Chhetri Academy

chhetriacademy.com

GCSE COMPUTER SCIENCE

6.5

Edexcel 1CP2 | Specification point 6.5.1

Be able to write programs that use arithmetic operators (addition, subtraction, division,...

Untiered

Answers and examiner comments

1	2025	Untiered
question groups	available papers	tier

Using this booklet

1. Answer numbers match the companion question booklet.
2. Follow the original question number on marking tables.
3. Examiner comments are included where available.

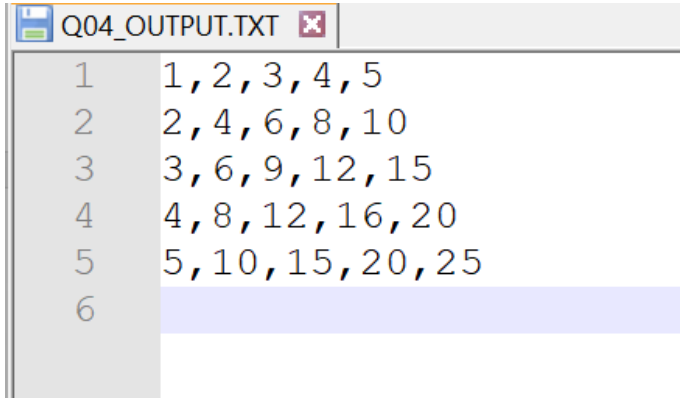
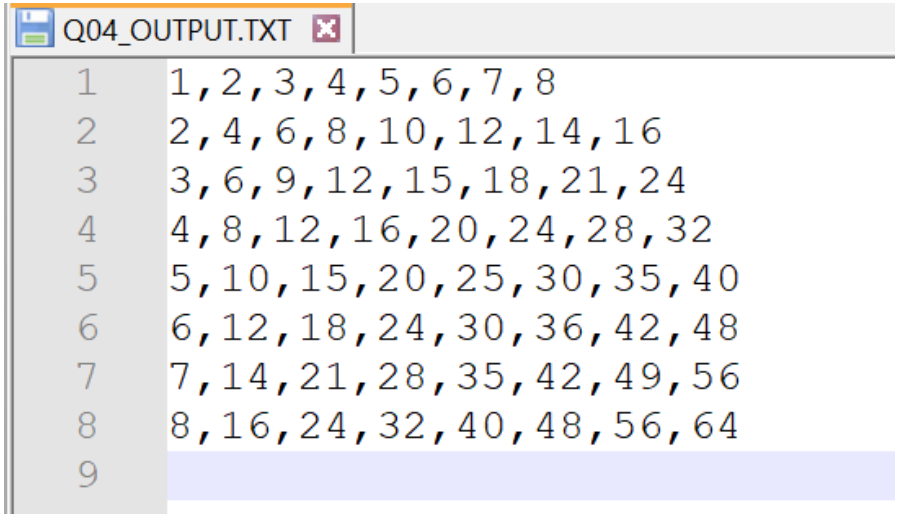
Name _____ **Date** _____

Some entries use full source marking material in an appendix. Any unavailable mark schemes are identified beside the relevant answer.

Question number	MP	Appx. Line	Answer	Additional guidance	Mark
4			Award marks as shown.		
			File and input		
	4.1	22	Output file opened for writing only (the 'w' must be present)	• Any method • Allow accurate hard-coded name matching the constant provided • Allow with open () • Where using open () - check for matching close ()	
			Building output line		
	4.2	30	Value is calculated as row * column	• Does not require assignment to a variable	
	4.3	31	New value is appended to outString	• Requires a type conversion • Ignore manipulation of comma for this mark • Allow any variable name for outString `outString = outString + str (num)`	
	4.4	35	Comma added as separator	• Ignore malformed output strings, but there must be the concept of a string being built • Allow hard-coded value instead of constant	
	4.5	38	Line feed added to last column in output line only	• Ignore comma, if present	
			Writing to file		
	4.6	41	Output line is added to file	• Do not award when the file.write() is over-indented to cause the writing of individual digits to the file. Full functionality marks are still available.	
			Closing file		**(11)**

