

# LAB-7.1

Name: Neela.Sai Shivathmika

Enroll num: 2403A54112

## Task1

prompt:

```
def is_prime(n):
```

```
    if n <= 1:
```

```
        return False
```

```
    for i in range(2, n):
```

```
        if n % i == 0:
```

```
            return False
```

```
    return True
```

```
print is_prime(7)
```

```
print is_prime(10)
```

```
print is_prime(1)
```

Find the syntax error in above code.

Code with error:

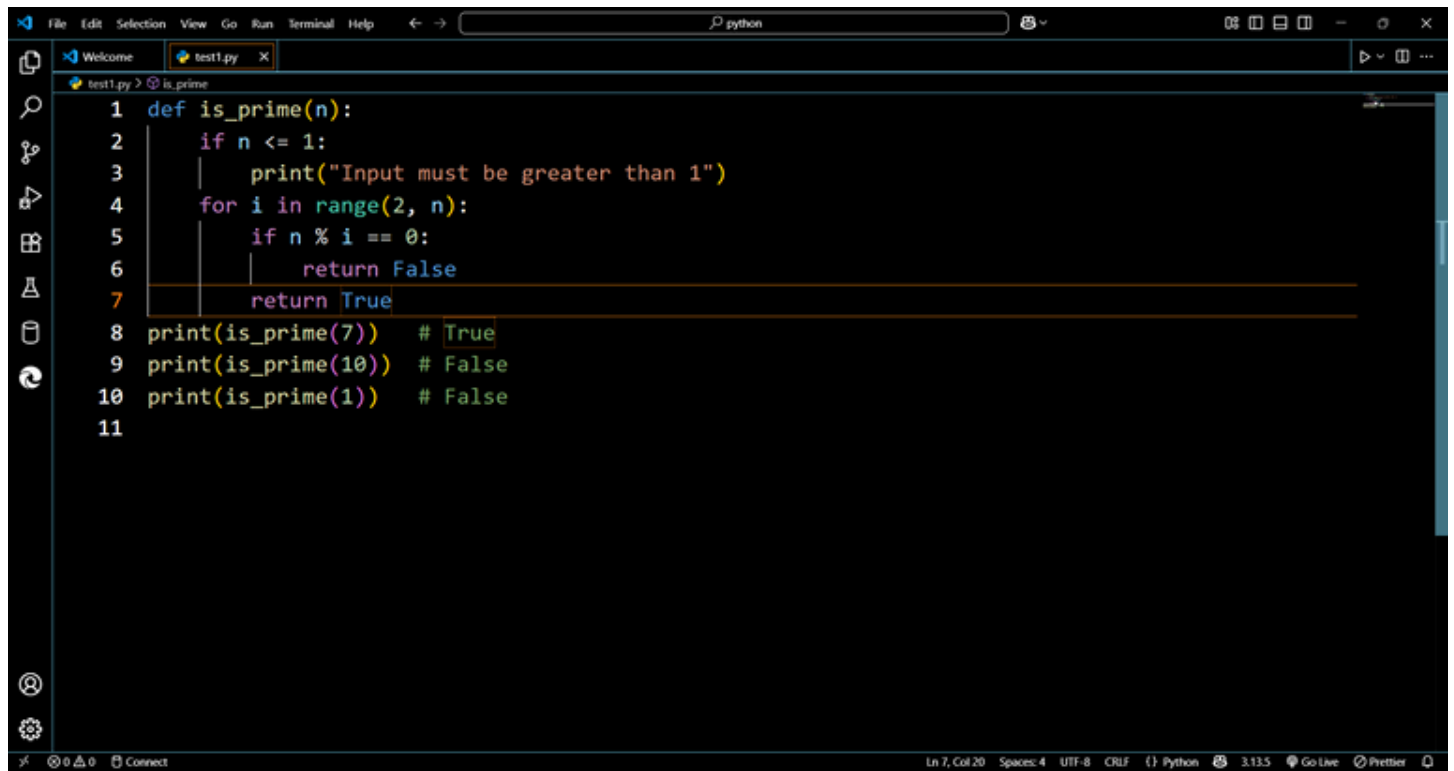
```
1 def is_prime(n):
2     if n <= 1:
3         return False
4     for i in range(2, n):
5         if n % i == 0:
6             return False
7     return True
8
9 print is_prime(7)
10 print is_prime(10)
11 print is_prime(1)
12
```

Error:

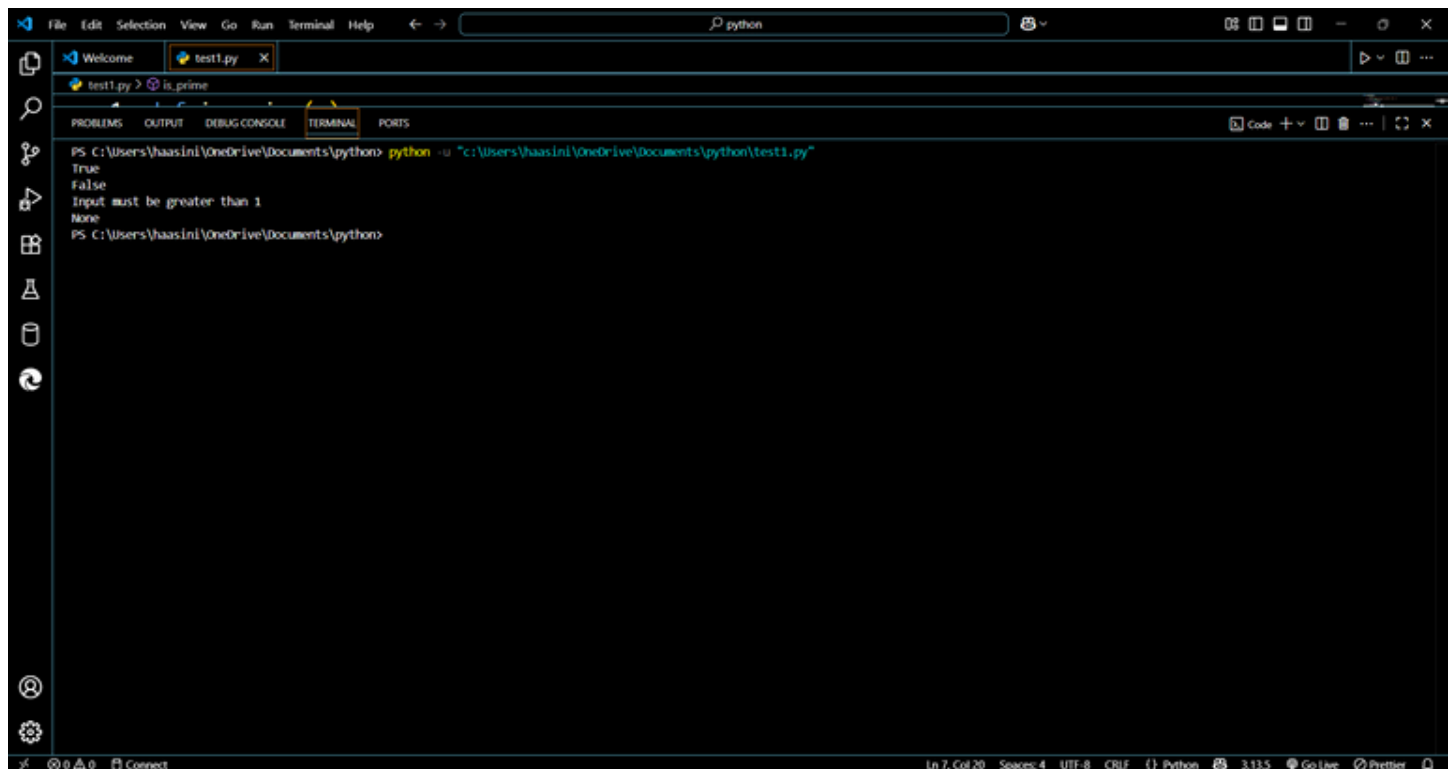
```
PS C:\Users\haasini\OneDrive\Documents\python> python -u "c:\Users\haasini\OneDrive\Documents\python\test1.py"
File "c:\Users\haasini\OneDrive\Documents\python\test1.py", line 9
    print is_prime(7)
    ~~~~~
SyntaxError: Missing parentheses in call to 'print'. Did you mean print(...)?
PS C:\Users\haasini\OneDrive\Documents\python>
```

Output:

Fixed code:



```
1 def is_prime(n):
2     if n <= 1:
3         print("Input must be greater than 1")
4     for i in range(2, n):
5         if n % i == 0:
6             return False
7     return True
8 print(is_prime(7))    # True
9 print(is_prime(10))  # False
10 print(is_prime(1))   # False
11
```



```
PS C:\Users\haasini\OneDrive\Documents\python> python -u "c:\Users\haasini\OneDrive\Documents\python\test1.py"
True
False
Input must be greater than 1
None
PS C:\Users\haasini\OneDrive\Documents\python>
```

## TASK 2:

Prompt:

```
def checknum(n):
```

if n%2=0:

print("even")

else:

print("odd")

n=int(input("Enter a number: "))

checknum(n)

identity and explain why this causes bug.

A screenshot of a code editor window titled 'test1.py'. The code is as follows:

```
1 def checknum(n):  
2     if n%2=0:  
3         print("even")  
4     else:  
5         print("odd")  
6 n=int(input("Enter a number: "))  
7 checknum(n)
```

The code is syntactically incorrect due to the use of '=' instead of '==' in the if statement on line 2. The editor's status bar at the bottom indicates 'Ln 7, Col 12', 'Spaces: 4', 'UTF-8', 'CRLF', and 'Python 3.13.5'.

Error:

The screenshot shows the Visual Studio Code editor with a file named `test1.py` open. The code defines a function `checksum(n)` with the following logic:

```
1 def checksum(n):
2     if n%2=0:
3         print("even")
4     else:
```

The error message in the terminal indicates a `SyntaxError` on line 2: `File "c:\Users\haasini\OneDrive\Documents\python\test1.py", line 2` `if n%2=0:` `^` `SyntaxError: cannot assign to expression here. Maybe you meant '=' instead of '=='?`

Output:

The bug is caused due to using `=` instead of `==`.

