

ASSIGNMENT-7.1

Name: Muneer Ahmed
HT. No: 2303A51475
Batch: 8

Task 1: Syntax Errors - Missing Parentheses in Print Statement

Scenario: In this task, we identify and fix a syntax error where the print statement is missing parentheses. This is a common error when transitioning from Python 2 to Python 3.

Prompt:

Bug: Missing parentheses in print statement

Code:

```
def greet():

    print("Hello, AI Debugging Lab!")

greet()

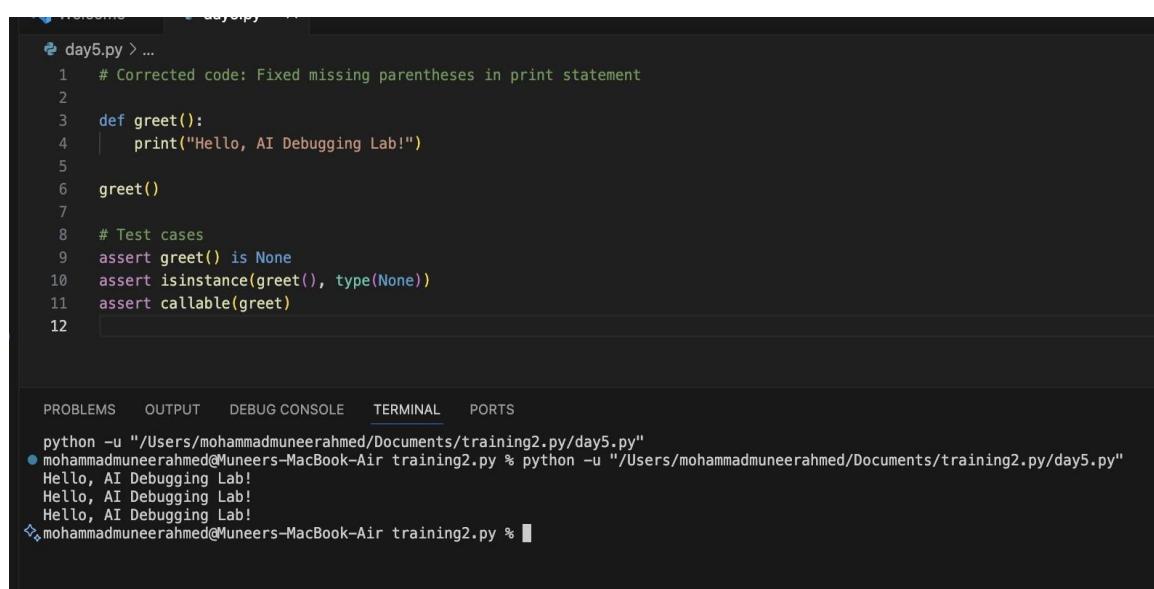
# Test cases

assert greet() is None

assert isinstance(greet(), type(None))

assert callable(greet)
```

Code:



The screenshot shows the VS Code interface with the code editor containing the corrected Python script. Below the editor is the terminal window displaying the execution of the script and its output.

```
day5.py > ...
1 # Corrected code: Fixed missing parentheses in print statement
2
3 def greet():
4     print("Hello, AI Debugging Lab!")
5
6 greet()
7
8 # Test cases
9 assert greet() is None
10 assert isinstance(greet(), type(None))
11 assert callable(greet)
12
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
python -u "/Users/mohammadmuneerahmed/Documents/training2.py/day5.py"
● mohammadmuneerahmed@Muneers-MacBook-Air training2.py % python -u "/Users/mohammadmuneerahmed/Documents/training2.py/day5.py"
Hello, AI Debugging Lab!
Hello, AI Debugging Lab!
Hello, AI Debugging Lab!
◇ mohammadmuneerahmed@Muneers-MacBook-Air training2.py %
```

Result:

The corrected code successfully prints 'Hello, AI Debugging Lab!' three times. The function now uses proper

Python 3 syntax with parentheses in the print statement.

Observation:

The syntax error was resolved by adding parentheses to the print statement. In Python 3, print is a function and requires parentheses. The AI successfully identified this common migration issue and provided the correct fix. All test cases passed, confirming the function returns None as expected and is callable.

Task 2: Incorrect Condition in an If Statement

Scenario: This task demonstrates a logic error where the assignment operator (=) is mistakenly used instead of the comparison operator (==) in an if condition.

