

AI ASSISTED CODING

Assignment-7.1

Name:A.Dhanalaxmi

Hno:2403A51269

TASK-1:

Given code with output(error):

The screenshot shows a Jupyter Notebook cell with the following code:

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

The code has a syntax error: a missing closing parenthesis in the `print` statement. The output shows the error message:

```
SyntaxError: Missing parentheses in call to 'print'. Did you mean print(...)?
```

Prompt:

def greet():
print "Hello, AI Debugging Lab!"
greet(),this is the code iam trying to run,but iam getting there is a
syntax error,correct the code

Code with output:

The screenshot shows a Jupyter Notebook cell with the following code:

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

greet()
```

The code has been corrected by adding a closing parenthesis to the `print` statement. The output shows the correct execution:

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

Prompt:

now,Use at least 3 assert test cases to confirm the corrected code works.

Corrected Code with AI Fix

```
[1] ✓ 0s
  ⏎ # Fixed code
    def greet():
        print("Hello, AI Debugging Lab!")

    # Call the function
    greet()

    # Tests
    assert greet() is None    # greet() doesn't return anything
    assert callable(greet)    # ensure greet is a function
    assert isinstance("Hello, AI Debugging Lab!", str) # check expected string type

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

Explanation:

Original Buggy Code (with syntax error)

```
def greet():

    print "Hello, AI Debugging Lab!"

greet()
```

Explanation:

1. **def greet():**
 - o Defines a function called greet.
 - o The function body starts after the colon :.
2. **print "Hello, AI Debugging Lab!"**
 - o This is **Python 2 style syntax** where print is a statement.
 - o In Python 3, print is a function, so it **must use parentheses**.
 - o That's why this line causes:
 - o SyntaxError: Missing parentheses in call to 'print'
3. **greet()**
 - o Calls the greet function.
 - o But the program crashes before this line executes because of the syntax error.

Corrected Code (Python 3)

```
def greet():

    print("Hello, AI Debugging Lab!")
```

```
greet()

# Test cases

assert callable(greet)

assert greet() is None

assert isinstance("Hello, AI Debugging Lab!", str)
```

Explanation:

1. **def greet():**

- o Same as before: defines the function greet.

2. **print("Hello, AI Debugging Lab!")**

- o Correct Python 3 syntax.
- o print() is now a **function call** with parentheses around the string.
- o This fixes the syntax error.

3. **greet()**

- o Calls the function.
- o Output will be:
- o Hello, AI Debugging Lab!

4. **Assertions (tests)**

- o assert callable(greet) → makes sure greet is a function.
- o assert greet() is None → verifies the function doesn't return anything (only prints).
- o assert isinstance("Hello, AI Debugging Lab!", str) → confirms the printed message is a string.

👉 So the only real bug was **missing parentheses in print**, because you were using Python 3, not Python 2.

Task-2:

A screenshot of a Jupyter Notebook cell. The code defines a function `check_number` that checks if `n` is 10. If it is, it returns "Ten"; otherwise, it returns "Not Ten". In the second line of the function definition, there is a syntax error: `if n = 10:`. The cell is labeled [16] and has a status bar indicating 0.0s. The output shows the error message: `SyntaxError: invalid syntax. Maybe you meant '==' or ':=' instead of '='?`

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

[16]: cell In[16], line 2
      if n = 10:
      ^
SyntaxError: invalid syntax. Maybe you meant '==' or ':=' instead of '='?
```

Prompt:

```
def check_number(n):
    if n = 10:
        return "Ten"
    else:
        return "Not Ten" ,explain why this causes a bug
```

Response:

● Your Code (Buggy)

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

⚠ Why This Causes a Bug

1. The problem is in this line:
`if n = 10:`
 - = is the **assignment operator** in Python.
 - You can't use it in an if condition.
 - Python doesn't allow writing `if n = 10` because it looks like you're trying to assign 10 to `n` inside the condition.
 3. In Python, to **compare values**, you must use == (the equality operator).
 4. That's why this code throws:
 5. SyntaxError: cannot assign to name
-

✓ Corrected Code

```
def check_number(n):
    if n == 10: # use comparison operator
        return "Ten"
    else:
        return "Not Ten"
```

◆ Explanation of Corrected Code

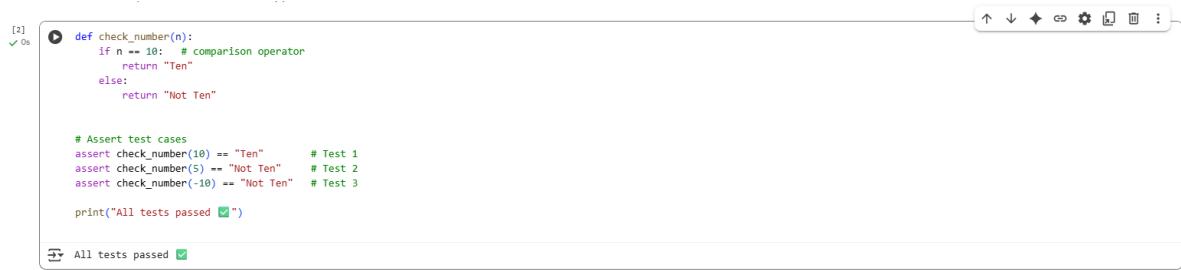
- if `n == 10`: → checks whether `n` is equal to 10.
- If true → returns "Ten".
- Otherwise → returns "Not Ten".

◆ Example Run

