Pearson Edexcel

# Mark Scheme

# Specimen Set 2

## Pearson Edexcel GCSE In Computer Science (1CP2)
## Paper 02: Application of Computational Thinking

**Edexcel and BTEC Qualifications**

Edexcel and BTEC qualifications are awarded by Pearson, the UK's largest awarding body. We provide a wide range of qualifications including academic, vocational, occupational and specific programmes for employers. For further information visit our qualifications websites at www.edexcel.com or www.btec.co.uk. Alternatively, you can get in touch with us using the details on our contact us page at www.edexcel.com/contactus.

**Pearson: helping people progress, everywhere**

Pearson aspires to be the world's leading learning company. Our aim is to help everyone progress in their lives through education. We believe in every kind of learning, for all kinds of people, wherever they are in the world. We've been involved in education for over 150 years, and by working across 70 countries, in 100 languages, we have built an international reputation for our commitment to high standards and raising achievement through innovation in education. Find out more about how we can help you and your students at: www.pearson.com/uk

All the material in this publication is copyright
© Pearson Education Ltd 2020

**Paper 2 Mark Scheme**

| Question number | Answer | Additional guidance | Mark |
|---|---|---|---|
| **1** | Award marks as shown.<br><br>• Complete creation using '[ ]' and any 9 integers between 0 and 100 (1)<br>Original: `myList =`<br>Amended: `myList = [22, 44, 66, 88, 11, 33, 55, 77, 99]`<br>• Provide text and missing double quotes (1)<br>Original: `mySentence =`<br>Amended: `mySentence = "Better a witty fool than a foolish wit"`<br>• Indent original line to be inside subprogram (1)<br>Original: `# =====> Fix the indentation error`<br>Original: `return (location)`<br>Amended: `# =====> Fix the indentation error`<br>Amended: `    return (location)`<br>• Add function call len() around 'mySentence' (1)<br>Original: `end = mySentence)`<br>Amended: `end = len (mySentence)`<br>• Add colon on end of while loop (1)<br>Original: `while (location != -1)`<br>Amended: `while (location != -1):`<br>• Add keyword 'if' to front of line (1)<br>Original: `(location != -1):`<br>Amended: `if (location != -1):`<br>• Add missing brackets around variable 'location' (1)<br>Original: `print (FOOL + " found at location: " + str location)`<br>Amended: `print (FOOL + " found at location: " + str (location))`<br>• Amend call to subprogram to fix NameError (1)<br>Original: `shwList (myList)`<br>Amended: `showList (myList)`<br>• Add type conversion to fix TypeError (1)<br>Original: `outString = outString + item + " "` | • Quote is from Twelfth Night: Better a witty fool than a foolish wit. (Feste, Act 1 Scene 5)<br>• Bullet 1: Accept alternative declaration of `myList = list ()` prior to initialisation with integers | **(10)** |

| | | | |
|---|---|---|---|
| | Amended: `outString = outString + str (item) + " "`<br>• Change – operator to a + operator to fix infinite loop (1)<br>Original: `start = location - 1`<br>Amended: `start = location + 1` | | |

```
# ----------------------------------------------------------------
# Constants
# ----------------------------------------------------------------
WIT = "wit"
FOOL = "fool"

# ----------------------------------------------------------------
# Global variables
# ----------------------------------------------------------------
start = 0
end = 0
location = 0

# =====> Complete the line to create a one-dimensional data structure
#        with 9 integers between 0 and 100, inclusive, in any order
myList = [22, 44, 66, 88, 11, 33, 55, 77, 99]

# ====> Initialise mySentence to this quote: Better a witty fool than a foolish wit

mySentence = "Better a witty fool than a foolish wit"

# ----------------------------------------------------------------
# Subprograms
# ----------------------------------------------------------------
def showList (inList):
    outString = ""

    for item in inList:
        # =====> Fix the type error
        outString = outString + str (item) + " "
    print (outString)

def findLocation (inString, inTarget, inStart, inEnd):
    location = -1
```

```
    location = inString.find (inTarget, inStart, inEnd)

# =====> Fix the indentation error
    return (location)

# ---------------------------------------------------------------------
# Main program
# ---------------------------------------------------------------------
# =====> Fix the name error
showList (myList)

# =====> Complete the line with a built-in function to find the
#        length of mySentence
end = len (mySentence)

# =====> Fix the syntax error
while (location != -1):
    location = findLocation (mySentence, FOOL, start, end)

    # =====> Complete the line with the keyword for selection
    if (location != -1):
        # =====> Fix the syntax error
        print (FOOL + " found at location: " + str (location))
        # =====> Fix the logic error
        start = location + 1
```

