Examiners' Report

Principal Examiner Feedback

Summer 2024

Pearson Edexcel GCSE In

Computer Science (1CP2/02)

Paper 2: 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

Summer 2024
Publications Code 1CP2_02_2406_ER

Introduction

This is the third examination of the Edexcel GCSE Computer Science (9-1), with the paper two onscreen exam. The programming language required is Python 3.

Students are supplied with a question paper, a programming language subset document, and a code file for each question. Students are required to amend the code files and save their work, using a different file name.

Centres compress the code file responses for each student. The compressed files are uploaded to Edexcel for external assessment, via the Learner Work Transfer platform.

Centre submissions

The ICE document for this series set out the format in which students' completed code files were to be submitted. The majority of centres were able to follow the instructions accurately, ensuring that a single zipped file of the COMPLETED_CODE folder was provided for each student. The submissions were correctly identified with the centre and student number.

General

Range of marks

A full range of marks was awarded for Paper 2. Examiners did see some submissions which achieved the full 75 marks available.

Attempting all questions

In common with previous years, there were a number of scripts where students did not attempt Q05 and Q06, thereby missing an opportunity to access some marks. There are partial marks that could be awarded in each question. Students are reminded to attempt all the questions on the paper.

Readability

It is not necessary to comment every line of code in a solution. In common with previous years, examiners saw some responses where the number of comments exceeded the number of code lines. Comments are to help understand the logic, so should be placed, more helpfully, at the start of blocks of code. Excessive commenting makes the response difficult to read.

White space also can help with readability, but there is no requirement to double space code. Use white space between blocks of logic. Single spacing is appropriate for code.

Execute and test the code

Marks are awarded in some questions, regardless if the code interprets and executes. However, in others, marks are awarded for interpretation and functionality. Students should always attempt to execute the code. The IDE will highlight syntax errors in the code editor or identify them with a runtime error during execution. Students can fix syntax and indentation errors this way.

In Q02, where students chose correct lines of code, the code should be executed with the test data given in the question paper. Execution would quickly identify that some incorrect lines were chosen.

Q01 – Fix the errors

This type of question has appeared in all previous papers.

Solutions required students to fix syntax errors, runtime errors, and logic errors. The resulting program does not have to translate nor execute.

The majority of students submitted good responses.

The most frequently lost marks were the corrections of the logic errors (MP1.8, MP1.9, and MP1.10).

Q01 Example 1

```
1  # -----------------------------------------------------------------
2  # Global variables
3  # -----------------------------------------------------------------
4  rainbow = ["Violet", "Indigo", "Blue", "Green", "Yellow", "Orange", "Red"]
5  waveTable = [380, 425, 450, 492, 577, 597, 622]
6  found = False
7  index = 0
8  wavelength = 123
9  colour = ""
10
11 # -----------------------------------------------------------------
12 # Main program
13 # -----------------------------------------------------------------
14 # User chooses a colour index
15 index = int(input("Enter an index: "))
16 if (index < 0):
17     print ("Indexes cannot be zero")
18 elif (index > 6):
19     print ("Indexes cannot be more than six")
20 else:
21     colour = rainbow[index]
22     print (colour)
23
24 # User chooses a colour based on wavelength
25 wavelength = int (input ("Enter a wavelength "))
26 if ((wavelength < 380) or (wavelength > 622)):
27     print ("Invalid wavelength")
28 else:
29     index = 0
30     # Look for a wavelength less than or equal to user's choice
31     while (not found):
32         if (wavelength == waveTable[index]):
33             found = True
34             print (rainbow[index])
35         elif (waveTable[index] <= wavelength):
36             found = True
37             print (rainbow[index])
38         else:
39             index = index + 1
40
```

This example was awarded eight marks. This response demonstrates an understanding of how to fix syntax errors and runtime errors. However, it has not correctly amended the code for all the logic errors.

