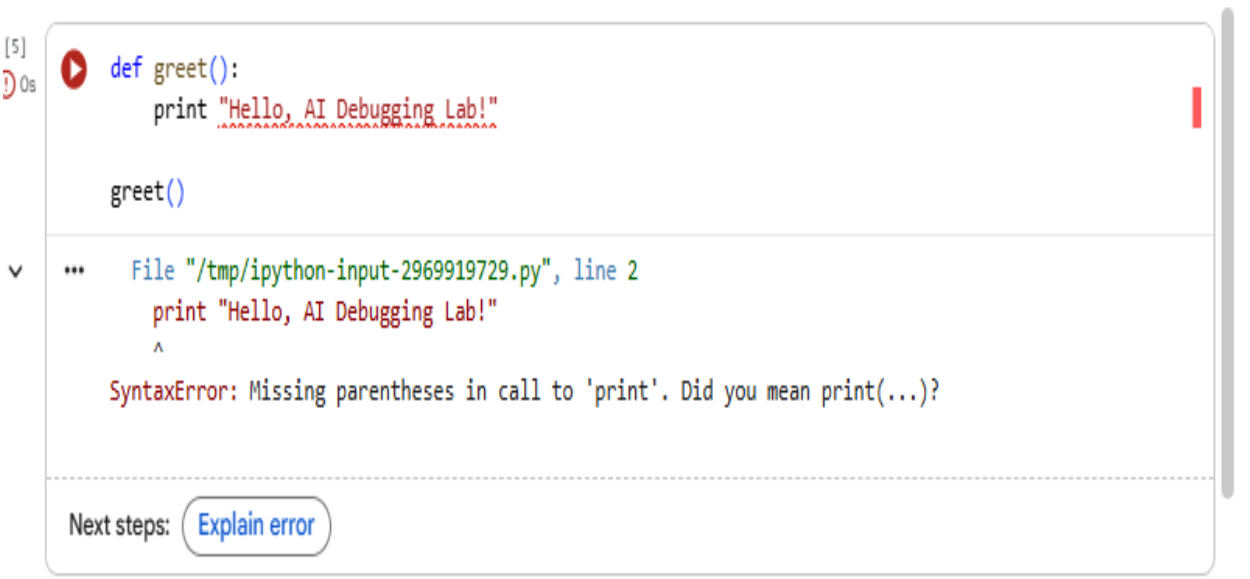
AI Assisted Coding

CHENCHU RAJU

ROLL NO:2303A52206

Assignment Number: 7.1

Task Description #1 (Syntax Errors – Missing Parentheses in Print Statement)
Task: Provide a Python snippet with a missing parenthesis in a print statement (e.g., print "Hello"). Use AI to detect and fix the syntax error.
# Bug: Missing parentheses in print statement
def greet():
print "Hello, AI Debugging Lab!"
greet()
Requirements:
• Run the given code to observe the error.
• Apply AI suggestions to correct the syntax.
• Use at least 3 assert test cases to confirm the corrected code works.
Expected Output #1:
• Corrected code with proper syntax and AI explanation



```
def greet():
-    print "Hello, AI Debugging Lab!"
+    print("Hello, AI Debugging Lab!")

greet()
```

```
  File "/tmp/ipython-input-2969919729.py", line 2
    print "Hello, AI Debugging Lab!"
    ^
SyntaxError: Missing parentheses in call to 'print'. Did you mean print(...)?
```

Next steps: Explain error

The above code will raise a SyntaxError because in Python 3, print is a function and requires parentheses around its arguments. AI debugging tools can easily detect and suggest this common fix. Here is the corrected code:

```
def greet():
    print("Hello, AI Debugging Lab!")

greet()
```

```
Hello, AI Debugging Lab!
```

Task Description #2 (Incorrect condition in an If Statement)
Task: Supply a function where an if-condition mistakenly uses = instead of ==. Let AI identify and fix the issue.
# Bug: Using assignment (=) instead of comparison (==)
def check_number(n):
if n = 10:
return "Ten"
else:
return "Not Ten"
Requirements:
• Ask AI to explain why this causes a bug.
• Correct the code and verify with 3 assert test cases.
Expected Output #2:
• Corrected code using == with explanation and successful test execution.

