Mark Scheme

Specimen Set 4

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

Paper 2 Mark Scheme

Question number	Answer	Additional guidance	Mark
1	Award marks as shown. • Alternative line numbers should be awarded, in the event that students change the code layout by inserting/deleting lines. • Award first response only. Do not skip over incorrect response to get to a correct response. • Do not award where a line number is required and a text response is provided. • Allow spelling/transcription errors. • time (1) • colours (1) • 9 / 10 / 11 (1) • 17 / 17-18 / 17,18 (1) • str() (1) • len() / range() (1) • + (1)	• Do not award line 18 alone • Ignore missing () • Ignore arguments if provided • Ignore missing () • Ignore arguments, if provided	**(7)**

```
# ------------------------------------------------------------------
# Import libraries
# ------------------------------------------------------------------
import time

# ------------------------------------------------------------------
# Global variables
# ------------------------------------------------------------------
colours = ["Red", "Red and Amber", "Green", "Amber"]
cycles = 0
waitTime = 1                # One second

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

while (cycles < 3):         # Three times around
    print ("Cycle " + str (cycles))

    # Show all the lights in sequence
    for index in range (len (colours)):
        print (colours[index])
        print ("...")
        time.sleep (waitTime)

    cycles = cycles + 1

print ("Finished")

# ------------------------------------------------------------------
# =====> Type your answers here in the right-hand column
# Left                                          # Right
# ------------------------------------------------------------------
# Name of a library used in this program        # time
# Name of an array used in this program         # colours
# Line number of a variable initialisation      # 10
# Line number of a repetition                   # 17
# Name of a data type conversion function
#     used in this program                      # str()
# Name of a built-in subprogram used
#     on line 21                                # len()
# An arithmetic operator used
#     in this program                           # +

```

Question number	Answer	Additional guidance	Mark
2	Award marks as shown. • Missing initial value (1) Original: `numYears =` Amended: `numYears = 0` • Missing) from input line (1) Original: `theYear = int (input ("Enter a year: ")` Amended: `theYear = int (input ("Enter a year: "))` • Incorrect conversion of input to float (1) Original: `numYears = float (input (… …` Amended: `numYears = int (input (… …` • Incorrect spelling of range (1) Original: `for count in rang (numYears … …` Amended: `for count in range (numYears … …` • Typo in variable name (1) Original: `theyear` Amended: `theYear` • Incorrect logical operator (1) Original: `or` Amended: `and` • Incorrect test for not divisible by 100 (1) Original: `(theYear % 100 == 0)` Amended: `(theYear % 100 != 0)` • Missing : from else (1)	• Allow any integer • Allow changing of ALL 'theYear' to 'theyear' • Allow creation of a variable named Goodbye with any reasonable message • This error causes the goodbye message to be printed every pass through the loop	**(11)**

	Original: `else` Amended: `else:` • Incorrect year displayed when not a leap year (1) Original: `theYear % 4` Amended: `theYear` • Missing quotes (1) Original: `print (Goodbye)` Amended: `print ("Goodbye")` • Incorrect indentation (1) Original: `indented inside for loop` Amended: `outdented inline with for loop`		

```
# ------------------------------------------------------------------
# Global variables
# ------------------------------------------------------------------
numYears = 0
theYear = 1900

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

numYears = int (input ("How many years do you want? "))

for count in range (numYears):
    theYear = int (input ("Enter a year: "))

    if (theYear % 400 == 0) and (theYear % 100 == 0):
        print (theYear, "is a leap year")
    elif (theYear % 4 == 0) and (theYear % 100 != 0):
        print (theYear, "is a leap year")
    else:
        print (theYear, "is not a leap year")

print ("Goodbye")
```

Question number	Answer	Additional guidance	Mark
3	Award marks as shown. **showMenu() Subprogram** • Choose correct definition, with no parameters `def showMenu ():` (1) **getMenuChoice() Subprogram** • Add a line to create a local variable (choiceStr) and set it to an empty string `choiceStr = ""` (1) • Choose initialisation to integer `choiceInt = 0` (1) • Call showMenu(), no arguments, no return value `showMenu ()` (1) • Complete the line to take a string input `choiceStr = input ("Enter an option: ")` (1) • Choose line with not <string>.isdigit() `if (not choiceStr.isdigit()):` (1) • Choose line with != 1 and !=2 and != 9 `if ((choiceInt != 1) and (choiceInt != 2) and (choiceInt != 9)):` (1)	• Do not award without brackets • Do not award if type conversion provided • Ignore str()	**(15)**