	4.7	44	Output file is closed outside the 'for row' loop	• Allow 'with open' where processing is indented properly	
			Overall		
	4.8	Any	Accurate use of at least one of the constants provided	• OUT_FILE, COMMA, LF	
			Levels-based mark scheme to a maximum of 3, from:	Considerations for levels-based mark scheme:	
	4.9 4.10 4.11		Functionality (3)	• [6.1.5] No syntax or runtime errors in a reasonable attempt to solve the problem • [6.1.5] Some logic errors in a reasonable attempt to address all the requirements • [6.1.6] Fully meets requirements of a solution • [6.1.2] Write in a high-level language	

Output file

User enters the number 5:	User enters the number 8:
	

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**



Examiners' Report
Principal Examiner Feedback

June 2025

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 candidates at: www.pearson.com/uk

June 2025
Publications Code 1CP2_02_2506_ER
All the material in this publication is copyright
© Pearson Education Ltd 2025

Introduction

This is the fourth 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.

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 any question that provides sample output or sample test data, it is important to execute the code.

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 (MP1.9). In those responses, the statement was outdented to the left margin. Students would have easily been able to see the correction if they run the code. Incorrect indentation permits the total to be printed, even if the input is invalid.

Q01 Example 1

```
# ------------------------------------------------------------------
# Global variables
# ------------------------------------------------------------------
cost2Courses = 15.00              Cost of meals
cost3Courses = 20.00
costDrinks = 2.00

num2Courses = 0                 # Number of meals
num3Courses = 0
numDrinks = 0

total = 0.0                     # Total cost of all food and drink

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

# Get the inputs from the user
num2Courses = int (input ("How many 2-course dinners? "))
num3Courses = int (input ("How many 3-course dinners? "))
numDrinks = int (input ("How many drinks? "))

# Check if everyone has a drink
if (numDrinks != (num2Courses + num3Courses)):
    print ("Meals and drinks don't match")
else:
    # Calculate the total cost of all meals and drinks
    total = num2Courses * cost2Courses
    total = total + num3Courses * cost3Courses
    total = total + numDrinks * costDrinks

    # Check if user gets a discount and apply it
    if (num2Courses - num3Courses > 8):     # Get 15% discount
        total = 0.85 + total
    elif (num3Courses > 4):                 # Get 10% discount
        total = 0.90 * total
    elif (num2Courses > 2):                 # Get 5% discount
        total = 0.95 * total

print ("Total is " + "total")
```

This response achieved five of the nine available marks. It demonstrates accurate correction of syntax error, runtime errors, and logic errors.

Q02 – 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. The indentation for the lines is shown in the comments.

The required screen output 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 mark was MP2.10, printing the 'Ignition' message. Students did not follow the indentation presented in the comments. This meant that 'Ignition' was often displayed even if the inputted countdown value was invalid. Students could have executed their solutions with an invalid input and seen that 'Ignition' was still displayed.

Q02 Example 1

```
# -----------------------------------------------------------------
# Import libraries
# -----------------------------------------------------------------
# =====> Add a line to import the time library
import time


# -----------------------------------------------------------------
# Constants
# -----------------------------------------------------------------
WAIT_TIME = 1.5

# -----------------------------------------------------------------
# Global variables
# -----------------------------------------------------------------
# =====> Add a line to create a variable named countDown and
#        set it to 0
countDown = 0

```

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

# =====> Complete the line to take the input from the user and
#        convert it to an integer
countDown = int(input("What is the countdown?"))

