Assignment-07

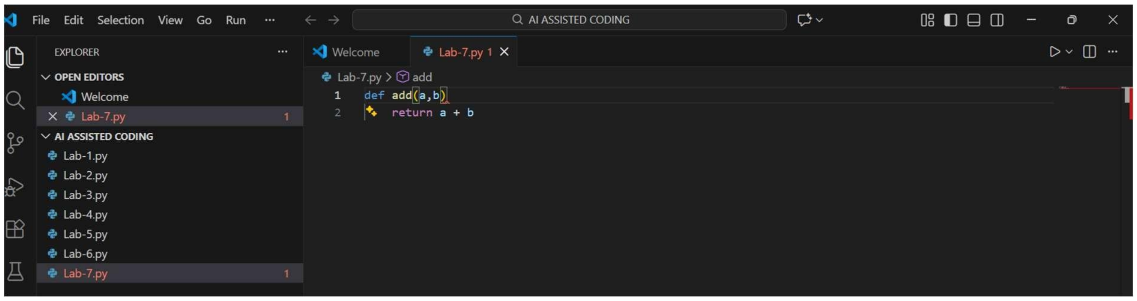
G.Sai teja

2303A52135

Batch 41

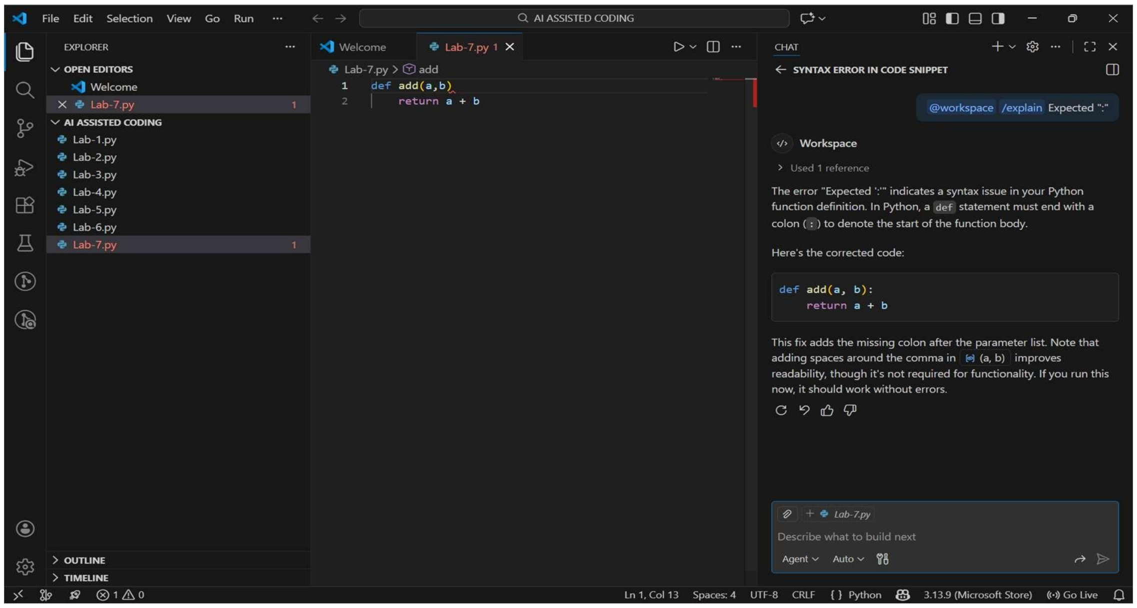
Task 1: Fixing Syntax Errors

Fix:



```
def add(a,b)
    return a + b
```

Review:



```
def add(a,b)
    return a + b
```

SYNTAX ERROR IN CODE SNIPPET

@workspace /explain Expected ":"

Workspace

Used 1 reference

The error "Expected ':'" indicates a syntax issue in your Python function definition. In Python, a `def` statement must end with a colon (`:`) to denote the start of the function body.