The screenshot shows the Visual Studio Code editor with the same file `test1.py`, but the code has been corrected. A tooltip from the AI assistant is visible, explaining the bug:

**fix** identify and explain why this causes bug

The bug is caused by using `=` instead of `==` in the if statement, which results in a syntax error because `=` is for assignment, not comparison.

Ask or edit in context

Accept Close

The corrected code is as follows:

```
1 def checknum(n):
2     if n%2==0:
3         print("even")
4     else:
5         print("odd")
6 n=int(input("Enter a number: "))
7 checknum(n)
```



```
1 def checknum(n):
2     if n%2==0:
3         print("even")
4     else:
5         print("odd")
6 n=int(input("Enter a number: "))
7 checknum(n)
```

Test cases:

N=5

N=10

N=16

This screenshot shows the Visual Studio Code interface with a terminal window open. The terminal displays the command to run a Python script and its output for the input number 5.

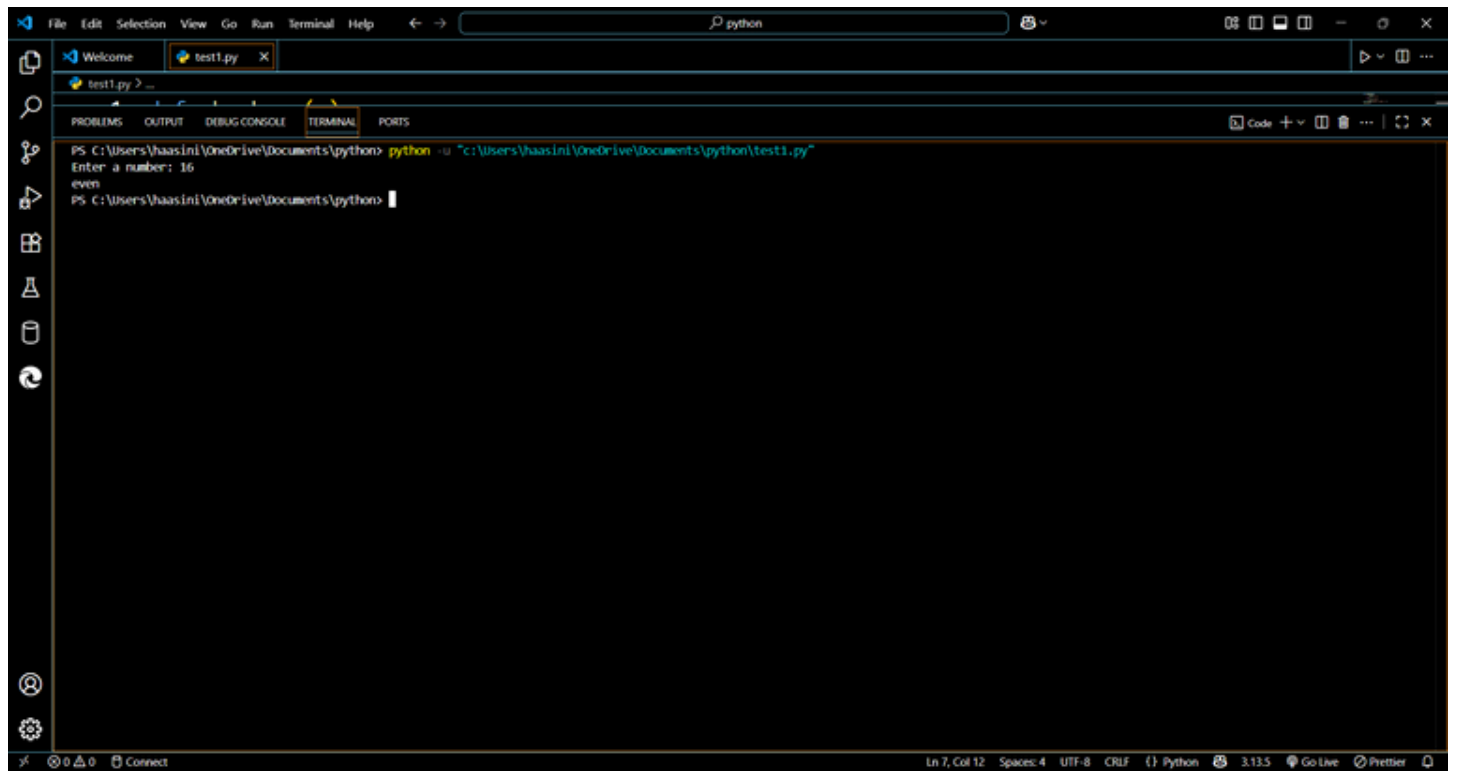
```
PS C:\Users\haasini\OneDrive\Documents\python> python -u "c:\Users\haasini\OneDrive\Documents\python\test1.py"
Enter a number: 5
odd
PS C:\Users\haasini\OneDrive\Documents\python> 
```

The status bar at the bottom indicates the current file is `test1.py` at line 7, column 12, with 4 spaces, using UTF-8 encoding and CRLF line endings. The Python version is 3.13.5, and the Prettier extension is active.

This screenshot shows the Visual Studio Code interface with a terminal window open. The terminal displays the command to run a Python script and its output for the input number 10.

```
PS C:\Users\haasini\OneDrive\Documents\python> python -u "c:\Users\haasini\OneDrive\Documents\python\test1.py"
Enter a number: 10
even
PS C:\Users\haasini\OneDrive\Documents\python> 
```

The status bar at the bottom indicates the current file is `test1.py` at line 7, column 12, with 4 spaces, using UTF-8 encoding and CRLF line endings. The Python version is 3.13.5, and the Prettier extension is active.



### TASK 3

Prompt:

```
def read_file(filename):
```

With open(filename, 'r') as f:

```
    Return f.read()
```

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

fix the error with proper file handling like reading the lines from text file , printing invalid path for wrong path and missing file if file path is not existing.

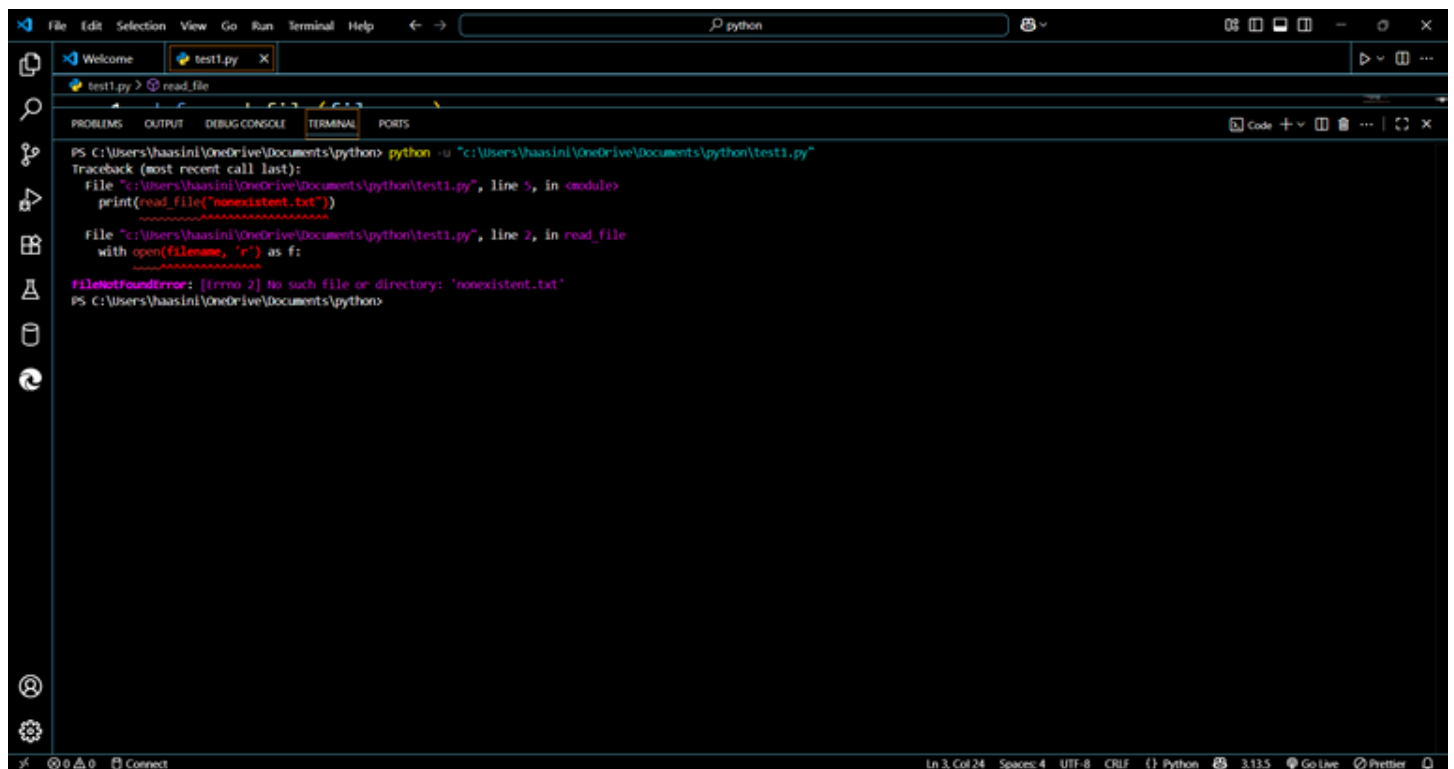
Code:





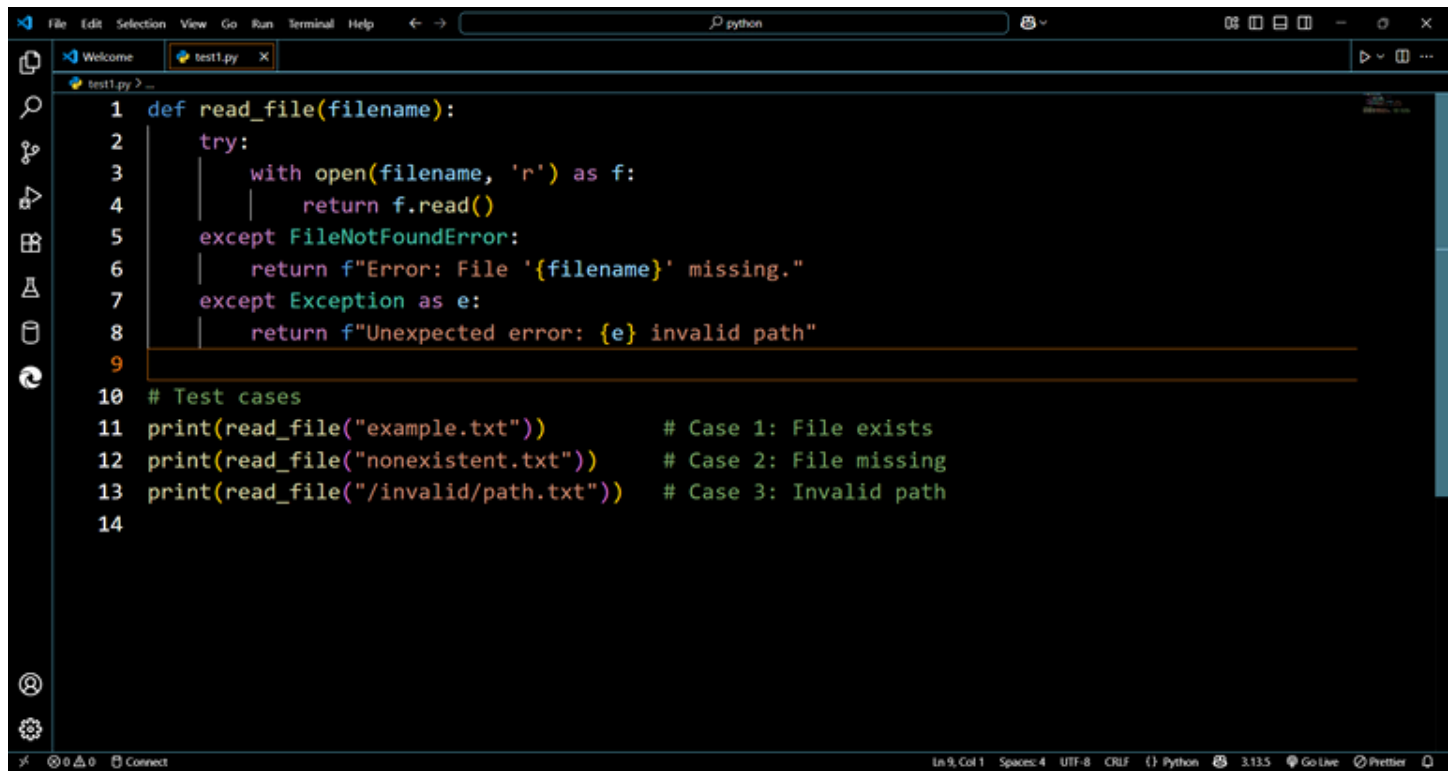
```
1 def read_file(filename):
2     with open(filename, 'r') as f:
3         return f.read()
4
5 print(read_file("nonexistent.txt"))
6
```

Error:



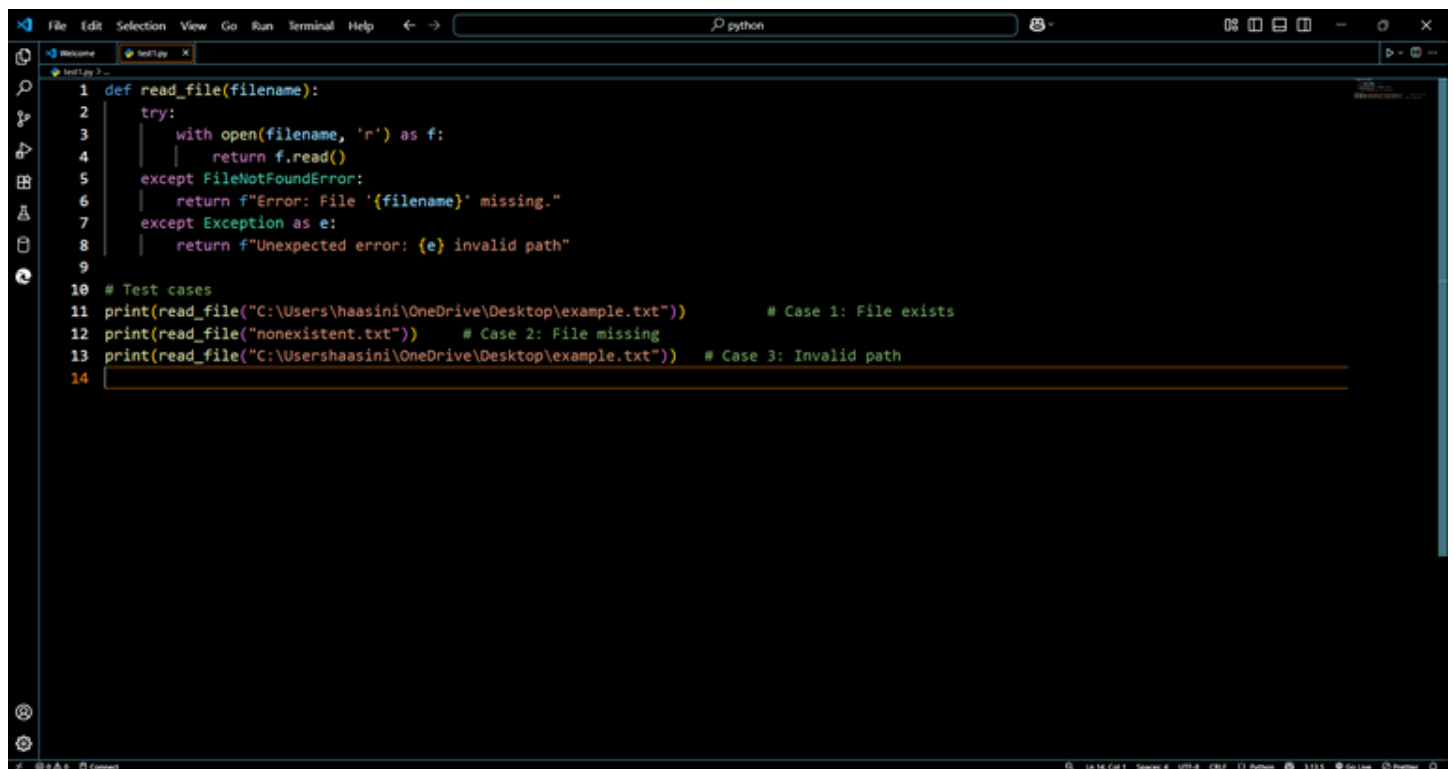
```
PS C:\Users\haasini\OneDrive\Documents\python> python -u "c:\Users\haasini\OneDrive\Documents\python\test1.py"
Traceback (most recent call last):
  File "c:\Users\haasini\OneDrive\Documents\python\test1.py", line 5, in <module>
    print(read_file("nonexistent.txt"))
  File "c:\Users\haasini\OneDrive\Documents\python\test1.py", line 2, in read_file
    with open(filename, 'r') as f:
FileNotFoundError: [Errno 2] No such file or directory: 'nonexistent.txt'
PS C:\Users\haasini\OneDrive\Documents\python>
```

Fixed code:



```
1 def read_file(filename):
2     try:
3         with open(filename, 'r') as f:
4             return f.read()
5     except FileNotFoundError:
6         return f"Error: File '{filename}' missing."
7     except Exception as e:
8         return f"Unexpected error: {e} invalid path"
9
10 # Test cases
11 print(read_file("example.txt"))      # Case 1: File exists
12 print(read_file("nonexistent.txt"))  # Case 2: File missing
13 print(read_file("/invalid/path.txt")) # Case 3: Invalid path
14
```

