Mark Scheme (Results)

Summer 2024

Pearson Edexcel GCSE In Computer Science (1CP2/02) Paper 2: Application of Computational Thinking

Edexcel and BTEC Qualifications

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

Pearson: helping people progress, everywhere

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

Summer 2024
Question Paper Log Number P75441
Publications Code 1CP2_02_2406_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.

2406 1CP2 02 Mark Scheme

Question number	MP	Appx. Line	Answer	Additional guidance	Mark
1			Award marks as shown.		
	1.1	5	Remove double quote from list of integers (1)	`… 577, 597, 622]`	
	1.2	6	Capitalise false to make it a keyword (1)	`found = False`	
	1.3	8	Change 0123 to any integer value (1)	• 0 • Allow any numeric value	
	1.4	15	Add closing bracket for integer conversion (1)	`index = int (input ("Enter an index:"))`	
	1.5	21	Correct 'color' to 'colour' (1)	`colour = rainbow[index]`	

	1.6	22	Correct type conversion attempting to print an int() to convert to string (1)	`print (colour)`
				`print ((colour))`
				`print (str (colour))`
				• Allow any method	

	1.7	26	Correct 'and' to 'or' (1)	`if ((wavelength < 380) or` `(wavelength > 622)):`	
	1.8	29	Correct 1 to 0 (1)	`index = 0`	
	1.9	35	Correct <= to > (1)	`elif (waveTable[index] > wavelength):` • Allow >= • Allow reversal of arguments: elif(wavelength <= waveTable[index])	
	1.10	37	Correct [index - 2] to [index – 1] (1)	`print (rainbow[index - 1])`	**(10)**

```
# ------------------------------------------------------------------
# Global variables
# ------------------------------------------------------------------
rainbow = ["Violet", "Indigo", "Blue", "Green", "Yellow", "Orange", "Red"]
waveTable = [380, 425, 450, 492, 577, 597, 622]
found = False
index = 0
wavelength = 0
colour = ""

# ------------------------------------------------------------------
# Main program
# ------------------------------------------------------------------
# User chooses a colour index
index = int (input ("Enter an index: "))
if (index < 0):
    print ("Indexes cannot be zero")
elif (index > 6):
    print ("Indexes cannot be more than six")
else:
    colour = rainbow[index]
    print (colour)

# User chooses a colour based on wavelength
wavelength = int (input ("Enter a wavelength "))
if ((wavelength < 380) or (wavelength > 622)):
    print ("Invalid wavelength")
else:
    index = 0
    # Look for a wavelength less than or equal to user's choice
    while (not found):
        if (wavelength == waveTable[index]):
            found = True
            print (rainbow[index])
        elif (waveTable[index] > wavelength):
            found = True
            print (rainbow[index - 1])
        else:
            index = index + 1

```

Question number	MP	Appx. Line	Answer	Additional guidance	Mark
2			Award marks as shown.	• Do not award mark if more than one line in each group of four is uncommented	
	2.1	20	`if (letter.isalpha ()): (1)`		
	2.2	29	`if (letter.isupper ()): (1)`		
	2.3	32	`if (value > ord ('Z')): (1)`		
	2.4	41	`elif (value < ord ('A')): (1)`		
	2.5	48	`elif (letter.islower ()): (1)`		
	2.6	55	`if (value > ord ('z')): (1)`		
	2.7	60	`elif (value < ord ('a')): (1)`		
	2.8	68	`newLetter = chr (value) (1)`		
	2.9	76	`cipherText = cipherText + newLetter (1)`		
	2.10	81	`cipherText = cipherText + letter (1)`		**(10)**

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

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

for letter in plainText:

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

        value = ord (letter)
        value = value + shift

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

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

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

                value = value + 26

```

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

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

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

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

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

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

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

Question number	MP	Appx. Line	Answer	Additional guidance	Mark
3			Award marks as shown.		
	3.1	18	purchaseType = 0 (1)	purchaseType = int (0) purchaseType = int () along with purchaseType = 0 Do not award spelling or transcription errors for this mark	
	3.2	26	PURCHASE_TYPE_ITEM (1)		
	3.3	26	and (1)		
	3.4	30	PURCHASE_TYPE_WEIGHT (1)	Allow • != PURCHASE_TYPE_ITEM	
	3.5	33	float (1)		
	3.6	38	* (1)		
	3.7	42	<= 0 (1)	Allow • < 1	
	3.8	48	> (1)	Allow • !=	
	3.9	51	print ("Total cost is", totalCost) (1)	• Must be syntactically correct • Must include both message and total • Allow any message text • Allow concatenation (+) if conversion to string provided for variable • Ignore formatting of currency, if attempted	
	3.10		Functions for test data given in paper (1)		**(10)**

Test Data

Purchase Type	Count of items	Weight in kilograms	Output
1	3		Total cost is 3.69
1	0		Invalid number of items
5		4.5	Total cost is 15.525
5		-6.6	Invalid weight
3			QP = Invalid category OR Code = Invalid purchase type

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

PRICE_PER_KILOGRAM = 3.45
PRICE_PER_ITEM = 1.23

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

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

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

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

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

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

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

    # =====> Add a line to display an informative message and the total cost
    print ("Total cost is", totalCost)
```

Question number	MP	Appx. Line	Answer	Additional guidance	Mark
4			Award marks as shown.		
			Inputs		
	4.1		Two integer inputs taken and assigned to different variables (1)		
			Crisps		
	4.2		Calculate partial bags of crisps (1)	`bagsCrisps = (numAdults * CRISPS_PER_ADULT) + (numChild * CRISPS_PER_CHILD)` • Allow 0.75 and 0.33 instead of constants	
			Cheese		
	4.3		Calculate grams of cheese required (1)	`gramsCheese = (numAdults * CHEESE_PER_ADULT) + (numChild * CHEESE_PER_CHILD)` • Allow 40, 30, and 500 instead of constants	
	4.4		Selection symbol translated to if/else for cheese (1)		
			Rolls		
	4.5		Calculate number of partial rolls required (1)	`numRolls = (numAdults * ROLLS_PER_ADULT) + (numChild * ROLLS_PER_CHILD)` • Allow 1.5, 0.5, and 24 instead of constants	**(15)**
	4.6		Selection symbol translated to if/else for rolls (1)		

			Overall		

	4.7		Conditional tests for both cheese and rolls use relational operator (<=) accurately (1)	`gramsCheese <= MIN_CHEESE`
				`numRolls <= MIN_ROLLS`
				
				Allow:
				• <	

	4.8		Input and output messages are informative and fit for purpose (1)	• Must be more than just the two inputs in the flowchart to award this mark. • Must include code for messages for five out of the nine possible messages in the flowchart	
	4.9		One or more use of helpful white space AND one or more use of helpful comments (1)	• Ignore excessive comments	
	4.10		Use of given constants throughout, rather than hard-coded values (1)	• Constants are involved in all relational and arithmetic expressions	
			Number conversions		
	4.11		At least one instance of a syntactically accurate expression to convert from decimal to whole number, even if the result is not correct	`math.ceil (<partial>)` Allow these, although the results produced will be incorrect: • round (<decimal>) • round (<decimal>, 0) • int (<decimal>) • // Do not award • % • /	

	4.12		math.ceil() used correctly to convert at least one decimal to whole number	`math.ceil (gramsCheese / MIN_CHEESE)`
				`math.ceil (numRolls)`
				
				● Can be awarded in addition to MP4.11	

			Levels-based mark scheme to a maximum of 3, from:		
	4.13 4.14 4.15		Functionality (3)	● Sequencing and selection used to control program flow ● input() and print() used to implement keyboard/console I/O ● Built-in subprograms used to abstract functionality	

Test Data

			Crisps		Cheese		Rolls	
Test Data	**Adults**	**Children**	**Partial required**	**Bags to order**	**Partial required**	**Order**	**Partial required**	**Order**
Set 1	10	3	8.49	9		1	16.5	1
Set 2	45	14	38.37	39		5	74.5	4

