

## AI ASSISTED CODING

### Lab assignment-7.3

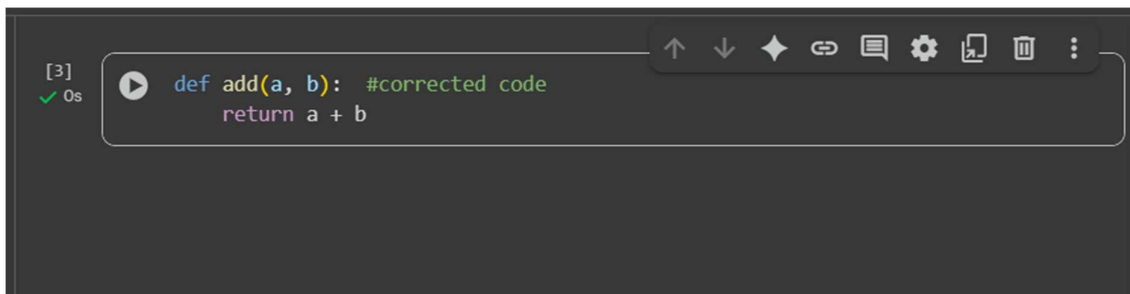
#### Task1:

Paste a function with a missing colon (add(a, b)), and let AI fix the syntax error.

```
python

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

Code output:



The screenshot shows a code editor interface with a dark background. On the left, there is a status bar indicating '[3] 0s' with a green checkmark. The main editor area displays the corrected Python code: `def add(a, b): #corrected code` followed by  `return a + b` on the next line. The code is color-coded: `def` is blue, `add` is green, `(a, b):` is green, `#corrected code` is green, `return` is blue, and `a + b` is green. A toolbar with various icons (up, down, search, link, chat, settings, copy, paste, delete, and more) is visible at the top right of the editor area.

Code explanation:

The error message `SyntaxError: expected ':'` indicates that there is a missing colon after the function definition `def add(a, b)`. Additionally, the `return` statement needs to be indented to be part of the function body. I will fix these errors in the code.

I've fixed the syntax error and the indentation in the `add` function. You can now use this function to add two numbers. Would you like to test it with some examples? Or perhaps you have another task in mind?

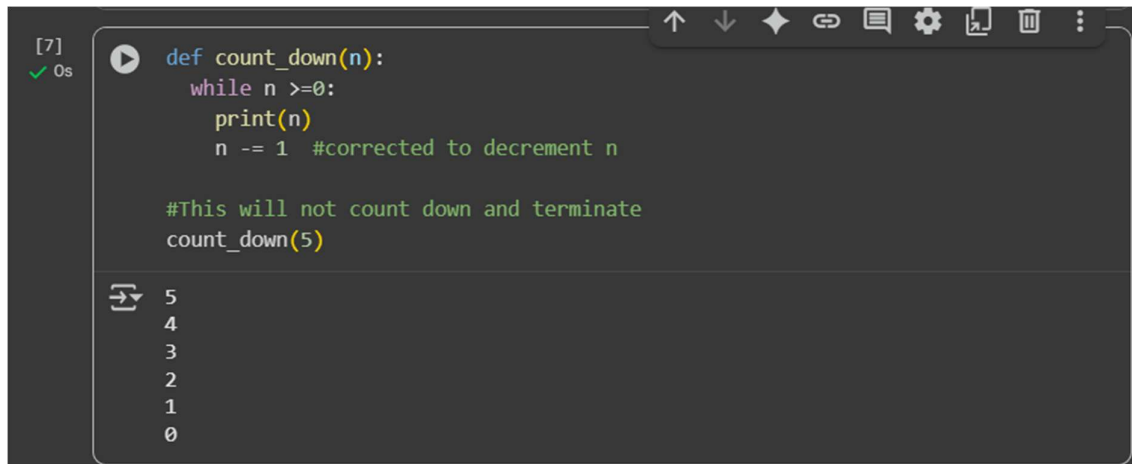
#### Task2:

- Identify and fix a logic error in a loop that causes infinite iteration.

```
python

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

Code output:



The screenshot shows a Jupyter Notebook interface. The top part is a code editor with the following Python code:

```
[7] ✓ 0s def count_down(n):
        while n >= 0:
            print(n)
            n -= 1 #corrected to decrement n

        #This will not count down and terminate
        count_down(5)
```

The bottom part of the notebook cell shows the output of the code, which is a vertical list of numbers from 5 down to 0:

```
5
4
3
2
1
0
```

Code explanation:

Certainly! I can explain the code you provided.

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

This code defines a function called `count_down` that takes one argument, `n`.

- `def count_down(n):`: This line defines a function named `count_down` that accepts a single parameter `n`.
- `while n >= 0:`: This is a while loop that continues to execute as long as the value of `n` is greater than or equal to 0.
- `print(n)`: Inside the loop, this line prints the current value of `n` to the console.
- `n -= 1`: This line decrements the value of `n` by 1 in each iteration of the loop. This is what makes the number count down.

```
#This will not count down and terminate
count_down(5)
```

This part of the code calls the `count_down` function with the argument 5. So, the function will start counting down from 5, printing each number until it reaches 0.