Here's the corrected code:

```
def add(a, b):
    return a + b
```

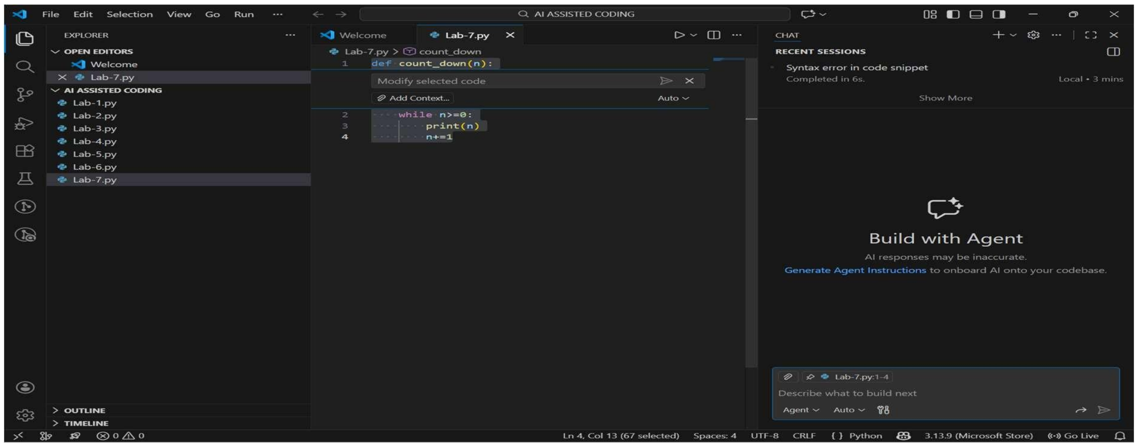
This fix adds the missing colon after the parameter list. Note that adding spaces around the comma in `(a, b)` improves readability, though it's not required for functionality. If you run this now, it should work without errors.

Justification:

The program had a small syntax error because the colon (:) was missing after the function definition. Python needs this colon to know where the function body starts. The AI tool detected the error, explained the reason clearly, and fixed it by adding the colon. After that, the function worked correctly without any errors.

Task 2: Debugging Logic Errors in Loop

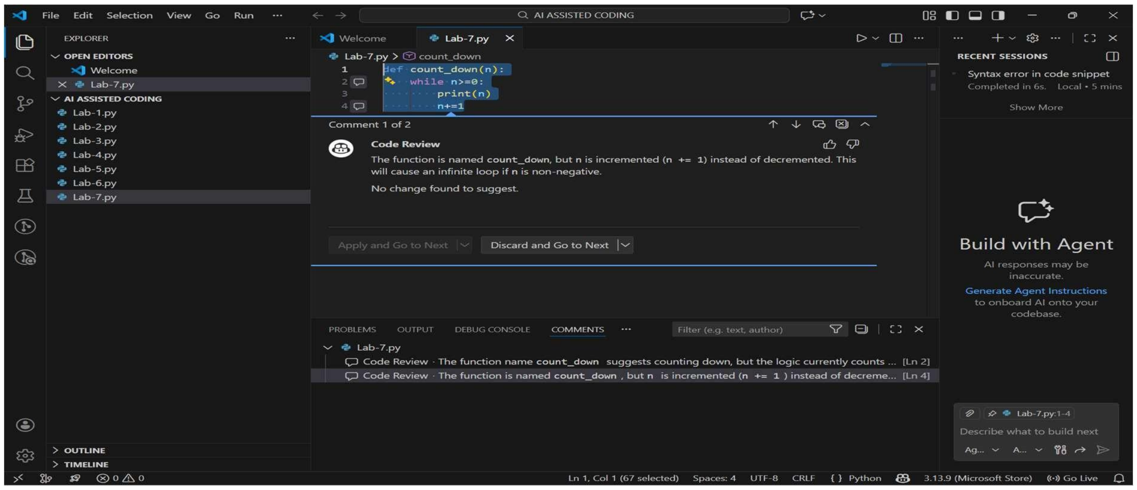
Modify:



```python
def count_down(n):
    while n>=0:
        print(n)
        n+=1
```

