Lab Assignment-7.2

Name : Saini.Deepthi

Hall ticket No. : 2403A52014

Batch No. : 24BTCAIAIB02

Course Title : Ai Assisted Coding

Task1:

Description :

>>Syntax Error in Conditionals.

```python
a = 10
if a = 10:
    print("Equal")
```

Expected Output:

>> Corrected function with syntax fix.

Prompt:

>> The following Python code contains a syntax error in its conditional statement: a = 10 if a = 10: print("Equal") Identify the error in the if statement. and rewrite the code to produce the

Expected Output.

```
a = 10
if a = 10:
    print("Equal")
```

```
  File "/tmp/ipython-input-3402872600.py", line 2
    if a = 10:
       ^
SyntaxError: invalid syntax. Maybe you meant '==' or ':=' instead of '='?
```

Next steps: Explain error

Correction of the code:

```
a = 10
if a ==10:
    print("Equal")
```

```
Equal
```

Observation:

>>Based on the execution of the code cell cecaf545, the variable a is assigned the value 10. The if statement then checks if the value of a is equal to 10. Since the condition is true, the code inside the if block, print("Equal"), is executed, resulting in the output "Equal".

Explanation:

>>We have a box labeled a, and we put the number 10 inside it.Then we ask, "Is the number in box a exactly the same as the number 10?"Since the number in box a *is* 10, the answer is

"Yes".Because the answer is "Yes", the code does what it was told to do when the answer is yes, which is to print the word "Equal".

Task2:

Description :

>> Loop Off-By-One Error.

```
def sum_upto_n(n):
    total = 0
    for i in range(1, n):
        total += i
    return total
```

Expected output:

>> AI fixes increment/decrement error

Prompt:

>>Review the Python function sum_upto_n(n) and identify if there's a loop off-by-one error. Explain whether the loop correctly includes all integers from 1 to n, and suggest a fix if the final sum excludes n unintentionally.

```
def sum_upto_n(n):
    total = 0
    for i in range(1, n):
        total += i
    return total

# Call the function and print the result to see the output with the error
print("Result before fixing the error:", sum_upto_n(5))
```

```
Result before fixing the error: 10
```

Correction of the code:

```
def sum_upto_n(n):
  total = 0
  for i in range(1, n + 1): # Changed n to n + 1
    total += i
  return total

# Example usage:
print(sum_upto_n(5)) # This should now correctly output 15 (1+2+3+4+5)
```

```
15
```

Observation:

>>The original loop excludes n because range(1, n) stops at n - 1. By updating it to range(1, n + 1), the loop correctly includes all integers from 1 to n, ensuring accurate summation.

Explanation:

>>The original Python function sum_upto_n(n) contains an off-by-one error because it uses range(1, n), which excludes the value n from the loop. In Python, the range() function is half-open—it includes the start value but stops just before the end value. As a result, when n = 5, the loop only sums 1 + 2 + 3 + 4, missing 5. To fix

this, the loop should use range(1, n + 1) so that it includes all integers from 1 through n.

Task3:

Description:

- Error: AttributeError

```
class User:
    def __init__(self, name):
        self.name = name

u = User("Alice")
print(u.getName())
```

Expected Output :

>>Detect missing self and initialize attributes properly.

Prompt:

>>Consider the following Python class definition and usage: class User: def _init_(self, name): self.name = name u = User("Alice") print(u.getName()) Running this code results in an AttributeError. Can you identify the cause of the error and modify the class so that it correctly prints the user's name?

```
class User:
    def __init__(self, name):
        self.name = name

u = User("Alice")
print(u.getName())
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
/tmp/ipython-input-2327273307.py in <cell line: 0>()
      4
      5 u = User("Alice")
----> 6 print(u.getName())

AttributeError: 'User' object has no attribute 'getName'
```

Correction of the code:

```
class User:
  def __init__(self, name): # Corrected the constructor name
    self.name = name

  def getName(self): # Added the getName method
    return self.name

u = User("Alice")
print(u.getName())
```

```
Alice
```

Observation:

1. We created a blueprint for a "User" that can hold a name.
2. We used this blueprint to make a specific user named "Alice".
3. Then, we asked this "Alice" user to tell us their name using the getName() instruction.
4. Since "Alice" knows her name is "Alice", she told us "Alice".