| Question number | Answer | Additional guidance | Mark |
|---|---|---|---|
| **2** | Award marks as shown.<br>• import time (1)<br>• `while (timesToTest != 0):` (1)<br>• `time.sleep (15)` for 15 seconds (1)<br>• `time.sleep (TIME_WALK)` use of constant over hard-coded, which will not meet requirement of 10 seconds (1)<br>• `for seconds in range (TIME_COUNTDOWN, 0, -1):` to count backwards (1)<br>• `time.sleep (1)` for counting down 1 second (1)<br>• `timesToTest = timesToTest – 1` to count down tests for three times (1)<br><br>Levels-based mark scheme to a maximum of 3, from:<br>• Functionality (3) | Considerations for levels-based mark scheme:<br>• [6.1.5] Program interprets and runs without syntax or runtime error<br>• [6.1.2] Program is refined to select appropriate constructs<br>• [6.4.1] Program produces the correct output meeting the timings set out in the requirements | **(10)** |

**Functionality (levels-based mark scheme)**

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

The hand, walk, and countdown should repeat three times, with the final ‘Happy crossing’ printing only once.

```
Showing image ... Hand
Showing image ... Walk
Showing image ... Countdown 20
Showing image ... Countdown 19
Showing image ... Countdown 18
Showing image ... Countdown 17
Showing image ... Countdown 16
Showing image ... Countdown 15
Showing image ... Countdown 14
Showing image ... Countdown 13
Showing image ... Countdown 12
Showing image ... Countdown 11
Showing image ... Countdown 10
Showing image ... Countdown 9
Showing image ... Countdown 8
Showing image ... Countdown 7
Showing image ... Countdown 6
Showing image ... Countdown 5
Showing image ... Countdown 4
Showing image ... Countdown 3
Showing image ... Countdown 2
Showing image ... Countdown 1
Happy crossing
```

```
# ---------------------------------------------------------------
# Import libraries
# ---------------------------------------------------------------
# ===> Choose one line
#import math
import time

# ---------------------------------------------------------------
# Constants
# ---------------------------------------------------------------
IMAGE_HAND  = "Hand"            # Images to be displayed once available
IMAGE_WALK = "Walk"
IMAGE_COUNTDOWN = "Countdown"

TIME_WALK = 10                  # Default time for walking
TIME_COUNTDOWN = 20             # Additional default countdown time

# ---------------------------------------------------------------
# Global variables
# ---------------------------------------------------------------
seconds = 0                 # For counting down the seconds
timesToTest = 3             # Times to run the test for

# ---------------------------------------------------------------
# Main program
# ---------------------------------------------------------------
# ===> Choose one line
while (timesToTest != 0):
#while (timesToTest < 3):
    print ("Showing image ... " + IMAGE_HAND)
    # ===> Choose one line
    time.sleep (15)
```

```
    #time.sleep (45)
    print ("Showing image ... " + IMAGE_WALK)
    # ===> Choose one line
    time.sleep (TIME_WALK)
    #time.sleep (20)
    # ===> Choose one line
    for seconds in range (TIME_COUNTDOWN, 0, -1):
    #for seconds in range (0, TIME_COUNTDOWN):
        print ("Showing image ... " + IMAGE_COUNTDOWN + " " + str (seconds))
        # ===> Choose one line
        time.sleep (1)
        #time.sleep (seconds)
    # ===> Choose one line
    timesToTest = timesToTest - 1
    #timesToTest = timesToTest + 1
print ("Happy crossing")
```

| Question number | Answer | Additional guidance | Mark |
|---|---|---|---|
| **3** | Award marks as shown.<br><br>• Variable 'name' set to "" (1)<br>• Get value of 'name' from user using 'input' (1)<br>• At least one 'if' statement used (1)<br>• Presence check using "" / len (name) == 0 (1)<br>• Length check using '< 3' (1)<br>• Length check using '> 20' (1)<br>• Use of 'if…elif…elif…else' rather than separate 'if' (1)<br>• Presence check identified in comments and at least one length check identified in comments (1)<br>• Both value of 'name' and 'All checks passed' printed on same line (1)<br>• 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)** |

