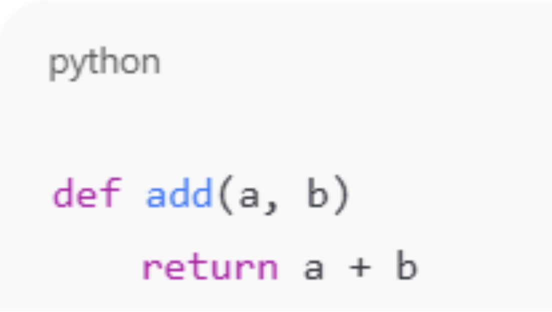
ASSIGNMENT

Task Description#1

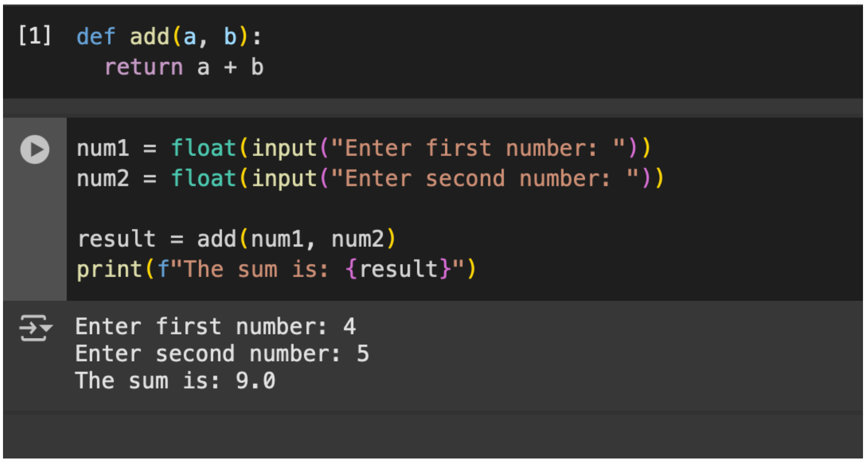
- Paste a function with a missing colon (add(a, b)), and let AI fix the syntax error.



python

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

Expected output with fixed errors given by google colab :

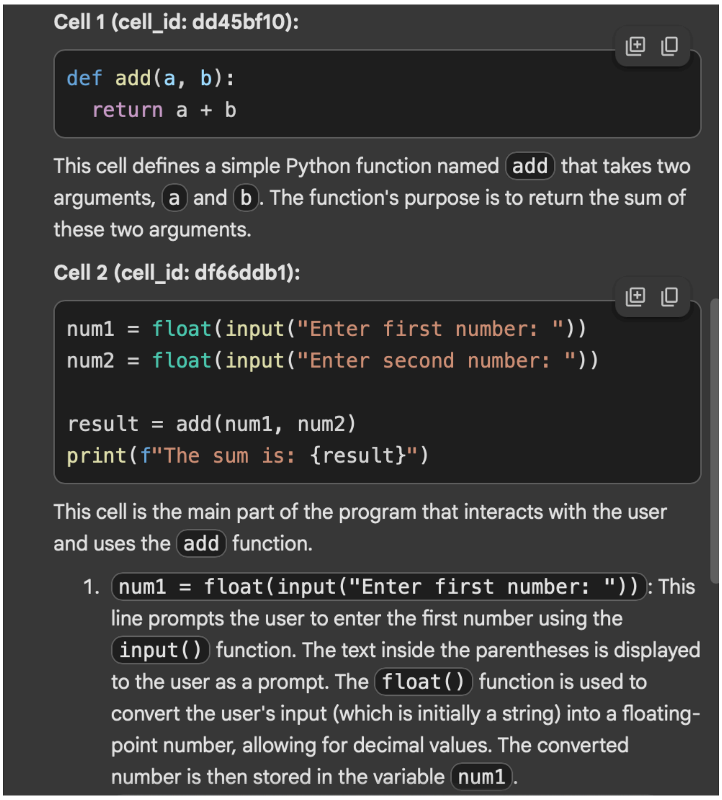


```python
[1] def add(a, b):
      return a + b
```

```python
num1 = float(input("Enter first number: "))
num2 = float(input("Enter second number: "))

result = add(num1, num2)
print(f"The sum is: {result}")
```

```
Enter first number: 4
Enter second number: 5
The sum is: 9.0
```

explanation of the code :



