a separate line in the output file • [6.2.2] Use of 'for' loop in preference to a 'while' loop for iteration across an entire data structure • [6.4.2] No comma on last field; no "\n" on last record; no additional spaces after commas; i.e. meets requirement of text file • [6.3.1] Same index variable used for all data structures, rather than three different ones. Minimum of one index value is required. • [6.4.2] File opened only for writing • [6.4.2] Close file to match open	**(13)**

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

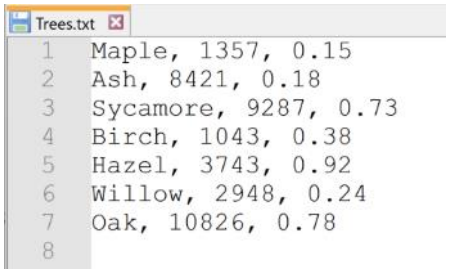
Functionality (levels-based mark scheme)

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

Student response

12 marks

```
# ----------------------------------------------------------------
# Constants
# ----------------------------------------------------------------
OUTFILE = "Trees.txt"

# ----------------------------------------------------------------
# Global variables
# ----------------------------------------------------------------
treeNames = ["Maple", "Ash", "Sycamore", "Birch", "Hazel", "Willow", "Oak"]
treeCounts = [1357, 8421, 9287, 1043, 3743, 2948, 10826]
treeScore = [0.15, 0.18, 0.73, 0.38, 0.92, 0.24, 0.78]
# ===> Write your code here
output = ""
i = 0
# ----------------------------------------------------------------
# Main program
# ----------------------------------------------------------------
# ===> Open the output file for writing
file = open(OUTFILE, "w")

# ===> Use iteration to process each item in treeNames
for i in range(len(treeNames)):
    # ===> Create a comma separated value output line
    output = (treeNames[i] + ", " + str(treeCounts[i]) + ", " +
              str(treeScore[i]) + "\n")
    # ===> Write the line to the file
    file.writelines(output)
# ===> Close the output file
file.close()

```



Parson's problem

Specimen 1 Q03

Question paper

Suggested time: 20 minutes

3 A program calculates the bus fare for passengers. The type of fare is entered using a menu. The fare is calculated and displayed. There is a choice to exit the program instead of calculating a fare.

The program is being developed. Some of the lines of code in the program are mixed up. Some sections of the program include different code options and only one of the options should be included in the program.

Open file **Q03.py**

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

You will need to rearrange lines of code or choose between alternative lines of code.

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

Do **not** add any additional functionality.

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

(Total for Question 3 = 15 marks)

Student code

```
# -----------------------------------------------------------------
# Import libraries
# -----------------------------------------------------------------

# -----------------------------------------------------------------
# Constants
# -----------------------------------------------------------------
# ===> Order the jumbled lines
CONCESSION = RETIRED * 2 / 10 * 4
FULL_FARE = 1.50
RETIRED = 2 * STUDENT
STUDENT = 0.25
CHILD = 0.0
# Finished jumbled lines

# -----------------------------------------------------------------
# Global variables
# -----------------------------------------------------------------
# ===> Choose one line
# layout = "Your fare is {:4.2f}"
# layout = "Your fare is {:4.4f}"

choice = ""
fare = 0.0

```

```
# ------------------------------------------------------------------
# Subprograms
# ------------------------------------------------------------------
def showMenu ():
    print ("----- Fares -----")
    print ("A - Full fare")
    print ("B - Student fare (8 - 16 years)")
    print ("C - Child fare (0 - 7 years)")
    print ("D - Retired (65+ years)")
    print ("E - Other concession")
    print ("Q - Quit")

def getUserChoice ():
    userChoice = ""
    validChoice = False

    # ===> Choose one line
    #while (validChoice):
    #while (not validChoice):
        userChoice = input ("Please choose a fare: ")
        userChoice = userChoice.upper ()
        # ===> Choose one long statement of 3 lines
        #if ((userChoice != "Q") and (userChoice != "A") and
        #        (userChoice != "B") and (userChoice != "C") and
        #        (userChoice != "D") and (userChoice != "E")):
        #if ((userChoice == "Q") or (userChoice == "A") or
        #        (userChoice == "B") or (userChoice == "C") or
        #        (userChoice == "D") or (userChoice == "E")):
            print ("Invalid entry")
        else:
            # Choose one line
            #validChoice = False
            #validChoice = True

    return (userChoice)

def calcFare (pCategory):
    theFare = 0.0

    # ===> Order and indent the jumbled lines
    else:
    elif (pCategory == "C"):
    if (pCategory == "A"):
    elif (pCategory == "B"):
    elif (pCategory == "D"):
    theFare = CONCESSION * FULL_FARE
    theFare = STUDENT * FULL_FARE
    theFare = FULL_FARE
    theFare = CHILD * FULL_FARE
    theFare = RETIRED * FULL_FARE
    # Finished jumbled lines

    return (theFare)

