Editorial-Solution-W3A1: Mutable Defaults, Higher-Order Functions, and File Operations

---

Question 1:

What will be the output of the following code?

```
def func(a, b=[]):
    b.append(a)
    return b

print(func(1))
print(func(2))
print(func(3))
```

Options: A)

[1]

[2]

[3]

B)

[1]

[1, 2]

[1, 2, 3]

C)

[1]

[2]

[3, 2, 1]

D)

[1]

[1]

[1]

Answer: B)

[1]

[1, 2]

[1, 2, 3]

Explanation:

- In Python, default arguments are evaluated **only once** when the function is defined, not each time the function is called.
- The default argument b=[] is a **mutable object** (a list). This means that the same list is reused across multiple function calls.
- When func(1) is called, b is initially [], and 1 is appended to it. The function returns [1].
- When func(2) is called, b is **not reset** to []. Instead, it retains its previous value [1], and 2 is appended to it. The function returns [1, 2].
- Similarly, when func(3) is called, b is now [1, 2], and 3 is appended to it. The function returns [1, 2, 3].
- This behavior occurs because the **same list object** is being modified across function calls.

Question 2:

What will be the output of the following code?

```
def outer(x):
    def inner(y):
        return x + y
    return inner

add_five = outer(5)
print(add_five(10))
```

Options: A) 5

B) 10

C) 15

D) Error

Answer: C) 15

Explanation:

- The outer function takes an argument x and defines an **inner function** inner that takes an argument y.

- The inner function returns the sum of x and y.
- When outer(5) is called, it returns the inner function with x fixed as 5. This is an example of a **closure**, where the inner function "remembers" the value of x from the outer function's scope.
- The returned function is assigned to add_five. Now, add_five is essentially inner with x=5.
- When add_five(10) is called, it computes 5 + 10 = 15.

Question 3:

What will be the output of the following code?

```
def func(x):
    return x * 2


lst = [1, 2, 3, 4]
result = list(map(lambda x: func(x), lst))
print(result)
```

Note:

- The map function applies a given function to each item in an iterable (in this case, the list lst).
- The lambda x: func(x) is an anonymous function that calls func(x) for each element x in lst.

Options: A) [2, 4, 6, 8]

B) [1, 2, 3, 4]

C) [1, 4, 9, 16]

D) Error

Answer: A) [2, 4, 6, 8]

Explanation:

- The map function applies a given function to each item in an iterable (in this case, the list lst).
- The lambda x: func(x) is an anonymous function that calls func(x) for each element x in lst.
- The func(x) function doubles the value of x.

- Therefore:
    - func(1) returns 2
    - func(2) returns 4
    - func(3) returns 6
    - func(4) returns 8
- The map function returns an iterator, which is converted to a list using list(). The final result is [2, 4, 6, 8].

Question 4:

What will be the output of the following code?

```
def func(a, b, c):
    return a + b + c


values = (1, 2, 3)
print(func(*values))
```

Options: A) 6

B) (1, 2, 3)

C) Error

D) None

Answer: A) 6

Explanation:

- The values syntax is used to **unpack** the tuple (1, 2, 3) into individual arguments for the function func.
- The function func takes three arguments: a, b, and c.
- After unpacking, the function call becomes func(1, 2, 3).
- The function computes 1 + 2 + 3 = 6.

Note : For Question 5 to Question 8 Please refer to the Kartik's Sir class Google Colab: Link to the Colab

Question 5:

What is the purpose of the game() function in the provided code?

Options:

A) To simulate a battle between two players with random attacks and defenses.

B) To calculate the probability of winning for each player.

C) To generate random numbers for a dice game.

D) To create a graphical user interface for a game.

Answer: A) To simulate a battle between two players with random attacks and defenses.

Explanation:

- The game() function simulates a turn-based battle between two players, Player 1 and Player 2.
- Each player has a health pool (Player1_HP and Player2_HP), and they take turns attacking or defending.
- The function uses random damage values between 10 and 20 for attacks and halves the damage if the opponent is defending.
- The game continues until one player's health drops to 0 or below, and the winner is declared

---

Question 6:

What happens when a player chooses to defend in the game() function?

Options:

A) The player's health is fully restored.

B) The player takes half damage during the opponent's attack.

C) The player's next attack deals double damage.

D) The player's health is reduced by half.

Answer: B) The player takes half damage during the opponent's attack.

Explanation:

- When a player chooses to defend, the Player1_defending or Player2_defending flag is set to True.
- If the opponent attacks while the player is defending, the damage is halved (damage = damage // 2).
- This effectively reduces the damage taken during the opponent's next attack, simulating a defensive action.

Question 7:

What is the role of the computer_choice() function in the code?

Options:

A) It randomly selects between attack and defend for Player 1.

B) It determines the optimal move for Player 2 based on the game state.

C) It calculates the total damage dealt by both players.

D) It ends the game when a player's health reaches 0.

Answer: B) It determines the optimal move for Player 2 based on the game state.