Cell 1 (cell_id: dd45bf10):

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

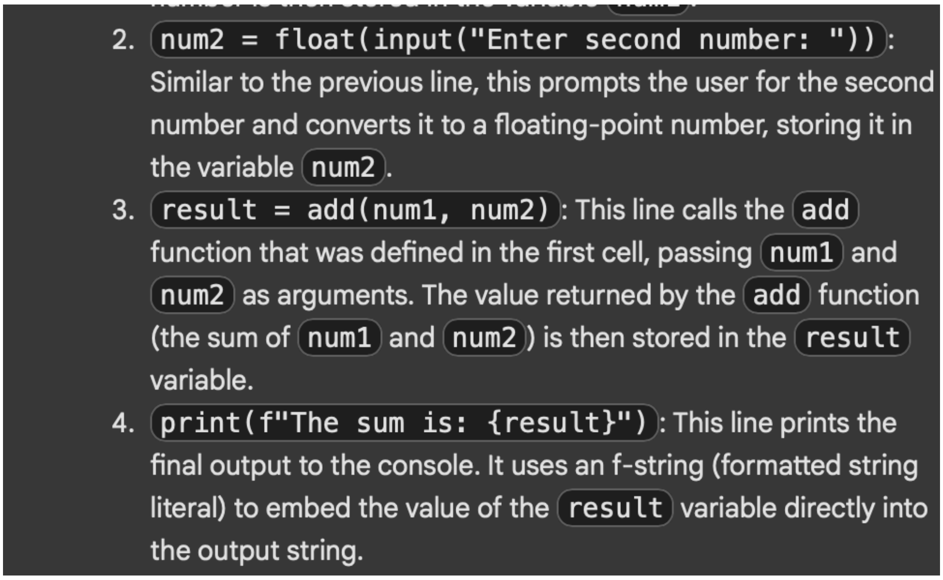
This cell defines a simple Python function named `add` that takes two arguments, `a` and `b`. The function's purpose is to return the sum of these two arguments.

Cell 2 (cell_id: df66ddb1):

```python
num1 = float(input("Enter first number: "))
num2 = float(input("Enter second number: "))

result = add(num1, num2)
print(f"The sum is: {result}")
```

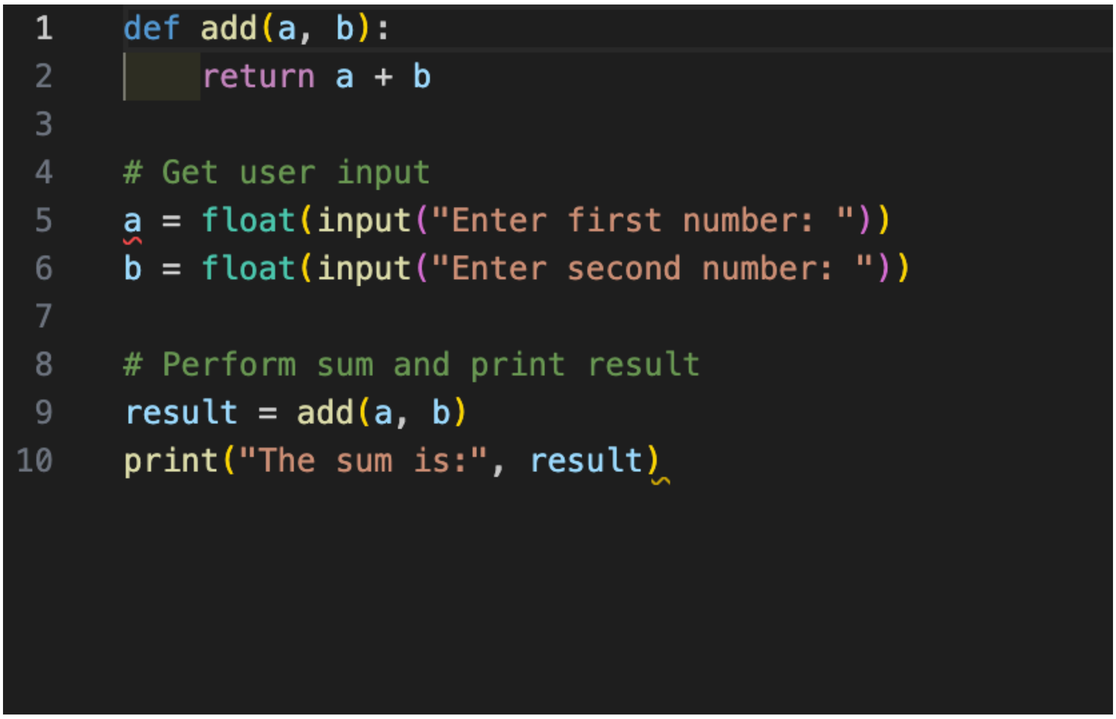
This cell is the main part of the program that interacts with the user and uses the `add` function.