The comment `#This will not count down and terminate` seems to be a remnant from a previous version of the code, as the current code **does** count down and terminate when `n` becomes less than 0.

0 / 2000

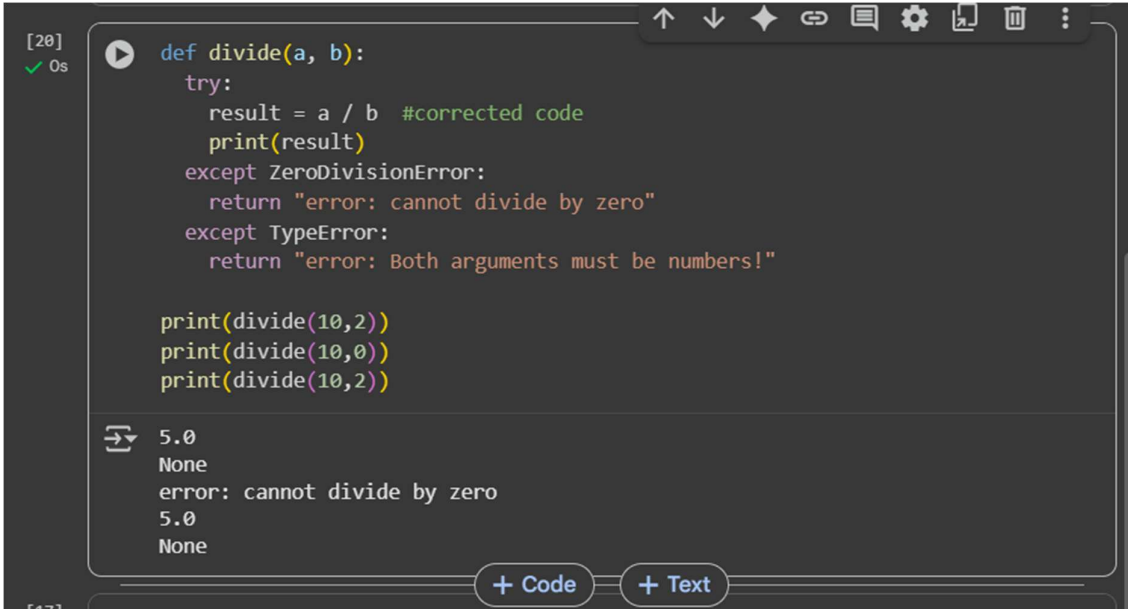
### Task3:

Debug a runtime error caused by division by zero. Let AI insert try-except.

```
# Debug the following code
def divide(a, b):
    return a / b

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

### Code output:



The screenshot shows a code editor with a dark theme. At the top, there's a toolbar with icons for undo, redo, search, and other editor functions. Below the toolbar, the code is as follows:

```
[20] ✓ 0s
def divide(a, b):
    try:
        result = a / b #corrected code
        print(result)
    except ZeroDivisionError:
        return "error: cannot divide by zero"
    except TypeError:
        return "error: Both arguments must be numbers!"

print(divide(10,2))
print(divide(10,0))
print(divide(10,2))
```

Below the code, the output is displayed:

```
5.0
None
error: cannot divide by zero
5.0
None
```

At the bottom of the editor, there are two buttons: "+ Code" and "+ Text".

### Code explanation:

The error in the selected cell is a `ZeroDivisionError`, which occurs when you try to divide a number by zero. This is mathematically undefined and therefore causes an error in programming. I will modify the code to include a check for division by zero and print an error message if it occurs.

I have successfully modified the divide function to handle division by zero by adding a check. You can now test the function with different inputs, including cases where `b` is not zero, to see the correct division results.

0 / 2000

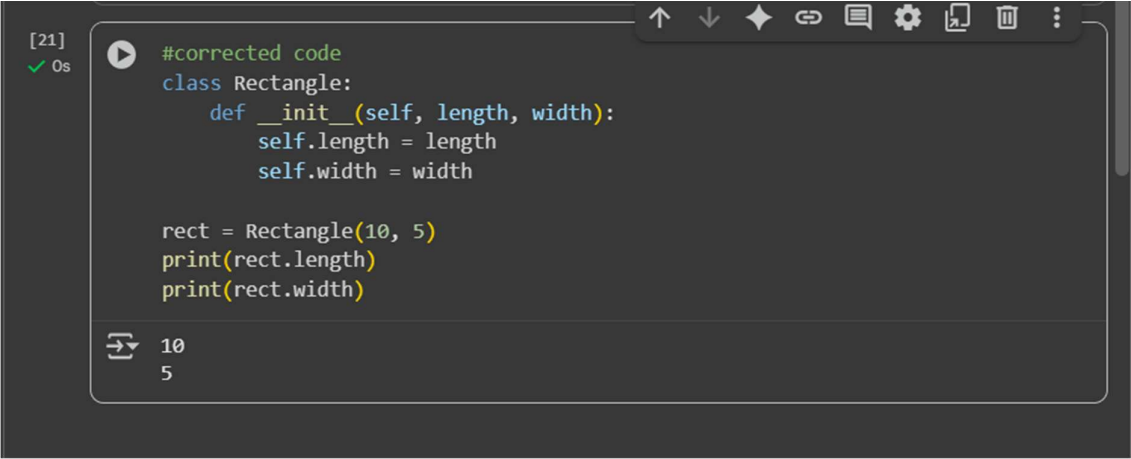
#### Task4:

Provide a faulty class definition (missing `self` in parameters). Let AI fix it