# =====> Complete the line to ensure the number inputted is valid
#        Use two relational operators and one logical operator
if ((countDown   <     10)    or     (countDown    >     0)):

    # =====> Complete the condition to ensure the loop
    #        stops when it reaches 0
    while ( countDown > 0 ):

        # =====> Add a line to display countDown for the user
        print(countDown)

        # =====> Complete the line to call a function in the time library
        #        to pause the program for WAIT_TIME
        time.sleep(1.5)

        # =====> Complete the line to reduce countDown by 1
        countDown = countDown - 1

    # =====> Add a line to display "Ignition - lift off!" for the user
    print("Ignition - lift off!")
```

This example achieved nine of the available ten marks. It demonstrates good knowledge and understanding of how to import and use library functions. It also used, accurately, relational operators, but not logical operators.

Q02 Example 2

```
# ----------------------------------------------------------------------
# Import libraries
# ----------------------------------------------------------------------
# =====> Add a line to import the time library

import time
# ----------------------------------------------------------------------
# Constants
# ----------------------------------------------------------------------
WAIT_TIME = 1.5

# ----------------------------------------------------------------------
# Global variables
# ----------------------------------------------------------------------
# =====> Add a line to create a variable named countDown and
#        set it to 0
countDown = 0

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

# =====> Complete the line to take the input from the user and
#        convert it to an integer
countDown = int(input("enter number between 1-10 for countdown"))

# =====> Complete the line to ensure the number inputted is valid
#        Use two relational operators and one logical operator
if ((countDown    <=    10)   and      (countDown   >=      0)):

    # =====> Complete the condition to ensure the loop
    #        stops when it reaches 0
    while (countDown != 0          ):

        # =====> Add a line to display countDown for the user
        print ("countDown")

        # =====> Complete the line to call a function in the time library
        #        to pause the program for WAIT_TIME
        time.sleep = WAIT_TIME

        # =====> Complete the line to reduce countDown by 1
        countDown = countDown - 1

    # =====> Add a line to display "Ignition - lift off!" for the user
print ("ignition - lift off!")
```

This example achieved six marks. It demonstrates good basic programming skills, such as importing libraries and declaration and assignment of variables.

Q03 – Parsons Problem

This question required reordering of the provided code lines to create a functional program that added up the ordinal values of characters in a string. The lines of code were presented as a main program. The levels of indentation for each line were correct in the given code file.

Many responses were seen that achieved full or nearly full marks on this question.

Some responses changed the levels of indentation for the supplied lines. Doing this means that the response will not execute as designed. As a result, they may not have been able to access MP14 and could not access MP15, which was based on the functionality of the final result.

Where there was no vertical movement, no marks could be awarded.

8

Q03 Example 1

```
# ------------------------------------------------------------------
# Constants
# ------------------------------------------------------------------
FIVE = 5
FOUR = 4
THREE = 3
ZERO = 0
# ------------------------------------------------------------------
# Global variables
# ------------------------------------------------------------------
total = 0
asciiValue = 0
inString = ""
# ------------------------------------------------------------------
# Main program
# ------------------------------------------------------------------

inString = input ("Enter a string (blank to exit): ")
while (inString != ""):
    if (len (inString) >= THREE):
        # Check if divisible by 4, 5, or neither
        if (total % FOUR == 0):
            inString = inString + str (FOUR)
        elif (total % FIVE == 0):
            inString = inString + str (FIVE)
        else:
            inString = inString + str (ZERO)
        for character in inString:
            asciiValue = ord(character)
            total = total + asciiValue
        print (total, inString)
        total = 0                       # Reset each pass
# Set up the input loop

    else:
        print ("String must be three or more characters long")

    inString = input ("Enter a string (blank to exit): ")

```

This response achieved 14 of the available 15 marks. It demonstrates an understanding of code layout, loop control, input validation, and error handling. Reversing the calculation and concatenation of the result have caused this response to miss MP15, the one for functionality.

Q03 Example 2

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


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


# -----------------------------------------------------------------------
# Main program
# -----------------------------------------------------------------------
ZERO = 0
THREE = 3
FOUR = 4
FIVE = 5
asciiValue = 0
inString = ""
total = 0

inString = input ("Enter a string (blank to exit): ")

# Set up the input loop
while (inString != ""):
    total = 0                          # Reset each pass
    inString = inString + str (ZERO)
    # Check if divisible by 4, 5, or neither
    if (len (inString) >= THREE):
        print ("String must be three or more characters long")
    else:
        if (total % FOUR == 0):
            inString = inString + str (FOUR)
        elif (total % FIVE == 0):
            inString = inString + str (FIVE)
        else:
            total = total + asciiValue
            print (total, inString)
        for character in inString:
            asciiValue = ord(character)
```

This response was awarded ten of the available 15 marks. It demonstrates the use of separation of declarations and code and loop control. The misinterpretation of the length check was observed frequently.