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

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

<table>
<tr><th>Question number</th><th>Answer</th><th>Additional guidance</th><th>Mark</th></tr>
<tr><td>4</td><td>Award marks as shown.<br><br>
• Import math library (1)<br>
• Constant, all capitals, set to 147 (1)<br>
• ‘if’ statement used to test for fullness (1)<br>
• ‘or’ statement used for single compound test for fullness (1)<br>
• Boolean value returned to indicate validity (1)<br>
• Two input parameters to calcNumberOfWidths (1)<br>
• Calculation of usable fabric width from constants (1)<br>
• Calculation for number of widths using:<br>
  ○ Pole length * fullness ratio in numerator (1)<br>
  ○ Usable fabric width in denominator (1)<br>
  ○ Ceiling function over entire fraction (1)<br>
• Return value for number of fabric widths (1)<br>
• Fullness ratio input converted to float (1)<br>
• Call to calcNumberOfWidths (1)<br>
• Call has two arguments, same order as calcNumberOfWidths header (1)<br>
• Functions correctly for all valid inputs (e.g. 2, 2.5, 3) and anticipated inputs (120, 180, 240) (1)<br><br>
Note: The calculation requires the ceiling function as accurate results will not be produced with either integer division or round.<br><br>
<table>
<tr><th>Pole</th><th>Fullness</th><th>Correct Widths</th><th>Round</th><th>Integer Division</th></tr>
<tr><td>120</td><td>2.0</td><td>2</td><td>2</td><td>1</td></tr>
<tr><td>120</td><td>2.5</td><td>3</td><td>2</td><td>2</td></tr>
<tr><td>120</td><td>3.0</td><td>3</td><td>3</td><td>2</td></tr>
</table></td><td>• Ignore additional parameters provided for calcNumberOfWidths()</td><td>(15)</td></tr>
</table>

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | 180 | 2.0 | 3 | 3 | 2 | | |
| | | 180 | 2.5 | 4 | 3 | 3 | | |
| | | 180 | 3.0 | 4 | 4 | 3 | | |
| | | 240 | 2.0 | 4 | 3 | 3 | | |
| | | 240 | 2.5 | 5 | 4 | 4 | | |
| | | 240 | 3.0 | 6 | 5 | 5 | | |

```
# ---------------------------------------------------------------------
# Libraries
# ---------------------------------------------------------------------
# =====> Import a needed library
import math

# ---------------------------------------------------------------------
# Constants
# ---------------------------------------------------------------------
SELVEDGE_WIDTH = 8               # The frayed edge on the side
# =====> Create a constant for the width of the fabric
#        and set it to 147
FABRIC_WIDTH = 147              # All fabric is 147 cm wide

# ---------------------------------------------------------------------
# Global variables
# ---------------------------------------------------------------------
poleLength = 0
fullnessRatio = 0.0
numberWidths = 0

# ---------------------------------------------------------------------
# Subprograms
# ---------------------------------------------------------------------
```

```
def isValidFullness (pFullness):
    # =====> Complete the code for this subprogram
    # 2, 2.5 or 3 only
    valid = False                   # Always assume invalid

    if ((pFullness == 2.0) or (pFullness == 2.5) or
        (pFullness == 3.0)):
        valid = True

    return (valid)

# =====> Complete the definition for this subprogram
def calcNumberOfWidths (pPoleLength, pFullness):
    # =====> Complete the code for this subprogram
    numWidths = 0                   # Initialise
    usableFabricWidth = 0

    # =====> Calculate usable fabric width
    usableFabricWidth = FABRIC_WIDTH - SELVEDGE_WIDTH

    # =====> Calculate number of fabric widths required
    numWidths = math.ceil ((pPoleLength * pFullness) / usableFabricWidth)
    return (numWidths)

# ---------------------------------------------------------------
# Main program
# ---------------------------------------------------------------
poleLength = int (input ("Enter the pole width (120, 180, 240): "))

# =====> Accept the fullness ratio from the user
fullnessRatio = float (input ("Enter the fullness ratio (2, 2.5, 3): "))
```

```
if (isValidFullness (fullnessRatio)):
    # =====> Call the subprogram to calculate number of widths
    numberWidths = calcNumberOfWidths (poleLength, fullnessRatio)
    print ("Number of widths required is " + str (numberWidths))
else:
    print ("Invalid fullness ratio")
```