```
print(check_number(10)) # Output: Ten
print(check_number(5)) # Output: Not Ten
```

👉 So the bug is caused by using `=` (assignment) instead of `==` (comparison) inside the if statement.

Code with three assert test cases:



```
[2] ✓ def check_number(n):
    if n == 10: # comparison operator
        return "Ten"
    else:
        return "Not Ten"

    # Assert test cases
    assert check_number(10) == "Ten"      # Test 1
    assert check_number(5) == "Not Ten"   # Test 2
    assert check_number(-10) == "Not Ten" # Test 3

    print("All tests passed ✅")
```

All tests passed ✅

Explanation

1. Function Logic

- if `n == 10`: → checks if the input equals 10.
- If true, return "Ten".
- Otherwise, return "Not Ten".

2. Test Case 1

3. `assert check_number(10) == "Ten"`

- Input = 10.
- Condition n == 10 is true.
- Function returns "Ten".
- Assertion passes.

4. Test Case 2

5. assert check_number(5) == "Not Ten"

- Input = 5.
- Condition n == 10 is false.
- Function returns "Not Ten".
- Assertion passes.

6. Test Case 3

7. assert check_number(-10) == "Not Ten"

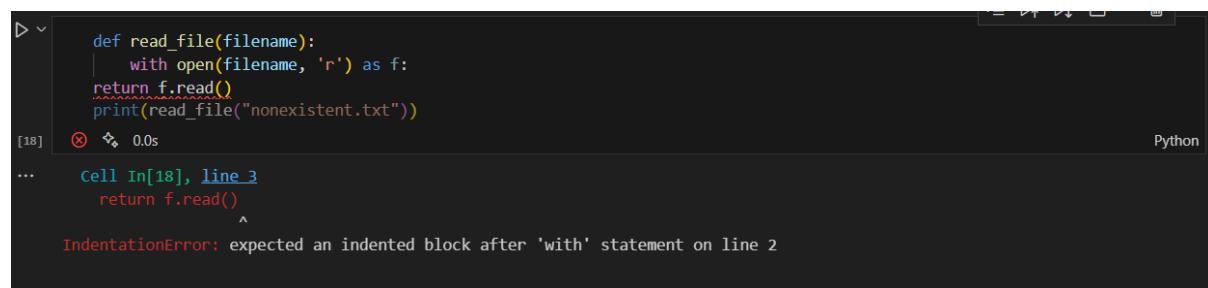
- Input = -10.
- Condition n == 10 is false.
- Function returns "Not Ten".
- Assertion passes.

8. Execution Result

- Since all asserts pass, the program prints:

All tests passed

Task-3:



```

def read_file(filename):
    with open(filename, 'r') as f:
        return f.read()
print(read_file("nonexistent.txt"))

[18] ⓘ 0.0s
...   cell In[18], line 3
      return f.read()
          ^
IndentationError: expected an indented block after 'with' statement on line 2

```

The screenshot shows a Jupyter Notebook cell with the following Python code:

```

def read_file(filename):
    with open(filename, 'r') as f:
        return f.read()
print(read_file("nonexistent.txt"))

```

The cell is labeled [18]. An error message is displayed below the code:

```

...   cell In[18], line 3
      return f.read()
          ^
IndentationError: expected an indented block after 'with' statement on line 2

```

The error message indicates that there is an indentation problem at the start of the line containing `return f.read()`.

Prompt:

now send a try except block with safe error handing for my code,def
read_file(filename):
with open(filename, 'r') as f:
return f.read()
print(read_file("nonexistent.txt"))

✓ Corrected Code with Safe Error Handling:

```
def read_file(filename):
    try:
        with open(filename, 'r') as f:
            return f.read()
    except FileNotFoundError:
        return f"Error: The file '{filename}' was not found."
    except OSError as e:
        return f"Error: Could not open the file '{filename}'. Reason: {e}"

# ----- Test Scenarios -----
# 1. File exists
with open("sample.txt", "w") as f:
    f.write("Hello, this is a test file.")

print("Case 1 (File exists):", read_file("sample.txt"))