10

Q04 – Comment-first coding

In this question students are writing lines of code from scratch. The logic to solve the problem is already designed for the student and is presented as comments in the code file. 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 comments to guide them in writing the code, very good marks were awarded. The indentation of the comments matches a fully functional solution.

The majority of responses correctly open a file for writing and most of those managed to close the file that was opened.

Some responses approached the file writing by using a write command for every single character required in the output, resulting in 11 I/O operations for every line in the file. Used correctly, this could result in a file that looked like the output shown in the question paper. As a programming pattern, that approach would not be viable when used on a larger problem.

11

Q04 Example 1

```
# ------------------------------------------------------------------
# Constants
# ------------------------------------------------------------------
OUT_FILE = "Q04_OUTPUT.TXT"
COMMA = ","
LF = "\n"

# ------------------------------------------------------------------
# Global variables
# ------------------------------------------------------------------
outString = ""
maxNum = 0
num = 0
row = 0
column = 0

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

# =====> Open the output file for writing
OUT_FILE = open(OUT_FILE, "w")

maxNum = int (input ("Enter a number: "))    # Get the number
for row in range (1, maxNum + 1):            # Going down the table
    outString = ""

    for column in range (1, maxNum + 1):     # Going across the table
        # =====> Calculate the new value
        num = column * row
        # =====> Add a comma to all except last column
        if (column < maxNum):
            outString = str(num) + COMMA
        # =====> Add a line feed to the last column
        outString = outString + LF
    # =====> Write the row to the file
    OUT_FILE.write (outString)

# =====> Close the file
OUT_FILE.close()
```

This response achieved seven of the 11 available marks. It demonstrates opening and closing a file for writing only, although it does misuse the constant as a variable. Examiners did see many responses that demonstrated the same misconception. The calculation from the question paper has been translated accurately. The output lines are not constructed accurately.

Q04 Example 2

```
# ----------------------------------------------------------------------
# Constants
# ----------------------------------------------------------------------
OUT_FILE = "Q04_OUTPUT.TXT"
COMMA = ","
LF = "\n"

# ----------------------------------------------------------------------
# Global variables
# ----------------------------------------------------------------------
outString = ""
maxNum = 0
num = 0
row = 0
column = 0

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

# =====> Open the output file for writing
file = open(OUT_FILE,"w")

maxNum = int (input ("Enter a number: "))    # Get the number
for row in range (1, maxNum + 1):            # Going down the table
    outString = ""

    for column in range (1, maxNum + 1):     # Going across the table
        # =====> Calculate the new value
    num = row * column
        # =====> Add a comma to all except last column
    file.write(",")
        # =====> Add a line feed to the last column
    file = line.readline()
    # =====> Write the row to the file
    file.writelines(row)
# =====> Close the file
file.close()
```

This response achieved five of the 11 available marks. It demonstrates accurately opening and closing a file for writing and using a constant correctly. The inaccurate indentation has caused an issue as well as the use of readline() and writelines().

13

Q05 – Comment-first coding

In this question students are writing lines of code from scratch. The high-level logic to solve the problem is already designed for the student and is presented as comments in the code file. This question uses the Design and Functionality Levels-based Mark Schemes.