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**

```
# ------------------------------------------------------------------
# Import libraries
# ------------------------------------------------------------------
import math

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

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

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

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

# =====> Write your code here
numAdults = 0
numChild = 0
gramsCheese = 0
orderCheese = 0
numRolls = 0.0
orderRolls = 0
bagsCrisps = 0.0
orderCrisps = 0

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

# =====> Write your code here

# Get the inputs
numAdults = int (input ("How many adults? "))
numChild = int (input ("How many children? "))

# Calculate the bags of crisps required
bagsCrisps = (numAdults * CRISPS_PER_ADULT) + (numChild * CRISPS_PER_CHILD)
print ("Require: " + str (bagsCrisps) + " bags of crisps")
orderCrisps = math.ceil (bagsCrisps)
print ("Order " + str (orderCrisps) + " bags of crisps")

# Calculate amount of cheese required
gramsCheese = (numAdults * CHEESE_PER_ADULT) + (numChild * CHEESE_PER_CHILD)
print ("Require: " + str (gramsCheese) + " grams of cheese")
if (gramsCheese <= MIN_CHEESE):
    print ("Order 1 pack of cheese")
else:
    orderCheese = math.ceil (gramsCheese / MIN_CHEESE)
    print ("Order " + str (orderCheese) + " packs of cheese")

# Calculate the number of rolls required
numRolls = (numAdults * ROLLS_PER_ADULT) + (numChild * ROLLS_PER_CHILD)
print ("Require: " + str (numRolls) + " rolls")
numRolls = math.ceil (numRolls)
if (numRolls <= MIN_ROLLS):
    print ("Order 1 pack of rolls")
else:
    orderRolls = math.ceil (numRolls / MIN_ROLLS)
    print ("Order " + str (orderRolls) + " packs of rolls")

```

Question number	MP	Appx. Line	Answer	Additional guidance	Mark
5			Award marks as shown.		
			getChoice()		
	5.1		menu item choice returned from getChoice() subprogram (1)		
			getShape()		
	5.2		Random number generated, even if upper bound is not correct (1)		

	5.3		Upper bound on random number is controlled by length of the array (1)	`random.randint (0, len (pTable) - 1)`
				`random.randint (0, len (pastaShapes) - 1)`
				• Allow omission of -1	

	5.4		One-dimensional indexing used to access shape in array (1)	• Ignore value of index inside the [ ] • Allow use of global pastaShapes, instead of pTable parameter	
			addShape()		
	5.5		String shape name is appended to array (1)	• Allow insert instead of append • Allow even if global pastaShapes is target	
			Subprograms		
	5.6		addShape() and getShape() use the parameter pTable to access the array (1)	• Do not award if pastaShapes appears inside either subprogram	
			Main Program		
	5.7		Condition-controlled loop (while not choosing to exit) (1)		
	5.8		Selection must handle **all** options (exit, get, add, show, error) (1)	• Allow switch statement • Allow hard-coded rather than constants	
	5.9		Final call to getChoice() inside main loop (1)		**(15)**

			Levels-based mark scheme to a maximum of 6, from:	Considerations for levels-based mark scheme:	
	5.10 5.11 5.12		Solution design (3)	• [6.3.2] Consistent use of provided constants for menu handling throughout or use of switch statement with hard-coded constant equivalents • [6.6.3] getShape() is implemented as a function that returns a value • [6.4.1] User message provided for invalid choices (could be in main program loop or in getChoice())	
	5.13 5.14 5.15		Functionality (3)	• [6.1.6] Get, add, and show function correctly • [6.4.1] Exits when option 4 is chosen at both prompts • [6.1.1] Subprogram calls match user numbers selected from the menu	

Test Data

Choice	Additional	Output	Note
3		Bigoli Strozzapreti Trofie Gigli Chitarra Penne Orecchiette Tagliatelle Chonchiglie Fusilli	10 items
2	ZZZZZ		
3		Bigoli Strozzapreti Trofie Gigli Chitarra Penne Orecchiette Tagliatelle Chonchiglie Fusilli ZZZZZ	11 items
1		A shape from the list	Check call to random.randint to make sure it will generate all items from 0 to the length of the table - 1
1		A shape from the list	
5		Invalid input	The program should display an error message and loop. It may display the menu again or it may not. Either way is acceptable.
4		Exits 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**

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**

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

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

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

shape = ""
choice = 0

# ------------------------------------------------------------------
# Subprograms
# ------------------------------------------------------------------
# Get a menu item from the user
def getChoice ():
     # =====> Write your code here
    menuItem = 0
    print ("1 - get a shape")
    print ("2 - add a shape")
    print ("3 - show the shapes")
    print ("4 - exit program")