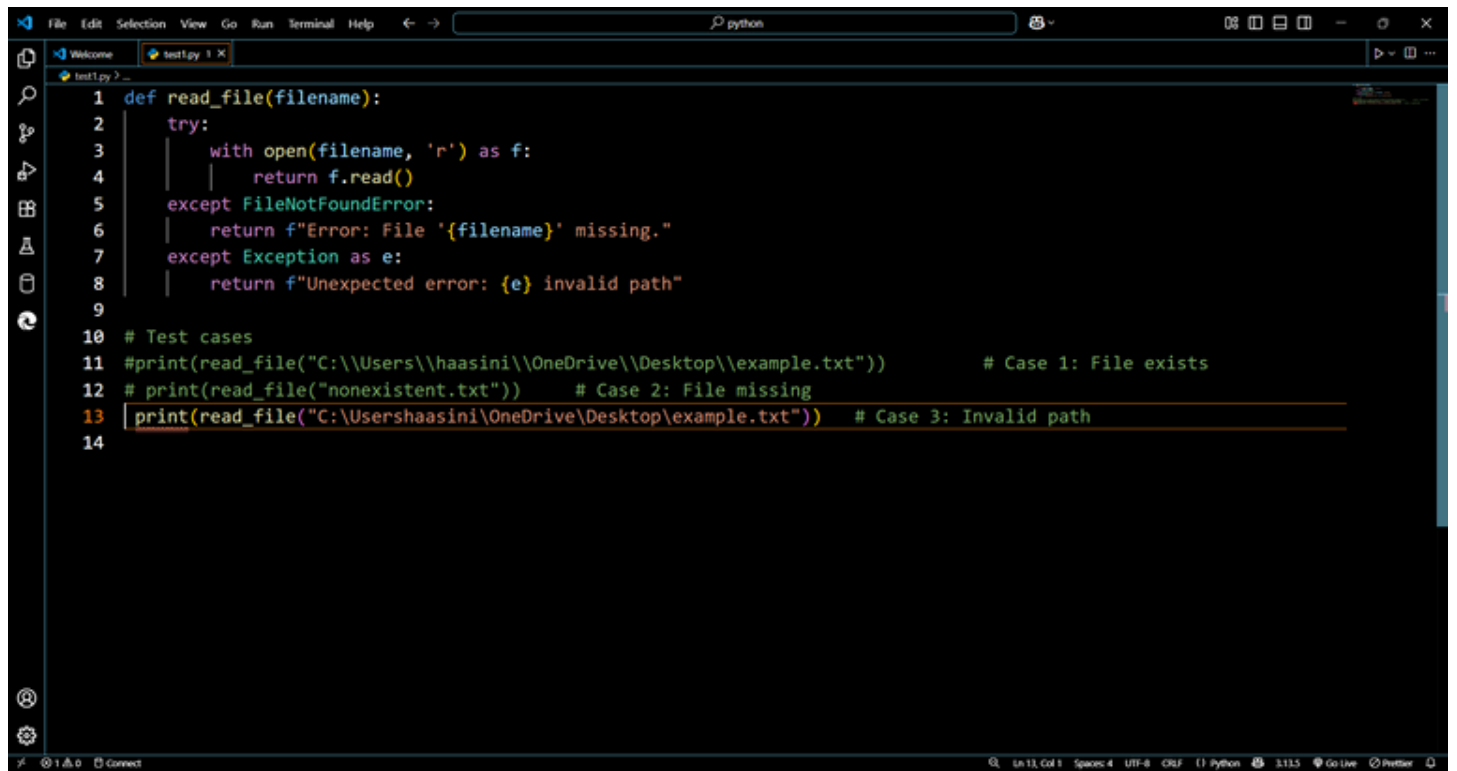
Output:



```
1 def read_file(filename):
2     try:
3         with open(filename, 'r') as f:
4             return f.read()
5     except FileNotFoundError:
6         return f"Error: File '{filename}' missing."
7     except Exception as e:
8         return f"Unexpected error: {e} invalid path"
9
10 # Test cases
11 print(read_file("C:\\Users\\haasini\\OneDrive\\Desktop\\example.txt")) # Case 1: File exists
12 print(read_file("nonexistent.txt")) # Case 2: File missing
13 print(read_file("C:\\Users\\haasini\\OneDrive\\Desktop\\example.txt")) # Case 3: Invalid path
14
```

Test case 1 invalid path

Code:



```
1 def read_file(filename):
2     try:
3         with open(filename, 'r') as f:
4             return f.read()
5     except FileNotFoundError:
6         return f"Error: File '{filename}' missing."
7     except Exception as e:
8         return f"Unexpected error: {e} invalid path"
9
10 # Test cases
11 #print(read_file("C:\\Users\\haasini\\OneDrive\\Desktop\\example.txt")) # Case 1: File exists
12 # print(read_file("nonexistent.txt")) # Case 2: File missing
13 print(read_file("C:\\Users\\haasini\\OneDrive\\Desktop\\example.txt")) # Case 3: Invalid path
14
```



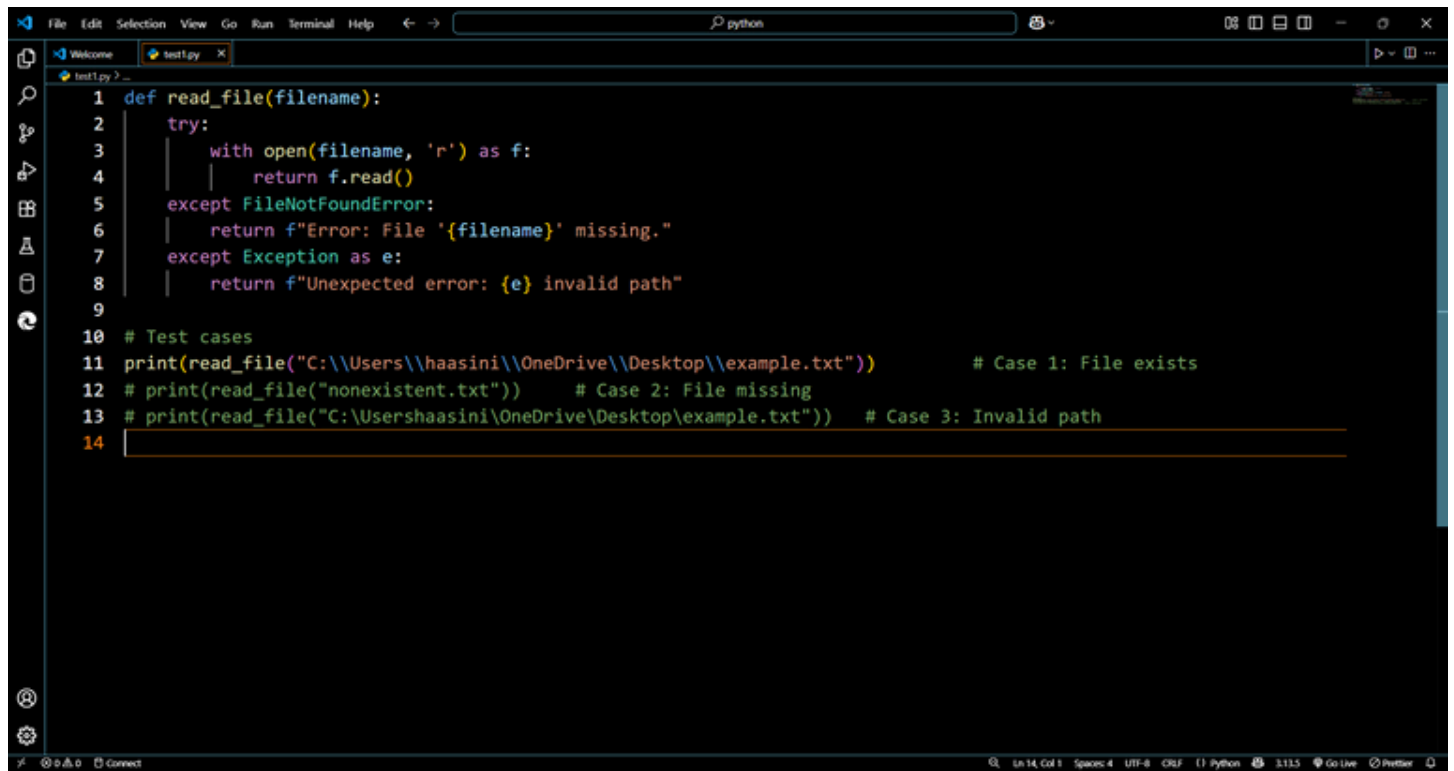
```
1 def read_file(filename):
2     try:
3         with open(filename, 'r') as f:
4             return f.read()
5     except FileNotFoundError:
6         return f"Error: File '{filename}' missing."
7     except Exception as e:
8         return f"Unexpected error: {e} invalid path"
9
10 # Test cases
11 #print(read_file("C:\\Users\\haasini\\OneDrive\\Desktop\\example.txt")) # Case 1: File exists
12 # print(read_file("nonexistent.txt")) # Case 2: File missing
13 print(read_file("C:\\Users\\haasini\\OneDrive\\Desktop\\example.txt")) # Case 3: Invalid path
```

PS C:\Users\haasini\OneDrive\Documents> python -u "C:\Users\haasini\OneDrive\Documents\python\test1.py"  
File "C:\Users\haasini\OneDrive\Documents\python\test1.py", line 13  
 print(read\_file("C:\\Users\\haasini\\OneDrive\\Desktop\\example.txt")) # Case 3: Invalid path  
~  
SyntaxError: (unicode error) 'unicodeescape' codec can't decode bytes in position 2-3: truncated 'U00000000' escape  
PS C:\Users\haasini\OneDrive\Documents>