Q01 Example 2

```
1  # ----------------------------------------------------------------
2  # Global variables
3  # ----------------------------------------------------------------
4  rainbow = ["Violet", "Indigo", "Blue", "Green", "Yellow", "Orange", "Red"]
5  waveTable = [380, 425, 450, 492, 577, 597, 622]
6  found = False
7  index = 0
8  wavelength = 123
9  colour = ""
10
11 # ----------------------------------------------------------------
12 # Main program
13 # ----------------------------------------------------------------
14 # User chooses a colour index
15 index = int(input("Enter an index:"))
16 if (index < 0):
17     print ("Indexes cannot be zero")
18 elif (index > 6):
19     print ("Indexes cannot be more than six")
20 else:
21     colour = rainbow[index]
22     print (input(colour))
23
24 # User chooses a colour based on wavelength
25 wavelength = int (input ("Enter a wavelength "))
26 if ((wavelength < 380 and wavelength > 622)):
27     print ("Invalid wavelength")
28 else:
29     [index] = 1
30     # Look for a wavelength less than or equal to user's choice
31     while (not found):
32         if (wavelength == waveTable[index]):
33             found = True
34             print (rainbow[index])
35         elif (waveTable[index] >= wavelength):
36             found = True
37             print (rainbow[index - 2])
38         else:
39             index = index + 1
40
```

This example was awarded six marks. This response demonstrates an understanding of how to fix syntax errors. The runtime error on line 22 was corrected, but the correction is not appropriate in the logic of the problem.

Q02 – Choose the lines

Solutions required selecting the correct line of code from four options.

A small number of responses deleted the lines of code that were not required. These were awarded appropriately, although they did not follow the instructions given on the paper.

Once, all the selections are made, students can execute the code to find and amend any lines where the wrong option has been chosen.

The majority of students submitted good responses.

The most frequently missed marks were those associated with the boundary conditions of the alphabet. The selections often included the boundary conditions (A, a, Z, z), rather than excluding them.

Q02 Example 1

```
# -----------------------------------------------------------------------
# Global variables
# -----------------------------------------------------------------------
plainText = ""
cipherText = ""
shift = 0

# -----------------------------------------------------------------------
# Main program
# -----------------------------------------------------------------------
plainText = input ("Enter a message: ")
shift = int (input ("Enter the shift: "))

for letter in plainText:

    # =====> Choose the correct line to check for alphabetic letters
    #if (letter.isalnum ()):
    #if (letter.islower ()):
    #if (letter.upper ()):
    if (letter.isalpha ()):

        value = ord (letter)
        value = value + shift

        # =====> Choose the correct line to check for upper case
        #if (letter.upper ()):
        #if (letter.isalpha ()):
        #if (letter.islower ()):
        if (letter.isupper ()):

            # =====> Choose the correct line to check if the letter is
            #if (value > ord ('Z')):
            if (value >= ord ('Z')):
            #if (value < chr ('Z')):
            #if (value < ord ('Z')):
                value = value - 26

            # =====> Choose the correct line to check if the letter is
            elif (value <= ord ('A')):
            #elif (value > chr ('A')):
            #elif (value < ord ('A')):
            #elif (value > ord ('A')):

                value = value + 26

```

```
45
46          # =====> Choose the correct line to check for lower case
47          #elif (letter.lower ()):
48          elif (letter.islower ()):
49          #elif (letter.isupper ()):
50          #elif (letter.isalpha ()):
51
52              # =====> Choose the correct line to check if the letter is
53              #if (value >= chr ('z')):
54              #if (value < ord ('z')):
55              if (value > ord ('z')):
56              #if (value <= chr ('z')):
57                  value = value - 26
58
59              # =====> Choose the correct line to check if the letter is
60              elif (value < ord ('a')):
61              #elif (value < chr ('z')):
62              #elif (value != ord ('a')):
63              #elif (value == chr ('z')):
64                  value = value + 26
65
66          # =====> Choose the correct line to set the variable newLetter
67          #newLetter = ord (value)
68          newLetter = chr (value)
69          #newLetter = ord (letter)
70          #newLetter = chr (letter)
71
72          # =====> Choose the correct line to create the encrypted strin
73          #cipherText = newLetter + cipherText
74          #newLetter = cipherText + newLetter
75          #newLetter = newLetter + cipherText
76          cipherText = cipherText + newLetter
77
78      else:
79          # =====> Choose the correct line to create the encrypted strin
80          #cipherText = letter + cipherText
81          cipherText = cipherText + letter
82          #letter = cipherText + letter
83          #letter = letter + cipherText
84
85  print ("Plain text: ", plainText)
86  print("Cipher text: ", cipherText)
87
```

This example was awarded eight marks. It demonstrates good use of the built-in string handling functions, but does include the boundary conditions on the alphabet.