Explanation:

- The computer_choice() function is used to decide whether Player 2 should attack or defend.
- It takes into account the current health of both players (Player1_HP and Player2_HP) and whether either player is defending (Player1_defending or Player2_defending).
- The function uses conditional logic to make decisions, such as attacking if Player 1 is defending or if Player 1's health is low, and defending if Player 2's health is low.
- This simulates a basic AI for Player 2, making the game more dynamic.

Question 8:

What is the significance of the turn variable in the game() function?

Options:

A) It keeps track of the total number of turns played in the game.

B) It determines which player's turn it is to attack or defend.

C) It calculates the remaining health of both players.

D) It decides the winner of the game.

Answer: B) It determines which player's turn it is to attack or defend.

Explanation:

- The turn variable alternates between 1 and 2 to indicate whose turn it is.
- When turn == 1, it is Player 1's turn to attack or defend.
- When turn == 2, it is Player 2's turn to attack or defend.
- After each turn, the value of turn is updated to switch to the other player's turn.

- This ensures that the game alternates between the two players until one of them loses all their health.

Question 9:

What will be the output of the following code?

```
f1 = open("student.txt", "w")
f1.write("Hello, World!")
f1.close()

f1 = open("student.txt", "r")
print(f1.read(5))
f1.close()
```

Note: student.txt file never existed before.

Options: A) Hello

B) Hello,

C) World

D) Error

Answer: A) Hello

Explanation:

1. **File Opening in Write Mode ('w'):**
    - The file "student.txt" is opened in write mode ('w').
    - The string "Python Programming" is written to the file.
    - The file is closed using f1.close().
2. **File Opening in Read-Write Mode ('r+'):**
    - The file is reopened in read-write mode ('r+'), which allows both reading and writing.
    - f1.write("Java") writes the string "Java" to the file. Since the file pointer is at the beginning, this overwrites the first 4 characters of the file, changing "Python" to "Java".
    - The file now contains "Java Programming".
3. **File Pointer Manipulation:**

- f1.seek(0) moves the file pointer to the beginning of the file.
- f1.read() reads the entire file from the current position (beginning) to the end, which is "Java Programming".

4. **File Closing:**
    - The file is closed again using f1.close().

Question 10:

What will be the output of the following code?

```
f1 = open("student.txt", "w")
f1.write("Line 1\nLine 2\nLine 3")
f1.close()

f1 = open("student.txt", "r")
print(len(f1.readlines()))
f1.close()
```

Note: student.txt file never existed before.

Options: A) 1

B) 2

C) 3

D) Error

Answer: C) 3

Explanation:

- The file is written with three lines: "Line 1", "Line 2", and "Line 3", each separated by a newline character (\n).
- f1.readlines() reads all lines from the file and returns them as a list.
- The length of the list is 3, so the output is 3.

Question 11:

What will be the output of the following code?

```
f1 = open("student.txt", "w")
f1.write("Python Programming")
```

```
f1.close()


f1 = open("student.txt", "r+")
f1.write("Java")
f1.seek(0)
print(f1.read())
f1.close()
```

Note: student.txt file never existed before.

Options: A) Python Programming

B) Java Programming

C) Java

D) Javaon Programming

Answer: D) Javaon Programming

Explanation:

1. The file **"student.txt"** is first opened in **write ("w") mode**, which creates the file (if it doesn't exist) and writes "Python Programming" into it. After writing, the file is closed.
2. The file is then reopened in **read+write ("r+") mode**. This allows both reading and writing.
3. The statement f1.write("Java") writes "Java" at the beginning of the file, **overwriting** the first four characters ("Pyth" from "Python Programming"). The file content now becomes "Javaon Programming".
4. f1.seek(0) moves the file pointer back to the beginning of the file.
5. print(f1.read()) reads the entire file content from the beginning, which is now **"Javaon Programming"**.

Thus, the correct output is **"Javaon Programming"**, making option **(B)** the correct answer.

Question 12:

What will be the output of the following code?

```
f1 = open("student.txt", "w")
f1.write("A\nB\nC\nD")
f1.close()
```

```
f1 = open("student.txt", "r")
f1.seek(2)
print(f1.read())
f1.close()
```

Note:

- Here, we are using Linux based operating system.
- student.txt file never existed before.

Options: A)

A

B

C

D

B)

B

C

D

C)

C

D

D) Error

Answer: B)

B

C

D

Explanation:

1. The file **student.txt** is first opened in write mode ("w") and the string "A\nB\nC\nD" is written to it. This means the file contains the following characters:
2. A\nB\nC\nD

Here, \n represents a newline character.

3. The file is then opened in read mode ("r") and seek(2) is used. The **seek()** function moves the file pointer to the specified position (character index 2).
4. Let's break down the file content with indexes:
5. Index: 0 1 2 3 4 5 6 7
6. Content: A \n B \n C \n D
    - A is at index 0
    - \n (newline) is at index 1
    - B is at index 2
    - \n (newline) is at index 3
    - C is at index 4
    - \n (newline) is at index 5
    - D is at index 6
7. Since seek(2) moves the pointer to **index 2**, reading from this position will output everything starting from "B", including the subsequent newline and characters.
8. The read() function then prints:
9. B
10. C
11. D

Thus, the correct answer is Option B.

