CS6.302 - Software System Development

Lab - 10 :: Python – Variables, Operations, Control Flow: Branching and Looping

Lab Activity :: Due: 10 Oct 2025, 05:00 PM :: Marks: 50

General Instructions:

- This is an individual activity. Please follow the questions properly and answer them
- We expect one submission from one student. Please follow submission guidelines mentioned below. If you don't follow the submission guidelines, you will receive '0'.
- Validate inputs and handle invalid or edge cases gracefully.

I. Write a Python program called `Q1.py` that asks the user to enter a positive integer n and then prints all the numbers from 1 to n that meet the following rule: **[2 marks]**

   a. If the number is divisible by 3, print "Three"
   b. If the number is divisible by 5, print "Five"
   c. If the number is divisible by both 3 and 5, print "ThreeFive"
   d. Otherwise, print the number itself

   Below is the output format:

```
Enter a positive integer: 5

1

2

Three

4

Five
```

II. Write a Python program named `Q2.py` that finds and prints all "mystical numbers" within a user-defined range.
   A number is considered "mystical" if it meets **all three** of the following conditions:

   - The number is **even**.
   - The sum of its digits is a **prime number**.
   - The digits of the number are in **strictly ascending order** (e.g., 135 is valid, but 122 or 153 are not).

   Your program should first prompt the user to enter a starting and ending number for the range. It should then find and print each mystical number, along with its digit sum, in the following format:

```
Mystical Number: [number] -> Digit Sum: [sum]
```

   [4 marks]

Below is the output format:

```
Please enter the starting number for the range: 100
Please enter the ending number for the range: 500
Searching for mystical numbers...
------------------------------
Mystical Number: 124 -> Digit Sum: 7
Mystical Number: 128 -> Digit Sum: 11
Mystical Number: 146 -> Digit Sum: 11
Mystical Number: 148 -> Digit Sum: 13
Mystical Number: 236 -> Digit Sum: 11
Mystical Number: 346 -> Digit Sum: 13
```

III. Write a Python program called `Q3.py` that tests the user's memory. The program should: **[3 marks]**

a. Ask the user how many numbers they want to memorize (between 3 and 5).
b. Generate a sequence of random numbers (between 1 and 9), one at a time, and display them to the user for a brief moment.
c. After the full sequence is shown, clear the screen (or simulate it using line breaks) and ask the user to input the numbers in the same order.
d. At the end, print:
    i. "Correct! You have a great memory!" if all guesses were correct.
    ii. "Oops! You remembered X out of Y correctly." if some numbers were wrong.

Below is the output format:

```
How many numbers do you want to memorize (between 3
and 5): 4
Remember these numbers:
7
2
9
5
(Press Enter when ready to continue...)
```

```
<Clear the screen once user press 'Enter'>
Now enter the numbers in the same order:
Enter number 1: 7
Enter number 2: 2
Enter number 3: 9
Enter number 4: 8
Oops! You remembered 3 out of 4 correctly!
```

IV. Write a Python program to construct the following patterns.

a) n = 5 **[5 Marks]**

```
    *
   * *
  * * *
 * * * *
* * * * *
```

b) n = 5 **[5 Marks]**

```
1  2  3  4  5
16 17 18 19 6
15 24 25 20 7
14 23 22 21 8
13 12 11 10 9
```

c) n = 5 **[5 Marks]**

```
*********
 *     *
  *   *
   * *
    *
   * *
  *   *
 *     *
*********
```

Note:

i. We are providing you with **template files** for each pattern question

   a. `Q4.py` for Question IV (a)
   b. `Q5.py` for Question IV (b)
   c. `Q6.py` for Question IV (c)

ii. **Do not modify anything else in the template** except replacing the pass statement with your implementation.

iii. You must submit **both the code file (`.py`) and the output file (`.txt`)**. Refer to Submission instructions for more details.

   a. To generate the output file, run the program using following command:

```
python .\Q4.py 5 > Q4_output.txt
```

   (Replace Q4.py with Q5.py or Q6.py as required. Also feel free to replace '5' with different 'n' value)

iv. The output format must match exactly as given in the question (spaces, newlines, symbols).