Q02 Example 2

```
# ---------------------------------------------------------------------
# Global variables
# ---------------------------------------------------------------------
plainText = ""
cipherText = ""
shift = 0

# ---------------------------------------------------------------------
# Main program
# ---------------------------------------------------------------------
plainText = input ("Enter a message: ")
shift = int (input ("Enter the shift: "))

for letter in plainText:

    # =====> Choose the correct line to check for alphabetic letters
    #if (letter.isalnum ()):
    #if (letter.islower ()):
    #if (letter.upper ()):
    if (letter.isalpha ()):

        value = ord (letter)
        value = value + shift

        # =====> Choose the correct line to check for upper case
        #if (letter.upper ()):
        #if (letter.isalpha ()):
        #if (letter.islower ()):
        if (letter.isupper ()):

            # =====> Choose the correct line to check if the letter is
            if (value > ord ('Z')):
            #if (value >= ord ('Z')):
            #if (value < chr ('Z'')):
            #if (value < ord ('Z')):
                value = value - 26

            # =====> Choose the correct line to check if the letter is
            #elif (value <= ord ('A')):
            #elif (value > chr ('A')):
            elif (value < ord ('A')):
            #elif (value > ord ('A')):

                value = value + 26

```

```
        # =====> Choose the correct line to check for lower case
        elif (letter.lower ()):
        #elif (letter.islower ()):
        #elif (letter.isupper ()):
        #elif (letter.isalpha ()):

            # =====> Choose the correct line to check if the letter is
            #if (value >= chr ('z')):
            #if (value < ord ('z')):
             if (value > ord ('z')):
            #if (value <= chr ('z')):
                value = value - 26

            # =====> Choose the correct line to check if the letter is
            #elif (value < ord ('a')):
            #elif (value < chr ('z')):
            elif (value != ord ('a')):
            #elif (value == chr ('z')):
                value = value + 26

        # =====> Choose the correct line to set the variable newLetter
        newLetter = ord (value)
        #newLetter = chr (value)
        #newLetter = ord (letter)
        #newLetter = chr (letter)

        # =====> Choose the correct line to create the encrypted strin
        cipherText = newLetter + cipherText
        #newLetter = cipherText + newLetter
        #newLetter = newLetter + cipherText
        #cipherText = cipherText + newLetter

    else:
        # =====> Choose the correct line to create the encrypted strin
        #cipherText = letter + cipherText
        #cipherText = cipherText + letter
        letter = cipherText + letter
        #letter = letter + cipherText

print ("Plain text: ", plainText)
print("Cipher text: ", cipherText)

```

This example was awarded five marks. Although the individual characters are handled accurately by the built-in string functions and the selections deal accurately with the boundary conditions of the alphabet, the construction of the new ciphertext is not accurate.

Q03 – Complete the code

Solutions required completion of the given code lines and addition of new code lines. The logic for the problem solution is provided in the comments.

Test data is given in the question paper so students can check if their solution functions correctly.

The majority of students submitted good responses.

The most frequently missed marks were those associated with the use of relational operators and the use of literals rather than the constants provided.

Q03 Example 1

```
# ------------------------------------------------------------------
# Constants
# ------------------------------------------------------------------
PURCHASE_TYPE_ITEM = 1
PURCHASE_TYPE_WEIGHT = 5

PRICE_PER_KILOGRAM = 3.45
PRICE_PER_ITEM = 1.23

# ------------------------------------------------------------------
# Global variables
# ------------------------------------------------------------------
weight = 0.0
count = 0
totalCost = 0.0

# =====> Create an integer variable named purchaseType and set it to 0
purchaseType = 0

# ------------------------------------------------------------------
# Main program
# ------------------------------------------------------------------
purchaseType = int (input ("Enter a purchase type (1 or 5) "))

# =====> Complete the line with the correct logical operator and the correct constant
if ((purchaseType !=PURCHASE_TYPE_ITEM)and(purchaseType != PURCHASE_TYPE_WEIGHT)):
    print ("Invalid purchase type")

# =====> Complete the line with the correct constant
elif (purchaseType == PURCHASE_TYPE_WEIGHT):

    # =====> Complete the line to accept a real value for the weight in kilograms
    weight = float(input ("Enter weight in kilograms "))
    if (weight <= 0):
        print ("Invalid weight")
    else:
        # =====> Complete the line to calculate the total cost based on weight
        totalCost = weight * PRICE_PER_KILOGRAM
else:
    count = int (input ("Enter count of items "))
    # =====> Complete the line to check for a 0 or negative count of items
    if (count <= 0 ):
        print ("Invalid number of items")
    else:
        totalCost = count * PRICE_PER_ITEM

# =====> Complete the line with the correct relational operator
if (totalCost >= 0.0):

    # =====> Add a line to display an informative message and the total cost

    print ("Your total cost is", totalCost)
```

This response was awarded eight marks. It demonstrates an understanding of data types, logical operators, and the use of constants. It highlights the common errors with relational operators.