Question 13:

What will be the output of the following code?

```
f1 = open("student.txt", "w")
f1.write("Hello\\\\nWorld")
f1.close()

f1 = open("student.txt", "a")
f1.write("\\\\nPython")
f1.close()

f1 = open("student.txt", "r")
```

print(f1.readlines())

f1.close()

Note: student.txt file never existed before.

Options: A) ['Hello\\\\n', 'World\\\\n', 'Python']

B) ['Hello\\\\n', 'World\\\\nPython']

C) ['Hello\\\\n', 'WorldPython']

D) ['Hello\\\\nWorld\\\\nPython']

Answer: D) ['Hello\\\\nWorld\\\\nPython']

Explanation:

1. **Writing to the file ("w" mode):**
2. f1 = open("student.txt", "w")
3. f1.write("Hello\\\\nWorld")
4. f1.close()
    - The write function stores the string **exactly** as it is, meaning "Hello\\\\nWorld" is written to the file.
    - The file content after this step:
    - Hello\\nWorld
5. **Appending to the file ("a" mode):**
6. f1 = open("student.txt", "a")
7. f1.write("\\\\nPython")
8. f1.close()
    - Since the file is opened in append mode ("a"), new content is added at the end without overwriting the previous data.
    - "\\\\nPython" is added, so now the file contains:
    - Hello\\nWorld\\nPython
9. **Reading the file ("r" mode)**
10. f1 = open("student.txt", "r")
11. print(f1.readlines())
12. f1.close()
    - The readlines() function reads the entire file and returns a list of strings, where each element represents a line.

- Since there are no actual newline characters (\n), the entire content remains in a single line.
- Thus, the output is:
- ['Hello\\\\nWorld\\\\nPython']

Question 14:

What will be the output of the following code?

```
try:
    with open("nonexistent.txt", "r") as f1:
        print(f1.read())
except FileNotFoundError:
    print("File not found")
else:
    print("File read successfully")
finally:
    print("Operation complete")
```

Note: nonexistent.txt file never existed before.

Options: A)

File not found

Operation complete

B)

File read successfully

Operation complete

C)

File not found

D) Error

Answer: A)

File not found

Operation complete

Explanation:

1. **Try Block:**

- The try block attempts to open the file "nonexistent.txt" in read mode ('r').
- Since the file does not exist, a FileNotFoundError is raised.

2. **Except Block:**
    - The except block catches the FileNotFoundError and executes its code.
    - print("File not found") is executed, printing "File not found".
3. **Else Block:**
    - The else block is skipped because an exception was raised.
4. **Finally Block:**
    - The finally block is always executed, regardless of whether an exception occurred.
    - print("Operation complete") is executed, printing "Operation complete".

Question 15:

What will be the output of the following code?

```
f1 = open("student.txt", "w")
f1.write("Reg_no\\\\tName\\\\tMark\\\\n1\\\\tAlice\\\\t90\\\\n2\\\\tBob\\\\t85")
f1.close()

with open("student.txt", "r") as f1:
    lines = f1.readlines()
    print(lines[0].split("\\\\t")[3])
```

Note: student.txt file never existed before.

Options: A) Reg_no

B) Alice

C) 90

D) Bob

Answer: B) Alice

Explanation:

1. **Writing to the File ("w" mode):**
2. f1 = open("student.txt", "w")
3. f1.write("Reg_no\\\\tName\\\\tMark\\\\n1\\\\tAlice\\\\t90\\\\n2\\\\tBob\\\\t85")

4. f1.close()
    - The string "Reg_no\\\\tName\\\\tMark\\\\n1\\\\tAlice\\\\t90\\\\n2\\\\tBob\\\\t85" is written exactly as it is.
    - The content stored in the file is:
    - Reg_no\\tName\\tMark\\n1\\tAlice\\t90\\n2\\tBob\\t85
    - The double backslashes (\\\\) indicate that Python is storing literal \t (tab) and \n (newline) as **two separate characters**, not escape sequences.
5. **Reading the File ("r" mode):**
6. with open("student.txt", "r") as f1:
7. lines = f1.readlines()
    - readlines() reads the file as a list of strings, where each element represents a line.
    - lines[0] contains the first line:
    - "Reg_no\\tName\\tMark\\n1\\tAlice\\t90\\n2\\tBob\\t85"
8. **Splitting the First Line Using "\\t":**
9. lines[0].split("\\\\t")
    - Since "\\\\" is stored as \\ in the file, the actual split happens on "\\t" (which is **not a tab character**, just the literal \t string).
    - Splitting "Reg_no\\tName\\tMark\\n1\\tAlice\\t90\\n2\\tBob\\t85" on "\\t" results in:
    - ["Reg_no", "Name", "Mark\\n1", "Alice", "90\\n2", "Bob", "85"]
10. **Accessing the Fourth Element ([3]):**
11. lines[0].split("\\\\t")[3]
    - The **index 3** corresponds to "Alice".

Thus, **the correct answer is:**
B) Alice