Pearson Edexcel International Advanced Level

Sample assessment material

Time 3 hours

Paper reference **WCP04/01**

Computer Science

International Advanced Level
Unit 4: Advanced Practical Programming and Problem-solving

You must have:
- a computer workstation with appropriate programming language code editing software and tools, including an IDE that you are familiar with that shows line numbers
- a 'STUDENT CODING' folder containing code and data files
- digital copy of the program language subset (PLS)

Instructions

- Answer **all** questions on your computer.
- Save the new or amended code in the 'COMPLETED CODING' folder using the name given in the question.
- Do **not** overwrite the original code and data files provided to you.
- You must **not** use the internet at any time during the examination.

Information

- The total mark for this paper is 80.
- The marks for **each** question are shown in brackets
  – use this as a guide as to how much time to spend on each question.
- The 'STUDENT CODING' folder in your user area includes all the code and data files you need.

Advice

- Read each question carefully before you start to answer it.
- Save your work regularly.
- Check your answers and work if you have time at the end.
- Try to answer every question.

S85797A

1/1/1/1/1/1/1/1/1

S 8 5 7 9 7 A

Pearson

Answer ALL questions.

Suggested time: 15 minutes

1 A program is intended to implement searching and validation with regular expressions.

Open file **Q01.py**

Amend the code to:

- fix the syntax error on original line 5
```
import
```
- fix the syntax error on original line 15
```
"4. UpDown 15 GB 12 months 120 minutes 50 texts £4.44
per month"
```
- fix the logic error on original line 25
```
regex = r" "
```
- fix the AttributeError and the NameError on original line 29
```
match = re.splot(regextra, offer)
```
- fix the logic error on original line 36, which should find all the offers with an unlimited number of minutes
```
regex = r"minutes"
```
- fix the logic error on original line 45
```
regex = r"^2"
```
- fix the logic error on original line 54
```
regex = r"[letter][number][letter]"
```
- fix the logic error on original line 65 to validate the values for DD in DD-MM-YY.
```
regex = r"^(0$"
```

The valid values for DD are:

- the digit zero followed by a digit one to nine:
  - 01, 02, 03, 04, 05, 06, 07, 08, 09
- the digit one or two followed by a digit zero to nine:
  - 10, 11, 12, 13, 14, 15, 16, 17, 18, 19
  - 20, 21, 22, 23, 24, 25, 26, 27 28, 29
- the digit three followed by the digit zero or one:
  - 30, 31

Execute the code to test its functionality.

Do **not** change the intended functionality of the provided code.

Do **not** add any additional functionality.

Save your amended code as **Q01FINISHED.py**

(Total for Question 1 = 12 marks)

Suggested time: 20 minutes

2 A program uses the functional programming techniques of pure functions and lazy evaluation.

Pure function

A pure function only changes states, variables and data that are created inside the function and always returns a result. Pure functions have no side effects, such as changing global variables.

Lazy evaluation

Lazy evaluation delays evaluation of an expression until its value is needed.

The program

The `add_member()` function adds a member to a new list.

The member key is the first item in each record of the list.

The user enters the birth year and the birth month.

The key is calculated by adding the birth year to the birth year modulo the birth month. Key collisions are ignored.

The `lazy_fives()` function produces a sequence of integers evenly divisible by five, using lazy evaluation.

The output is from a fully-functional program when the user enters a birth month of 6 and a birth year of 2005.

Output

```
Enter your birth month (1 is January): 6
Enter your birth year in four digits: 2005

Original members before add_member call
[[1996, 'Adams', 'Amanda', 1, 1996],
 [2002, 'Brown', 'Brenda', 6, 2000],
 [2000, 'Collins', 'Christopher', 3, 1999]]

Original members after add_member call
[[1996, 'Adams', 'Amanda', 1, 1996],
 [2002, 'Brown', 'Brenda', 6, 2000],
 [2000, 'Collins', 'Christopher', 3, 1999]]

New list of members after add_member call
[[1996, 'Adams', 'Amanda', 1, 1996],
 [2002, 'Brown', 'Brenda', 6, 2000],
 [2000, 'Collins', 'Christopher', 3, 1999],
 [2006, 'Wilson', 'Wilma', 6, 2005]]

0 5 10
```

Open file **Q02.py**

Amend the code to:

- implement `add_member()` as a pure function
- implement `lazy_fives()` so that it can be evaluated lazily
- implement the main program to call the functions.