Q03 Example 2

```
# ---------------------------------------------------------------------
# Constants
# ---------------------------------------------------------------------

PURCHASE_TYPE_ITEM = 1
PURCHASE_TYPE_WEIGHT = 5

PRICE_PER_KILOGRAM = 3.45
PRICE_PER_ITEM = 1.23

# ---------------------------------------------------------------------
# Global variables
# ---------------------------------------------------------------------
weight = 0.0
count = 0
totalCost = 0.0
# =====> Create an integer variable named purchaseType and set it to 0
purchaseType = 0

# ---------------------------------------------------------------------
# Main program
# ---------------------------------------------------------------------
purchaseType = int (input ("Enter a purchase type (1 or 5) "))

# =====> Complete the line with the correct logical operator and the correct constant
if ((purchaseType != PRICE_PER_KILOGRAM) and (purchaseType != PURCHASE_TYPE_WEIGHT)):
    print ("Invalid purchase type")

# =====> Complete the line with the correct constant
elif (purchaseType == PURCHASE_TYPE_WEIGHT):

    # =====> Complete the line to accept a real value for the weight in kilograms
    weight =      int    (input ("Enter weight in kilograms "))
    if (weight <= 0):
        print ("Invalid weight")
    else:
        # =====> Complete the line to calculate the total cost based on weight
        totalCost = weight  +    PRICE_PER_KILOGRAM
else:
    count = int (input ("Enter count of items "))
    # =====> Complete the line to check for a 0 or negative count of items
    if (count <=  0      ):
        print ("Invalid number of items")
    else:
        totalCost = count * PRICE_PER_ITEM

# =====> Complete the line with the correct relational operator
if (totalCost      ==    0.0):

    # =====> Add a line to display an informative message and the total cost
    print ("you havn't purchased anything")
```

This response was awarded four marks. It demonstrates the use of logical operators, but does not deal with all data types and relational tests accurately.

Q04 – Implement a flowchart

In this question students are given a description of a scenario, a flowchart algorithm that solves the problem in the scenario, and test data.

The logic to solve the problem is already designed for the student and is presented as a flowchart in the question paper. This is the first question in the paper that uses the Functionality Levels-based Mark Scheme.

Where students followed the logic set out in the flowchart to guide them in writing the code, very good marks were awarded.

The majority of responses correctly constructed the calculations to determine the partial packs of crisps, the number of rolls, and the grams of cheese required. Less successful was the logic to convert these to numbers of whole packs. Using math.ceil(), from the provided library, is the preferred method for conversion. Students were creative and demonstrated many different types of approaches. However, while many were awarded partial marks, not all approaches deal with the edge conditions accurately.

The question paper states to use the library and constants provided, use informative messages, comments, white space and layout. Where requirements are explicitly stated, students should attempt to meet them.

Q04 Example 1

```
# ---------------------------------------------------------------------
# Import libraries
# ---------------------------------------------------------------------
import math
# ---------------------------------------------------------------------
# Constants
# ---------------------------------------------------------------------
CHEESE_PER_ADULT = 40        # Grams
CHEESE_PER_CHILD = 30        # Grams
MIN_CHEESE = 500             # 500 grams in a pack

ROLLS_PER_ADULT = 1.5        # Count
ROLLS_PER_CHILD = 0.5        # Count
MIN_ROLLS = 24               # Count of rolls in a pack

CRISPS_PER_ADULT = 0.75      # Of a bag
CRISPS_PER_CHILD = 0.33      # Of a bag

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

# =====> Write your code here

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

# =====> Write your code here
print("Please enter the number of adults:") #asking for number of
adults

Adults = int(input()) #recieving number of adults as inputted and
assigning them a variable

print("Please enter the number of children:") #asking for number of
children

Children = int(input()) # recieving number of children as inputted
and assigning them a variable

AdultCrisps = Adults * 0.75 # calculating the total number of
partial crisps needed for adults

ChildCrisps = Children * 0.33 # calculating the total number of
partial crisps needed for children

TotalCrisps = AdultCrisps + ChildCrisps # adding the total number of
partial crisps for both adults and children

print(f"{TotalCrisps} partial bags of crisps required.") #
displaying how many partial bags of crisps are required

print(f"Order {int(TotalCrisps)} bags of crisps.") # using the int
function to remove the decimal and only return the actual number of
bags of crisps that need to be ordered, as crisps can only be
ordered in whole bags

