AI Assisted Coding Lab 7.3

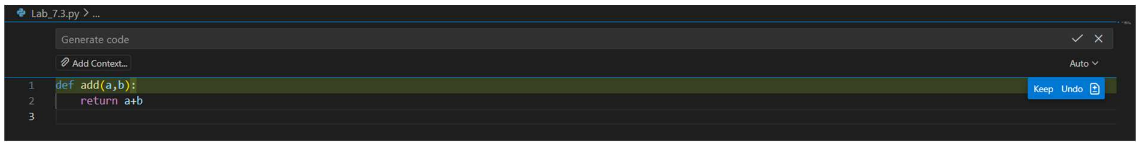
Veekshitha Adharasani

2303A52175

Batch – 41

Question 1: Fixing Syntax Errors

Fix:



Lab_7.3.py > ...

Generate code

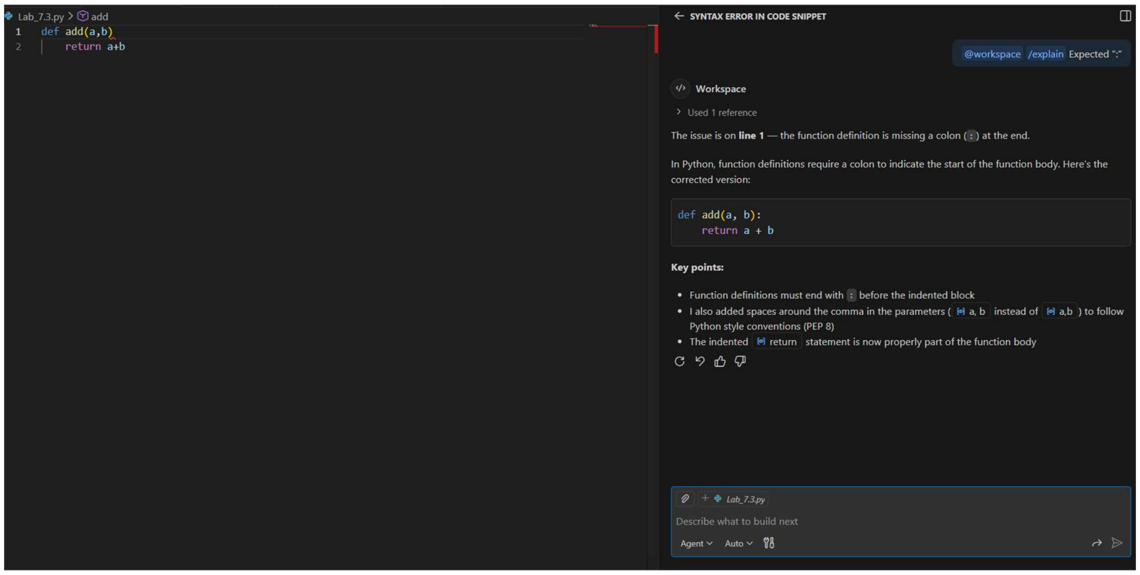
Add Context...

Auto

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

Keep Undo

Review:



Lab_7.3.py > add

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

SYNTAX ERROR IN CODE SNIPPET

@workspace /explain Expected ":"

Workspace

Used 1 reference

The issue is on **line 1** — the function definition is missing a colon (:) at the end.

In Python, function definitions require a colon to indicate the start of the function body. Here's the corrected version:

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

Key points:

- Function definitions must end with : before the indented block
- I also added spaces around the comma in the parameters (a, b instead of a,b) to follow Python style conventions (PEP 8)
- The indented return statement is now properly part of the function body

Lab_7.3.py

Describe what to build next

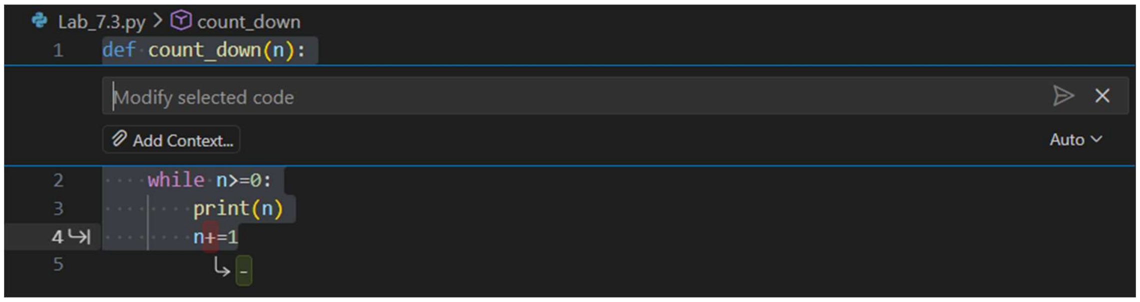
Agent Auto

JUSTIFICATION:

The syntax error occurred because the function definition was missing a colon at the end of the header line. In Python, a colon is mandatory to indicate the start of an indented block that forms the function body. The AI tool automatically detected the issue and suggested adding the colon in the correct place. After fixing it, the function executed normally and correctly returned the sum of the two inputs.

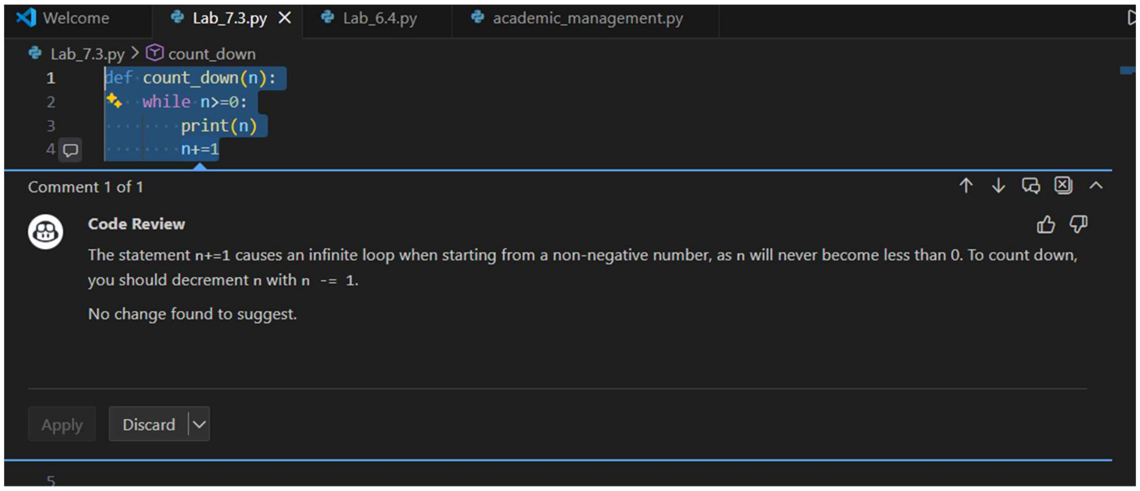
Question 2: Debugging Logic Errors in Loops

Fix:



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

Review:



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

The statement n+=1 causes an infinite loop when starting from a non-negative number, as n will never become less than 0. To count down, you should decrement n with n -= 1.

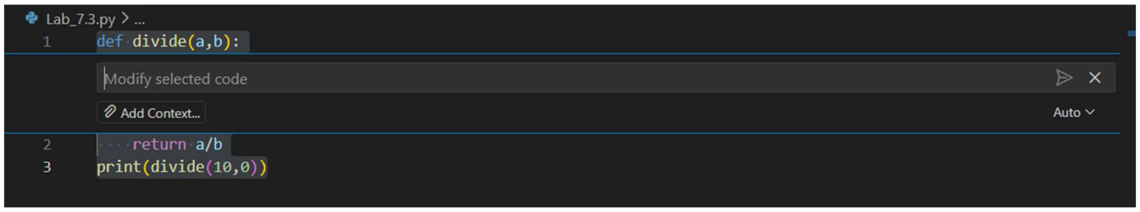
No change found to suggest.

JUSTIFICATION:

The infinite loop occurred because the variable n was updated using n += 1, which increased the value instead of decreasing it. Since the loop condition checks n >= 0, increasing n prevents it from ever becoming negative, so the loop never stops. This logical mistake caused continuous execution. The AI identified the incorrect increment and suggested replacing it with n -= 1 to properly decrease the value. After the fix, the loop counts down step-by-step and terminates correctly when the condition becomes false.