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

[8]
0s

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

# Attempting to call the function will raise a SyntaxError
# print(check_number(10))
```

```
  File "/tmp/ipython-input-2969447443.py", line 2
    if n = 10:
       ^
SyntaxError: invalid syntax. Maybe you meant '==' or ':=' instead of '='?
```

Next steps: Explain error

[9]
0s

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

# Test cases
assert check_number_corrected(10) == "Ten", "Test Case 1 Failed: Should return 'Ten' for 10."
assert check_number_corrected(5) == "Not Ten", "Test Case 2 Failed: Should return 'Not Ten' for 5."
assert check_number_corrected(0) == "Not Ten", "Test Case 3 Failed: Should return 'Not Ten' for 0."
assert check_number_corrected(-10) == "Not Ten", "Test Case 4 Failed: Should return 'Not Ten' for -10

print("All test cases passed for the corrected function!")
```

```
All test cases passed for the corrected function!
```

Task Description #3 (Runtime Error – File Not Found)
Task: Provide code that attempts to open a non-existent file and crashes. Use AI to apply safe error handling.
# Bug: Program crashes if file is missing
def read_file(filename):
with open(filename, 'r') as f:
return f.read()
print(read_file("nonexistent.txt"))
Requirements:
• Implement a try-except block suggested by AI.
• Add a user-friendly error message.
• Test with at least 3 scenarios: file exists, file missing, invalid path.
Expected Output #3:
• Safe file handling with exception management.

Bug: Program crashes if file is missing

```
[10]
0s
def read_file(filename):
    with open(filename, 'r') as f:
        return f.read()

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

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
/tmp/ipython-input-573643107.py in <cell line: 0>()
      3         return f.read()
      4
----> 5 print(read_file("nonexistent.txt"))

/tmp/ipython-input-573643107.py in read_file(filename)
      1 def read_file(filename):
----> 2     with open(filename, 'r') as f:
      3         return f.read()
      4
      5 print(read_file("nonexistent.txt"))

FileNotFoundError: [Errno 2] No such file or directory: 'nonexistent.txt'
```

Next steps: Explain error

The code above will raise a FileNotFoundError because the file nonexistent.txt does not exist. When a program tries to open a file that cannot be found, Python's open() function raises this specific type of OSError.

To handle this gracefully and prevent the program from crashing, we can use a try-except block. This allows us to 'try' an operation and 'catch' specific errors that might occur, providing a user-friendly message or an alternative action instead of terminating the program. Here's the corrected code with error handling for several scenarios:

```
[11]
✓ 0s
```

```python
import os

def read_file_safe(filename):
    try:
        with open(filename, 'r') as f:
            content = f.read()
        return f"Successfully read file '{filename}':\n{content}"
    except FileNotFoundError:
        return f"Error: The file '{filename}' was not found. Please check the file path and name."
    except IsADirectoryError:
        return f"Error: '{filename}' is a directory, not a file. Please provide a file path."
    except Exception as e:
        return f"An unexpected error occurred while reading '{filename}': {e}"

# --- Test Scenarios ---

# Scenario 1: File exists (create a dummy file first)
dummy_file_name = "existent.txt"
with open(dummy_file_name, 'w') as f:
    f.write("This is a test file.\nLine 2.")
print(read_file_safe(dummy_file_name))
os.remove(dummy_file_name) # Clean up the dummy file
print("\n")

# Scenario 2: File missing
print(read_file_safe("nonexistent.txt"))
print("\n")

# Scenario 3: Invalid path (e.g., trying to open a directory as a file)
# On most systems, opening the current directory will cause IsADirectoryError
```

```
[11]
✓ 0s
```