1. `num1 = float(input("Enter first number: "))`: This line prompts the user to enter the first number using the `input()` function. The text inside the parentheses is displayed to the user as a prompt. The `float()` function is used to convert the user's input (which is initially a string) into a floating-point number, allowing for decimal values. The converted number is then stored in the variable `num1`.

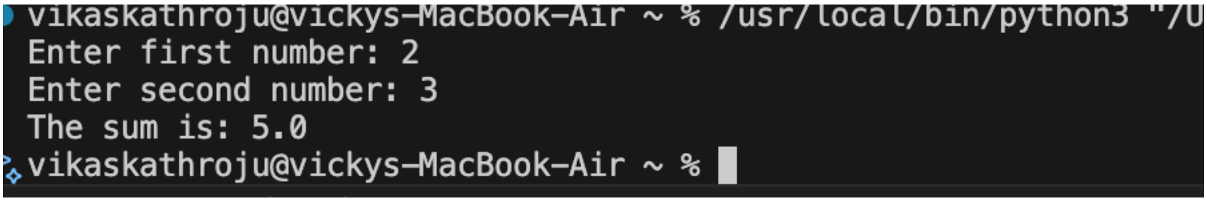


2. `num2 = float(input("Enter second number: "))`: Similar to the previous line, this prompts the user for the second number and converts it to a floating-point number, storing it in the variable `num2`.
3. `result = add(num1, num2)`: This line calls the `add` function that was defined in the first cell, passing `num1` and `num2` as arguments. The value returned by the `add` function (the sum of `num1` and `num2`) is then stored in the `result` variable.
4. `print(f"The sum is: {result}")`: This line prints the final output to the console. It uses an f-string (formatted string literal) to embed the value of the `result` variable directly into the output string.

CODE AND OUPUT PROVIDED BY GITHUB COPILOT :