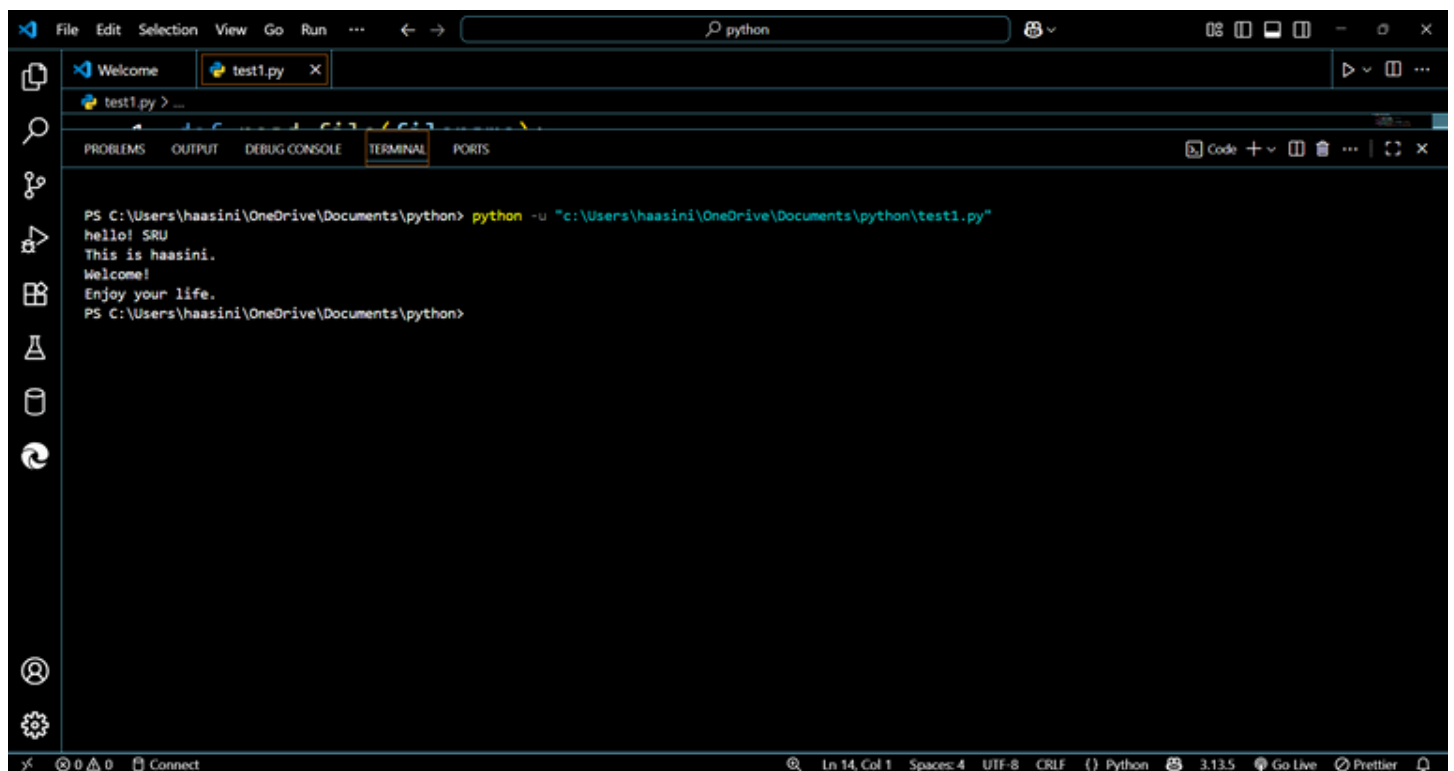
Test case 2

Reading the lines from the text

Code:



```
1 def read_file(filename):
2     try:
3         with open(filename, 'r') as f:
4             return f.read()
5     except FileNotFoundError:
6         return f"Error: File '{filename}' missing."
7     except Exception as e:
8         return f"Unexpected error: {e} invalid path"
9
10 # Test cases
11 print(read_file("C:\\Users\\haasini\\OneDrive\\Desktop\\example.txt")) # Case 1: File exists
12 # print(read_file("nonexistent.txt")) # Case 2: File missing
13 # print(read_file("C:\\Users\\haasini\\OneDrive\\Desktop\\example.txt")) # Case 3: Invalid path
14
```



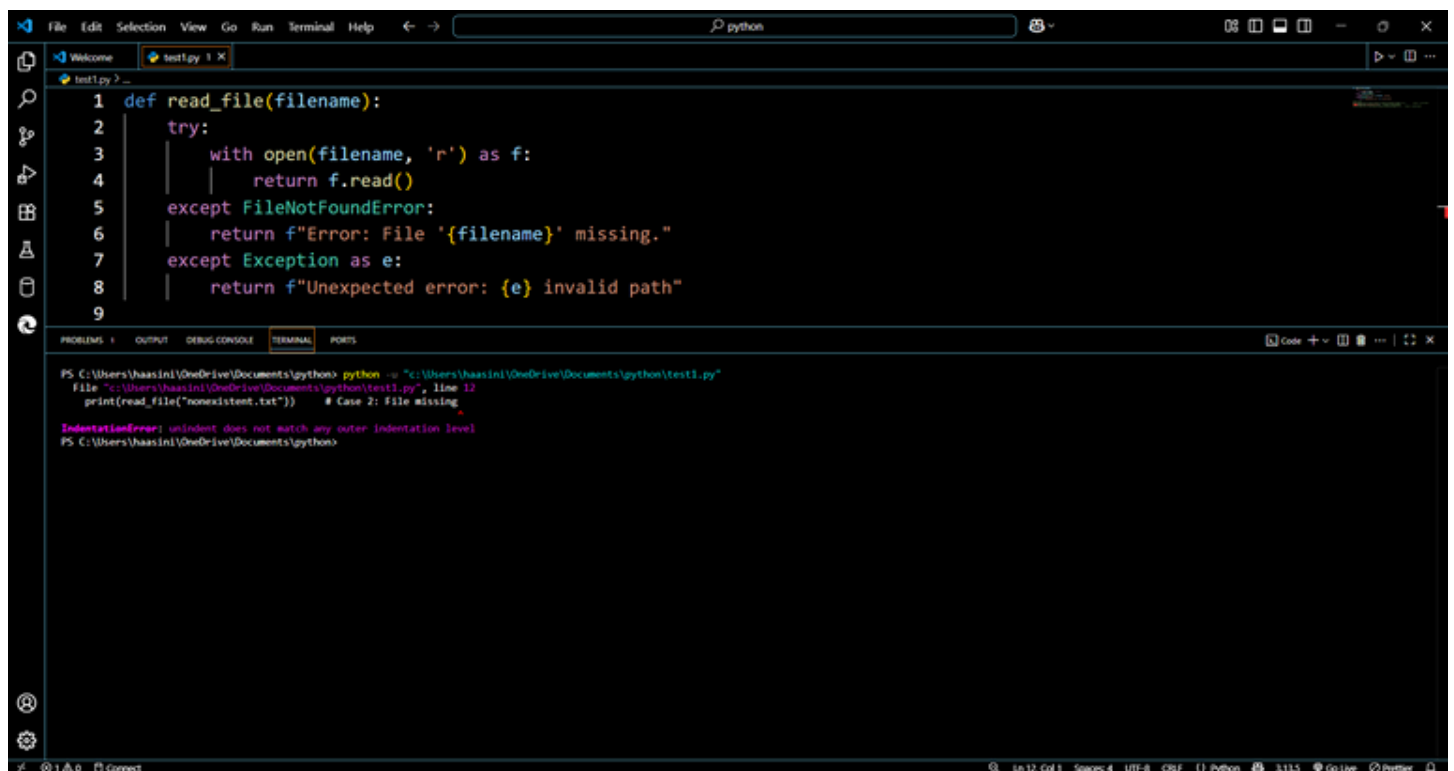
```
PS C:\Users\haasini\OneDrive\Documents\python> python -u "c:\Users\haasini\OneDrive\Documents\python\test1.py"
hello! SRU
This is haasini.
Welcome!
Enjoy your life.
PS C:\Users\haasini\OneDrive\Documents\python>
```

Text case 3 : file missing

Code:



```
1 def read_file(filename):
2     try:
3         with open(filename, 'r') as f:
4             return f.read()
5     except FileNotFoundError:
6         return f"Error: File '{filename}' missing."
7     except Exception as e:
8         return f"Unexpected error: {e} invalid path"
9
10 # Test cases
11 #print(read_file("C:\\Users\\haasini\\OneDrive\\Desktop\\example.txt")) # Case 1: File exists
12 print(read_file("nonexistent.txt")) # Case 2: File missing
13 # print(read_file("C:\\Users\\haasini\\OneDrive\\Desktop\\example.txt")) # Case 3: Invalid path
14
```



```
1 def read_file(filename):
2     try:
3         with open(filename, 'r') as f:
4             return f.read()
5     except FileNotFoundError:
6         return f"Error: File '{filename}' missing."
7     except Exception as e:
8         return f"Unexpected error: {e} invalid path"
9
10 # Test cases
11 #print(read_file("C:\\Users\\haasini\\OneDrive\\Desktop\\example.txt")) # Case 1: File exists
12 print(read_file("nonexistent.txt")) # Case 2: File missing
13 # print(read_file("C:\\Users\\haasini\\OneDrive\\Desktop\\example.txt")) # Case 3: Invalid path
14
```

```
PS C:\Users\haasini\OneDrive\Documents\python> python -p "c:\Users\haasini\OneDrive\Documents\python\test1.py"
File "c:\Users\haasini\OneDrive\Documents\python\test1.py", line 12
print(read_file("nonexistent.txt")) # Case 2: File missing
IndentationError: unindent does not match any outer indentation level
PS C:\Users\haasini\OneDrive\Documents\python>
```

#### TASK 4:

class Car:

def start(self):

return "Car started"

```
my_car = Car()
```

```
print(my_car.drive())
```

debug and fix this code with 3 assert cases

code:



```
1 class Car:
2     def start(self):
3         return "Car started"
4
5 my_car = Car()
6 print(my_car.drive()) # This will raise an AttributeError since 'drive' method is not defined in Car class.
```

Error:

The screenshot shows a VS Code editor with a file named `test1.py`. The code defines a `Car` class with a `start` method and creates an instance `my_car`. The `print(my_car.drive())` line on line 6 is highlighted. The terminal at the bottom shows the command `python -i "c:\Users\haasini\OneDrive\Documents\python\test1.py"` and the resulting `AttributeError: 'Car' object has no attribute 'drive'`.

```
1 class Car:
2     def start(self):
3         return "Car started"
4
5 my_car = Car()
6 print(my_car.drive())
```

PS C:\Users\haasini\OneDrive\Documents\python> python -i "c:\Users\haasini\OneDrive\Documents\python\test1.py"

Traceback (most recent call last):  
File "c:\Users\haasini\OneDrive\Documents\python\test1.py", line 6, in <module>  
 print(my\_car.drive())  
AttributeError: 'Car' object has no attribute 'drive'