5. Finally, we showed what "Alice" said, which is why you see "Alice" as the output.

Explanation:

>>The _init_ method is the constructor; it's called when you create a new User object. It takes self (which refers to the instance of the class) and a name argument, and it stores the name in an attribute called self.name. The code then creates an instance of the User class named u, passing "Alice" as the name. Finally, it attempts to call a method named getName() on the u object and print its return value. This is where the AttributeError occurs because no method named getName is defined within the User class.

Task4:

Description:

>>incorrect Class Attribute Initialization

```
class Car:
    def start():
        print("Car started")

mycar = Car()
mycar.start()
```

Expected output:

>>Detect missing self and initialize attributes properly

Prompt:

>>Examine the following Python class definition and method call: class Car: def start(): print("Car started") mycar = Car() mycar.start() Running this code results in a TypeError because of incorrect method definition. Can you identify the issue with the start() method and rewrite the class so that the method works properly ?

```
class Car:
    def start():
        print("Car started")

mycar = Car()
mycar.start()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/tmp/ipython-input-1533921467.py in <cell line: 0>()
      4
      5 mycar = Car()
----> 6 mycar.start()

TypeError: Car.start() takes 0 positional arguments but 1 was given
```

Correction of the code:

Gemini

```
class Car:
    def start(self): # Added 'self' as the first parameter
        print("Car started")

mycar = Car()
mycar.start() # Now this call works correctly
```

```
Car started
```

Observation :

In Python, when you define a method inside a class, it must take self as the first parameter. This self refers to the instance of the class

(like mycar in your example). Without it, Python doesn't know how to pass the object to the method, and that's why you get a TypeError.

Explanation:

>> Python automatically tries to pass the object mycar to the method as the first argument. Since start() wasn't expect ing any arguments, Python throws a TypeError saying it got 1 argument but expected 0.

Task5:

Description:

>> Conditional Logic Error in Grading System.

```
def grade_student(score):
    if score < 40:
        return "A"
    elif score < 70:
        return "B"
    else:
        return "C"
```

Expected Output :

>> Detect illogical grading and correct the grade levels.

Prompt:

>> Write a Python function called grade_student(score) that returns a grade based on the score:

Return "A" if the score is less than40

Return "B" if the score is between 40 and 69 (inclusive of 40, exclusive of 70)

Return "C" if the score is 70 or above

Then, test your function with the following scores: 35, 55, and 85. What grades do you get?

```
def grade_student(score):
    if score < 40:
        return "A"
    elif score < 70:
        return "B"
    else:
        return "C"

# Test cases to show the incorrect output
print(f"Score 35 gets grade: {grade_student(35)}") # Expected: something lower than A, Actual: A
print(f"Score 65 gets grade: {grade_student(65)}") # Expected: something lower than B, Actual: B
print(f"Score 85 gets grade: {grade_student(85)}") # Expected: something higher than C, Actual: C
```

```
Score 35 gets grade: A
Score 65 gets grade: B
Score 85 gets grade: C
```

Correction of the code:

```python
def grade_student_corrected(score):
    if score >= 90:
        return "A"
    elif score >= 80:
        return "B"
    elif score >= 70:
        return "C"
    elif score >= 60:
        return "D"
    else:
        return "F"

# Test cases for the corrected function
print(f"Score 95 gets grade: {grade_student_corrected(95)}")
print(f"Score 85 gets grade: {grade_student_corrected(85)}")
print(f"Score 75 gets grade: {grade_student_corrected(75)}")
print(f"Score 65 gets grade: {grade_student_corrected(65)}")
print(f"Score 55 gets grade: {grade_student_corrected(55)}")
```

```
Score 95 gets grade: A
Score 85 gets grade: B
Score 75 gets grade: C
Score 65 gets grade: D
Score 55 gets grade: F
```

Observation:

>> Score 35 is less than 40 → returns "A"

Score 55 is between 40 and 69 → returns "B"

Score 85 is 70 or more → returns "C"

The function works correctly and assigns grades based on the score ranges you specified. It uses simple conditional logic (if, elif, else) to check where the score falls and returns the appropriate grade.

Explanation:

The function grade_student(score) is designed to return a grade based on the numeric value of a student's score. It uses conditional statements to evaluate the score and assign a grade accordingly. If the score is less than 40, it returns "A"; if the score falls between 40 and 69 (inclusive of 40 but exclusive of 70), it returns "B"; and if the score is 70 or above, it returns "C".