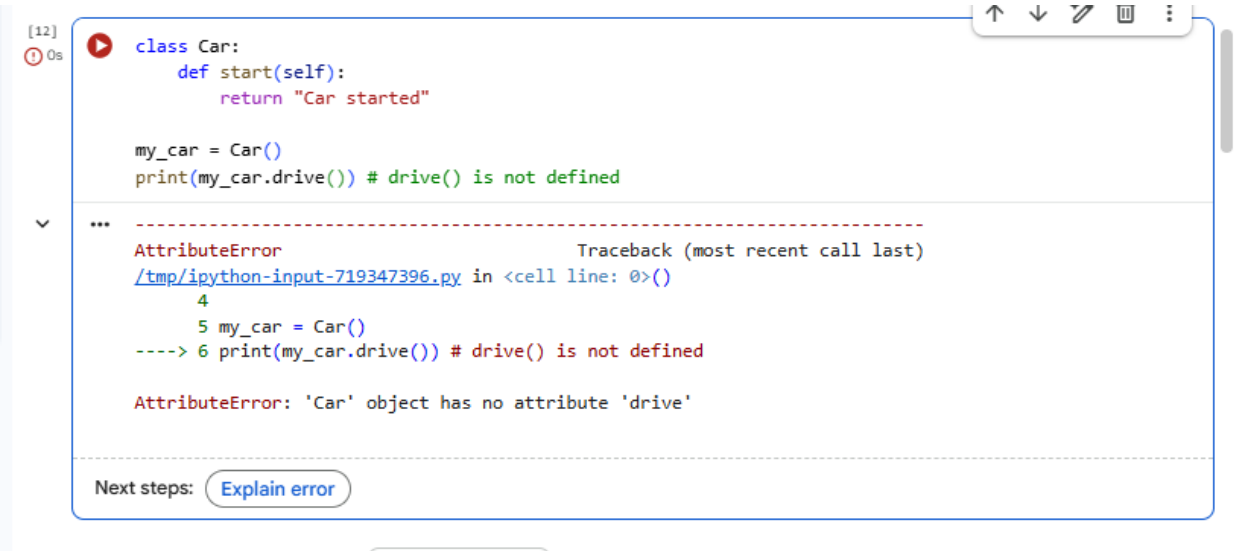
```python
print("\n")

# Scenario 3: Invalid path (e.g., trying to open a directory as a file)
# On most systems, opening the current directory will cause IsADirectoryError
invalid_path = "."
print(read_file_safe(invalid_path))
print("\n")

# Scenario 4: Another unexpected error (e.g., permission denied - difficult to simulate reliably acro
# For demonstration, we'll just show the generic exception handling with a non-file path
# This might also trigger a FileNotFoundError or PermissionError depending on the OS and permissions
print(read_file_safe("/root/no_permission_file.txt")) # Usually permission denied
```

```
Successfully read file 'existent.txt':
This is a test file.
Line 2.


Error: The file 'nonexistent.txt' was not found. Please check the file path and name.


Error: '.' is a directory, not a file. Please provide a file path.


Error: The file '/root/no_permission_file.txt' was not found. Please check the file path and name.
```

Task Description #4 (Calling a Non-Existent Method)
Task: Give a class where a non-existent method is called (e.g., obj.undefined_method()). Use AI to debug and fix.
Bug: Calling an undefined method

class Car:
def start(self):
return "Car started"
my_car = Car()
print(my_car.drive()) # drive() is not defined
Requirements:
• Students must analyze whether to define the missing method or correct the method call.
• Use 3 assert tests to confirm the corrected class works.
Expected Output #4:
• Corrected class with clear AI explanation.

