Mark Scheme (Results)

Summer 2023

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 2023
Publications Code 1CP2_02_2306_MS

General Marking Guidance

- All candidates must receive the same treatment. Examiners must mark the first candidate in exactly the same way as they mark the last.
- Mark schemes should be applied positively. Candidates must be rewarded for what they have shown they can do rather than penalised for omissions.
- Examiners should mark according to the mark scheme not according to their perception of where the grade boundaries may lie.
- There is no ceiling on achievement. All marks on the mark scheme should be used appropriately.
- All the marks on the mark scheme are designed to be awarded. Examiners should always award full marks if deserved, i.e. if the answer matches the mark scheme. Examiners should also be prepared to award zero marks if the candidate's response is not worthy of credit according to the mark scheme.
- Where some judgement is required, mark schemes will provide the principles by which marks will be awarded and exemplification may be limited.
- When examiners are in doubt regarding the application of the mark scheme to a candidate's response, the team leader must be consulted.
- Crossed out work should be marked UNLESS the candidate has replaced it with an alternative response.

Question number	MP	Appx. Line	Answer	Additional guidance	Mark
1			Award marks as shown.		
	1.1	37	DISCOUNT_5 / DISCOUNT_10 (1)	Alternative line numbers should be awarded, in the event that students change the code layout by inserting/deleting lines. Award first response only. Do not skip over incorrect response to get to a correct response. Allow spelling/transcription errors. Do not award where a line number is required and a text response is provided. Do not award repetition for iteration or vice versa. Do not award assignments, e.g. line 21, for initialisation, as it is not the first time the variable is used.	
	1.2	38	theTemperatures (1)		
	1.3	40	10 / 11 / 12 (1)		
	1.4	41	27 / 27, 29 / 27-30 (1)		
	1.5	42	22 / 22-24 (1)		
	1.6	43	17 / 17-18 (1)		
	1.7	44	18 / 21 / 24 / 32 (1)		**(7)**

Question number	MP	Appx. Line	Answer	Additional guidance	Mark
2			Award marks as shown.	Award equivalent expressions, if accurate and fix the error	
			Import libraries		
	2.1	4	Misspelling of randum changed to random (1) Original: `import randum` Amended: `import random`		
			Global variables		
	2.2	8	Single/double quotes added to 'pushups' in array (1) Original: `exerciseTable = ["squats", "planks", pushups, "lunges", "burpees"]` Amended: `exerciseTable = ["squats", "planks", "pushups", "lunges", "burpees"]`		
			Main program – printing akk exercises		
	2.3	16	Missing colon added to for loop (1) Original: `for exercise in exerciseTable` Amended: `for exercise in exerciseTable:`		
			Main program – choosing exercises		
	2.4	19	-1 removed (1) Original: `for count in range (numExercises - 1):` Amended: `for count in range (numExercises):`		
	2.5	20	Call to random completed with randint (0,4) (1) Original: `index = random.` Amended: `index = random.randint (0, 4)`		
	2.6	21	+1 removed (1)		**(8)**

			Original: `name = exerciseTable[index + 1]` Amended: `name = exerciseTable[index]`		
	2.7	22	Misspelling of naime changed to name (1) Original: `print (naime)` Amended: `print (name)`		
			Whole code file		
	2.8	-	At least one additional use of white space, in a correct location that improves readability (1)	Ignore excessive white space	

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

# ------------------------------------------------------------------
# Global variables
# ------------------------------------------------------------------
exerciseTable = ["squats", "planks", "pushups", "lunges", "burpees"]
index = 0
name = ""
numExercises = 0

# ------------------------------------------------------------------
# Main program
# ------------------------------------------------------------------
print ("Here is the exercise table")
for exercise in exerciseTable:
    print (exercise)