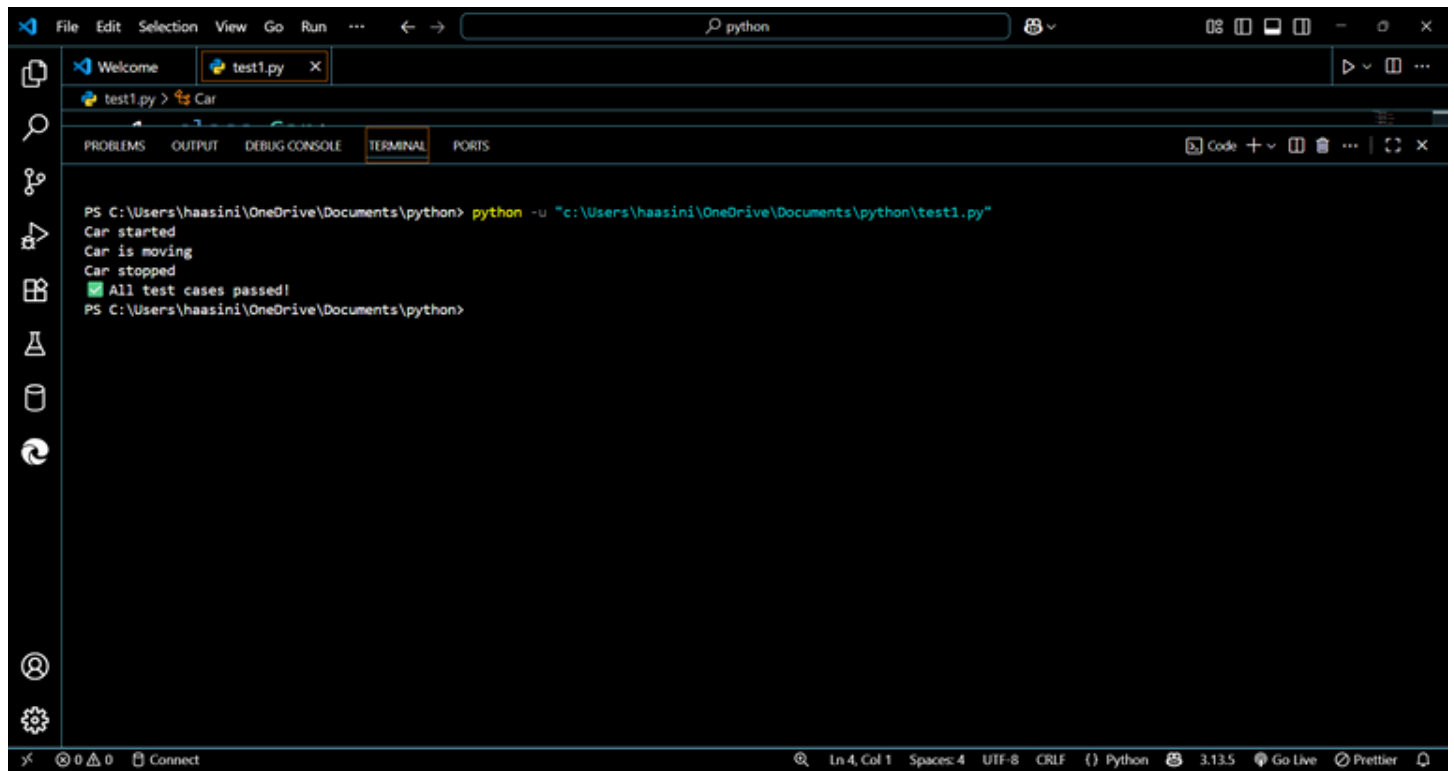
PS C:\Users\haasini\OneDrive\Documents\python>

Fixed code:

The screenshot shows the same VS Code editor with the fixed `test1.py`. The `Car` class now has `start`, `drive`, and `stop` methods. The script creates a `car` instance and runs three unit tests using `print` and `assert` to verify the methods work correctly. The final output is `print("✅ All test cases passed!")`.

```
1 class Car:
2     def start(self):
3         return "Car started"
4
5     def drive(self):
6         return "Car is moving"
7
8     def stop(self):
9         return "Car stopped"
10
11 car = Car()
12 # Test 1
13 print(car.start()) # Expected: Car started
14 assert car.start() == "Car started"
15
16 # Test 2
17 print(car.drive()) # Expected: Car is moving
18 assert car.drive() == "Car is moving"
19
20 # Test 3
21 print(car.stop()) # Expected: Car stopped
22 assert car.stop() == "Car stopped"
23
24 print("✅ All test cases passed!")
```

Output:



```
PS C:\Users\haasini\OneDrive\Documents\python> python -u "c:\Users\haasini\OneDrive\Documents\python\test1.py"
Car started
Car is moving
Car stopped
All test cases passed!
PS C:\Users\haasini\OneDrive\Documents\python>
```

## Explanation:

Before :

The error occurs because drive() was **called but not defined** inside the class.

Fix: Either **define drive()** or correct the call to an existing method.

In this case, I defined a new drive() method.

After fixing error:

In the **original code**, only the start() method was defined, but drive() was called → causing an AttributeError.

To fix this, I added drive() and stop() methods to the Car class.

Now the class supports **three actions**: start, drive, and stop.

I wrote **three test cases** to check each method:

1. car.start() → returns "Car started"
2. car.drive() → returns "Car is moving"
3. car.stop() → returns "Car stopped"

I used both print() (to see actual output) and assert (to verify correctness).

## TASK 5:

Prompt:

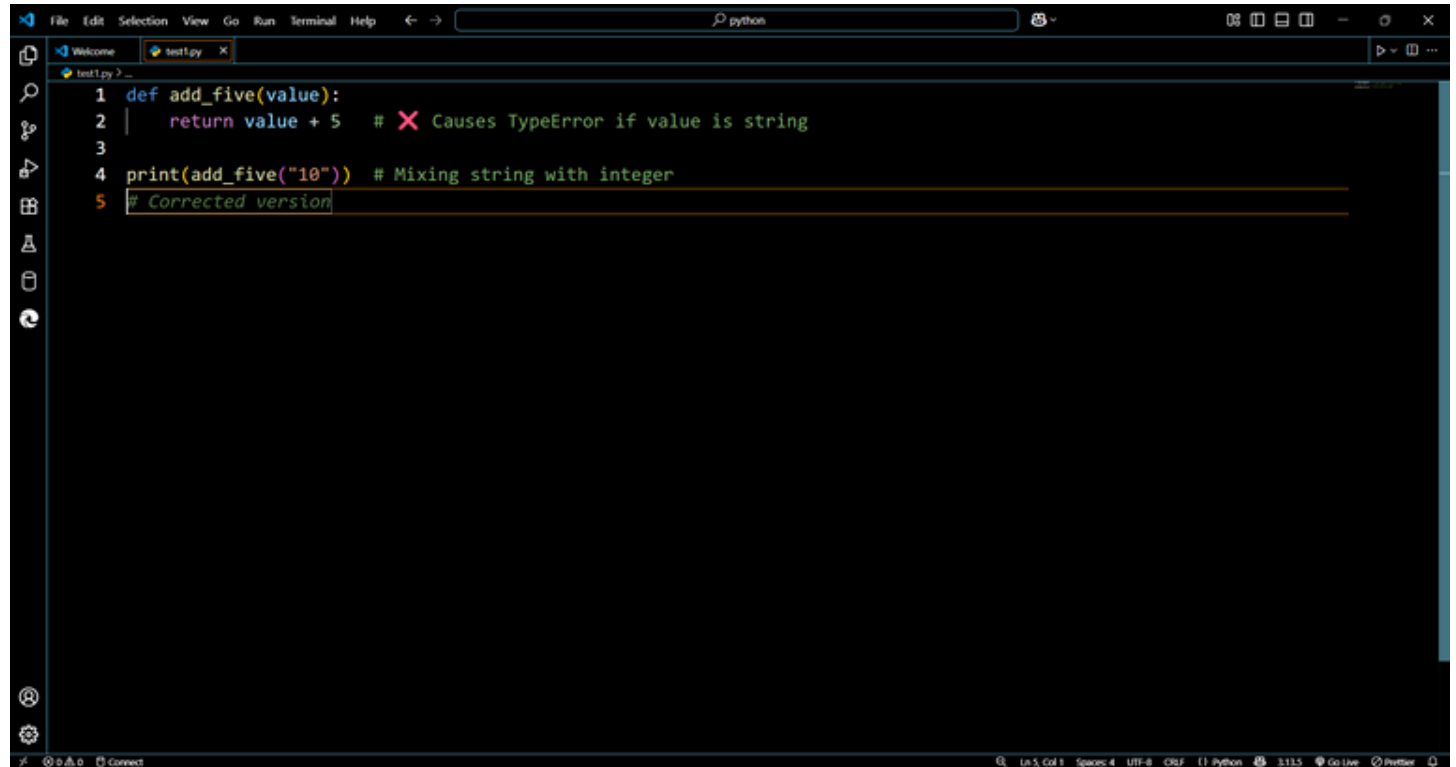
```
def add_five(value):
```

```
    return value + 5 # ✗ Causes TypeError if value is string
```