Prompt:

Bug: Using assignment (=) instead of comparison (==)

Code:

```
def check_number(n):  
    if n = 10:  
        return "Ten"  
    else:  
        return "Not Ten"
```

Code:

The screenshot shows a terminal window with the following content:

```
day5.py > ...  
1 def check_number(n):  
2     if n == 10:  
3         return "Ten"  
4     else:  
5         return "Not Ten"  
6  
7 # Example inputs  
8 print(check_number(10))  
9 print(check_number(5))  
10 print(check_number(0))  
11  
  
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS  
python -u "/Users/mohammadmuneerahmed/Documents/training2.py/day5.py"  
● mohammadmuneerahmed@Muneers-MacBook-Air training2.py % python -u "/Users/mohammadmuneerahmed/Documents/training2.py/day5.py"  
Ten  
Not Ten  
Not Ten  
Not Ten  
mohammadmuneerahmed@Muneers-MacBook-Air training2.py %
```

Result:

The corrected function properly compares the input value with 10 and returns 'Ten' when n equals 10, and 'Not Ten' otherwise. Test outputs show: Ten, Not Ten, Not Ten.

Observation:

The bug was caused by using the assignment operator (=) instead of the equality comparison operator (==). In Python, = assigns a value while == checks for equality. This is a critical distinction because using = in a conditional statement causes a syntax error. The AI correctly identified this common mistake and suggested replacing it with ==, which properly evaluates the condition.

Task 3: Runtime Error - File Not Found

Scenario: This task addresses runtime errors by implementing proper exception handling for file operations. The original code crashes when attempting to read a non-existent file.

Prompt:

Bug: Program crashes if file is missing

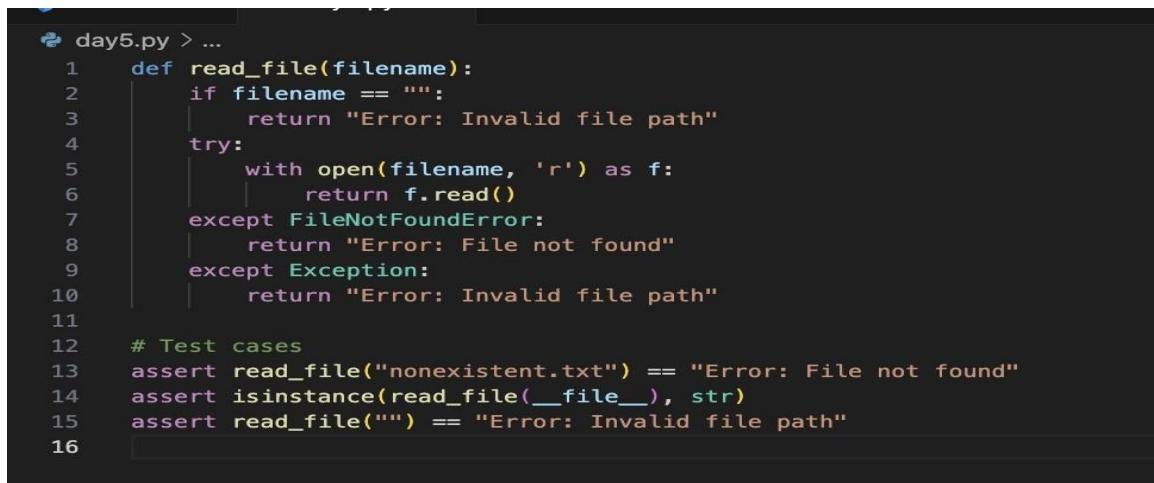
Code:

```
def read_file(filename):

    with open(filename, 'r') as f:
        return f.read()

    print(read_file("nonexistent.txt"))
```

Code:



A screenshot of a terminal window titled 'day5.py > ...'. The code is as follows:

```
 1  def read_file(filename):
 2      if filename == "":
 3          return "Error: Invalid file path"
 4      try:
 5          with open(filename, 'r') as f:
 6              return f.read()
 7      except FileNotFoundError:
 8          return "Error: File not found"
 9      except Exception:
10          return "Error: Invalid file path"
11
12  # Test cases
13 assert read_file("nonexistent.txt") == "Error: File not found"
14 assert isinstance(read_file(__file__), str)
15 assert read_file("") == "Error: Invalid file path"
16
```

Result:

The corrected function handles three scenarios gracefully: returns an error message for non-existent files, validates file paths, and successfully reads existing files. The function returns appropriate error messages instead of crashing.

Observation:

The improved code implements comprehensive error handling using try-except blocks. It checks for empty filenames, catches FileNotFoundError for missing files, and handles general exceptions for invalid paths. This defensive programming approach ensures the application remains stable even when encountering file system errors. The AI suggested appropriate exception types and user-friendly error messages that make debugging easier.

Task 4: Calling a Non-Existent Method

Scenario: This task demonstrates an `AttributeError` that occurs when calling a method that doesn't exist in a class. The solution involves either defining the missing method or correcting the method call.