Bug: Calling an undefined method

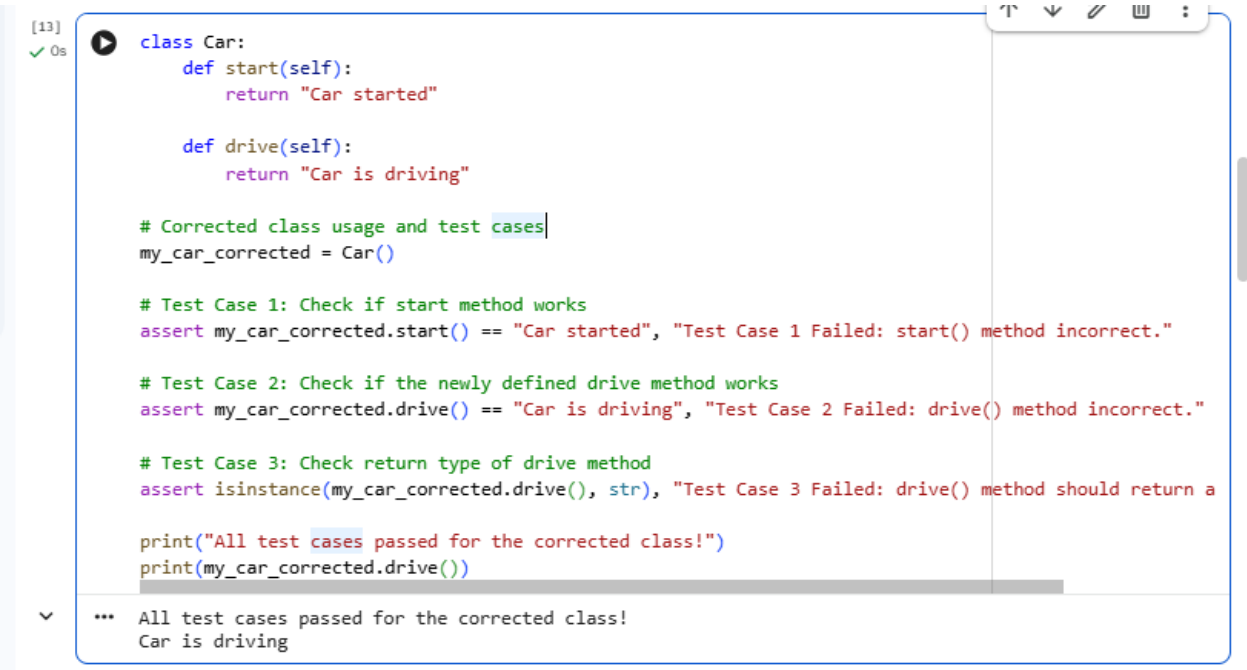


The code above will raise an AttributeError. This error occurs when you try to access an attribute (like a method or a variable) on an object that does not possess that attribute. In this case, the Car class has a start method but no drive method, yet the code attempts to call my_car.drive().

To fix this, we have two main options:

1. Define the missing method: If the intention was for Car objects to have a drive method, we need to add it to the class definition.
2. Correct the method call: If drive() was a typo and the intention was to call an existing method (like start()), then we should change the method call accordingly.

Given the context, I will proceed by defining the missing drive method within the Car class to fulfill the intended call. Here is the corrected class and its verification with test cases:



Task Description #5 (TypeError – Mixing Strings and Integers in Addition)

Task: Provide code that adds an integer and string ("5" + 2) causing a TypeError. Use AI to resolve the bug.

Bug: TypeError due to mixing string and integer

def add_five(value):

return value + 5

print(add_five("10"))

Requirements:

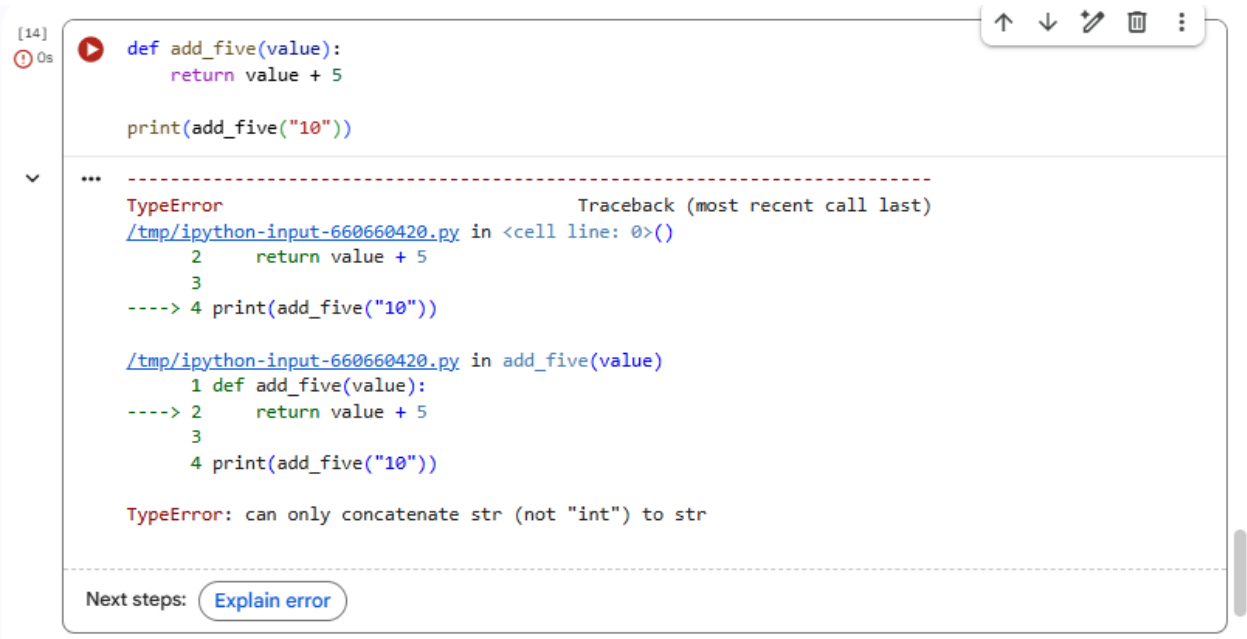
• Ask AI for two solutions: type casting and string concatenation.