| Question number | Answer | Additional guidance | Mark |
|---|---|---|---|
| **5** | Award marks as shown.<br>• Use of 2-dimensional table of data (1)<br>• Layout string for headings is left justified, centre justified, and left justified (1)<br>• The separator print statement used the repeat '*' operator (1)<br>• The layout string for the percent field is formatted to one decimal place (1)<br>• Use of a loop for iteration across data structure(s) (1)<br>• Total votes is initialised as integer and total percent is initialised as real (1)<br>• Values in each record are added to totals for votes and percent (1)<br>• Layout string for footer is right justified, left justified, and right justified (1)<br>• Use of white space, comments, and layout aids readability (1)<br>Levels-based mark scheme to a maximum of 6, from:<br>• Solution design (3)<br>• Functionality (3) | • Only one mark awarded for choice of data structure. Other marks are awardable regardless of choice of one or three data structures.<br>• Column widths may vary, but justification and order must match requirements<br>• Data type of totals can be discerned from first assignment, if not initialised separately<br><br>Considerations for levels-based mark scheme:<br>• [6.1.2] Translates without error, even if reduced functionality<br>• [6.1.6] Each row is output on a separate line in the output<br>• [6.1.6] Rows and totals match example output<br>• [6.2.2] Use of 'for' loop in preference to a 'while' loop for iteration across an entire data structure because every record must be processed<br>• [6.3.1] Same index variable used for all data structures, rather than three different ones.<br>• [6.3.3] Separators have been used to aid readability in output | **(15)** |

**Solution design (levels-based mark scheme)**

| 0 | 1 | 2 | 3 | Max. |
|---|---|---|---|---|
| *No rewardable material* | • There has been little attempt to decompose the problem.<br>• Some of the component parts of the problem can be seen in the solution, although this will not be complete.<br>• Some parts of the logic are clear and appropriate to the problem.<br>• The use of variables and data structures, appropriate to the problem, is limited.<br>• The choice of programming constructs, appropriate to the problem, is limited. | • There has been some attempt to decompose the problem.<br>• Most of the component parts of the problem can be seen in the solution.<br>• Most parts of the logic are clear and appropriate to the problem.<br>• The use of variables and data structures is mostly appropriate.<br>• The choice of programming constructs is mostly appropriate to the problem. | • The problem has been decomposed clearly into component parts.<br>• The component parts of the problem can be seen clearly in the solution.<br>• The logic is clear and appropriate to the problem.<br>• The choice of variables and data structures is appropriate to the problem.<br>• 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)**<br>• The component parts of the program are incorrect or incomplete, providing a program of limited functionality that meets some of the given requirements.<br>• Program outputs are of limited accuracy and/or provide limited information.<br>• Program responds predictably to some of the anticipated input.<br>• Solution is not robust and may crash on anticipated or provided input. | **Functionality (when the code is run)**<br>• The component parts of the program are complete, providing a functional program that meets most of the stated requirements.<br>• Program outputs are mostly accurate and informative.<br>• Program responds predictably to most of the anticipated input.<br>• Solution may not be robust within the constraints of the problem. | **Functionality (when the code is run)**<br>• The component parts of the program are complete, providing a functional program that fully meets the given requirements.<br>• Program outputs are accurate, informative, and suitable for the user.<br>• Program responds predictably to anticipated input.<br>• Solution is robust within the constraints of the problem. | **3** |

```
# ------------------------------------------------------------------
# Global variables
# ------------------------------------------------------------------
# =====> Use only the most appropriate structure(s)
resultsTable =[["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]]

partyTable = ["Conservative Party", "Labour Party", "Liberal Democrat Party", "Scottish National Party", "Green Party"]
voteTable = [13830975, 10181243, 3564231, 1131279, 853632]
percentTable = [43.7, 32.3, 11.6, 3.9, 2.8]
# =====> End of choice of data structure

layout = ""                     # For formatting

## =====> Initialise the variables
totalVote = 0                   # Running total
totalPercent = 0.0

# ------------------------------------------------------------------
# Main program
# ------------------------------------------------------------------
# =====> Complete the layout string to format the headings
layout = "{:<25} | {:^10} | {:<9}"     # Headings
# =====> Print the headings
print (layout.format ("Party", "Votes", "Percent"))

# =====> Complete the code to print a separator
print ("-"*50)                      # Separator
```

```
# =====> Complete the layout string to format the data
layout = "{:<25} | {:<10} | {:>6.1f}"  # Decimal format

# =====> Run totals and print the rows
for row in resultsTable:
    totalVote = totalVote + row[1]
    totalPercent = totalPercent + row[2]
    print (layout.format (row[0], row[1], row[2]))

# =====> Add the code to print a separator
print ("="*50)                      # Separator

# =====> Complete the layout string to format the footer
layout = "{:>25} | {:<10} | {:>6.1f}"  # Footer

# =====> Print the footer
print (layout.format ("Total", totalVote, totalPercent))
```