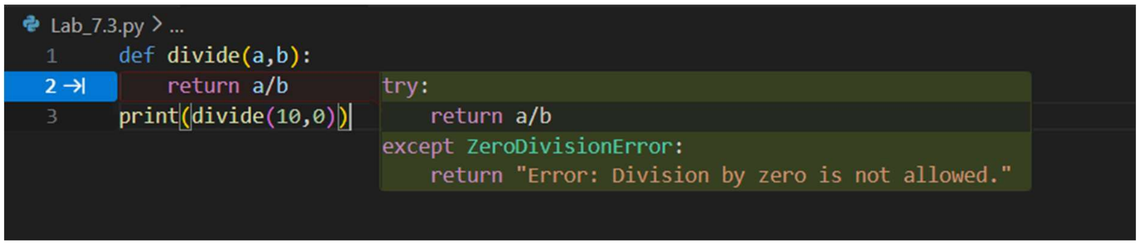
Question 3: Handling Runtime Errors (Division by Zero)

Fix:



Lab_7.3.py > ...

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

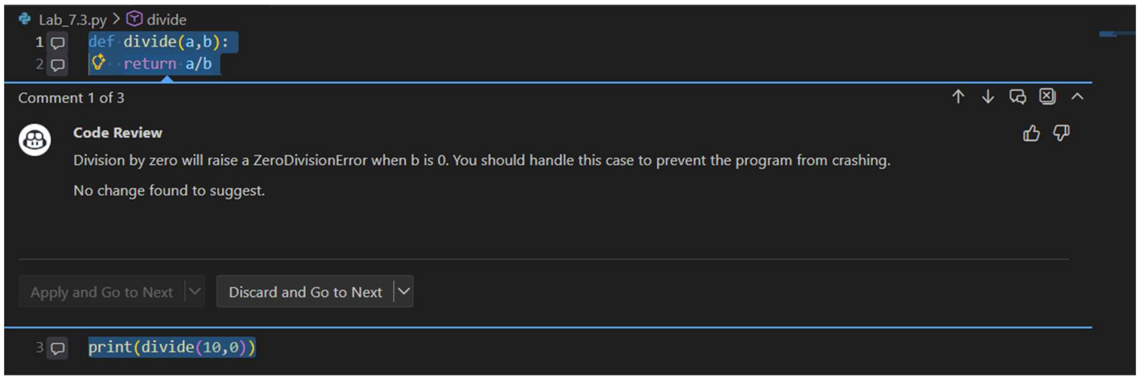


Lab_7.3.py > ...

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

```
try:
    return a/b
except ZeroDivisionError:
    return "Error: Division by zero is not allowed."
```

Review:



Lab_7.3.py > divide

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

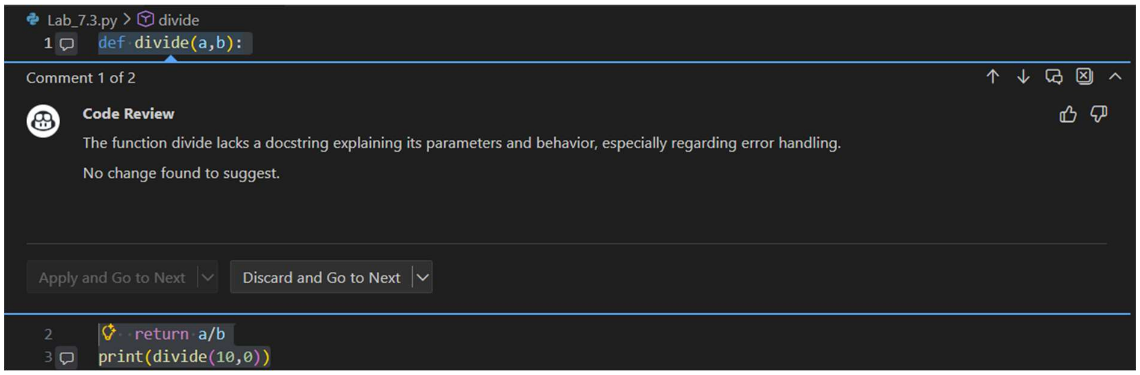
Comment 1 of 3

Code Review

Division by zero will raise a ZeroDivisionError when b is 0. You should handle this case to prevent the program from crashing.