• Validate with 3 assert test cases.

Expected Output #5:

• Corrected code that runs successfully for multiple inputs.

Bug: TypeError due to mixing string and integer



```
[14]
0s
def add_five(value):
    return value + 5

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

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/tmp/ipython-input-660660420.py in <cell line: 0>()
      2     return value + 5
      3
----> 4 print(add_five("10"))

/tmp/ipython-input-660660420.py in add_five(value)
      1 def add_five(value):
----> 2     return value + 5
      3
      4 print(add_five("10"))

TypeError: can only concatenate str (not "int") to str
```

Next steps: Explain error

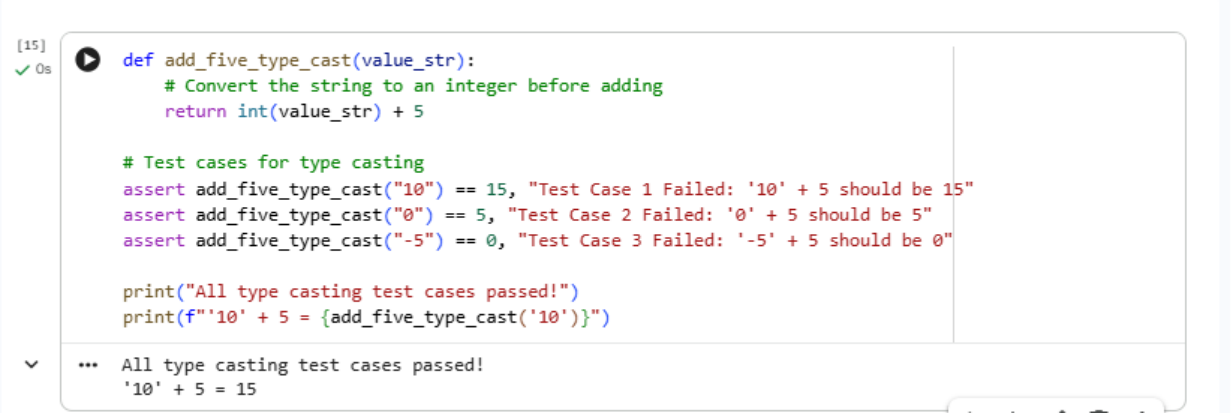
The code above will raise a TypeError. This error occurs because Python cannot directly perform arithmetic addition between a string and an integer. When you use the + operator, Python expects both operands to be of compatible types for the operation: either both numbers for arithmetic addition, or both strings for string concatenation.

To fix this, we have two primary approaches:

1. Type Casting (convert string to integer): If the intention is to perform arithmetic addition, the string must first be converted into an integer using int(). This is suitable when the string represents a valid number.
2. String Concatenation (convert integer to string): If the intention is to combine the string and the number as textual data, the integer must first be converted into a string using str(). This is suitable when you want to join them as text.

I will demonstrate both solutions below with verification tests.

Solution 1: Type Casting (Converting string to integer)



```
[15]
0s
def add_five_type_cast(value_str):
    # Convert the string to an integer before adding
    return int(value_str) + 5

# Test cases for type casting
assert add_five_type_cast("10") == 15, "Test Case 1 Failed: '10' + 5 should be 15"
assert add_five_type_cast("0") == 5, "Test Case 2 Failed: '0' + 5 should be 5"
assert add_five_type_cast("-5") == 0, "Test Case 3 Failed: '-5' + 5 should be 0"

print("All type casting test cases passed!")
print(f"'10' + 5 = {add_five_type_cast('10')}")
```

```
All type casting test cases passed!
'10' + 5 = 15
```

Solution 2: String Concatenation (Converting integer to string)

[16] ✓ 0s

```
def add_five_str_concat(value_str):
    # Convert the integer to a string before concatenating
    return value_str + str(5)

# Test cases for string concatenation
assert add_five_str_concat("10") == "105", "Test Case 1 Failed: '10' + '5' should be '105'"
assert add_five_str_concat("abc") == "abc5", "Test Case 2 Failed: 'abc' + '5' should be 'abc5'"
assert add_five_str_concat("Hello") == "Hello5", "Test Case 3 Failed: 'Hello' + '5' should be 'Hello5

print("All string concatenation test cases passed!")
print(f"'10' + '5' = {add_five_str_concat('10')}")
```

```
All string concatenation test cases passed!
'10' + '5' = 105
```