numExercises = int (input ("How many exercises do you need (1-5)? "))
for count in range (numExercises):
    index = random.randint (0, 4)
    name = exerciseTable[index]
    print (name)
```

Question number	MP	Appx. Line	Answer	Additional guidance	Mark
3			Award marks as shown.	Only one response allowed for MCQ. Do not award if more than one line uncommented.	**(15)**
			Constants		
	3.1	7	Add "Screws.txt", including quotes `INPUT_FILE = "Screws.txt"` (1)	• Must be name of supplied file, which is "Screws.txt"	
	3.2	10	Add .txt to Bricks file name `OUTPUT_FILE = "Bricks.txt"` (1)	• Allow .txt after the quotes • Allow .csv	
			Global variables		
	3.3	16	Add brickTable as name of array before assignment symbol `brickTable = ["Rustic", …` (1)		
	3.4	31	Choose integer initialisation `total = 0` (1)		
	3.5	36	Choose string initialisation: `outLine = ""` (1)		
			Processing copper screws		
	3.6	50	Choose constant file name and open for read only: `inFile = open (INPUT_FILE, "r")` (1)		
	3.7	56	Choose result of find() != -1 `if (line.find (SPECIFIED_MATERIAL) != -1):` (1)		
	3.8	61	Add code to increment total by one `total + 1` (1)	• Allow `total += 1`	
	3.9	64	Choose closing that matches the correct opening on line 48		

			`inFile.close ()` (1)		
	3.10	73	Choose output that will create the one given in the question paper `print ("Total screws: " + str(total) + " " + SPECIFIED_MATERIAL +" screws: " + str(foundCount))` (1)		
			Processing bricks		
	3.11	79	Choose opening the file for writing only `outFile = open (OUTPUT_FILE, "w")` (1)	• As file does not exist on first run, the "a" will create the file. If run again, the program will append bricks, resulting in an incorrect output file.	
	3.12	85	Choose the line to convert the brick name to uppercase `brick = brick.upper ()` (1)		
	3.13	93	Choose the line to add a line feed so each brick name is on a separate line `outLine = brick + "\n"` (1)		
	3.14	98	Choose the line to write the correct variable to the output file `outFile.write (outLine)` (1)		
	3.15	105	Choose output that will create the one given in the question paper `print ("Wrote", len (brickTable), "brick names to file")` (1)		

Display output:

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

Bricks.txt file contents:

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

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

# =====> Add the correct extension to this file name
INPUT_FILE = "Screws.txt"

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

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

inFile = ""
outFile = ""
foundCount = 0

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

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

```

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

# Process the screws

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

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

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

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

```

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

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

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

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

outFile.close ()

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

Question number	MP	Appx. Line	Answer	Additional guidance	Mark
4			Award marks as shown.		
	4.1		Three individual inputs taken from user (1)	• `base = input (…)` • `height = input (…)` • `length = input (…)`	
			Take and prepare inputs		
	4.2		Inputs converted from strings to real numbers prior to being used in calculations (1)	• `base = float (input (…))` • `height = float (input (…))` • `length = float (input (…))` • Allow conversion at any point before the calculation	
			Check for invalid inputs (relational and logical operators)		
	4.3		Relational operator used to check for invalid input (<= 0) (at least one input variable) (1)	• `height <= 0.0` • `base <= 0.0` • `length <= 0.0` • Allow 0 as equivalent to 0.0 • Allow < for <= • This mark can be awarded anywhere in the response that demonstrates the use of a relational operator	
	4.4		Correct logical operator used to create at least one compound test (1)	• `(height <= 0.0) or (base <= 0.0) or (length <= 0.0)` • `(width or height or length) < 0` • This mark can be awarded anywhere in the response that demonstrates the use of a logical operator	
	4.5		An error message for invalid input is displayed (1)		
			Process the triangle		
	4.6		Formula used to calculate the area of a triangle is translated correctly (1)	• Must use variables taken as input • `area = (1/2) * base * height`	**(15)**

				• `area = 0.5 * base * height` • `area = base * height / 2`	
	4.7		The area of the triangle rounded to two decimal places (using any method) (1)	• `round (area, 2)` • `print("{:0.2f}".format(area))`	
	4.8		Formula used to calculate the volume of a prism is translated correctly (1)	• Must use variables taken as input and previously calculated area • `volume = area * length`	
	4.9		Printing of final volume uses a string formatting function) (1)	• Allow f-strings • Allow <string>.format() • Do not award round(a,2) as string formatting	
	4.10		Format of decimal output is 8 columns with 2 decimal places. (1)	• `{:<8.2f} cubic units` • Ignore omission of 'cubic units'	
	4.11		A goodbye message is displayed before program terminates, in all cases, valid or invalid inputs (1)		
			Whole code file		
	4.12		Meaningful variable names used throughout (1)	• Allow b, h, l, a, and v as they're given in the question paper	
			Levels-based mark scheme to a maximum of 3 from:		
	4.13 4.14 4.15		Functionality (3) Execute with test data given in question paper. Execute with negative numbers to check validation.	Considerations for levels-based mark scheme: • Functionality - Translates without syntax and runtime errors • Functionality - Calculations are accurate, regardless of output • Functionality - Output messages are accurate and fit for purpose • Functionality - Fully meets requirements	

Prism 1:
Extra spaces after 250.00 is correct, due to the eight columns.

```
Enter the width of the base of the triangle: 4.567
Enter the height of the triangle: 1.23
Enter the length of the prism: 89.01
Area of the triangle is 2.81
Volume is 250.00   cubic units
Goodbye
```

Prism 2:
Extra spaces after 250.00 is correct, due to the eight columns.

```
Enter the width of the base of the triangle: 2.74
Enter the height of the triangle: 6.01
Enter the length of the prism: 5.55
Area of the triangle is 8.23
Volume is 45.70    cubic units
Goodbye
```

Invalid input:
Any input value less than or equal to zero.