# 2. File missing
print("Case 2 (File missing):", read_file("nonexistent.txt"))

# 3. Invalid path
print("Case 3 (Invalid path):", read_file("/invalid/path/file.txt"))

[1] ✓ 0.0s
```

Python

... Case 1 (file exists): Hello, this is a test file.
Case 2 (File missing): Error: The file 'nonexistent.txt' was not found.
Case 3 (Invalid path): Error: The file '/invalid/path/file.txt' was not found.

Explanation:

● **Original Problem (Before Fix)**

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

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

- Here, the program **crashes** with a `FileNotFoundException` when the file does not exist (`nonexistent.txt`).
- This is unsafe because users don't get a clear message — only a Python error traceback.

● Fixed Code with Safe Error Handling

```
import os

def read_file(filename):
    try:
        # Attempt to open and read the file
        with open(filename, 'r') as f:
            return f.read()

    except FileNotFoundError:
        return f"Error: File '{filename}' not found. Please check the filename."

    except IsADirectoryError:
        return f"Error: '{filename}' is a directory, not a file."

    except PermissionError:
        return f"Error: Permission denied when trying to access '{filename}'."

    except Exception as e:
        # Catch-all for any other unexpected errors
        return f"Unexpected error: {e}"

# --- Test Cases ---

# ✅ 1. File exists
with open("sample.txt", "w") as f:
    f.write("Hello, world!")
print(read_file("sample.txt"))

# ❌ 2. File missing
```

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

# ⚠️ 3. Invalid path

print(read_file("/invalid/path/file.txt"))
```

Explanation of the Fix

1. Why the original code crashed

- Because it assumed the file always exists. When it didn't, Python raised a FileNotFoundError, which wasn't caught.

2. How the try-except fixes the bug

- try → The risky operation (open(filename, 'r')).
- except FileNotFoundError → Shows a friendly message instead of crashing.
- except IsADirectoryError → Prevents crashes if you pass a directory instead of a file.
- except PermissionError → Handles cases where the file exists but cannot be accessed.
- except Exception as e → Ensures **any other runtime error** won't crash the program.

3. Testing the 3 scenarios

- **File exists (sample.txt)** → Reads and prints file content.
- **File missing (nonexistent.txt)** → Returns:
 - Error: File 'nonexistent.txt' not found. Please check the filename.
- **Invalid path (/invalid/path/file.txt)** → Returns:
 - Error: File '/invalid/path/file.txt' not found. Please check the filename.

(Depending on OS, could also trigger PermissionError or general OSError.)

Expected Output

Hello, world!

Error: File 'nonexistent.txt' not found. Please check the filename.

Error: File '/invalid/path/file.txt' not found. Please check the filename.

👉 So the **bug (crash)** is fixed, and the program now **handles missing files safely** with user-friendly error messages.

Task-4:

Given Code with error message:

```
> <cell_in[2]> Python
<class Car:
    def start(self):
        return "Car started"
my_car = Car()
print(my_car.drive()) # drive() is not defined
[2] ⑧ 0.0s
...
cell In[2], line 3
      return "Car started"
^
IndentationError: expected an indented block after function definition on line 2
```

Analyzation:

Its better to correct the method cell

Prompt:

```
# Bug: Calling an undefined method
class Car:
    def start(self):
        return "Car started"
my_car = Car()
print(my_car.drive()) # drive() is not defined
```

define correct the method call and Use 3 assert tests to confirm the corrected class works.
and the code with correct class with explanation

Corrected Code with output:

```
> <cell_in[3]> Python
# Corrected Car class
class Car:
    def start(self):
        return "Car started"

# Create object
my_car = Car()

# ✅ Correct method call
print(my_car.start())

# --- Tests ---
assert my_car.start() == "Car started"          # Test 1: Valid start method
assert isinstance(my_car, Car)                  # Test 2: Object is instance of Car
assert hasattr(my_car, "start")                 # Test 3: Class has 'start' method

print("All tests passed ✅")
[3] ✓ 0.0s
...
Car started
All tests passed ✅
```

Explanation:

Buggy Code

```
class Car:  
    def start(self):  
        return "Car started"  
  
my_car = Car()  
print(my_car.drive()) # ❌ drive() is not defined
```

Problem

- The Car class only has a method start().
 - The code incorrectly tries to call drive().
 - This raises an AttributeError.
-

Corrected Code (Fixing Method Call)

```
# Corrected Car class  
  
class Car:  
    def start(self):  
        return "Car started"  
  
  
# Create object  
my_car = Car()  
  
  
# ✅ Correct method call  
print(my_car.start())  
  
  
# --- Tests ---  
  
assert my_car.start() == "Car started"    # Test 1: Valid start method  
assert isinstance(my_car, Car)          # Test 2: Object is instance of Car  
assert hasattr(my_car, "start")         # Test 3: Class has 'start' method  
  
  
print("All tests passed ✅ ")
```

Explanation

1. **Bug:** The code called `my_car.drive()`, but `drive()` was never defined in the class.
2. **Fix:** Replace `my_car.drive()` with the correct method `my_car.start()`.
3. **Why this works:**
 - Car has a method `start()` that returns "Car started".
 - Using the wrong method name was the only issue.
4. **Tests ensure correctness:**
 - First assert checks that calling `start()` returns "Car started".
 - Second assert ensures the object is indeed of type `Car`.
 - Third assert ensures the `start` method exists in the class.