No change found to suggest.

```
print(divide(10,0))
```



Lab_7.3.py > divide

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

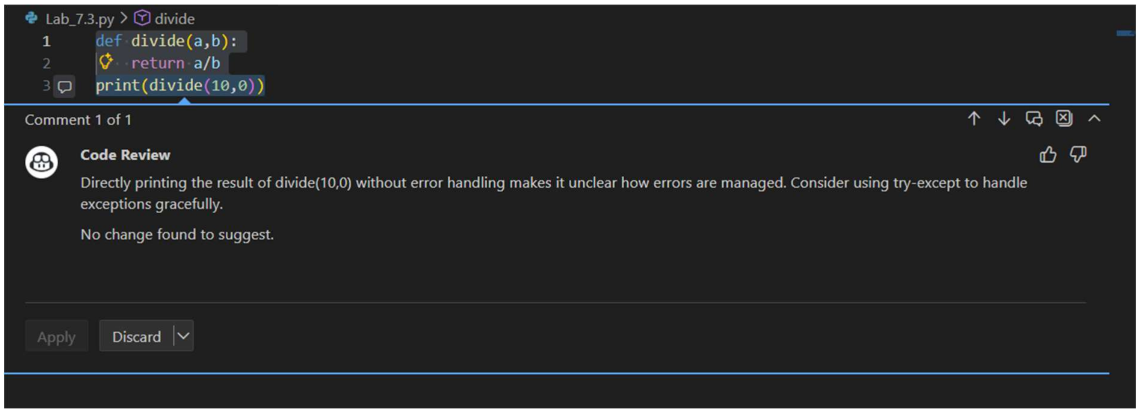
Comment 1 of 2

Code Review

The function divide lacks a docstring explaining its parameters and behavior, especially regarding error handling.

No change found to suggest.

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



Lab_7.3.py > divide

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

Comment 1 of 1

Code Review

Directly printing the result of divide(10,0) without error handling makes it unclear how errors are managed. Consider using try-except to handle exceptions gracefully.

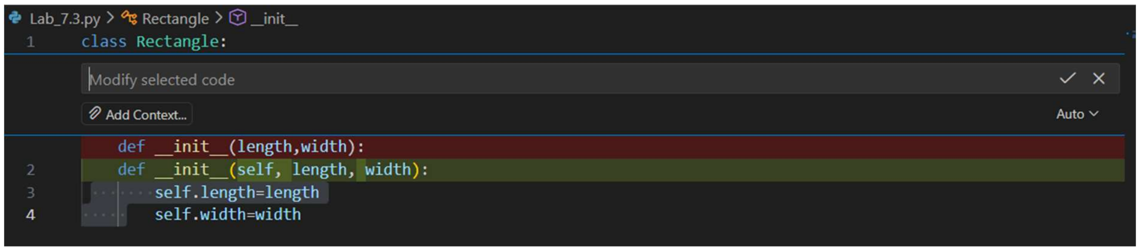
No change found to suggest.

JUSTIFICATION:

The original function directly divided two numbers without checking if the denominator was zero, which caused a runtime ZeroDivisionError and crashed the program. The AI detected this unsafe operation and recommended adding a try-except block to handle the exception. The try block attempts the division, while the except block catches the specific error and returns a friendly message. This prevents the program from terminating unexpectedly. After the fix, the function executes safely even when division by zero occurs.