RECENT SESSIONS
Syntax error in code snippet
Completed in 6s. Local • 3 mins

Review:



```python
def count_down(n):
    while n>=0:
        print(n)
        n+=1
```

Comment 1 of 2

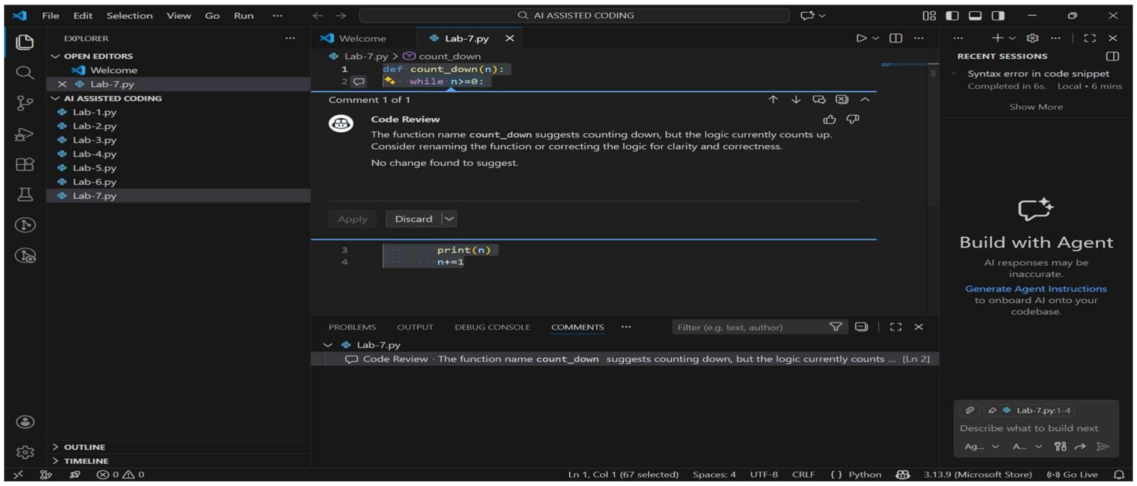
Code Review

The function is named count_down, but n is incremented (n += 1) instead of decremented. This will cause an infinite loop if n is non-negative.

No change found to suggest.

COMMENTS

Lab-7.py
Code Review · The function name count_down suggests counting down, but the logic currently counts ... [Ln 2]
Code Review · The function is named count_down , but n is incremented (n += 1) instead of decreme... [Ln 4]



```python
def count_down(n):
    while n>=0:
        print(n)
        n+=1
```

Comment 1 of 1

Code Review

The function name count_down suggests counting down, but the logic currently counts up. Consider renaming the function or correcting the logic for clarity and correctness.

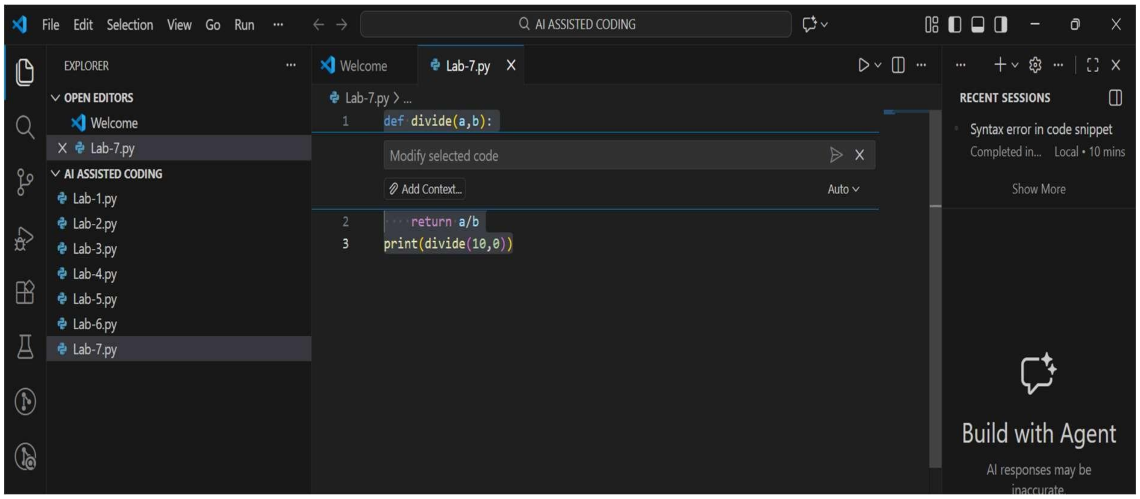
No change found to suggest.