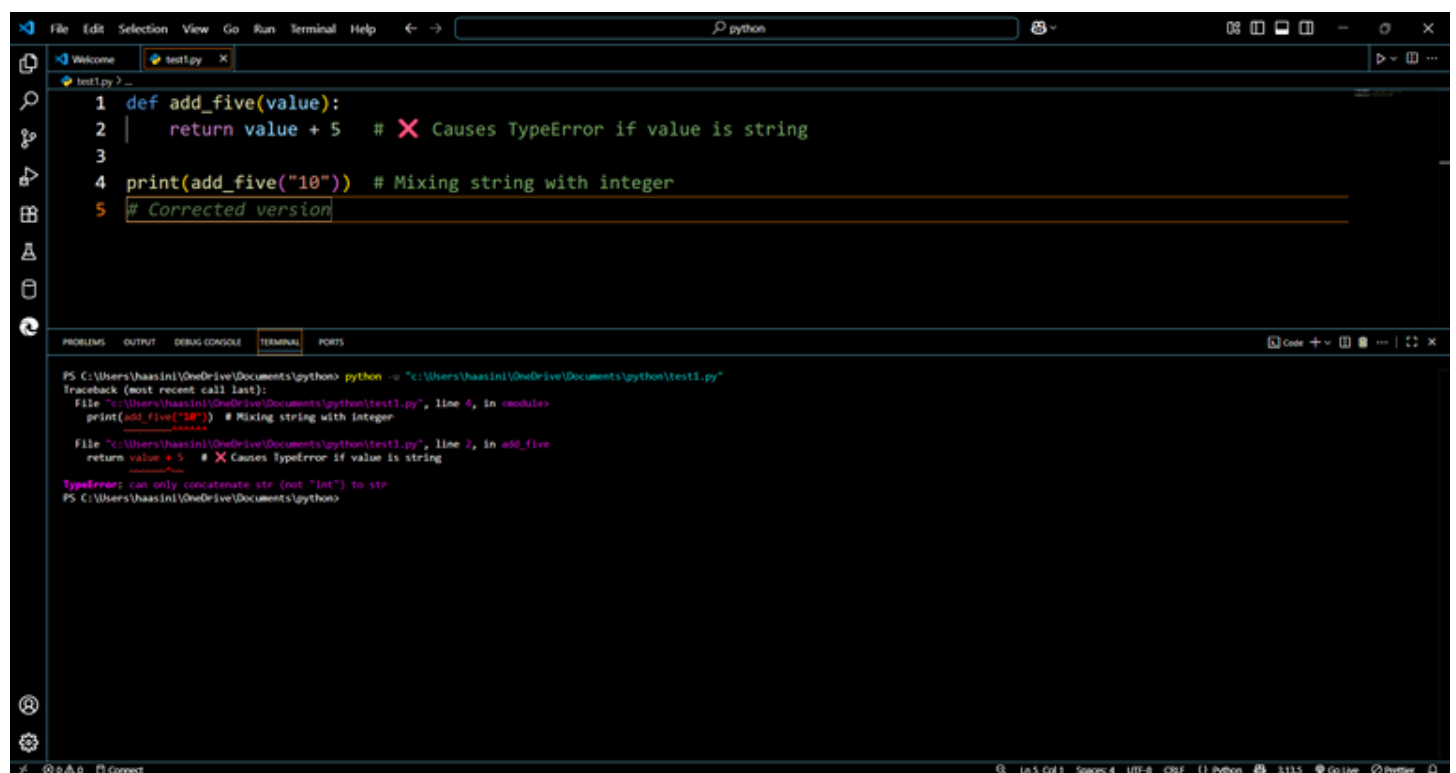
```
print(add_five("10")) # Mixing string with integer
```

debug the error and give the code with type casting and string concatenation.



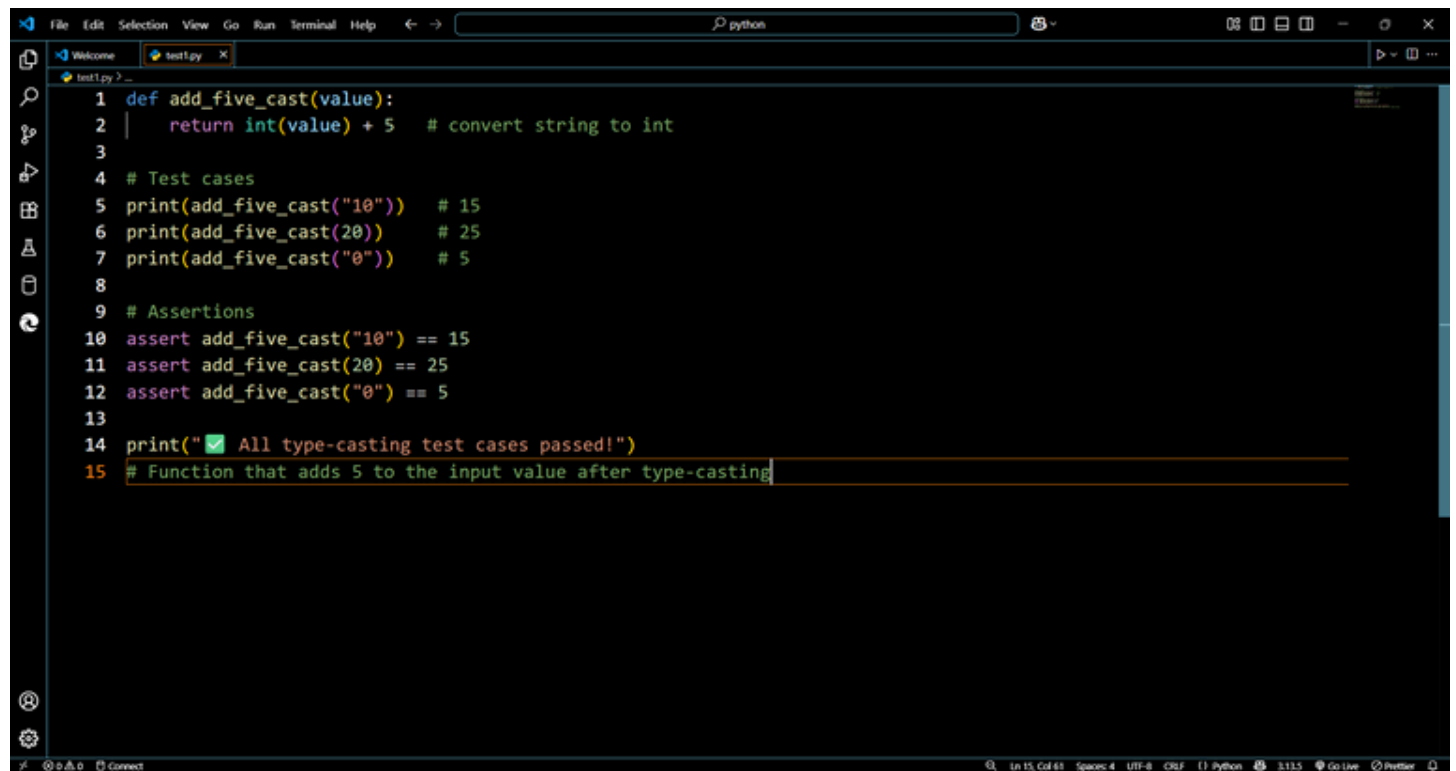
```
1 def add_five(value):
2     return value + 5 # X Causes TypeError if value is string
3
4 print(add_five("10")) # Mixing string with integer
5 # Corrected version
```

Error:



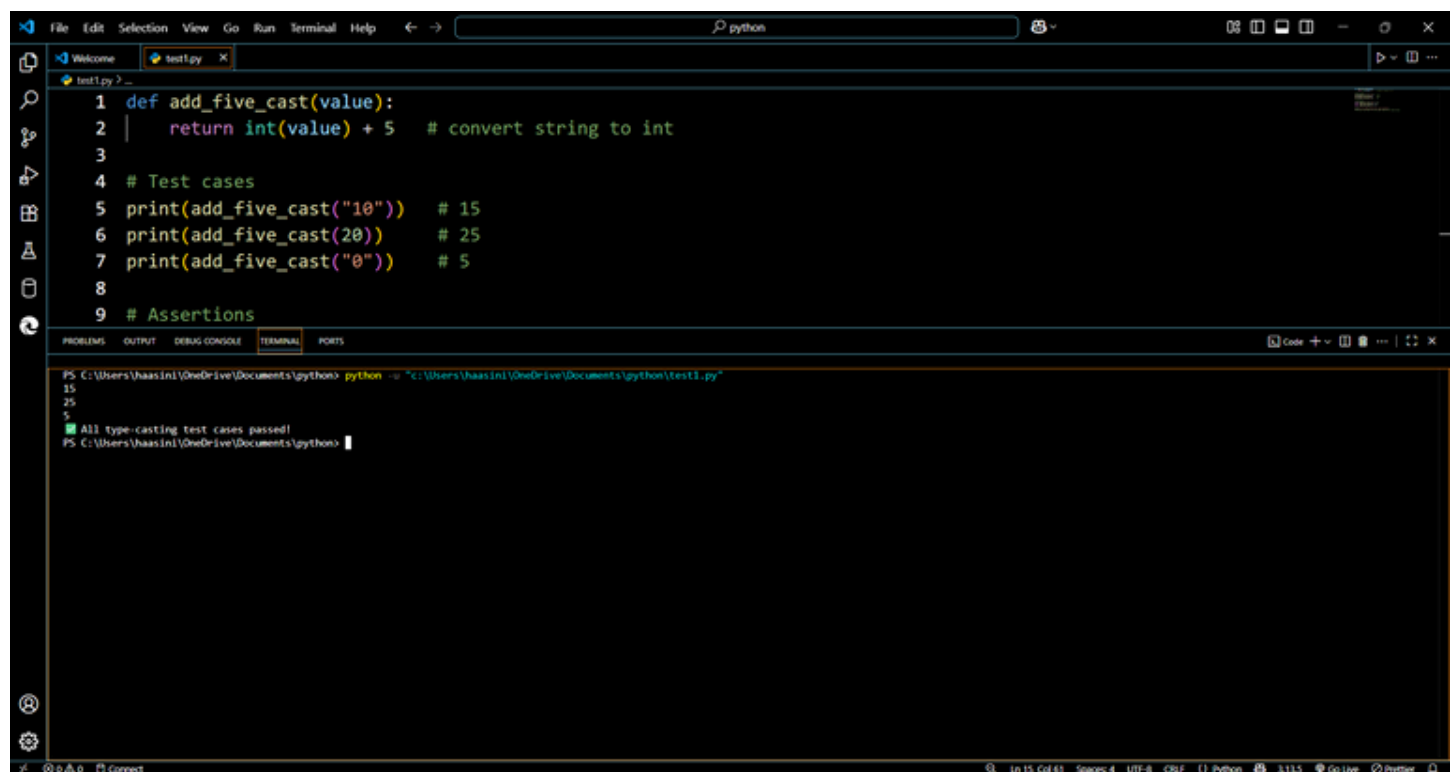
```
PS C:\Users\haasini\OneDrive\Documents\python> python -> "c:\Users\haasini\OneDrive\Documents\python\test1.py"
Traceback (most recent call last):
  File "c:\Users\haasini\OneDrive\Documents\python\test1.py", line 4, in <module>
    print(add_five("10")) # Mixing string with integer
          ~~~~~^~~~~~
  File "c:\Users\haasini\OneDrive\Documents\python\test1.py", line 2, in add_five
    return value + 5 # X Causes TypeError if value is string
           ~~~~~^~~~~
TypeError: can only concatenate str (not "int") to str
PS C:\Users\haasini\OneDrive\Documents\python>
```