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

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

# ---------------------------------------------------------------
# Global variables
# ---------------------------------------------------------------
# =====> Write your code here
greenPath = []          # List for each coloured path
yellowPath = []
redPath = []
maxGreen = 0            # Maximum weight in each path
maxYellow = 0
maxRed = 0
numPackages = 0         # Number of packages to process

# ---------------------------------------------------------------
# Subprograms
# ---------------------------------------------------------------
# =====> Write your code here
# Generate the number of random weights
def generateWeights (pNum):
    count = 0
    weight = 0

    while (count < pNum):                   # As many as the user wants
        weight = random.randint (1, 1000)    # Get a random number

        # Find out which path it should go in
        if (weight <= 100):
            greenPath.append (weight)
        elif (weight <= 750):
```

```
            yellowPath.append (weight)
        else:
            redPath.append (weight)
        count = count + 1

# Function to find the maximum number in any list using a linear search
def findMax (pList):
    maxValue = 0

    # Use iteration to look at the whole list
    for weight in pList:
        if (weight > maxValue):
            maxValue = weight

    return (maxValue)
# ---------------------------------------------------------------------
# Main program
# ---------------------------------------------------------------------
# =====> Write your code here
numPackages = int (input ("Enter a number of packages to process: "))
if (numPackages < 1) or (numPackages > 100):
    # Any invalid entry defaults to 100
    numPackages = 100

generateWeights (numPackages)        # Fill data structures with weights

# Get the maximum weight for each data structure
maxGreen = findMax (greenPath)
maxYellow = findMax (yellowPath)
maxRed = findMax (redPath)

# Print the statistics
print ("Green path has " + str (len (greenPath)) + " items. The maximum is " + str (maxGreen) + ".")
print ("Yellow path has " + str (len (yellowPath)) + " items. The maximum is " + str (maxYellow) + ".")
```

```
print ("Red path has " + str (len (redPath)) + " items. The maximum is " + str (maxRed) + ".")

print ("Goodbye")
```

**Solution design (levels-based mark scheme)**

| 0 | 1 | 2 | 3 | Max. |
|---|---|---|---|---|
| *No rewardable material* | • There has been little attempt to decompose the problem.<br>• Some of the component parts of the problem can be seen in the solution, although this will not be complete.<br>• Some parts of the logic are clear and appropriate to the problem.<br>• The use of variables and data structures, appropriate to the problem, is limited.<br>• The choice of programming constructs, appropriate to the problem, is limited. | • There has been some attempt to decompose the problem.<br>• Most of the component parts of the problem can be seen in the solution.<br>• Most parts of the logic are clear and appropriate to the problem.<br>• The use of variables and data structures is mostly appropriate.<br>• The choice of programming constructs is mostly appropriate to the problem. | • The problem has been decomposed clearly into component parts.<br>• The component parts of the problem can be seen clearly in the solution.<br>• The logic is clear and appropriate to the problem.<br>• The choice of variables and data structures is appropriate to the problem.<br>• 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.<br>• Some use of meaningful variable names.<br>• Limited or excessive commenting.<br>• 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.<br>• Uses mostly meaningful variable names.<br>• Some use of appropriate commenting, although may be excessive.<br>• 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.<br>• Meaningful variable names and subprogram interfaces are used where appropriate.<br>• Effective commenting is used to explain logic of code blocks.<br>• 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)**<br>• The component parts of the program are incorrect or incomplete, providing a program of limited functionality that meets some of the given requirements.<br>• Program outputs are of limited accuracy and/or provide limited information.<br>• Program responds predictably to some of the anticipated input.<br>• Solution is not robust and may crash on anticipated or provided input. | **Functionality (when the code is run)**<br>• The component parts of the program are complete, providing a functional program that meets most of the stated requirements.<br>• Program outputs are mostly accurate and informative.<br>• Program responds predictably to most of the anticipated input.<br>• Solution may not be robust within the constraints of the problem. | **Functionality (when the code is run)**<br>• The component parts of the program are complete, providing a functional program that fully meets the given requirements.<br>• Program outputs are accurate, informative, and suitable for the user.<br>• Program responds predictably to anticipated input.<br>• Solution is robust within the constraints of the problem. | **3** |