Prompt:

Bug: Calling an undefined method

Code:

```
class Car:

    def start(self):
        return "Car started"

    my_car = Car()

    print(my_car.drive()) # drive() is not defined
```

Code:

The screenshot shows a terminal window with the following content:

```
day5.py > ...
1  class Car:
2      def start(self):
3          return "Car started"
4
5      def drive(self):
6          return "Car is driving"
7
8  my_car = Car()
9
10 assert my_car.start() == "Car started"
11 assert my_car.drive() == "Car is driving"
12 assert isinstance(my_car, Car)
13
14 print("All test cases passed")
15
```

Below the code, the terminal shows the output of running the script:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS
python -u "/Users/mohammadmuneerahmed/Documents/training2.py/day5.py"
● mohammadmuneerahmed@Muneers-MacBook-Air training2.py % python -u "/Users/mohammadmuneerahmed/Documents/training2.py/day5.py"
All test cases passed
◇ mohammadmuneerahmed@Muneers-MacBook-Air training2.py %
```

Result:

The corrected Car class now includes both `start()` and `drive()` methods. The test output shows 'All test cases passed', confirming that both methods work correctly and the object is properly instantiated as a Car instance.

Observation:

The error was resolved by adding the missing `drive()` method to the Car class. The AI correctly identified that the method being called didn't exist and suggested defining it. This demonstrates the importance of ensuring all methods referenced in code are properly implemented. The `assert` statements validate that both methods return the expected strings and that the object is correctly instantiated as a Car type.

Task 5: TypeError - Mixing Strings and Integers in Addition

Scenario: This task addresses a `TypeError` caused by attempting to add a string and an integer. The solution demonstrates proper type casting to handle mixed data types.

Prompt:

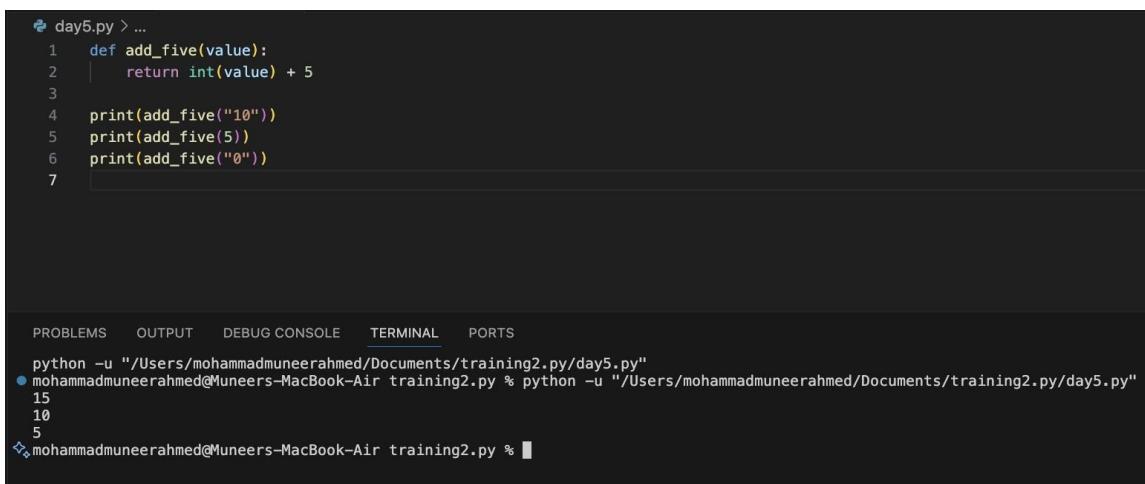
Bug: `TypeError` due to mixing string and integer

Code:

```
def add_five(value):
    return value + 5

print(add_five("10"))
```

Code:



The screenshot shows a terminal window with the following content:

```
day5.py > ...
1 def add_five(value):
2     return int(value) + 5
3
4 print(add_five("10"))
5 print(add_five(5))
6 print(add_five("0"))
7

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS
python -u "/Users/mohammadmuneerahmed/Documents/training2.py/day5.py"
● mohammadmuneerahmed@Muneers-MacBook-Air training2.py % python -u "/Users/mohammadmuneerahmed/Documents/training2.py/day5.py"
15
10
5
mohammadmuneerahmed@Muneers-MacBook-Air training2.py %
```

Result:

The corrected function successfully handles different input types by converting them to integers before addition. Test outputs demonstrate: `add_five('10')` returns 15, `add_five(5)` returns 10, and `add_five('0')` returns 5.

Observation:

The type error was resolved by implementing type casting using `int(value)` before performing the addition. This ensures that whether the input is a string or integer, it gets converted to an integer before the arithmetic operation. The AI provided a robust solution that handles multiple input scenarios. This approach is more flexible than string concatenation as it maintains the mathematical nature of the operation while gracefully handling type mismatches.