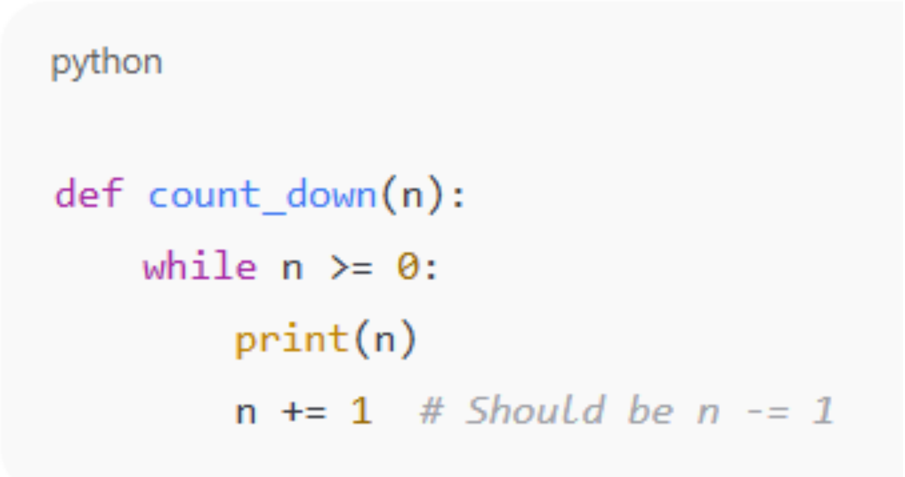
```python
def add(a, b):
    return a + b

# Get user input
a = float(input("Enter first number: "))
b = float(input("Enter second number: "))

# Perform sum and print result
result = add(a, b)
print("The sum is:", result)
```



```
vikaskathroju@vickys-MacBook-Air ~ % /usr/local/bin/python3 "/U
Enter first number: 2
Enter second number: 3
The sum is: 5.0
vikaskathroju@vickys-MacBook-Air ~ % 
```

Task Description 2

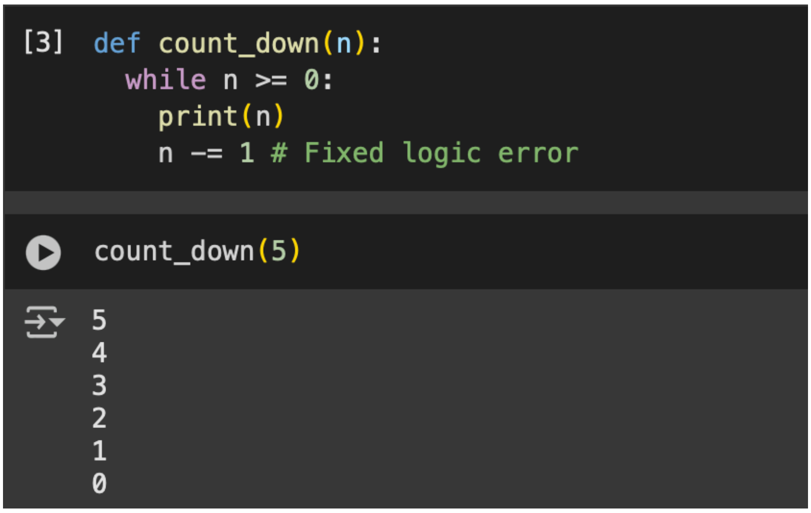
- **Identify and fix a logic error in a loop that causes infinite iteration.**



python

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

EXPECTED CODE AND OUTPUT WITH FIXED ERRORS IN GOOGLE COLAB :