	milesToKilometres Subprogram • Choose pMiles / MILE_KILOMETRE `km = pMiles / MILE_KILOMETRE` (1) • Add a line to return the variable km to the caller `return (km)` (1) **Main Program** • Choose subprogram call with no arguments and return caught in variable `myChoice = getMenuChoice ()` (1) • Choose selection statement == 1 `if (myChoice == 1):` (1) • Choose to accept a float input `miles = float (input ("Enter the number of miles: "))` (1) • Choose subprogram call with miles argument and return caught in variable kilometres `kilometres = milesToKilometres (miles)` (1) • Choose print with string concatenation and integers converted to strings `print (str (miles) + " miles is " + str (kilometres) + " kilometres")` (1) • Choose selection with myChoice == 2 `elif (myChoice == 2):` (1)	• `float (input ())` Ignore prompt, if provided.	

```
# ------------------------------------------------------------------
# Constants
# ------------------------------------------------------------------
MILE_KILOMETRE = 1.6093         # One mile = 1.6093 kilometre
INCH_CENTIMETRE = 2.54          # One inch = 2.54 centimetres

# ------------------------------------------------------------------
# Global variables
# ------------------------------------------------------------------
myChoice = -9
miles = 0.0
kilometres = 0.0
inches = 0.0
centimetres = 0.0

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

# ==========> Choose the correct subprogram definition header
def showMenu ():
#def showMenu (kilometres):
#def showMenu (pMiles):
#def showMenu (pMiles, pKilometres):

    print ("==============================")
    print ("1 - Miles to kilometres")
    print ("2 - Inches to centimetres")
    print ("9 - Exit")

```

```
def getMenuChoice ():
    # =====> Add a line to create a local variable named choiceStr
    #        and set it to an empty string
    choiceStr = ""

    # =====> Add a line to create a local variable named choiceInt
    #        and set it to 0
    choiceInt = 0

    error = True

    # =====> Choose the correct line to display the menu
    #choiceStr = showMenu()
    showMenu ()
    #showMenu (choiceInt)
    #choiceInt = showMenu(choiceStr)

    while (error):
        # =====> Complete the line to take a string input
        choiceStr = input ("Enter an option: ")

        # =====> Choose the correct line to validate the input
        if (not choiceStr.isdigit()):
        #if (choiceStr.isalnum()):
        #if (choiceStr.isalpha()):
        #if ("{:<8.2f}".format(choiceStr):
            error = True
        else:
            choiceInt = int (choiceStr)

            # =====> Choose the correct line to check the menu option
            #if ((choiceInt != 1) or (choiceInt != 2) or (choiceInt != 9)):
            #if ((choiceInt == 1) or (choiceInt == 2) or (choiceInt == 9)):
            if ((choiceInt != 1) and (choiceInt != 2) and (choiceInt != 9)):
            #if ((choiceInt == 1) and (choiceInt == 2) and (choiceInt == 9)):
                error = True
            else:
                error = False

        if (error):
            print ("Enter an option from the menu")


    return (choiceInt)

def milesToKilometres (pMiles):
    km = 0.0

    # =====> Choose the correct line to calculate kilometres
    #km = MILES_KILOMETRE / pMiles
    km = pMiles * MILE_KILOMETRE
    #km = MILES_KILOMETRE + pMiles
    #km = pMiles / MILE_KILOMETRE

    km = round (km, 4)

    # =====> Add a line to return the variable km to the caller
    return (km)

```

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

while (myChoice != 9):

    # =====> Choose the correct line to use a subprogram to get
    #        the user's menu option choice
    #getMenuChoice ()
    #getMenuChoice (miles)
    #kilometres = getMenuChoice (miles)
    myChoice = getMenuChoice ()

    # =====> Choose the correct line to direct the menu choice
    if (myChoice == 1):
    #if (myChoice == -9):
    #if (myChoice == 2):
    #if (myChoice == 9):
        print ("Going to change miles to kilometres")

        # =====> Complete the line to take a real number input
        #        and store it in the variable miles
        miles = float (input ("Enter the number of miles: "))

        # =====> Choose the correct line to call the subprogram
        #milesToKilometres (miles)
        #milesToKilometres (kilometres)
        kilometres = milesToKilometres (miles)
        #miles = milesToKilometres (kilometres)

        # =====> Choose the line to display the correct output
        #print (miles + " miles is " + kilometres + " kilometres")
        #print (miles,  "kilometres", kilometres, "miles")
        print (str (miles) + " miles is " + str (kilometres) + " kilometres")
        #print (int (miles) + " miles is " + int (kilometres) + " kilometres")