Use the variables provided.

Execute the code to test its functionality.

Do **not** change the intended functionality of the provided code.

Do **not** add any additional functionality.

Save your amended code as **Q02FINISHED.py**

(Total for Question 2 = 13 marks)

Suggested time: 45 minutes

3 A department is modelled using object-oriented programming. The department has employees. Each employee is a person.

The output is from a fully-functional program.

Output

```
========== Create and show a person ==========
Person 1
Ansel Alice
========== Create and show employees ==========
Employee 1
Banks Barbara 30/04/2015
Employee 2
Cash Connie 29/04/2020
========== Create and show the department ==========
The Department
Banks Barbara 30/04/2015
Cash Connie 29/04/2020
```

Open file **Q03.py**

Amend the code to:

- create and show a person, by adding code to the Person class and the main program
- create and show employees, by adding code to the Employee class and the main program
- create and show the department, by adding code to the Department class and the main program.

Use the classes, attributes, methods and variables provided.

Execute the code to test its functionality.

Do **not** change the intended functionality of the provided code.

Do **not** add any additional functionality.

Save your amended code as **Q03FINISHED.py**

(Total for Question 3 = 17 marks)

Suggested time: 45 minutes

4 The names of eight foods are stored in a list of records. The names must be inserted into a binary search tree.

Binary search tree

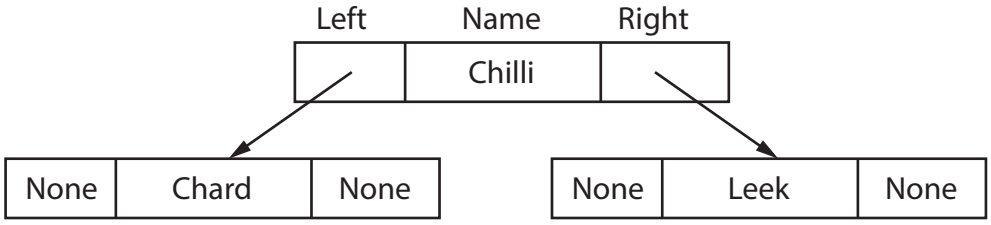
Each node, in the binary search tree, contains a name and two pointer values.

The binary search tree is ordered:

- a left pointer points to a node with a name earlier in the alphabet
- a right pointer points to a node with a name later in the alphabet
- a pointer value of None represents no pointer.

The binary search tree shows the initial state of the tree given in the program code file.

Binary search tree



The output is from a fully-functional program.

Output

```
========== Building tree ==========
========== Tree complete ==========
========== Post Order Traversal ==========
Carrot
Beet Root
Chard
Fennel
Peas
Sweet Corn
Squash
Lettuce
Tomato
Leek
Chilli
```

Open file **Q04.py**

Amend the code to:

- create a procedure to insert a node into the binary search tree. The node is passed as a parameter
- create a new node for each food in the list of food. The program must work for any length of list
- insert each new node into the binary search tree, using the insert procedure
- create a procedure that implements a recursive post-order traversal of the binary search tree to display the food name in each node
- call the post-order traversal procedure.

Use the class and variables provided.

Execute the code to test its functionality.

Do **not** change the intended functionality of the provided code.

Do **not** add any additional functionality.

Save your amended code as **Q04FINISHED.py**

(Total for Question 4 = 18 marks)

Suggested time: 55 minutes

5 A program is required to calculate the scores of dives and display leader boards for a 10-metre platform synchronised diving competition.