```

```
48  AdultCheese = Adults * 40 # calculating the total grams of cheese
    needed for the adults
49
50  ChildCheese = Children * 30 # calculating the total grams of cheese
    needed for the children
51
52  TotalCheese = AdultCheese + ChildCheese # adding together the total
    grams of cheese needed for both children and adults to find the
    total amount of packs required
53
54  if TotalCheese <= 500: # checking if the total amount of cheese is
    within 500 grams or one pack of cheese
55
56      print("Order one pack of cheese.") # ordering one pack of cheese
        as the required total is within the amount provided for by a
        single pack of cheese
57
58  else: # if the total amount of cheese exceeds 500 grams or one pack
    of cheese, the following events will occur
59
60      ExtraCheese = TotalCheese // 500 # dividing the total cheese by
        500 to identify how many whole packs of cheese are needed and
        assigning that value a variable
61
62      print(f"Order {ExtraCheese} packs of cheese.") # using the
        variable assigned in the previous line to display how many packs
        of cheese are needed
63
64  AdultRolls = Adults * 1.5 # calculating the total amount of rolls
    needed for the adults
65
66  ChildRolls = Children * 0.5 # calculating the total amount of rolls
    needed for the children
67
68  TotalRolls = AdultRolls + ChildRolls # calculating the total amount
    of rolls needed for everyone
69
70  if TotalRolls <= 24: # checking if the total amount of rolls exceeds
    24 or one pack of rolls
71
72      print("Order one pack of rolls.") # displaying an instruction
        which states that one pack of rolls should be ordered as it will
        suffice to feed everyone within a single pack of rolls
73
74  else: # if the total amount of rolls exceeds 24 or one pack of rolls
    the following events will occur
75
76      ExtraRolls = TotalRolls // 24 # dividing the total amount of
        rolls by 24 to identify how many whole packs of rolls are needed
        and assigning that value a variable
77
78      print(f"Order {ExtraRolls} packs of rolls.") # using the
        variable assigned in the previous line to display how many packs
        of rolls are needed
79
```

This response was awarded 11 of the 15 available marks.

It is a good example of code that follows the logic of the flowchart. It has not used the provided constants or library.

It has, however, used excessive white space and comments. As a result, the code is very difficult to read. Students are reminded that examiners are knowledgeable 3rd parties, who are assumed to be able to understand Python code without line-by-line commenting. Commenting blocks of logic is more appropriate.

Q04 Example 2

```
import math

# ---------------------------------------------------------------------
# Constants
# ---------------------------------------------------------------------
CHEESE_PER_ADULT = 40        # Grams
CHEESE_PER_CHILD = 30        # Grams
MIN_CHEESE = 500             # 500 grams in a pack

ROLLS_PER_ADULT = 1.5        # Count
ROLLS_PER_CHILD = 0.5        # Count
MIN_ROLLS = 24               # Count of rolls in a pack

CRISPS_PER_ADULT = 0.75      # Of a bag
CRISPS_PER_CHILD = 0.33      # Of a bag

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

# =====> Write your code here
adults = int(input("how many adults will be attending the community event: "))

children = int(input("how many children will be attending the community event: "))

partialCrisps = (adults*CRISPS_PER_ADULT) + (children*CRISPS_PER_CHILD)
print("the amount of partial Crisps Bags needed is : ", partialCrisps)

wholeCrisps = int(partialCrisps) +1
print("the whole bags of crips needed is", wholeCrisps)
# ---------------------------------------------------------------------

gramsOfCheese = (adults*CHEESE_PER_ADULT)+(children*CHEESE_PER_CHILD)

if gramsOfCheese <= MIN_CHEESE:
    print("order one pack of cheese")
if gramsOfCheese >= MIN_CHEESE:
    gramsOfCheese = int(gramsOfCheese/MIN_CHEESE)+1
    print("the amount of packs of cheese you need to order is :", gramsOfCheese)

# ---------------------------------------------------------------------

partialRolls = (adults*ROLLS_PER_ADULT)+(children*ROLLS_PER_CHILD)
print("the partial number of rolls required is: ", partialRolls)

numberRolls = int(partialRolls)+1

if numberRolls <= MIN_ROLLS:
    print("order one pack of rolls")
if numberRolls >= MIN_ROLLS:
    numberRolls = int(numberRolls/MIN_ROLLS)+1
    print("the amount of packs of rolls you need to order is :", numberRolls)

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

# =====> Write your code here

```

This response was awarded 10 of the 15 marks. It is a good example of calculating the decimal values for the ingredients. There is an attempt to convert to whole numbers, which was awarded a mark. However, the outputs are not completely accurate.