    # =====> Choose the correct line to direct the menu choice
    #elif (myChoice == 1):
    #elif (myChoice == 9):
    #else:
    elif (myChoice == 2):
        print ("Going to change inches to centimetres (Not implemented yet)")

print ("Goodbye")

```

Question number	Answer	Additional guidance	Mark

4	Award marks as shown.
	
	• File is opened using constant OPEN_FILE for reading only (1)
	
	• For each loop used to read every line (1)
	
	• Line feed has been removed from the record (1)
	
	• The record has been split, into a 1D structure, using constant COMMA (1)
	
	• The record contents are displayed, i.e. flavour, number, cost (1)
		• `theFile = open (IN_FILE, "r")`
		
		Do not award response rewritten using other approaches to opening files, such as 'with open'
		Do not award without brackets
		Do not award literal "Q04_INPUT.TXT"
		
		• `for line in theFile:`
		
		Iterator variable name may be different to 'line', but must follow through for the remainder of the loop
		
		• Use of <string>.function() expected
		
		`record = line.strip ()`
		`record = line.replace (LF,"")`
		`record = line.replace ("\n","")`
		
		Allow literal ("\n") if using <string>.replace()
		
		• `fields = record.split (COMMA)`
		Do not award literal ","
		
		• Any method
		Ignore any type conversion
		Award string concatenation
			(12)

	• A subtotal is calculated, multiplying the number in stock by the cost (1)
	
	• The subtotal is added to the total (1)
	
	• "6.2f" is added inside the curly braces { } (1)
	
	• The file is closed (1)
	
	• Levels-based mark scheme to a maximum of 3, from: Functionality (3)
		`print (fields[0], fields[1], fields[2])`
		`print (fields)`
		
		• Any method
		
		`subtotal = float (fields[1]) * float (fields[2])`
		
		• `total = total + subtotal`
		
		• `print ("Total {:<6.2f}".format(total))` Allow anywhere inside the curly braces
		
		• `theFile.close ()`
		
		Considerations for levels-based mark scheme:
		
		• [6.1.5] Be able to identify, locate and correct program errors (logic, syntax, runtime)
		• [6.1.2] Translates and executes without error, even if reduced functionality
		• [6.1.6] Be able to use logical reasoning and test data to evaluate a program's fitness for purpose	

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**

```
# ------------------------------------------------------------------
# Constants
# ------------------------------------------------------------------
IN_FILE = "Q04_INPUT.TXT"
COMMA = ","
LF = "\n"

# ------------------------------------------------------------------
# Global variables
# ------------------------------------------------------------------
theFile = ""
data = []
total = 0.0
subtotal = 0.0
record = ""
fields = []

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

# =====> Complete the line to open the file for reading
theFile = open (IN_FILE, "r")

# =====> Complete the line to read every line from the file
for line in theFile:

    # =====> Complete the line to remove the line feed
    record = line.strip ()

    # =====> Complete the line to separate the record into fields
    fields = record.split (COMMA)

    # =====> Complete the line to display the flavour, the number
    #        in stock, and the cost of each item
    print (fields[0], fields[1], fields[2])

    # =====> Complete the line to calculate the total value
    #        of this item in stock
    subtotal = float (fields[1]) * float (fields[2])

    # =====> Complete the line to calculate the total value
    #        of all items in stock
    total = total + subtotal

# =====> Complete the line to display the variable total
#        formatted with a field width of 6 and a precision of 2
print ("Total {:<6.2f}".format(total))