V. You are a "Mastermind Logician." Your task is to write a single Python program named `Q7.py` that simulates a game over 10 rounds and calculates a "Final Score." Your program must take user input to determine the moves for each player.
Your program must prompt the user to enter two sequences of 10 numbers each, representing the moves for Player A and Player B. The numbers should be entered as a single line, separated by spaces.
Your program must apply the following rules, in a logical order, for each of the 10 rounds.

   a. **Base Score:** For each round, add a number to the total score. This number is the value of **Player A's Move** for that round.
   b. **Penalty Condition:** A penalty is applied if a specific condition is met. If the penalty applies, subtract the value of **Player B's Move** for the round. The condition for the penalty is met if **Player A's move is less than Player B's move**.
   c. **Bonus Condition:** A bonus is applied if a separate condition is met. If the bonus applies, add the value of **Player B's Move** for the round. The condition for the bonus is met if **Player A's and Player B's moves are identical.**

[4 marks]

Below is the output format:

```
Enter 10 moves for Player A (1-6), separated by
spaces: 4 6 2 5 1 3 5 6 4 1

Enter 10 moves for Player B (1-6), separated by
spaces: 4 5 2 3 1 2 5 6 3 4

The final numeric score after all 10 rounds: 45
```

VI. Implement a text-based adventure game (**Crystal Cavern Explorer**) named `Q8.py` where a player navigates through a mystical cavern, collects crystals, and avoids traps. Your program must calculate a final score based on a complex set of rules.

Game Setup:

- Player starts with 100 health points and 0 crystals
- The cavern has 8 chambers (numbered 1-8)
- Player begins in chamber 1 and must reach chamber 8
- Each chamber contains either a crystal, trap, or is empty

The program should prompt user to enter the chamber configuration and it should accept exactly 8 characters: 'C' (Crystal), 'T' (Trap), 'E' (Empty) in upper case.

Game Rules: Your program must apply the following rules, in a specific logical order, for each chamber the player enters.

1. **Crystal Collection:**
   - Each crystal found adds **15 points** to the player's score.
   - If the current chamber number is a prime number (2,3,5,7), the crystal gives **double points**.
   - If the number of crystals collected **so far** is an even number, the crystal's value is also **doubled**. This multiplier stacks.
2. **Trap Encounters:**
   - Each trap reduces health by **20 points**.
   - If the player's **current health** is divisible by the **current chamber number**, the trap damage is **halved**.
3. **Empty Chambers:**
   - The player rests and recovers **5 health points**. Health cannot exceed 100.
4. **Survival Check:**
   - If health drops to **0 or below** at any point, the game ends immediately, and the final score is **0**.
   - If the player reaches chamber 8 alive, calculate the final score.
5. **Final Score Calculation:**
   - **Base Score:** (crystals collected × crystal points earned)
   - **Health Bonus:** remaining health ÷ 2
   - **Final Score:** Base Score + Health Bonus

[8 marks]

Below is the output format:

```
Enter chamber configuration (8 characters): CTECTTCE
=== Crystal Cavern Explorer ===
Chamber 1 (C): Found crystal! +15 points
Chamber 2 (T): Hit trap! -20 health
Chamber 3 (E): Empty chamber, resting... +5 health
Chamber 4 (C): Found crystal! +30 points
Chamber 5 (T): Hit trap! -10 health
Chamber 6 (T): Hit trap! -20 health
Chamber 7 (C): Found crystal! +60 points (Bonus applied!)
Chamber 8 (E): Safe exit!

Final Status: Health=55, Crystals=4, Crystal Points=120
Final Score: 180
```

VII. Write a Python program called `Q9.py` which calculates scores from character sequences using specific rules. The program should exactly accept 5 characters from the user (letters A-Z and digits 0-9).