Q05 – Complete the Code

In this question, students are required to create a programmed solution to a problem. Students are provided with the requirements of the problem in the question paper. This is followed up in the student code file with partially complete code representing the logic of a programmed solution.

This question requires knowledge and understanding of using subprograms effectively to decompose a solution.

The use of random.choice(pTable) to find a random pasta shape was not awarded MP 5.4. Instructions in the question paper state that a random number is to be generated and used as an index into the pasta table. The marks for functionality were not affected.

Examiners saw these strengths:

- Returning a value from getChoice()
- Calling showShapes() to display the pasta table
- Appending a shape to the pasta table

Examiners saw these recurring errors:

- Ignoring input parameters (pTable) to subprograms, using the global variable instead
- Upper bound on random number not controlled by length of the pasta table or off-by-one errors
- No loop in main program

Q05 Example 1

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

# ---------------------------------------------------------------------
# Constants
# ---------------------------------------------------------------------
GET = 1
ADD = 2
SHOW = 3
EXIT = 4

# ---------------------------------------------------------------------
# Global variables
# ---------------------------------------------------------------------
pastaShapes = ["Bigoli", "Strozzapreti", "Trofie", "Gigli", "Chitarra",
               "Penne", "Orecchiette", "Tagliatelle", "Chonchiglie",
               "Fusilli"]

shape = ""
choice = 0

# ---------------------------------------------------------------------
# Subprograms
# ---------------------------------------------------------------------
# Get a menu item from the user
def getChoice ():

        print ("1 - get a shape")
        print ("2 - add a shape")
        print ("3 - show the shapes")
        print ("4 - exit program")
        menuChoice = int(input("Enter your menu choice: "))
    # =====> Write your code here
        return menuChoice

# Display all the shapes
def showShapes (pTable):
    for pasta in pTable:
        print (pasta)

# Get a random shape
def getShape (pTable):
    # =====> Write your code here
    index =random.randint(0, len(pTable)-1)
    return(pTable[index])

```

```
49  # Add a shape
50  def addShape (pTable):
51      # =====> Write your code here
52      newShape = input("Enter the name of the new pasta shape: ")
53      pTable.append(newShape)
54
55  # -----------------------------------------------------------------------
56  # Main program
57  # -----------------------------------------------------------------------
58
59  choice = getChoice ()
60  # =====> Write your code here
61  while (choice != EXIT):
62      if choice == GET:
63          print("The random pasta shape is:",getShape(pastaShapes))
64      elif choice == ADD :
65          addShape(pastaShapes)
66      elif choice == SHOW:
67          showShapes(pastaShapes)
68      else:
69          print("ERROR: You must select a menu option")
70      choice = getChoice()
```

This response was awarded the maximum of 15 marks. It demonstrates good use of subprograms to decompose a problem. In addition, the program demonstrates good design decisions. It accurately uses the parameters passed into each subprogram. The random number generation makes a generalised solution that works with any number of shapes in the pasta table. There are no additional global variables used, thereby reducing the probability of errors and making debugging easier.

Q05 Example 2

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

# ---------------------------------------------------------------------
# Constants
# ---------------------------------------------------------------------
GET = 1
ADD = 2
SHOW = 3
EXIT = 4

# ---------------------------------------------------------------------
# Global variables
# ---------------------------------------------------------------------
pastaShapes = ["Bigoli", "Strozzapreti", "Trofie", "Gigli", "Chitarra",
               "Penne", "Orecchiette", "Tagliatelle", "Chonchiglie",
               "Fusilli"]

shape = ""
choice = 0

# ---------------------------------------------------------------------
# Subprograms
# ---------------------------------------------------------------------
# Get a menu item from the user
def getChoice ():
    # =====> Write your code here
    print ("1 - get a shape")
    print ("2 - add a shape")
    print ("3 - show the shapes")
    print ("4 - exit program")
    choice = int(input("Enter menu number: "))

    # =====> Write your code here
    return choice

# Display all the shapes
def showShapes (pTable):
    for pasta in pTable:
        print (pasta)

# Get a random shape
def getShape (pTable):
    # =====> Write your code here
    shape = pastaShapes[random.randint(0,(len(pTable)-1))]
    return shape

# Add a shape
def addShape (pTable):
    # =====> Write your code here
    shape = input("Enter the name of the shape: ")
    pTable.append(shape)