```

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

# ===> Here is a group of jumbled lines
# ===> Order the lines by moving them into their correct location
# ===> Indentation levels are shown for you with commented arrows
# ===> The number of lines to place is shown in brackets after the arrows

#    if (choice != "Q"):
# while (choice != "Q"):
#        print (layout.format (fare))
#    choice = getUserChoice ()
# print ("Goodbye")
#        fare = calcFare (choice)
#    showMenu ()


# ===> (1 line)

   # ===> (3 lines)



      # ===> (2 lines)


# ===> (1 line)

# ----- End of the file -----

```

Mark scheme

Question number	Answer	Additional guidance	Mark
3	Award marks as shown. • Order of arithmetic required in creation of constants: ○ RETIRED creation must come after creation of STUDENT (1) ○ CONCESSION creation must come after RETIRED creation (1) • Choice of appropriate instruction statement: ○ layout = "Your fare is {:4.2f}" (1) ○ while (not validChoice) (1) ○ if ((userChoice != "Q") and ... (1) ○ validChoice = True (1) • Calculated fares match category of test: ○ Concession fare must go with 'else' (1) ○ All four fares match tests (1) • Ordering of lines in main program with correct indentation, one each for a maximum of 7 ○ while (choice != "Q"): (1) ○ showMenu() (1) ○ choice = getUserChoice() (1) ○ if (choice != "Q"): (1) ○ fare = calcFare (choice) (1) ○ print (layout.format(fare)) (1) ○ print ("Goodbye") (1)		**(15)**

Student response

14 marks

```
# ----------------------------------------------------------------
# Import libraries
# ----------------------------------------------------------------

# ----------------------------------------------------------------
# Constants
# ----------------------------------------------------------------
# ===> Order the jumbled lines
# Finished jumbled lines
FULL_FARE = 1.50
CHILD = 0.0
STUDENT = 0.25
RETIRED = 2 * STUDENT
CONCESSION = RETIRED * 2 / 10 * 4
# ----------------------------------------------------------------
# Global variables
# ----------------------------------------------------------------
# ===> Choose one line
layout = "Your fare is {:4.2f}"
#layout = "Your fare is {:4.4f}"

choice = ""
fare = 0.0

```

```
# ---------------------------------------------------------------------
# Subprograms
# ---------------------------------------------------------------------
def showMenu ():
    print ("----- Fares -----")
    print ("A - Full fare")
    print ("B - Student fare (8 - 16 years)")
    print ("C - Child fare (0 - 7 years)")
    print ("D - Retired (65+ years)")
    print ("E - Other concession")
    print ("Q - Quit")

def getUserChoice ():
    userChoice = ""
    validChoice = False

    # ===> Choose one line
    while (validChoice):
    #while (not validChoice):
        userChoice = input ("Please choose a fare: ")
        userChoice = userChoice.upper ()
        # ===> Choose one long statement of 3 lines
        if ((userChoice != "Q") and (userChoice != "A") and
                (userChoice != "B") and (userChoice != "C") and
                (userChoice != "D") and (userChoice != "E")):
        #if ((userChoice == "Q") or (userChoice == "A") or
        #        (userChoice == "B") or (userChoice == "C") or
        #        (userChoice == "D") or (userChoice == "E")):
            print ("Invalid entry")
        else:
            # Choose one line
            #validChoice = False
            validChoice = True

    return (userChoice)

def calcFare (pCategory):
    theFare = 0.0

    # ===> Order and indent the jumbled lines
    # Finished jumbled lines
    if (pCategory == "A"):
            theFare = FULL_FARE
    elif (pCategory == "B"):
            theFare = STUDENT * FULL_FARE
    elif (pCategory == "C"):
            theFare = CHILD * FULL_FARE
    elif (pCategory == "D"):
            theFare = RETIRED * FULL_FARE
    else:
            theFare = CONCESSION * FULL_FARE
    return (theFare)
```

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

# ===> Here is a group of jumbled lines
# ===> Order the lines by moving them into their correct location
# ===> Indentation levels are shown for you with commented arrows
# ===> The number of lines to place is shown in brackets after the arrows

# ===> (1 line)
while (choice != "Q"):
    # ===> (3 lines)
    showMenu ()
    choice = getUserChoice ()
    if (choice != "Q"):
        # ===> (2 lines)
        fare = calcFare (choice)
        print (layout.format (fare))
# ===> (1 line)
print ("Goodbye")
# ----- End of the file -----
```