    # =====> Write your code here
    menuItem = int (input ("Enter a choice "))
    return (menuItem)

```

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

# Get a random shape
def getShape (pTable):
    # =====> Write your code here
    aNumber = 0
    theShape = ""

    aNumber = random.randint (0, len (pTable) - 1)
    theShape = pTable[aNumber]
    return (theShape)

# Add a shape
def addShape (pTable):
    # =====> Write your code here
    shapeName = ""

    shapeName = input ("Type the name to add ")
    pTable.append (shapeName)

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

choice = getChoice ()
# =====> Write your code here

while (choice != EXIT):
    if (choice == GET):
        shape = getShape(pastaShapes)
        print ("Your shape is", shape)
    elif (choice == ADD):
        addShape (pastaShapes)
    elif (choice == SHOW):
        showShapes (pastaShapes)
    else:
        print ("That choice is invalid")
    choice = getChoice ()
```

Question number	MP	Appx. Line	Answer	Additional guidance	Mark
6			Award marks as shown.		
	6.1		Process every record in the input file (1)		
	6.2		Strip off the line feed on the last field (1)		
	6.3		Split the line on the commas (1)	• Requires argument of comma	
	6.4		First two characters of breed or name extracted (1)		

	6.5		Number component of key calculated correctly (1)	`int(tagNumb) // 100`
				`int (fields[2]) // 100`
				
				• Award calculation, even if number not in correct place in the key	

	6.6		Subprogram called to display table (1)	• Must be accurate call with correct parameter and no other extraneous operations	
			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)	• [6.4.2] Open file for reading and close if required • [6.2.2] Iteration is used appropriately • [6.3.1] Data types and structures are used appropriately	
	6.10 6.11 6.12		Good programming practices (3)	• [6.1.4] Program code is laid out in clear sections; white space is used to show different parts of the solution/functionality • [6.1.4] Variable names are meaningful; comments are provided and are helpful in explaining logic	**(15)**

	6.13 6.14 6.15		Functionality (3)	• [6.3.1] Correct order for each field in the record (key, tag, name, breed) and each record in the table • [6.1.1] Use decomposition to solve problem and create solution • [6.1.2] Write in a high-level language	

Output:

```
['Hi25An', '2569', 'Annabelle', 'Highland']
['Sh37Bo', '3798', 'Bonnie', 'Shetland']
['Be45Ca', '4521', 'Carmine', 'Belted Galloway']
['Hi57De', '5736', 'Delores', 'Highland']
['Sh65Ev', '6504', 'Evette', 'Shetland']
['Be77Fr', '7713', 'Francis', 'Belted Galloway']
```

Award where tag number is an integer:

```
['Hi25An', 2569, 'Annabelle', 'Highland']
['Sh37Bo', 3798, 'Bonnie', 'Shetland']
['Be45Ca', 4521, 'Carmine', 'Belted Galloway']
['Hi57De', 5736, 'Delores', 'Highland']
['Sh65Ev', 6504, 'Evette', 'Shetland']
['Be77Fr', 7713, 'Francis', 'Belted Galloway']
```

Award tuple output:

```
('Hi25An', '2569', 'Annabelle', 'Highland')
('Sh37Bo', '3798', 'Bonnie', 'Shetland')
('Be45Ca', '4521', 'Carmine', 'Belted Galloway')
('Hi57De', '5736', 'Delores', 'Highland')
('Sh65Ev', '6504', 'Evette', 'Shetland')
('Be77Fr', '7713', 'Francis', 'Belted Galloway')
```

Do not award where each record is a single string

```
'Hi25An,2569,Annabelle,Highland'
'Sh37Bo,3798,Bonnie,Shetland'
'Be45Ca,4521,Carmine,Belted Galloway'
'Hi57De,5736,Delores,Highland'
'Sh65Ev,6504,Evette,Shetland'
'Be77Fr,7713,Francis,Belted Galloway'
```

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
# ------------------------------------------------------------------
cowTable = []

# =====> Write your code here
fields = []
key = ""
record = []

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

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

# Process all lines in the file
for line in file:
    line = line.strip()
    fields = line.split (",")

    # Build the key
    key = fields[1][0] + fields[1][1]
    key = key + str (int (fields[2]) // 100)
    key = key + fields[0][0] + fields[0][1]

    # Construct the record
    record = [key, fields[2], fields[0], fields[1]]

    # Save the new record
    cowTable.append (record)

showTable (cowTable)

file.close ()
```