Question 4: Debugging Class Definition Errors

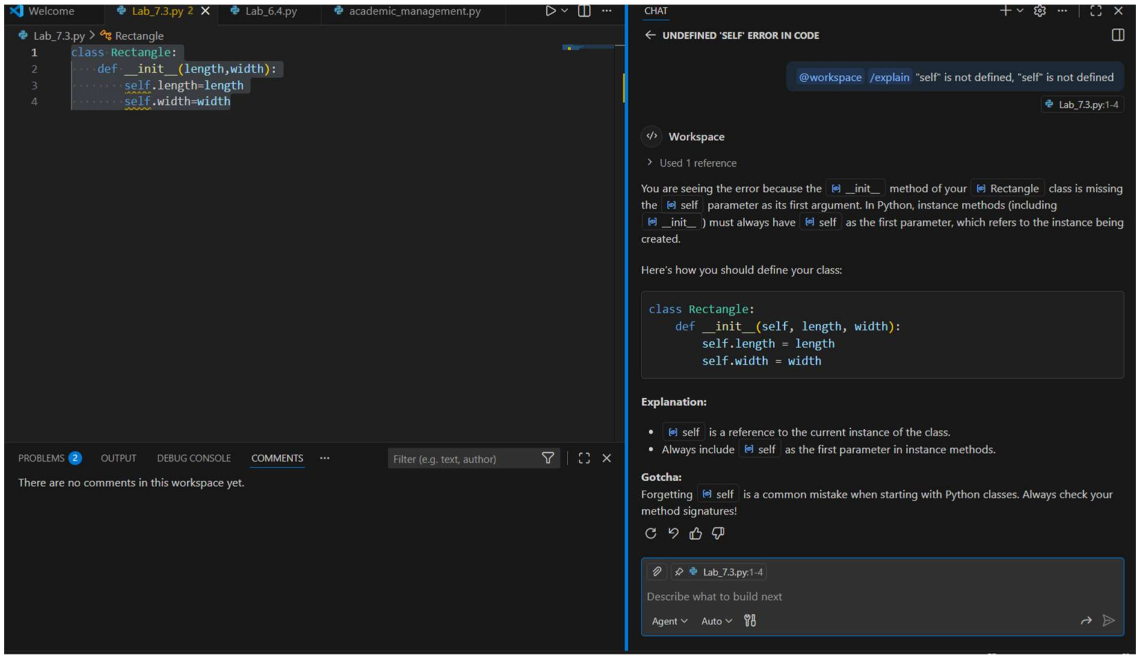
Fix:



Lab_7.3.py > Rectangle > __init__

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

Modify:



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

There are no comments in this workspace yet.

@workspace /explain "self" is not defined, "self" is not defined

You are seeing the error because the __init__ method of your Rectangle class is missing the self parameter as its first argument. In Python, instance methods (including __init__) must always have self as the first parameter, which refers to the instance being created.

Here's how you should define your class:

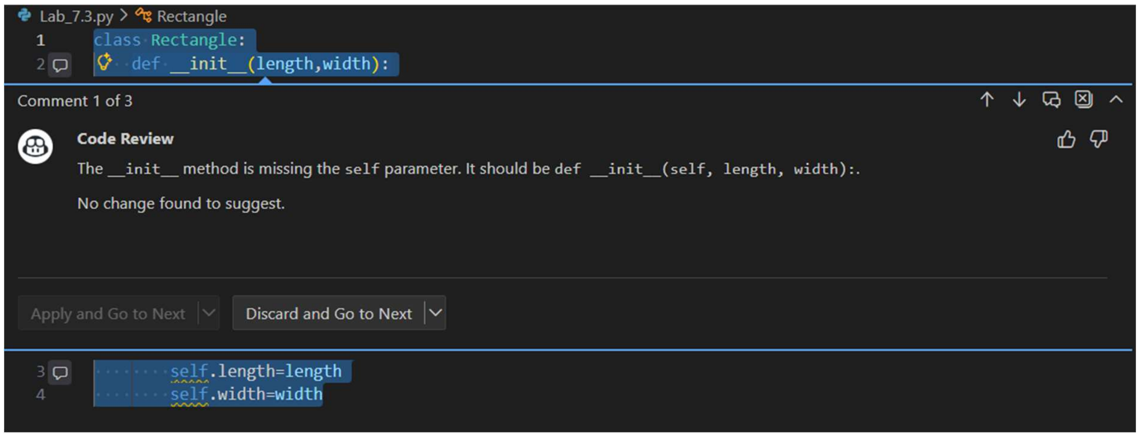
```python
class Rectangle:
    def __init__(self, length, width):
        self.length = length
        self.width = width
```

Explanation:

- self is a reference to the current instance of the class.
- Always include self as the first parameter in instance methods.

Gotcha:
Forgetting self is a common mistake when starting with Python classes. Always check your method signatures!

Review:



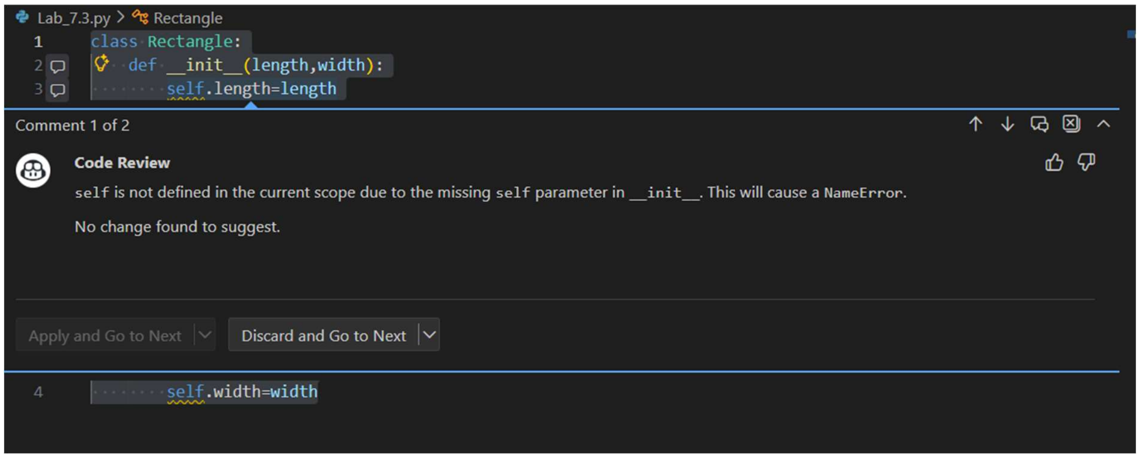
```python
class Rectangle:
    def __init__(length,width):
```

Code Review

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

No change found to suggest.

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



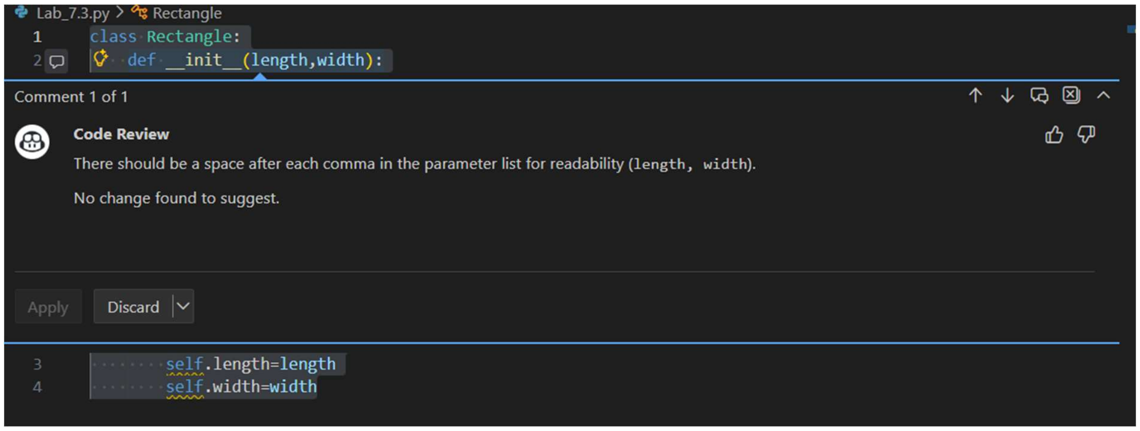
```python
class Rectangle:
    def __init__(length,width):
        self.length=length
```

Code Review

`self` is not defined in the current scope due to the missing `self` parameter in `__init__`. This will cause a `NameError`.

No change found to suggest.

```python
        self.width=width
```



Lab_7.3.py > Rectangle

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

Comment 1 of 1

Code Review

There should be a space after each comma in the parameter list for readability (`length, width`).

No change found to suggest.

Apply | Discard

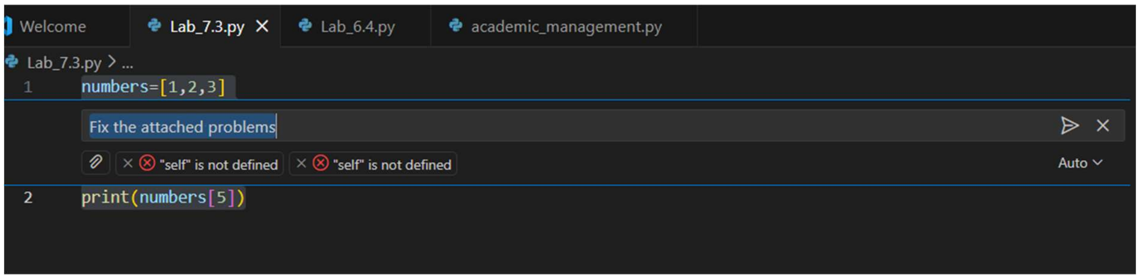
```
        self.length=length
        self.width=width
```

JUSTIFICATION:

The original constructor missed the self parameter, which caused self to be undefined inside the method and resulted in an error. In Python, self refers to the current object instance and is required in all instance methods. Without it, attributes cannot be assigned to the object properly. The AI detected this issue and corrected the method signature by adding self as the first parameter. After the fix, the class initializes objects correctly and stores the rectangle's dimensions without errors.