COMMENTS

Lab-7.py
Code Review · The function name count_down suggests counting down, but the logic currently counts ... [Ln 2]

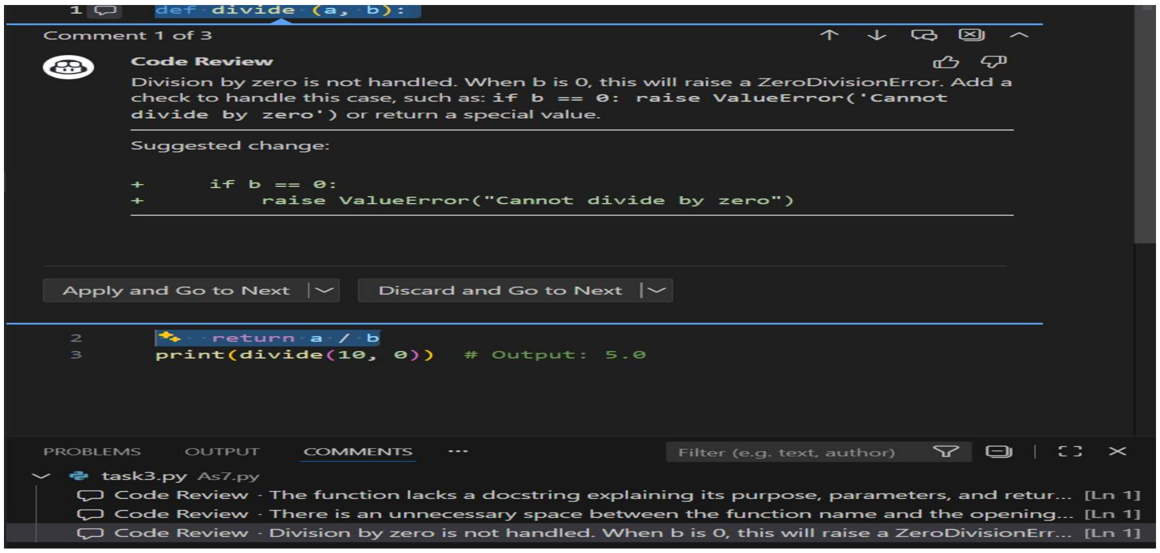
Justification: In this case, the loop was running infinitely because of a logical error in how the variable was updated. The function was named count_down, but instead of decreasing the value, it was increasing it (n += 1), so the loop condition never became false. The AI tool identified this mismatch between the function name and the logic and explained why it caused an infinite loop. It then fixed the issue by correcting the increment/decrement logic so the value changes in the right direction. After the fix, the loop worked as expected and stopped correctly.

Task 3: Handling Runtime Errors (Division by Zero) Fix:



```python
def divide(a,b):
    return a/b
print(divide(10,0))
```

Review:



```
def divide (a, b):
```

Comment 1 of 3

Code Review

Division by zero is not handled. When b is 0, this will raise a ZeroDivisionError. Add a check to handle this case, such as: `if b == 0: raise ValueError('Cannot divide by zero')` or return a special value.