Q05 Example 1

```
# -----------------------------------------------------------------
# Import libraries
# -----------------------------------------------------------------
# =====> Import the required libraries
import turtle
import random
# -----------------------------------------------------------------
# Constants
# -----------------------------------------------------------------
WIDTH = 800
HEIGHT = 600
# -----------------------------------------------------------------
# Global variables
# -----------------------------------------------------------------
myColours = ["peru", "olive", "gold"]
xPos = -120     # Location of circle 1, turtle pointing east
yPos = -195
# -----------------------------------------------------------------
# Main program
# -----------------------------------------------------------------
turtle.mode ("standard")    # Turtle points to the east
                            # Angles are counterclockwise
screen = turtle.Screen ()
screen.setup (WIDTH, HEIGHT)
turtle.screensize (WIDTH, HEIGHT)
# -----------------------------------------------------------------
# =====> Prepare the turtle, including speed
dia = turtle.Turtle()
dia.speed(10)
dia.pensize(3)
# =====> Draw Line A and Line B
dia.penup()
dia.setposition(-200,0)
dia.pendown()
dia.forward(400)
dia.penup()
dia.setposition(0,200)
dia.pendown()
dia.right(90)
dia.forward(400)
```

```
# =====> Draw Line C and Line D
dia.penup()
dia.setposition(-200,-200)
dia.pendown()
dia.left(135)
dia.forward(566)
dia.penup()
dia.setposition(200,-200)
dia.pendown()
dia.left(90)
dia.forward(566)
dia.right(135)
# =====> Draw nine coloured circles
for cc in range(3):
    myColours = ["peru", "olive", "gold"]
    for cir in range(3):
        dia.pencolor(myColours[cir])
        pensize = random.randint(5,10)
        dia.pensize(pensize)
        dia.penup()
        dia.setposition(xPos,yPos)
        dia.pendown()
        dia.circle(75)
        xPos = xPos + 30
        yPos = yPos + 30


# ------------------------------------------------------------------------
turtle.done ()
```

This response achieved 12 of the 15 available marks. It demonstrates adherence to the pre-defined logic. It has identified programming constructs suitable to the problem.

15

Q05 Example 2

```
# ------------------------------------------------------------------
# Import libraries
# ------------------------------------------------------------------
# =====> Import the required libraries
import turtle
# ------------------------------------------------------------------
# Constants
# ------------------------------------------------------------------
WIDTH = 800
HEIGHT = 600
# ------------------------------------------------------------------
# Global variables
# ------------------------------------------------------------------
myColours = ["peru", "olive", "gold"]
xPos = -120     # Location of circle 1, turtle pointing east
yPos = -195
# ------------------------------------------------------------------
# Main program
# ------------------------------------------------------------------
turtle.mode ("standard")    # Turtle points to the east
                            # Angles are counterclockwise
screen = turtle.Screen ()
screen.setup (WIDTH, HEIGHT)
turtle.screensize (WIDTH, HEIGHT)
# ------------------------------------------------------------------
# =====> Prepare the turtle, including speed
turtle = turtle.Turtle()
turtle.speed("fastest")

# =====> Draw Line A and Line B
turtle.setposition(-200, 0)
turtle.pendown ()
turtle.forward(400)
turtle.penup ()

turtle.setposition(0, 200)
turtle.pendown ()
turtle.right(90)
turtle.forward(400)
turtle.penup ()

```

16

```
42  # =====> Draw Line C and Line D
43  turtle.setposition(-200, -200)
44  turtle.pendown ()
45  turtle.left(135)
46  turtle.forward(566)
47  turtle.penup ()
48
49  turtle.setposition(200, -200)
50  turtle.pendown()
51  turtle.left(90)
52  turtle.forward(566)
53  turtle.penup()
54
55  # =====> Draw nine coloured circles
56
57  turtle.setposition(xPos, yPos)
58  turtle.circle(75)
59  turtle.pencolor("red")
60
61  for i in range(8):
62          xPos = xPos + 30
63          yPos = xPos + 30
64          turtle.setposition(xPos, yPos)
65          turtle.circle(75)
66
67  # ------------------------------------------------------------------
68  turtle.done ()
```

This response achieved seven of the 15 available marks. It demonstrates meeting some of the requirements set out in the question paper, including setting the turtle speed, the starting positions for the lines, and the length of the lines. It also demonstrates the common error of overwriting the system turtle. This will cause the code to crash, particularly on the last line that holds the canvas open. Where this occurred, examiners ignored the recreation of the turtle.

Q06 – Open

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 string manipulation and user-devised subprograms.