```

```
56
57  # ------------------------------------------------------------------------
58  # Main program
59  # ------------------------------------------------------------------------
60
61  choice = getChoice ()
62  # =====> Write your code here
63  pTable = pastaShapes
64  while choice != EXIT:
65      if choice == GET:
66          shape = getShape(pTable)
67          print("The shape is",shape)
68      elif choice == ADD:
69          addShape(pTable)
70      elif choice == SHOW:
71          showShapes(pTable)
72      elif choice != GET and ADD and SHOW and EXIT:
73          print("ERROR! The option you selected is not in the menu.")
74      choice = getChoice ()
75
```

This response was awarded 13 marks. It demonstrates a good understanding of using subprograms to decompose a problem. It is inconsistent in the use of passed in parameters. The conditional test on line 72 does not function as the author may believe it does. The outputs remain accurate as a side-effect of the way the if/elif choices are arranged.

Q05 Example 3

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

# ---------------------------------------------------------------------
# Constants
# ---------------------------------------------------------------------
GET = 1
ADD = 2
SHOW = 3
EXIT = 4

# ---------------------------------------------------------------------
# Global variables
# ---------------------------------------------------------------------
pastaShapes = ["Bigoli", "Strozzapreti", "Trofie", "Gigli", "Chitarra",
               "Penne", "Orecchiette", "Tagliatelle", "Chonchiglie",
               "Fusilli"]

shape = ""
choice = 0

# ---------------------------------------------------------------------
# Subprograms
# ---------------------------------------------------------------------
# Get a menu item from the user
def getChoice ():
    # =====> Write your code here

    print ("1 - get a shape")
    print ("2 - add a shape")
    print ("3 - show the shapes")
    print ("4 - exit program")

    # =====> Write your code here
    answer = int(input("please enter your choice from the menu: "))
    #user inputs choice
    return (answer)

# Display all the shapes
def showShapes (pTable):
    for pasta in pTable: #iterating through table
        print (pasta)

# Get a random shape
def getShape (pTable):
    # =====> Write your code here
    index = random.randint (0, 9) #picking a random number
    shape = pTable[index] #finding the shape with that index
    return(shape)

```

```
52  # Add a shape
53  def addShape (pTable):
54      # =====> Write your code here
55      shape = input("enter the name of the shape of pasta that you would
        like to add to the list") #allowing user to enter shape
56      pTable.append(shape) #adding shape to list
57
58  # -----------------------------------------------------------------------
59  # Main program
60  # -----------------------------------------------------------------------
61
62  choice = getChoice ()
63  # =====> Write your code here
64
65  while choice != EXIT: #loop to continue choice
66
67      if choice == GET:
68          result = getShape(pastaShapes) #calling subprogram with
            parameter
69          print(result)
70          choice = getChoice ()
71
72      elif choice == ADD:
73          addShape(pastaShapes) #calling subprogram with parameter
74          choice = getChoice ()
75
76      elif choice == SHOW:
77          showShapes(pastaShapes) #calling subprogram with parameter
78          choice = getChoice ()
79
80      else:
81          print("that input is not valid") #telling user that input is
            invalid
82          choice = getChoice ()
83
```

This response was awarded 11 marks. It also demonstrates a good understanding of using subprograms to decompose a problem. It has consistently used the parameters passed into the subprograms. However, it is not a generalised solution, as it will not work for any number of items in the pasta table. There is a duplication of code in the main loop, as only a single call to getChoice() is required, regardless of the menu item chosen.

Q06 – Files and strings

In this question, students are required to create a programmed solution to a problem. There is very little scaffolding provided in this question.

This question requires knowledge and understanding of reading lines from a file, manipulating strings and numbers, and storing records in a table.

Students are asked to read in data from a comma-separated value text file and break the line into separate fields. String fields are indexed, integer fields are used in arithmetic, and then they are recombined to make a key. The new key and original record are added to a table and the table is displayed.

There was a range of creative solutions which demonstrated the main requirements of the problem. Some solutions demonstrated decomposition and abstraction by using subprograms.

Examiners saw these strengths:

- Opening and closing files
- Processing every line from the file
- Removing line feed characters
- Breaking the line from the file into separate fields
- Appending a record to a table

Examiners saw these recurring errors:

- Inadvertent conversions to tuples, when building the final record
- Incorrect extraction (index, slice) of characters from strings
- Attempts at using temporary data structures (copies) or traversing records/fields multiple times
- Inconsistent code layout, resulting in nesting of the supplied subprogram in main program code

Q06 Example 1