```
Enter the width of the base of the triangle: 0
Enter the height of the triangle: 1.23
Enter the length of the prism: 89.01
Invalid input
Goodbye
```

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**

```
# ------------------------------------------------------------------
# Global variables
# ------------------------------------------------------------------
# =====> Write your code here
height = 0.0
base = 0.0
length = 0.0
area = 0.0
volume = 0.0
layout = "Volume is {:<8.2f} cubic units"

# ------------------------------------------------------------------
# Main program
# ------------------------------------------------------------------
# =====> Write your code here
# Take three decimal inputs from the user
base = float (input ("Enter the width of the base of the triangle: "))
height = float (input ("Enter the height of the triangle: "))
length = float (input ("Enter the length of the prism: "))

# Check for invalid inputs, using relational and logical operators
if ((height <= 0.0) or (base <= 0.0) or (length <= 0.0)):
# Display an error message if any input is invalid.
# Invalid input should not be processed.
    print ("Invalid input")
else:
# Process all valid inputs
# Calculate the area of the triangle
    area = (1/2) * base * height

# Display the area of the triangle, rounded to two decimal places
    print ("Area of the triangle is", round (area, 2))

# Calculate the volume of the prism
    volume = area * length

# Display the volume of the prism using the <string>.format() function
# in eight columns with two decimal places
    print (layout.format (volume))

# In all cases, display a goodbye message just before terminating
print ("Goodbye")

```

Question number	MP	Appx. Line	Answer	Additional guidance	Mark
5			Award marks as shown.		
			makeID subprogram		
	5.1	14	[definition line] any one of the parameter names changed to a different name (1)		
	5.2	14	[definition line] all three parameter names changed to a different name (1)		
	5.3	14	[definition line] Any changed names are meaningful (1)		
	5.4	24	int() replaced with ord() (1)	• `numberPart = numberPart + ord (character)`	
			Welcome subprogram		
	5.5	32	Welcome subprogram is defined using keyword def, a meaningful name, and brackets (1)	• Ignore inclusion of parameters and return() statement	
	5.6	33	Welcome message is fit for purpose (1)		
			Main program		
	5.7	39	Welcome subprogram called in main, prior to taking any other inputs (1)	• Ignore inclusion of arguments	**(15)**

	5.8		String (lastName, firstName, or myID) converted to lower case (1)	• Do not award if missing brackets
				• Conversion on input:
				`lastName = lastName.lower()`
				`firstName = firstName.lower()`
				• Conversion before output:
				`myID.lower()`	

	5.9	49	Digits in date of birth are validated as being 0-9	• Examples: o A loop over all digits o Use of <string>.isdigit()	
			Levels-based mark scheme to a maximum of 6, from:	Considerations for levels-based mark scheme:	
	5.10		Solution design (3)	• Variables in makeID subprogram	

	5.11 5.12			changed to match new header variable names • Welcome subprogram must have a body (indented line) and no return() statement • Uses [<string>.isdigit()] rather than loop over all characters	
	5.13 5.14 5.15		Functionality (3)	• Program translates • Program runs without runtime errors • A welcome message is displayed • Displays an error message if date of birth is invalid • Fully meets requirements	

Test data:

Last name	**First name**	**Date of birth (ddmmyyyy)**	**ID**	
Bassir	Viola	15062005	bassirv403	
BASSIR	VIOLA	15062005	bassirv403	
Jon35	pen7	15062005	jon35p403	No requirement to validate for all characters in the first and last names
Jon35	pen7	01AB2005	Invalid date of birth.	Processing should not take place
Jon35	pen7	A5062005	Invalid date of birth.	Processing should not take place

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**

```
# ---------------------------------------------------------------
# Global variables
# ---------------------------------------------------------------
lastName = ""
firstName = ""
dob = ""
myID = ""

# ---------------------------------------------------------------
# Subprograms
# ---------------------------------------------------------------
# =====> Change the names of the local variables to distinguish them
#        from the global variables with the same name
def makeID (pLast, pFirst, pDob):
    namePart = ""
    numberPart = 0

    namePart = pLast + pFirst[0]     # Letter part

    # =====> Correct the logic error caused by using the int() function
    #        in the number part calculation rather than using a function
    #        that returns the ASCII value of the character
    for character in pDob:
        numberPart = numberPart + ord (character)

    yourID = namePart + str (numberPart)

    return (yourID)

# =====> Add a procedure, with no parameters, to display a
#        welcome message for the user
def welcomeUser ():
    print ("Welcome to the program")

```