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

Examiners saw these strengths:

- Loop processing, use for each
- Two-dimensional indexing used to access the key and the signal as separate items
- Use of string manipulation functions, such as .isalnum() and .isdigit()

Examiners saw these recurring issues:

- Multiple passes over the entire data structure
- Multiple passes over each key
- Premature adding up of the digits, without fully validating each key
- Isolating last character in a string, i.e. the check digit in the key

Q06 Example 1

```
# ------------------------------------------------------------------
# Constants
# ------------------------------------------------------------------
MIN_KEY_LENGTH = 3       # A00
MAX_SIGNAL = 5.0         # Max five
MIN_SIGNAL = 1.0         # Min 1

# ------------------------------------------------------------------
# Global variables
# ------------------------------------------------------------------
theRecords = [["7JA3B1", 3.41], ["A7B", 2.33], ["Y8R4K", 2.78],
              ["O9N1K0", 5.0], ["A1&2X3", 1.25],
              ["A12B3", 2.47], ["B1P6Y7", -1.23], ["F8D7L5", 5.17],
              ["AB23A5", 2.47], ["X0B9A9", 0], ["Q6B7T3", 0.5],
              ["A15B6C2", 2.56], ["A12340", 2.5], ["P3Y1M4V7", 4.35],
              ["J1H0Q1", 1.0], ["X", 2.3], ["W64T18B1", 1.51],
              ["A00", 1.99], ["A3B1C14", 4.59]
             ]

keyValid = False
signalValid = False                          # init boolean

# ------------------------------------------------------------------
# Subprograms
# ------------------------------------------------------------------
def validateKey(pKey):
    length = len(pKey)               # works for any number of records
    valid = True                     # True if valid, False if invalid
    numTotal = 0

    if (length < MIN_KEY_LENGTH):        # if too short
        valid = False
    else:                                # otherwise
        for char in pKey[:length-1]:     # calculate correct check digit
            if char.isdigit():
                numTotal = numTotal + int(char)

        # right-most digit of sum
        checkDigit = (str(numTotal))[len(str(numTotal))-1]
        if (pKey[length-1] != checkDigit):  # if digits don't match
            valid = False

    return valid

```

```
45  def validateSignal(pSignal):
46      valid = True
47
48      # if not between MIN_SIGNAL and MAX_SIGNAL
49      if not(MIN_SIGNAL <= pSignal <= MAX_SIGNAL):
50          valid = False
51
52      return valid
53
54  # ---------------------------------------------------------------------
55  # Main program
56  # ---------------------------------------------------------------------
57
58  for record in theRecords:
59
60      keyValid = validateKey(record[0])
61      signalValid = validateSignal(record[1])
62
63      if not(keyValid) or not(signalValid):    # if either invalid
64          print(record)
65
66          if not(keyValid):                    # if key invalid
67              print('Invalid key')
68          if not(signalValid):                 # if signal invalid
69              print('Invalid signal')
70
71          print('\n')                          # easier to read
72
```

This example achieved 13 of the available 15 marks. It demonstrates effective decomposition and abstraction by using meaningful subprograms. This helps the main program to be easily understandable. It does not however, exclude non-alphanumeric characters in the validation process.

Q06 Example 2

```
# ----------------------------------------------------------------------
# Constants
# ----------------------------------------------------------------------
MIN_KEY_LENGTH = 3      # A00
MAX_SIGNAL = 5.0        # Max five
MIN_SIGNAL = 1.0        # Min 1

# ----------------------------------------------------------------------
# Global variables
# ----------------------------------------------------------------------
theRecords = [["7JA3B1", 3.41], ["A7B", 2.33], ["Y8R4K", 2.78],
              ["O9N1K0", 5.0], ["A1&2X3", 1.25],
              ["A12B3", 2.47], ["B1P6Y7", -1.23], ["F8D7L5", 5.17],
              ["AB23A5", 2.47], ["X0B9A9", 0], ["Q6B7T3", 0.5],
              ["A15B6C2", 2.56], ["A12340", 2.5], ["P3Y1M4V7", 4.35],
              ["J1H0Q1", 1.0], ["X", 2.3], ["W64T18B1", 1.51],
              ["A00", 1.99], ["A3B1C14", 4.59]
             ]

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

# Checks that the length of the key in each record is a minimum of 3
def valid_length():
    valid = False

    for record in theRecords:
        if len(record[0]) < 3:
            print ("Record", record, "has an invalid key")
            valid = False
        else:
            valid = True

    return valid

# Checks that the key in each record is a group of letters and digits
def valid_letters():
    valid = False

    for record in theRecords:
        if (record[0]).isdigit == False:
            print ("Record", record, "has an invalid key")
            valid = False
        else:
            valid = True

    return valid