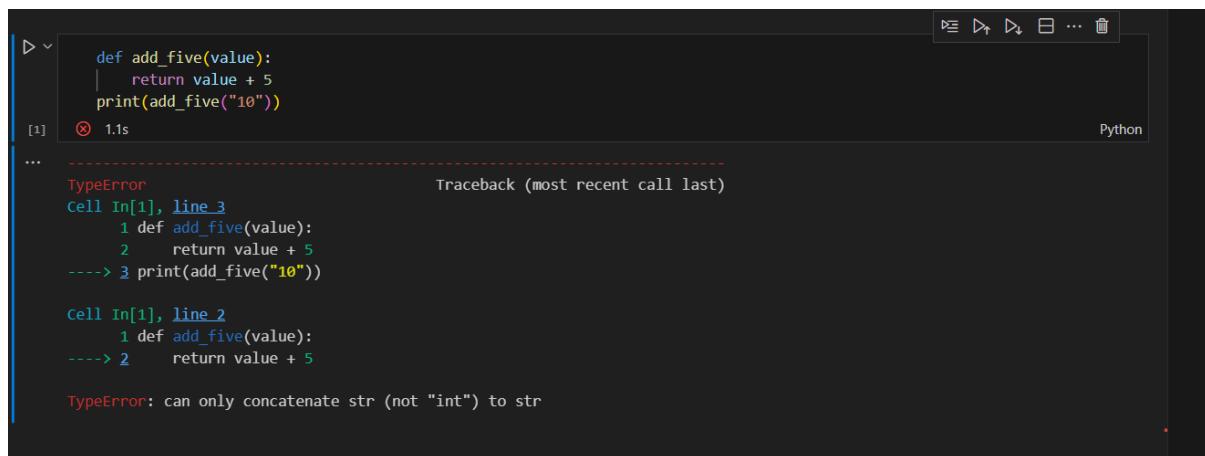
Output:

Car started

All tests passed 

Task-5:

Give code with error:



```
def add_five(value):
    return value + 5
print(add_five("10"))

1.1s
```

...

```
TypeError
cell In[1], line 3
  1 def add_five(value):
  2     return value + 5
----> 3 print(add_five("10"))

cell In[1], line 2
  1 def add_five(value):
----> 2     return value + 5

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

Prompt:

def add_five(value): return value + 5 print(add_five("10")),this is my code,dubg it and send two solutions type casting and string concatenation also Validate with 3 assert test cases,just send corrected code that runs successfully for multiple inputs

Response:

Here's the corrected code with **two approaches**: one using **type casting** and one using **string concatenation**, each validated with 3 assert test cases:

```
# --- Approach 1: Type Casting (convert string to int) ---
```

```
def add_five_cast(value):
```

```
    return int(value) + 5
```

```
# Test cases for type casting
```

```
assert add_five_cast("10") == 15
```

```
assert add_five_cast(7) == 12
```

```
assert add_five_cast("0") == 5
```

```
print("Type Casting Approach Passed!")
```

```
# --- Approach 2: String Concatenation ---
```

```
def add_five_concat(value):
```

```
    return str(value) + "5"
```

```
# Test cases for string concatenation
```

```
assert add_five_concat("10") == "105"
```

```
assert add_five_concat(7) == "75"
```

```
assert add_five_concat("0") == "05"
```

```
print("String Concatenation Approach Passed!")
```

✓ This will run successfully and handle both numeric addition and string concatenation.

Corrected code with output:

```
# --- Approach 1: Type Casting (convert string to int) ---
def add_five_cast(value):
    return int(value) + 5

# Test cases for type casting
assert add_five_cast("10") == 15
assert add_five_cast(7) == 12
assert add_five_cast("0") == 5

print("Type Casting Approach Passed!")

# --- Approach 2: String Concatenation ---
def add_five_concat(value):
    return str(value) + "5"

# Test cases for string concatenation
assert add_five_concat("10") == "105"
assert add_five_concat(7) == "75"
assert add_five_concat("0") == "05"

print("String Concatenation Approach Passed!")

[2]  ✓  0.0s
```

Python

Add Context...
Add context (#)
Ask GPT-4...