Suggested change:

```
+       if b == 0:
+           raise ValueError("Cannot divide by zero")
```

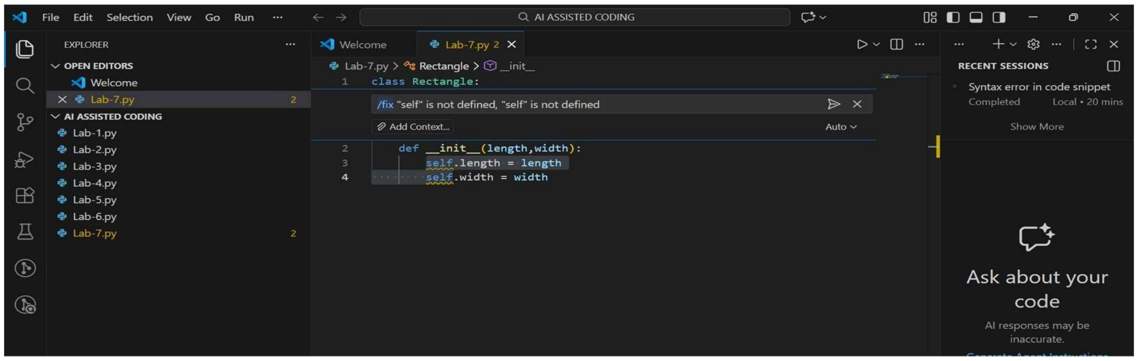
```
    return a / b
print(divide(10, 0))  # Output: 5.0
```

```
task3.py As7.py
  Code Review · The function lacks a docstring explaining its purpose, parameters, and retur...  [Ln 1]
  Code Review · There is an unnecessary space between the function name and the opening...  [Ln 1]
  Code Review · Division by zero is not handled. When b is 0, this will raise a ZeroDivisionErr...  [Ln 1]
```

Justification: In this scenario, the division function caused a runtime error because it tried to divide a number by zero, which is not allowed in Python. The AI tool identified this issue when the function was executed and explained that it would raise a ZeroDivisionError. To fix this, the AI added a try-except block to handle the error safely. With this change, the program no longer crashes and instead shows a clear message when division by zero occurs, making the code safer and more reliable.

Task 4: Debugging Class Definition Errors

Fix:



/fix "self" is not defined, "self" is not defined

```
class Rectangle:
    def __init__(length,width):
        self.length = length
        self.width = width
```

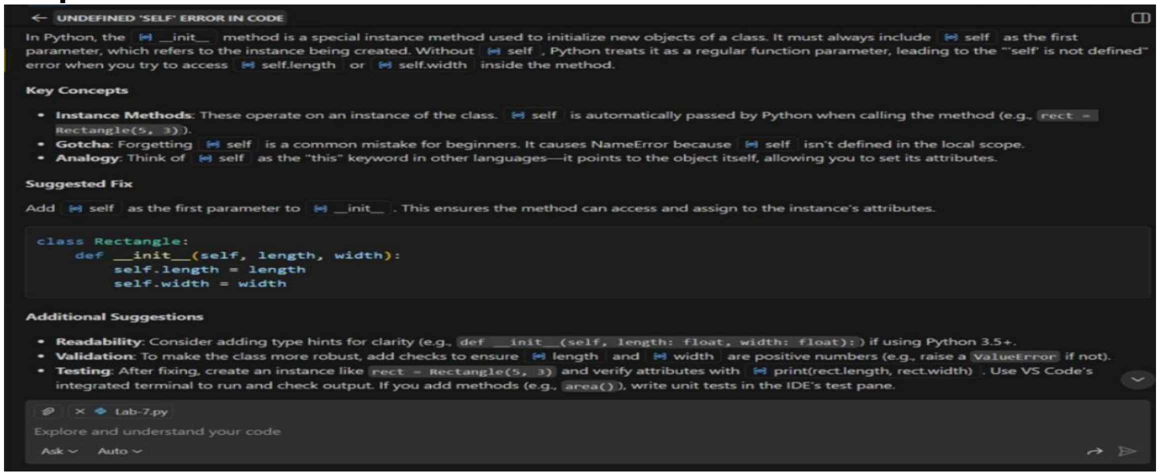
RECENT SESSIONS

Syntax error in code snippet — Completed — Local • 20 mins

Ask about your code

AI responses may be inaccurate.