```
35 # ---------------------------------------------------------------------
36 # Main program
37 # ---------------------------------------------------------------------
38 # =====> Call the welcome procedure before taking input from the user
39 welcomeUser ()       # Welcome the user
40
41 # Get last name and first name from the user
42 lastName = input ("Enter your last name: ")
43 firstName = input ("Enter your first name: ")
44
45
46 # =====> Convert last name and first name to lowercase after they
47 #        are inputted by the user
48 lastName = lastName.lower()
49 firstName = firstName.lower()
50
51 # Get date of birthdate from the user
52 dob = input ("Enter your date of birth (ddmmyyyy): ")
53
54 # =====> Check that only the digits 0 to 9 appear in the date of birth
55
56 if (dob.isdigit ()):
57     # =====> Call the makeID() function, if the date of birth is valid
58     myID = makeID (lastName, firstName, dob)
59     print (myID)
60 else:
61 # =====> Tell the user, if the date of birth is invalid
62     print ("Invalid date of birth")
63
```

Question number	MP	Appx. Line	Answer	Additional guidance	Mark
6			Award marks as shown.		**(15)**
	6.1		Two string inputs taken (1)		
	6.2		Check for blank input for name/password using a relational operator (1)	• `len (name) == 0` • `len (password) == 0` • `name == ""` • `password == ""`	
	6.3		Table is traversed using a loop (1)	• `for (…)` • `while (…)` • Allow membership	
	6.4		Use of a logical operator to form a compound test / use of loops to keep invalid input out (1)	• Allow logical operator anywhere in program • `(len (name) == 0) or (len (password) == 0)` • `(not foundName) and (index < len (userTable)` • `while (name == "")…`	
	6.5		Individual fields of each record accessed (1)	• `userTable[index][0] == name` • `userTable[index][1] == password)` • Allow membership	
	6.6		Mechanism for distinguishing between states (1)	• Selection, which may be nested • Three states are: name not found; name found no password match; full match	
			Levels-based mark scheme to a maximum of 9, from:	Considerations for levels-based mark scheme:	
	6.7 6.8 6.9		Solution design (3)	• Solution decomposed into component parts • Stops loop when name and password match found • Stops loop when name match found, but not password • Minimum number of passes through the data (i.e. visits each record only once)	
	6.10 6.11 6.12		Good programming practice (3)	• Meaningful variable names • Layout with white space improves readability • Commenting is sufficient to completely follow the logic, without being excessive	

	6.13 6.14 6.15		Functionality (3)	• User messages are informative and fit for purpose • Robust, does not crash with syntax or runtime errors • Fully meets requirements, including working with any length of array	

Test data:

User name	**Password**	**Output**	**Note**
LLemon8	BeigeDresser	Welcome	Valid user and password found in the array
LLemon8	GreyOttoman	Incorrect password	The user name is found. The password belongs to a different user.
llemon8	BeigeDresser	User not found	The user name has lowercase letters so doesn't match.
OOrange99	WhiteNights	User not found	The user name is not there. The password belongs to a different user.
BJones33	GoldBed	User not found	The user does not exist. The password is not in the table.
<empty>	GreenCouch	Invalid input	User name cannot be blank.
LLemon8	<empty>	Invalid input	Password cannot be blank.

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**

```
# ----------------------------------------------------------------------
# Global variables
# ----------------------------------------------------------------------
userTable = [["LArmstrong3", "RougeChairBean"],
             ["SBarrett7", "AmarilloDeskLemon"],
             ["EChisholm4", "JauneStoolCarrot"],
             ["VDunn1", "AzulFutonLime"],
             ["DElms5", "BleuCouchBroccoli"],
             ["EFirsova13", "RojoMattressOrange"],
             ["JGolland6", "VertTableSquash"],
             ["FHartley13", "VerdeMirrorApple"],
             ["DJohnstone12", "RoseBedOnion"],
             ["GKirkhope8", "RosaNightstandPear"],
             ["LLemon8", "BlancDresserPepper"],
             ["HMacCunn6", "RosaOttomanGrapefruit"],
             ["PNewland10", "NoirWardrobeChilli"],
             ["AOldham5", "BlancoPillowStrawberry"],
             ["JPook8", "VioletCabinetAubergine"]]

# =====> Write your code here
name = ""
password = ""            # User types in
foundName = False        # Found user name
letIn = False            # Found full match
index = 0

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

# =====> Write your code here

# Get the user input
name = input ("Enter your name: ")
password = input ("Enter your password: ")

# Check if input is valid
if ((len (name) == 0) or (len (password) == 0)):
    print ("Invalid input")          # No blanks allowed
else:
    # Can be processed
    while ((not foundName) and (index < len (userTable))):
        if (userTable[index][0] == name):
            foundName = True         # Stops the loop
            if (userTable[index][1] == password):
                letIn = True         # Passes both checks
        else:
            index = index + 1

    # Determine which state we're in based on flags
    if (letIn):
        print ("Welcome")
    elif (foundName):
        print ("Incorrect password")
    else:
        print ("User not found")

```