Question paper

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)

Student code

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

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

# ----------------------------------------------------------------------
# Subprograms
# ----------------------------------------------------------------------
# =====> Write your code here

# ----------------------------------------------------------------------
# Main program
# ----------------------------------------------------------------------
# =====> Write your code here

```

Mark scheme

Question number	Answer	Additional guidance	Mark
6	Award marks as shown. Points-based mark scheme: • Accepts string input and converts to integer (1) • One-dimensional data structures (list) created for each of the colour paths (1) • Use of concatenation/<string>.format() to form English sentences with spacing and punctuation (1) • One or more subprograms created and called (1) • Import random and use of random.randint() (1) • Range check used for validation with invalid items defaulting to 100 (1) Levels-based mark scheme to a maximum of 9, from: • Solution design (3) • Good programming practices (3) • Functionality (3)	Considerations for levels-based mark scheme: • [6.1.2] Write in a high-level language • [6.2.2] Main program code is laid out in clear sections; white space is used to show different parts of the solution/functionality; variable names are meaningful; comments are provided and are helpful • [6.2.2] Use of iteration to find maximum weight • [6.4.1] Messages match content of data structures • [6.1.6] Functions correctly for all numeric input (i.e. edge conditions and defaulting to 100) • [6.1.1] Use decomposition to solve problem and create solution • [6.2.2] Use of 'for' loop to iterate over a data structure, rather than a 'while' loop • [6.3.1] Conversion of input types to those required by program, e.g. two strings and two integers • [6.1.6] Tracking of maximum value for each list as numbers are generated rather than sorting or searching each list after building	**(15)**

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

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

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

Student response

13 marks

```
# ----------------------------------------------------------------
# Libraries
# ----------------------------------------------------------------
# =====> Write your code here
import random

# ----------------------------------------------------------------
# Global variables
# ----------------------------------------------------------------
# =====> Write your code here
greenPath = list ()             # Data structure for each path
yellowPath = list ()
redPath = list ()

# ----------------------------------------------------------------
# Subprograms
# ----------------------------------------------------------------
# =====> Write your code here
def numPackages ():
    num = int (input ("Enter the number of packages "))
    if (num < 1) or (num > 100):            # Validate
        num = 100               # Invalid, so default
    return (num)

# ----------------------------------------------------------------
# Main program
# ----------------------------------------------------------------
# =====> Write your code here
num = numPackages ()            # Get number of packages

# Fill data structures from random numbers
for count in range (num):
    weight = random.randint (1, 1000)
    if weight <= 100:
        greenPath.append (weight)
    elif weight <= 750:
        yellowPath.append (weight)
    else:
        redPath.append (weight)

# Do the outputs
print ("Green: ", "Num = ", len (greenPath), "Max = ", max (greenPath))
print ("Yellow: ", "Num = ", len (yellowPath), "Max = ", max (yellowPath))
print ("Red: ", "Num = ", len (redPath), "Max = ", max (redPath))
```

Key Documents

Document	Link
Computer Science subject page	https://qualifications.pearson.com/en/qualifications/edexcel-gcses/computer-science-2020.html
Specification Sample assessment materials	https://qualifications.pearson.com/en/qualifications/edexcel-gcses/computer-science-2020.coursematerials.html#filterQuery=Pearson-UK:Category%2FSpecification-and-sample-assessments
Past papers	https://qualifications.pearson.com/en/qualifications/edexcel-gcses/computer-science-2020.coursematerials.html#filterQuery=Pearson-UK:Category%2FExam-materials
Specimen materials Scheme of Work	https://qualifications.pearson.com/en/qualifications/edexcel-gcses/computer-science-2020.coursematerials.html#filterQuery=Pearson-UK:Category%2FTeaching-and-learning-materials
Instructions for the Conduct of the Examination (ICE) document	https://qualifications.pearson.com/en/qualifications/edexcel-gcses/computer-science-2020.coursematerials.html#%2FfilterQuery=Pearson-UK:Category%2FForms-and-administration
Switching to Pearson	https://qualifications.pearson.com/en/qualifications/edexcel-gcses/computer-science-2020/switch-to-pearson.html