```

21

```
# Checks that each record has a valid check digit
def valid_checkd():
    valid = False
    character = ""

    for record in theRecords:
        sum_digits = 0
        digit_check = []

        for character in record[0]:
            if character.isdigit():
                sum_digits = sum_digits + int(character)

        digit_check.append(sum_digits)
        check_digit = digit_check[0]

        if check_digit == record[0][len(record[0])-1]:
            valid = True
        else:
            print ("Record", record, "has an invalid key")
            valid = False

    return valid

# Checks that the signal in each record is greater than or equal to
# 1.0 and less than or equal to 5.0
def valid_signal():
    valid = False

    for record in theRecords:
        if (record[1] >= 1.0) and (record[1] <= 5.0):
            valid = True
        else:
            print ("Record", record, "has an invalid signal")
            valid = False

    return valid

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

# Calls each subprogram to comlete the checks on each record
valid_length()
valid_letters()
valid_checkd()
valid_signal()
```

This response achieved nine of the 15 available marks. It demonstrates meaningful use of subprograms. The use of two-dimensional indexing is accurate in the valid_signal() subprogram. There is adherence to good practice in code layout, variable naming conventions, and in commenting. The solution does translate.

Q06 Example 3

```
# ------------------------------------------------------------------
# Constants
# ------------------------------------------------------------------
MIN_KEY_LENGTH = 3      # A00
MAX_SIGNAL = 5.0        # Max five
MIN_SIGNAL = 1.0        # Min 1

# ------------------------------------------------------------------
# Global variables
# ------------------------------------------------------------------
theRecords = [["7JA3B1", 3.41], ["A7B", 2.33], ["Y8R4K", 2.78],
              ["O9N1K0", 5.0], ["A1&2X3", 1.25],
              ["A12B3", 2.47], ["B1P6Y7", -1.23], ["F8D7L5", 5.17],
              ["AB23A5", 2.47], ["X0B9A9", 0], ["Q6B7T3", 0.5],
              ["A15B6C2", 2.56], ["A12340", 2.5], ["P3Y1M4V7", 4.35],
              ["J1H0Q1", 1.0], ["X", 2.3], ["W64T18B1", 1.51],
              ["A00", 1.99], ["A3B1C14", 4.59]
             ]

# ------------------------------------------------------------------
# Subprograms
# ------------------------------------------------------------------
#function to sheck key
def check_key(keyparameter):
    key_right = TRUE
    for i in keyparameter:
        find sum of the digets only in the key
        checldigittrue =
    if len(keyparameter) < MIN_KEY_LENGTH:
        key_right = FALSE
    elif keyparameter.isalpha() :
        key_right = FALSE
    elif checkdigitentered != checkdigittrue:
        key_right = FALSE
    return key_right


```

23

```
38  def check_signal(signalparameter):
39      signal_right = TRUE
40      if signalparameter < MIN_SIGNAL or signalparameter > MAX_SIGNAL:
41          signal_right = FALSE
42      return signal_right
43  # --------------------------------------------------------------------
44  # Main program
45  # --------------------------------------------------------------------
46
47  for i in theRecords:
48      check_key(i(0))
49      check_signal(i(1))
50      if check_key = FALSE or check_signal = FALSE:
51          print(i)
52          if check_key = FALSE:
53              print("invalid key")
54          else:
55              print("invalid signal")
56
```

This response achieved six of the available 15 marks. It demonstrates understanding of loop processing. It attempts to demonstrate two-dimensional indexing. Although the indexing is inaccurate, it does not preclude awarding other marks.

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 all code, using the data supplied in the question, where given
- 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