Synchronised diving is when two people dive at the same time, side by side, from the same platform.

The competition has four rounds, with one dive per team. There are five teams. Each team has two divers.

Each dive is scored by 11 judges:

- three judges score diver 1
- three judges score diver 2
- five judges score the synchronisation.

A judge's score is a real number, between 0 and 10, in 0.5 increments.

Input

- the team number and name of the diving teams, stored in a dictionary
- header titles for the leader board, stored in a list
- a leader board, holding the team numbers, in numerical order by team, stored in a two-dimensional list
- four rounds, holding information about one dive for each team, in numerical order by team, stored in a two-dimensional list.

Round 1 input shows the information given in the program code file.

Round 1 input

Team number	Difficulty ↓	Judges' scores diver 1	Judges' scores diver 2	Judges' scores synchronisation
[[17,	2.0,	8.5, 9.0, 8.5,	9.0, 8.5, 9.5,	8.5, 9.5, 9.0, 9.0, 9.5],
[23,	2.0,	9.0, 8.0, 9.0,	8.0, 9.0, 9.0,	8.5, 8.5, 8.0, 8.0, 8.5],
[28,	3.3,	9.5, 9.0, 9.0,	9.0, 9.5, 9.0,	8.5, 9.5, 8.0, 8.5, 9.5],
[45,	3.0,	9.5, 9.5, 9.5,	8.0, 9.5, 8.5,	8.5, 9.0, 8.0, 9.5, 9.0],
[83,	3.0,	9.0, 9.5, 8.0,	8.0, 8.0, 9.0,	9.5, 9.5, 8.5, 9.0, 9.5]
]				

Processes

- The raw score for a dive is calculated as the sum of three median scores:
  - median of diver 1's three scores
  - median of diver 2's three scores
  - median three scores of the five scores for synchronisation.

  For team 17 in **round 1 input**, the calculation of the raw score is:

  8.5 + 9.0 + (9.0 + 9.0 + 9.5) = 45.0
- The round score for a dive is calculated to two decimal places as:
  - raw score × scale adjustment of 0.6 × difficulty

  For team 17 in **round 1 input**, the calculation of the round score is:

  45.0 × 0.6 × 2.0 = 54.00
- The total score after each round is the sum of the round scores so far.

Output

- Four leader boards, showing the round scores and the total score for the team. The leader board is sorted in descending order by total score.

Round 1 output shows the leader board at the end of round 1, from a fully-functional program.

Round 1 output

Team	Name	Round 1	Round 2	Round 3	Round 4	Total
28	Warriors	88.11	-	-	-	88.11
83	Chargers	81.00	-	-	-	81.00
45	Orcas	80.10	-	-	-	80.10
17	Pirates	54.00	-	-	-	54.00
23	Hawks	51.60	-	-	-	51.60

Round 4 output shows the leader board at the end of round 4, from a fully-functional program.

Round 4 output

Team	Name	Round 1	Round 2	Round 3	Round 4	Total
28	Warriors	88.11	88.32	87.72	44.37	308.52
83	Chargers	81.00	72.24	84.48	49.59	286.41
45	Orcas	80.10	63.36	54.81	71.28	270.45
17	Pirates	54.00	70.47	56.10	82.77	263.34
23	Hawks	51.60	76.50	73.71	58.74	260.55

Open file **Q05.py**

Write the code to:

- calculate the scores after each round of dives
- display the leader board after each round of dives
- ensure the original dive data is not changed by the processing.

Use best practice techniques to:

- design effective and efficient code
- write clean code
- ensure the solution functions as intended.

Use the constants and variables provided.

Execute the code to test its functionality.

Do **not** change the intended functionality of the provided code.

Do **not** add any additional functionality.

Save your amended code as **Q05FINISHED.py**

(Total for Question 5 = 20 marks)

TOTAL FOR PAPER = 80 MARKS