Fixed code type casting:



```
1 def add_five_cast(value):
2 |     return int(value) + 5 # convert string to int
3
4 # Test cases
5 print(add_five_cast("10")) # 15
6 print(add_five_cast(20)) # 25
7 print(add_five_cast("0")) # 5
8
9 # Assertions
10 assert add_five_cast("10") == 15
11 assert add_five_cast(20) == 25
12 assert add_five_cast("0") == 5
13
14 print("✅ All type-casting test cases passed!")
15 # Function that adds 5 to the input value after type-casting"
```

Output:

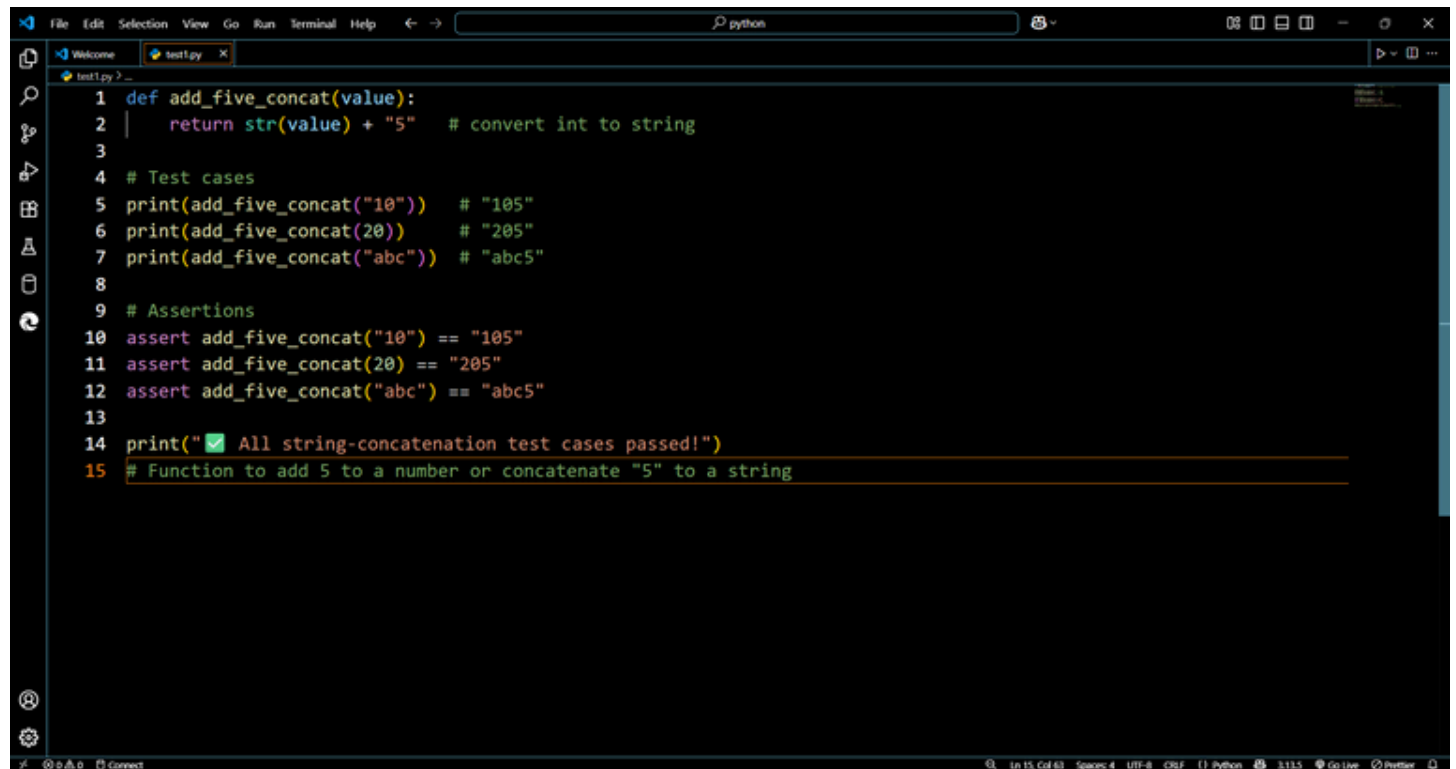


```
1 def add_five_cast(value):
2 |     return int(value) + 5 # convert string to int
3
4 # Test cases
5 print(add_five_cast('10')) # 15
6 print(add_five_cast(20)) # 25
7 print(add_five_cast('0')) # 5
8
9 # Assertions
10 assert add_five_cast('10') == 15
11 assert add_five_cast(20) == 25
12 assert add_five_cast('0') == 5
13
14 print('✅ All type-casting test cases passed!')
15 # Function that adds 5 to the input value after type-casting"
```

```
PS C:\Users\haasini\OneDrive\Documents\python> python -p "c:\Users\haasini\OneDrive\Documents\python\test1.py"
15
25
5
✅ All type-casting test cases passed!
PS C:\Users\haasini\OneDrive\Documents\python>
```

Fixed code 2 String concatenation

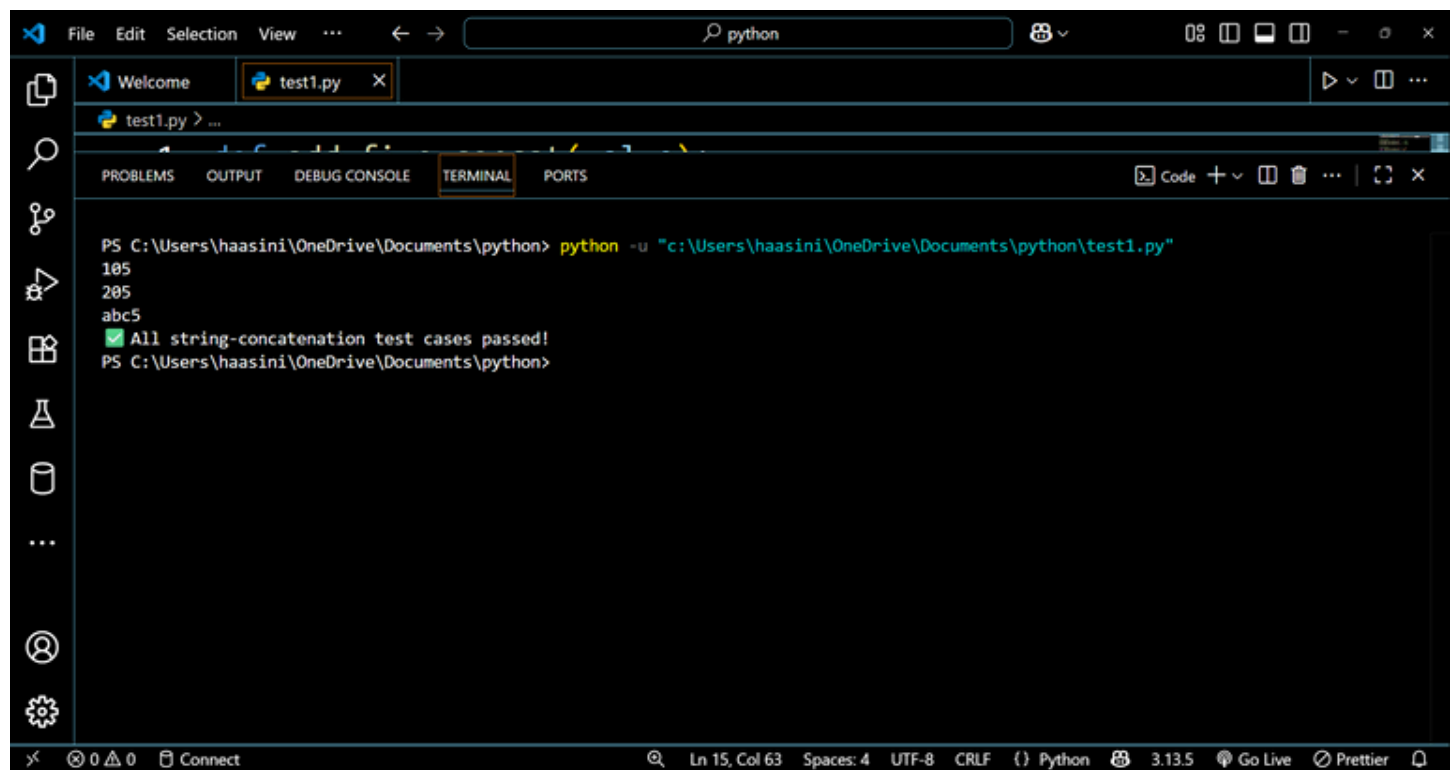
Code:



The screenshot shows a code editor with a dark theme. The file 'test1.py' is open. The code defines a function 'add\_five\_concat' that takes a 'value' and returns 'str(value) + "5"'. It includes test cases for '10', '20', and 'abc', followed by assertions to verify the results. A green checkmark icon is used in the print statement to indicate success.

```
1 def add_five_concat(value):
2     return str(value) + "5" # convert int to string
3
4 # Test cases
5 print(add_five_concat("10")) # "105"
6 print(add_five_concat(20))  # "205"
7 print(add_five_concat("abc")) # "abc5"
8
9 # Assertions
10 assert add_five_concat("10") == "105"
11 assert add_five_concat(20) == "205"
12 assert add_five_concat("abc") == "abc5"
13
14 print("✅ All string-concatenation test cases passed!")
15 # Function to add 5 to a number or concatenate "5" to a string
```

Output:



The screenshot shows the terminal output of running 'python -u "c:\Users\haasini\OneDrive\Documents\python\test1.py"'. The output matches the test cases and assertions in the code, showing '105', '205', 'abc5', and a success message with a green checkmark.

```
PS C:\Users\haasini\OneDrive\Documents\python> python -u "c:\Users\haasini\OneDrive\Documents\python\test1.py"
105
205
abc5
✅ All string-concatenation test cases passed!
PS C:\Users\haasini\OneDrive\Documents\python>
```