Scoring Rules: Apply these rules in order:

1. **Basic Points:**
   - Each letter gets points equal to its alphabet position (A=1, B=2, ..., Z=26)
   - Each digit gets points equal to the digit value × 3
2. **Position Bonus:**
   - If a character is in an odd position (1st, 3rd, 5th), add 5 extra points
   - If a character is in an even position (2nd, 4th), add 10 extra points
3. **Special Rule:**
   - If the total score so far is even, multiply it by 1.2 (use integer division)
   - If the total score so far is odd, add 15 to it

[**4 marks]**

Below is the output format:

```
Enter 5 characters: A1B2C
Final Score: 60


Enter 5 characters: X3Y4Z
Final Score: 146
```

VIII. Write a Python program called `Q10.py` that processes multiple rounds of character sequences through various computational stages. **[10 marks]**

Input Requirements:

- First prompt: "Enter number of rounds (2-5):"
- For each round, prompt: "Round X - Enter 4 characters:"
- Accept exactly 4 characters per round (letters A-Z and digits 0-9)

Processing Stages:

Stage 1: Individual Round Processing For each round, process the 4 characters:

- Letters: Points = alphabet position (A=1, B=2, etc.)
- Digits: Points = digit value × position in round (1st digit ×1, 2nd digit ×2, etc.)
- Apply position modifier: Characters in odd positions get +3 bonus, even positions get +7 bonus
- Calculate round total

Stage 2: Round-to-Round Effects

- If current round total is greater than previous round total, multiply current round by 1.5 (integer division)
- If current round total is divisible by the round number, add 20 bonus points
- If current round total contains the digit '7' when written as a number, subtract 10 points
- Store modified round total

Stage 3: Cross-Round Analysis After processing all rounds:

- Find the highest round score and lowest round score
- Calculate average of all round scores (integer division)
- Count how many rounds had scores above the average

Stage 4: Final Calculation

- Base score = sum of all modified round totals
- Apply multiplier based on number of above-average rounds:
  - 0-1 above average: multiply by 0.9
  - 2-3 above average: multiply by 1.1
  - 4+ above average: multiply by 1.3
- If base score is palindrome (reads same forwards/backwards), add 50 bonus
- Throughout the calculation, result uses integer arithmetic

Below is the output format:

```
Enter number of rounds (2-5): 3

Round 1 - Enter 4 characters: A2B3

Round 2 - Enter 4 characters: X1Y4

Round 3 - Enter 4 characters: M5N6

Processing Results:

Round 1 Score: 89

Round 2 Score: 124

Round 3 Score: 156

Above Average Rounds: 2

Final Score: 447
```

Submission Instructions:

Please follow below instructions carefully. A strict zero will be given if the submission format doesn't adhere to this.

- Programs must follow the output format provided. This includes each blank line, colons (:), and other symbols.
- Programs must be working correctly.
- Programs must be written in Python.
- Programs must be submitted with the correct .py format. You must submit **both the code file (`.py`) and the output file (`.txt`)**
- Programs must be saved in files with the correct file name given in each question.
- Please feel free to make any **valid assumptions** wherever the question seems ambiguous. Clearly mention these assumptions as **comments in your code**. Marks will be awarded **only if the assumptions are stated explicitly**. Any **invalid assumptions will lead to penalties**.
- Submissions that do not adhere to the file structure, naming convention, or output format **will not be graded**.
- **Submission Format**:
  - You are required to submit this lab activity as <roll_number>.zip. For example, if you roll number 20162153, then your submission file should be 20162153.zip.
  - Inside this .zip, create **separate folders for each question (Q1, Q2,...Q10).**
  - Each question folder must contain:
    - The respective **Python code file** (Q1.py, Q2.py, ... Q10.py)
    - The respective **output file** (Q1_output.txt, Q2_output.txt, ... Q10_output.txt)