# =====> Complete the line to close the opened file
theFile.close ()
```

Question number	Answer	Additional guidance	Mark

5	Award marks as shown.
	
	Calculating sales for each employee
	• Loop used to process all items in attempt to calculate sales for each employee (1)
	
	• Length of array with 8 elements used to bound loop (1)
	
	• Total for employee calculated by adding item at same index in all four salesQuater arrays (1)
	
	• Total added to salesTotalPerName array (1)
	
	Display the headers and divider
	• Header printed using <string>.format() (1)
		`for index in range`
		`while index <`
		
		Ignore bound
		
		`range (len (salesQuarter1)`
		`< len (names)`
		Do not award length of headers array
		
		`total = salesQuarter1[index] +`
		`salesQuarter2[index] +`
		`salesQuarter3[index] +`
		`salesQuarter4[index]`
		
		`salesTotalPerName.append (total)`
		`salesTotalPerName.insert (<index>, total)`
		
		Allow any method
		
		`{:<9s}|{:^5s}|{:^9s}|{:^9s}|{:^9s}|{:^9s}|{:^10s}`
		
		Allow any layout format that includes the correct field widths, even if not functional
		Allow formatted string literals (f-strings)
		Allow omission of 's' for type, as it is default
		Allow omission of pipe, "|" or alternative symbol between columns
			(15)

	• Divider printed using multiplication and TABLE_WIDTH (1)
	
	Display rows
	• Data on the row is in the correct order (Name, ID, Q1, Q2, Q3, Q4) and total (1)
	
	Calculate and display footer information
	• Total of all yearly subtotals calculated (1)
		`print ("-"*TABLE_WIDTH)`
		`print ("="*80)`
		
		Allow any symbol for divider
		Allow literal, 66 for TABLE_WIDTH
		Allow literal, 60 for forgetting the pipes
		Allow width exactly as wide as the output rows for row lengths in error
		Do not award for a very long print line.
		
		`(names[index], empID[index],`
		`salesQuarter1[index],`
		`salesQuarter2[index],`
		`salesQuarter3[index],`
		`salesQuarter4[index],`
		`salesTotalPerName[index])`
		
		Allow, even if total not stored in array salesTotalPerName array
		
		Any method
		
		`for index in range (len (salesTotalPerName)):`
		`salesYearTotal = salesYearTotal + salesTotalPerName[index]`
		
		`total = salesQuarter1[index] + salesQuarter2[index] + salesQuarter3[index] + salesQuarter4[index]`
		
		Allow, even if totals not stored in	

	• Mean of total sales is calculated (1) **Levels-based mark scheme to a maximum of 3, from:** • Design (3) • Functionality (3)	salesTotalPerName array Allow even if total of individual records is inaccurate Allow literal count, 8, which is the given data length [6.3.3] Length of arrays used for bounds on all loops and <string>.format() (f-strings) attempted for all outputs, even if layout is inaccurate [6.3.2] Constant used for table width. [6.4.1] Currencies must be right aligned. [6.1.5] Translates. [6.1.3] Translates and executes without runtime errors. Some calculations are accurate. [6.1.6] Table output is fit for purpose. Alignment as given in exemplar, even if field widths are inaccurate	

Solution design (levels-based mark scheme)

0	1	2	3	Max.
No rewardable material	• There has been little attempt to decompose the problem. • Some of the component parts of the problem can be seen in the solution, although this will not be complete. • Some parts of the logic are clear and appropriate to the problem. • The use of variables and data structures, appropriate to the problem, is limited. • The choice of programming constructs, appropriate to the problem, is limited.	• There has been some attempt to decompose the problem. • Most of the component parts of the problem can be seen in the solution. • Most parts of the logic are clear and appropriate to the problem. • The use of variables and data structures is mostly appropriate. • The choice of programming constructs is mostly appropriate to the problem.	• The problem has been decomposed clearly into component parts. • The component parts of the problem can be seen clearly in the solution. • The logic is clear and appropriate to the problem. • The choice of variables and data structures is appropriate to the problem. • The choice of programming constructs is accurate and appropriate to the problem.	**3**

Functionality (levels-based mark scheme)

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

```
# ----------------------------------------------------------------------
# Constants
# ----------------------------------------------------------------------
TABLE_WIDTH = 66            # Divider lines