Question 5: Resolving Index Errors in Lists

Fix:



Welcome | Lab_7.3.py | Lab_6.4.py | academic_management.py

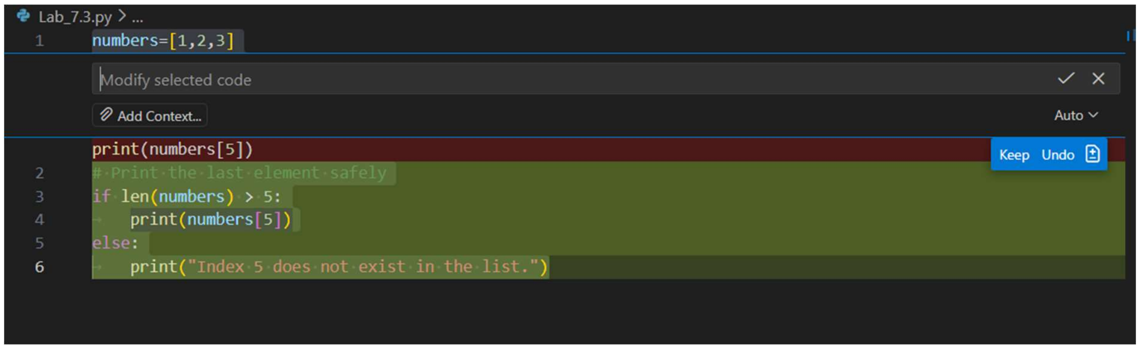
Lab_7.3.py > ...

```
numbers=[1,2,3]
```

Fix the attached problems

"self" is not defined | "self" is not defined | Auto

```
print(numbers[5])
```



Lab_7.3.py > ...

```
numbers=[1,2,3]
```

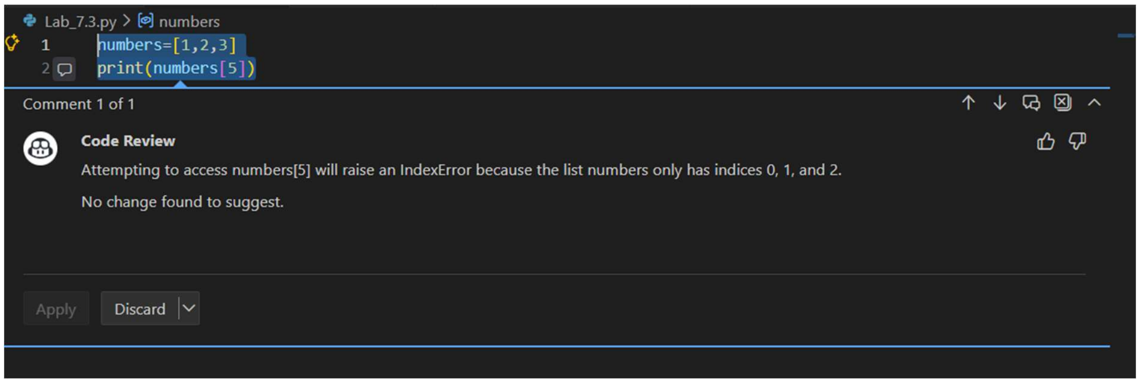
Modify selected code

Add Context... | Auto

```
print(numbers[5])
# Print the last element safely
if len(numbers) > 5:
    print(numbers[5])
else:
    print("Index 5 does not exist in the list.")
```

Keep Undo

Review:



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

Code Review

Attempting to access numbers[5] will raise an IndexError because the list numbers only has indices 0, 1, and 2.

No change found to suggest.

JUSTIFICATION:

The original code attempted to access numbers[5], but the list contains only three elements, so valid indices are 0 to 2. This caused an index error at runtime because the position was out of range. The AI detected this unsafe access and recommended checking the list length before indexing. By using a bounds check with len(numbers), the program safely verifies whether the index exists. After applying the fix, the program handles invalid indices gracefully without crashing.