Explain:



UNDEFINED 'SELF' ERROR IN CODE

In Python, the `__init__` method is a special instance method used to initialize new objects of a class. It must always include `self` as the first parameter, which refers to the instance being created. Without `self`, Python treats it as a regular function parameter, leading to the ""self" is not defined" error when you try to access `self.length` or `self.width` inside the method.

Key Concepts

- **Instance Methods**: These operate on an instance of the class. `self` is automatically passed by Python when calling the method (e.g., `rect = Rectangle(5, 3)`).
- **Gotcha**: Forgetting `self` is a common mistake for beginners. It causes NameError because `self` isn't defined in the local scope.
- **Analogy**: Think of `self` as the "this" keyword in other languages—it points to the object itself, allowing you to set its attributes.

Suggested Fix

Add `self` as the first parameter to `__init__`. This ensures the method can access and assign to the instance's attributes.

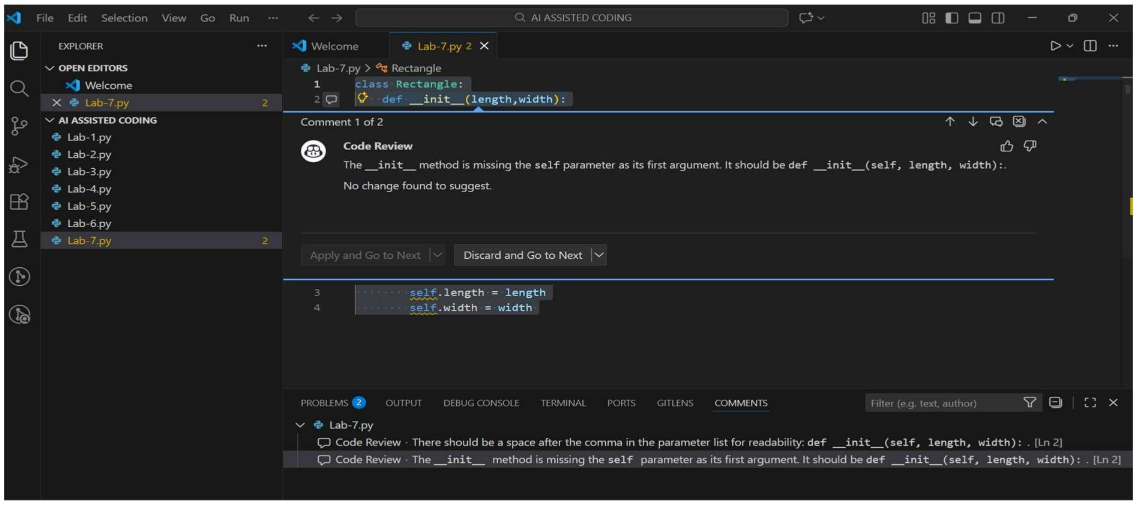
```
class Rectangle:
    def __init__(self, length, width):
        self.length = length
        self.width = width
```

Additional Suggestions

- **Readability**: Consider adding type hints for clarity (e.g., `def __init__(self, length: float, width: float):`) if using Python 3.5+.
- **Validation**: To make the class more robust, add checks to ensure `length` and `width` are positive numbers (e.g., raise a `ValueError` if not).
- **Testing**: After fixing, create an instance like `rect = Rectangle(5, 3)` and verify attributes with `print(rect.length, rect.width)`. Use VS Code's integrated terminal to run and check output. If you add methods (e.g., `area()`), write unit tests in the IDE's test pane.

Explore and understand your code

Review:



```
class Rectangle:
    def __init__(length,width):
```

Comment 1 of 2

Code Review

The `__init__` method is missing the `self` parameter as its first argument. It should be `def __init__(self, length, width):`.