# ----------------------------------------------------------------------
# Global variables
# ----------------------------------------------------------------------
headers = ["Name", "ID", "Q 1", "Q 2", "Q 3",
           "Q 4", "Yr Total"]
names = ["Adams", "Baker", "Collins", "Dalton", "East", "Ford",
         "Green", "Hill"]
empID = [434, 161, 427, 285, 460, 889, 275, 789]
salesQuarter1 = [942.45, 1566.99, 924.59, 197.71, 764.20,
                 279.43, 867.03, 880.43]
salesQuarter2 = [865.78, 337.10, 1597.64, 171.13, 552.89,
                 495.23, 637.09, 469.96]
salesQuarter3 = [973.25, 466.54, 288.54, 979.36, 2780.42,
                898.52, 522.45, 979.11]
salesQuarter4 = [535.84, 919.83, 661.63, 852.07, 315.02,
                120.78, 2748.53, 755.71]
salesTotalPerName = []
salesYearTotal = 0.0
salesMean = 0.0
headerLayout = "{:<9s}|{:^5s}|{:^9s}|{:^9s}|{:^9s}|{:^9s}|{:^10s}"
rowLayout = "{:<9s}|{:^5d}|{:>9.2f}|{:>9.2f}|{:>9.2f}|{:>9.2f}|{:>10.2f}"
yearTotalLayout = "{:>55s}{:>11.2f}"
meanSalesLayout = "{:>55s}{:>11.2f}"

```

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

# Calculate total sales for each employee and add it
#   to the salesTotalPerName array
for index in range (len (salesQuarter1)):
    total = salesQuarter1[index] + salesQuarter2[index] + \
            salesQuarter3[index] + salesQuarter4[index]
    salesTotalPerName.append (total)

# Display the headers with a pipe symbol separating columns
print (headerLayout.format (headers[0], headers[1], headers[2],
                            headers[3], headers[4], headers[5],
                            headers[6]))
# Display a divider for the width of the table
print ("-"*TABLE_WIDTH)

# Display a row to show data for each employee
for index in range (len (names)):
    print (rowLayout.format (names[index], empID[index],
                             salesQuarter1[index], salesQuarter2[index],
                             salesQuarter3[index], salesQuarter4[index],
                             salesTotalPerName[index]))
# Display a divider for the width of the table
print ("-"*TABLE_WIDTH)

# Calculate total of all sales for all employees for the year
for index in range (len (salesTotalPerName)):
    salesYearTotal = salesYearTotal + salesTotalPerName[index]

# Display a label and the total sales for the year
print (yearTotalLayout.format ("Year Total", salesYearTotal))

# Calculate mean of the total sales for the year
salesMean = salesYearTotal / len (names)

# Display a label and the mean of the total sales for the year
print (meanSalesLayout.format ("Mean Sales", salesMean))
```

Question number	Answer	Additional guidance	Mark

6	Award marks as shown.
	
	• Looping for displaying the grid (1)
	
	• Logical (AND, OR, NOT) operator used (1)
	
	• 2D indexing (1)
	
	• Looping for checking rows and columns (1)
		• `for row in pGrid:`
		
		For loop only
		
		Allow any row-oriented output:
		
		`['X', '_', '_']`
		`['O', 'O', 'O']`
		`['X', '_', 'X']`
		
		`X _ _`
		`O O O`
		`X _ X`
		
		`(pGrid[row][0] == pGrid[row][1]) and`
		`(pGrid[row][0] == pGrid[row][2]) and`
		`(pGrid[row][0] != B)`
		
		`(pGjrid[row][0] == pGrid[row][1] == pGrid[row][2]) and`
		`(pGrid[row][0] != B))`
		
		Any accurately used expression
		
		`(pGrid[0][column] == pGrid[1][column])`
		
		Used accurately anywhere in the program
		
		`while (row < 3)`
		`for row in pGrid:`
		
		Any loop method
			(15)

	• Early exit from loop, when finding winner (1) • User-devised subprogram (1) **Levels-based mark scheme to a maximum of 3, from:** • Design (3) • Programming practice (3) • Functionality (3)	`while ((column < 3) and (not foundWinner)):` Use of while loop with Boolean variable Use of for loop with break/return One user-devised subprogram (decomposition) makes a logical contribution to the problem solution [6.1.2] While loop with Booleans, rather than for loop with break/return [6.6.2] Use of parameters in subprogram and arguments on the call [6.1.1] Fully decomposed with a variety of subprograms to ensure code is not duplicated and the number of global variables is minimised. [6.1.4] Meaningful variable names [6.1.4] Layout with white space improves readability [6.1.4] Commenting is sufficient to completely follow the logic, without being excessive [6.1.2] Translates. [6.1.5] Translates and executes without runtime errors. Produces some accurate output. [6.1.6] Fully meets requirements of the program, producing fully accurate results with the data provided in the program	

Solution design (levels-based mark scheme)

0	1	2	3	Max.
No rewardable material	• There has been little attempt to decompose the problem. • Some of the component parts of the problem can be seen in the solution, although this will not be complete. • Some parts of the logic are clear and appropriate to the problem. • The use of variables and data structures, appropriate to the problem, is limited. • The choice of programming constructs, appropriate to the problem, is limited.	• There has been some attempt to decompose the problem. • Most of the component parts of the problem can be seen in the solution. • Most parts of the logic are clear and appropriate to the problem. • The use of variables and data structures is mostly appropriate. • The choice of programming constructs is mostly appropriate to the problem.	• The problem has been decomposed clearly into component parts. • The component parts of the problem can be seen clearly in the solution. • The logic is clear and appropriate to the problem. • The choice of variables and data structures is appropriate to the problem. • The choice of programming constructs is accurate and appropriate to the problem.	**3**