```
python

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

#### Code output:



```
[21]
✓ Os
#corrected code
class Rectangle:
    def __init__(self, length, width):
        self.length = length
        self.width = width

rect = Rectangle(10, 5)
print(rect.length)
print(rect.width)
```

10  
5

#### Code explanation:

I can explain this code for you.

This code defines a simple class called `Rectangle` and then creates an object (an instance) of that class.

```
class Rectangle:
```

```
    def __init__(self, length, width):
```

```
        self.length = length
```

```
        self.width = width
```

- **class Rectangle::** This line defines a new class named Rectangle. Classes are blueprints for creating objects that have specific properties (attributes) and behaviors (methods).
- **def \_\_init\_\_(self, length, width)::** This is a special method called the constructor. It's automatically called when you create a new object of the Rectangle class.
  - **self:** This refers to the instance of the class being created. It's a convention to use self as the first parameter.
  - **length and width:** These are parameters that you pass when you create a Rectangle object.
  - **self.length = length:** This line takes the value passed for the length parameter and assigns it to an attribute of the object called length.
  - **self.width = width:** Similarly, this line takes the value passed for the width parameter and assigns it to an attribute of the object called width.

In essence, the `__init__` method initializes the attributes of a Rectangle object when it's created.

```
rect = Rectangle(10, 5)
```

```
print(rect.length)
```

```
print(rect.width)
```

- **rect = Rectangle(10, 5):** This line creates a new instance of the Rectangle class. It calls the `__init__` method with 10 for length and 5 for width. The newly created Rectangle object is assigned to the variable rect.
- **print(rect.length):** This line accesses the length attribute of the rect object and prints its value (which is 10).
- **print(rect.width):** This line accesses the width attribute of the rect object and prints its value (which is 5).

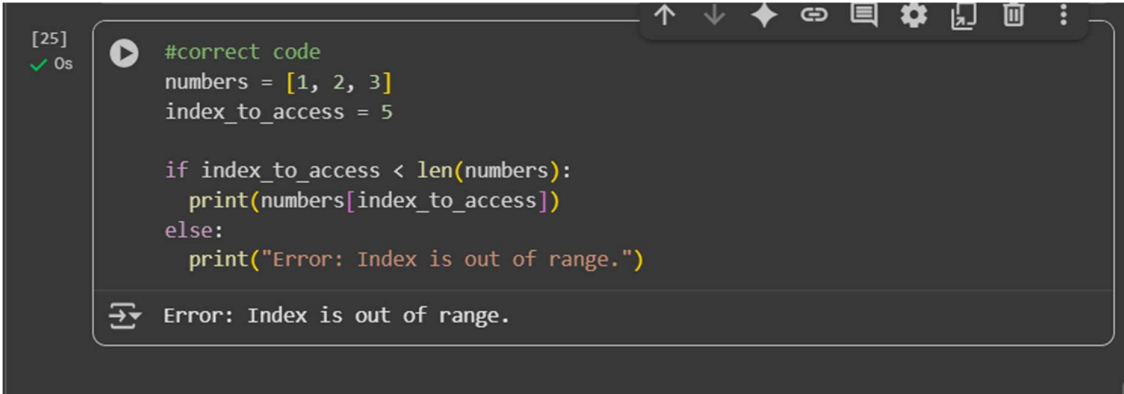
So, the code defines what a Rectangle is (it has a length and a width) and then creates a specific rectangle with a length of 10 and a width of 5, and finally prints those dimensions

#### Task5:

Access an invalid list index and use AI to resolve the Index Error.

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

#### Code output:



```
[25]  
✓ 0s  
#correct code  
numbers = [1, 2, 3]  
index_to_access = 5  
  
if index_to_access < len(numbers):  
    print(numbers[index_to_access])  
else:  
    print("Error: Index is out of range.")  
  
Error: Index is out of range.
```

#### Code explanation:

The code in the selected cell has two errors. First, there's a `SyntaxError: expected ':'` because the `else` statement is missing a colon at the end. Second, even if the syntax error is fixed, there will be an `IndexError: list index out of range` because you are trying to access an index (5) that is outside the valid range of the `numbers` list (which only has indices 0, 1, and 2).

I will fix the syntax error by adding a colon to the `else` statement and correct the code to check if the `index_to_access` is within the valid range of the list before attempting to access it.

I have fixed the `SyntaxError` by adding the missing colon to the `else` statement and added a check to prevent the `IndexError`. The code now correctly identifies when the index is out of the list's range.

**You can now try changing the value of `index_to_access` to a number between 0 and 2 (inclusive) to see the element at that index being printed. You could also try modifying the numbers list.**