```
# ---------------------------------------------------------------------
# Global variables
# ---------------------------------------------------------------------
cowTable = []
cowDetails = [] #Store the cow details from the text file in this 2
dimensional list

# =====> Write your code here

#Open Cow.txt and store the cow detaqils in a list

file1 = open("Cows.txt","r")
for line in file1:
    line = line.strip('\n')
    cowDetails.append(line.split(','))
file1.close()

# ---------------------------------------------------------------------
# Subprograms
# ---------------------------------------------------------------------
def showTable (pTable):
    for cow in pTable:
        print (cow)
# create the key for each cow and store the cow details and key as a
record in the cowTable list
def createKey(pTable, dTable):
    for details in dTable:
        name,breed,tag = details[0], details[1], details[2]
        breedDetails = breed[0:2]
        tagDetails = str(int(tag)//100)
        nameDetails = name[0:2]

        key = breedDetails+tagDetails+nameDetails

        pTable.append([key, tag, name, breed])


# ---------------------------------------------------------------------
# Main program
# ---------------------------------------------------------------------
# =====> Write your code here
createKey(cowTable, cowDetails)

showTable(cowTable)

```

This example was awarded 12 marks. It demonstrates an effective way to open a file, read lines from a file, strip off the line feed from each line, and close a file. The strings are split apart, arithmetic is accurate, and the key has been formed using type conversions. There is an effective use of a subprogram. However, the design of the solution has introduced the need for an additional data structure to hold the contents of the file. Each line of the file can be processed one at a time, without the need for duplicate storage. The layout of the code has nested the subprogram definitions inside the main code. This should be avoided to better demonstrate an understanding of scope.

Q06 Example 2

```
# ---------------------------------------------------------------------
# Global variables
# ---------------------------------------------------------------------
cowTable = []

# =====> Write your code here

# ---------------------------------------------------------------------
# Subprograms
# ---------------------------------------------------------------------
def showTable (pTable):
    for cow in pTable:
        print (cow)

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

#open the file in read mode
theFile = open("Cows.txt","r")

#itereation for loop, reads each line and creates a lists with the data separated, stripped and split
for line in theFile:
    line = line.strip ("\n")
    datas = line.split (",")
    # making the tag number an integer
    tag = int(datas [2])

    #key with the first 2 letters of the breed, the tag number divided by 100, the name
    key = (datas[1](1,3), tag/100 , datas [0])

    record = key , datas [2], datas [0], datas [1]
    cowTable.append (record)

showTable (cowTable)

```

This response was awarded nine marks. It demonstrates opening a file and processing each line in the file, one line at a time. It uses both strip() and split() appropriately. Indexing a string and using integer division are not accurately implemented. There is no matching close() for the file open(). While the solution does translate, it generates runtime errors on execution.

Q06 Example 3

```
# ------------------------------------------------------------------
# Global variables
# ------------------------------------------------------------------
cowTable = []

# =====> Write your code here
keyPartOne = ""
keyPartTwo = ""
keyPartThree = ""
cowName = ""
cowBreed = ""
cowTag = 0
cowKey = ""
pTable = ""

# ------------------------------------------------------------------
# Subprograms
# ------------------------------------------------------------------
def showTable (pTable):
    for cow in pTable:
        print (cow)

# ------------------------------------------------------------------
# Main program
# ------------------------------------------------------------------
# =====> Write your code here
cowFile = open("Cows.txt", "r")

#iterates through each cow - to make their key
for cow in cowFile:
    print(cow)

    cow.split()
    cow.strip()

    cowBreed = cow[1]
    keyPartOne = cowBreed[0:4]

    cowTag = cow[2]
    keyPartTwo = cowTag

    cowName = cow[0]
    keyPartThree = cowName[0:2]

    #create the key
    cowKey = keyPartOne + keyPartTwo + keyPartThree

    #create record
    cowRecord = (cowKey , cowTag , cowName , cowBreed)
    cowTable.append(cowRecord)

#calls the subprogram to display the contents of the table
pTable = cowTable
print(showTable(pTable))

cowFile.close()
```

This response was awarded seven marks. It demonstrates good handling of resources by using both open() and close() for the file. The character parts of the key are handled accurately. Each record in the table is a tuple, rather than a list.

Summary

Students should:

- Follow the instructions in the paper and do not rewrite the supplied code
- Remove all the syntax errors from code so that it will translate
- Execute and test code with the data supplied in the question
- Consider the design of the overall solution, not just the single lines of code
- Use effective, but not excessive, commenting and white space to make the program logic clear