Good programming practices (levels-based mark scheme)

0	1	2	3	Max.
No rewardable material	• There has been little attempt to lay out the code into identifiable sections to aid readability. • Some use of meaningful variable names. • Limited or excessive commenting. • Parts of the code are clear, with limited use of appropriate spacing and indentation.	• There has been some attempt to lay out the code to aid readability, although sections may still be mixed. • Uses mostly meaningful variable names. • Some use of appropriate commenting, although may be excessive. • Code is mostly clear, with some use of appropriate white space to aid readability.	• Layout of code is effective in separating sections, e.g. putting all variables together, putting all subprograms together as appropriate. • Meaningful variable names and subprogram interfaces are used where appropriate. • Effective commenting is used to explain logic of code blocks. • Code is clear, with good use of white space to aid readability.	**3**

Functionality (levels-based mark scheme)

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

```
# ------------------------------------------------------------------
# Constants
# ------------------------------------------------------------------
X = "X"
O = "O"
B = "_"

# ------------------------------------------------------------------
# Global variables
# ------------------------------------------------------------------
Grid_01 = [[X, B, B],
           [O, O, O],
           [X, B, X]]

Grid_02 = [[X, O, X],
           [O, B, X],
           [O, B, X]]

Grid_03 = [[B, B, X],
           [B, B, B],
           [O, B, B]]

Grid_04 = [[B, B, X],
           [O, X, B],
           [X, B, O]]

Grid_05 = [[O, B, B],
           [X, O, B],
           [X, B, O]]

```

```
# ---------------------------------------------------------------------
# Subprograms
# ---------------------------------------------------------------------
# ==========> Write your code here
def displayGrid (pGrid):
    for row in pGrid:
        print (row)

def checkRows (pGrid):
    winner = B              # No winner to start
    foundWinner = False
    row = 0

    # Go down each of the three rows
    while ((row < 3) and (not foundWinner)):
        # Check if match all three and not blank
        if ((pGrid[row][0] == pGrid[row][1]) and
            (pGrid[row][0] == pGrid[row][2]) and
            (pGrid[row][0] != B)):
            foundWinner = True
            winner = pGrid[row][0]
        row = row + 1

    return (winner)

def checkColumns (pGrid):
    winner = B              # No winner to start
    foundWinner = False
    column = 0

    # Go across each of the three columns
    while ((column < 3) and (not foundWinner)):
        # Check if match all three and not blank
        if ((pGrid[0][column] == pGrid[1][column]) and
            (pGrid[0][column] == pGrid[2][column]) and
            (pGrid[0][column] != B)):
            foundWinner = True
            winner = pGrid[0][column]
        column = column + 1

    return (winner)

```

```
73  def checkDiagonals (pGrid):
74      winner = B              # No winner to start
75
76      # Check left to right
77      if ((pGrid[0][0] == pGrid[1][1]) and
78              (pGrid[0][0] == pGrid[2][2]) and
79              (pGrid[0][0] != B)):
80          winner = pGrid[0][0]
81
82      # Check right to left
83      elif ((pGrid[0][2] == pGrid[1][1]) and
84              (pGrid[0][2] == pGrid[2][0]) and
85              (pGrid[0][2] != B)):
86          winner = pGrid[0][2]
87
88      return (winner)
89
90  def displayWinner (pWinner):
91      if (pWinner == B):
92          print ("No winner")
93      elif (pWinner == X):
94          print ("Winner: ", X)
95      elif (pWinner == O):
96          print ("Winner: ", O)
97      else:
98          print ("Unknown error")
99
100 # Main processing loop for each grid
101 def processGrid (pGrid):
102     theWinner = ""
103
104     displayGrid (pGrid)
105     theWinner = checkRows (pGrid)
106     if (theWinner == B):
107         theWinner = checkColumns (pGrid)
108         if (theWinner == B):
109             theWinner = checkDiagonals (pGrid)
110     displayWinner (theWinner)
111
112 # ---------------------------------------------------------------------
113 # Main program
114 # ---------------------------------------------------------------------
115 # ==========> Write your code here
116 processGrid (Grid_01)       # Every grid
117 processGrid (Grid_02)
118 processGrid (Grid_03)
119 processGrid (Grid_04)
120 processGrid (Grid_05)
121
```