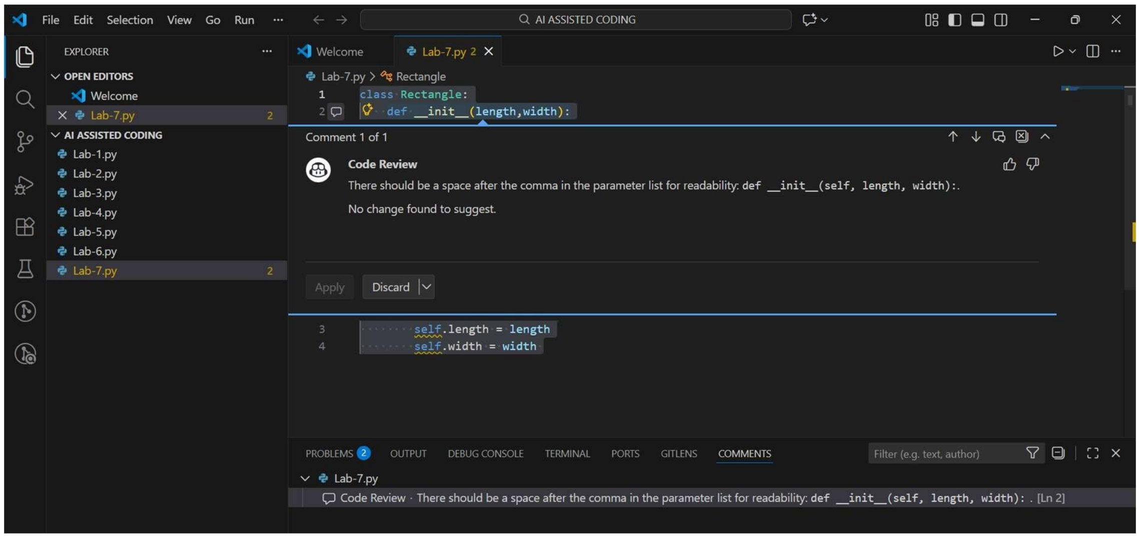
No change found to suggest.

Apply and Go to Next | Discard and Go to Next

```
        self.length = length
        self.width = width
```

COMMENTS

Lab-7.py
- Code Review · There should be a space after the comma in the parameter list for readability: `def __init__(self, length, width):` . [Ln 2]
- Code Review · The `__init__` method is missing the `self` parameter as its first argument. It should be `def __init__(self, length, width):` . [Ln 2]



```
class Rectangle:
    def __init__(length,width):
```

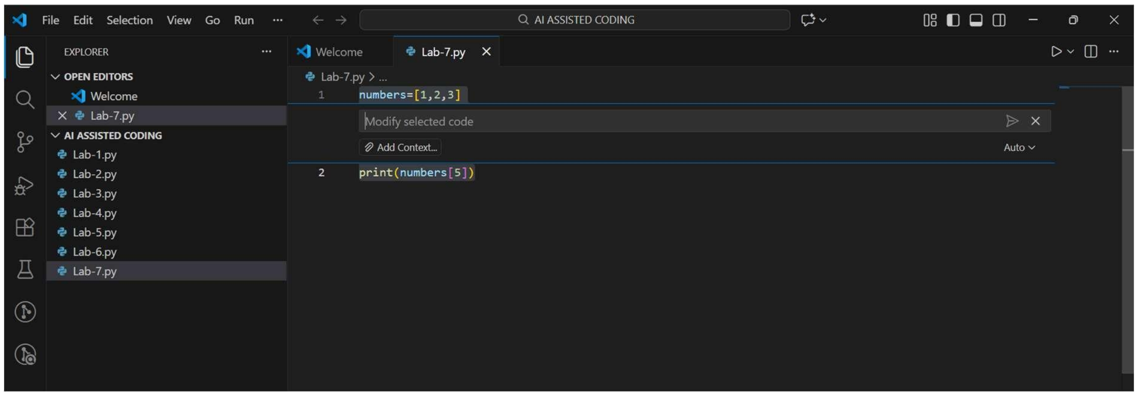
Code Review
There should be a space after the comma in the parameter list for readability: `def __init__(self, length, width):`.
No change found to suggest.

```
        self.length = length
        self.width = width
```

Code Review · There should be a space after the comma in the parameter list for readability: `def __init__(self, length, width):` . [Ln 2]

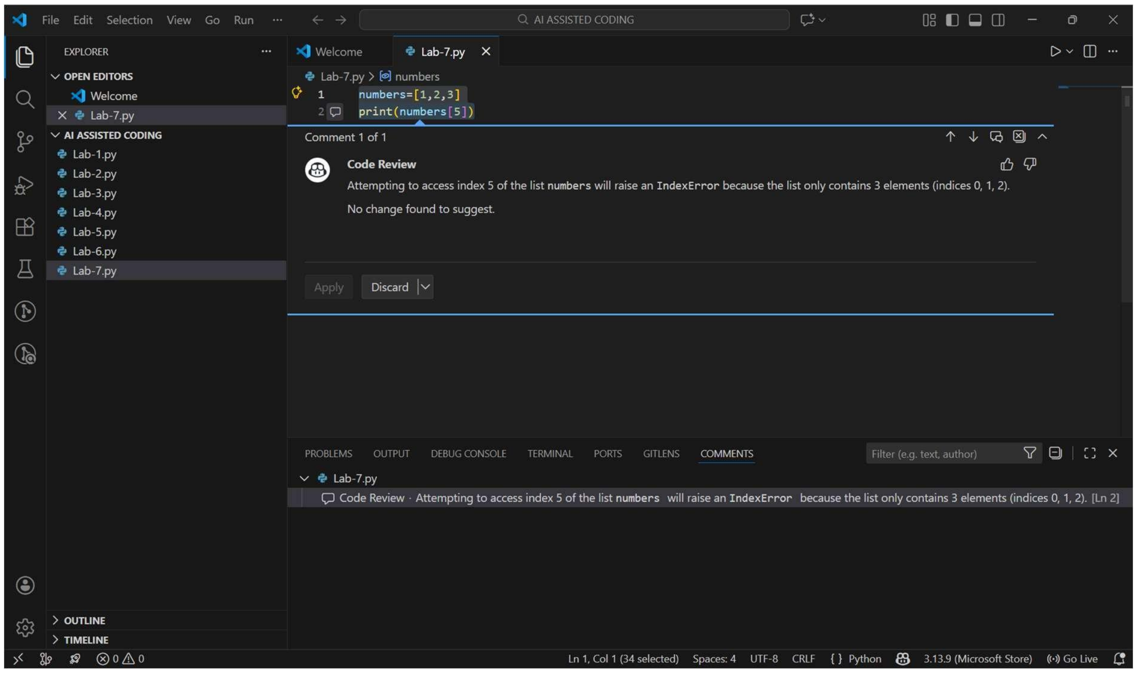
Justification: In this scenario, the class constructor had an error because the self parameter was missing in the __init__() method. The AI tool detected this problem and explained that self is required to refer to the current object and store instance variables. It then corrected the class by adding self to the constructor and using it properly for attributes. After the fix, the class worked correctly, showing how self is essential in Python class definitions.

Task 5: Resolving Index Errors in Lists Modify:



```
numbers=[1,2,3]
print(numbers[5])
```

Review:


```
numbers=[1,2,3]
print(numbers[5])
```

Comment 1 of 1

Code Review

Attempting to access index 5 of the list `numbers` will raise an `IndexError` because the list only contains 3 elements (indices 0, 1, 2).

No change found to suggest.

Lab-7.py

Code Review · Attempting to access index 5 of the list `numbers` will raise an `IndexError` because the list only contains 3 elements (indices 0, 1, 2). [Ln 2]

Justification: In this scenario, the program crashed because it tried to access a list index that does not exist, which caused an IndexError. The AI tool identified this problem and explained that list indices must be within the valid range. To fix it, the AI suggested safe access methods such as checking the list length before accessing an index or using a try-except block to handle the error. After applying these changes, the program ran safely without crashing.