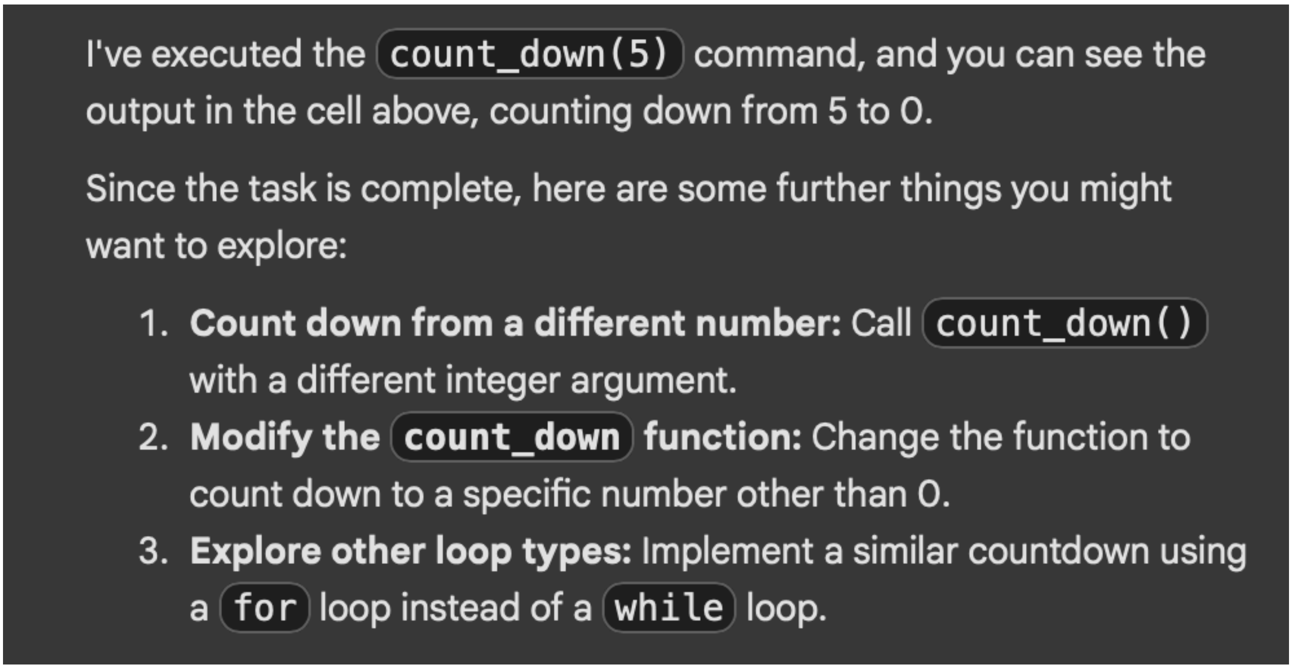


```
[3] def count_down(n):
      while n >= 0:
        print(n)
        n -= 1 # Fixed logic error
```

```
count_down(5)
```

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

EXPLANATION OF THE CODE PROVIDED BY THE GOOGLE COLAB :

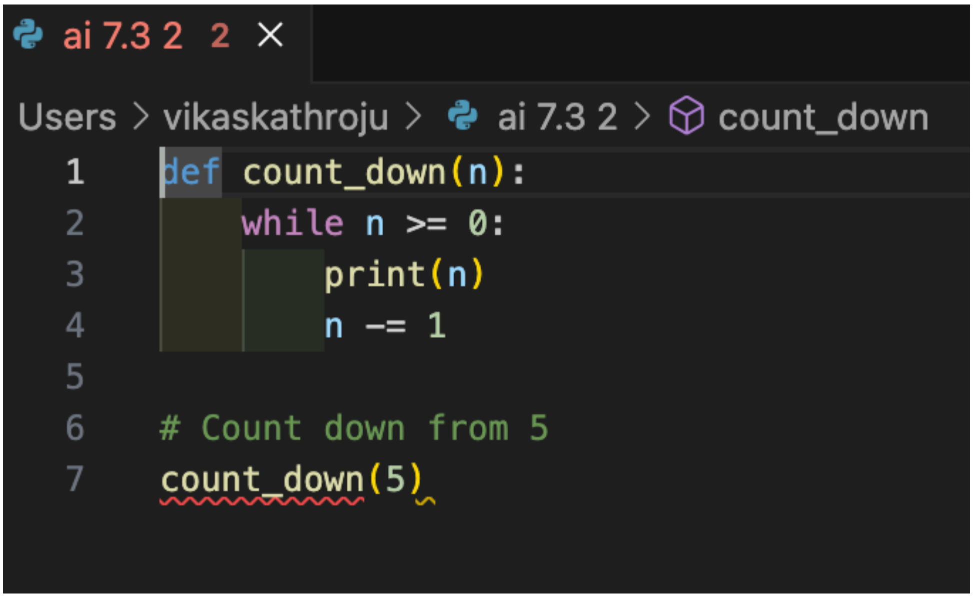


I've executed the `count_down(5)` command, and you can see the output in the cell above, counting down from 5 to 0.

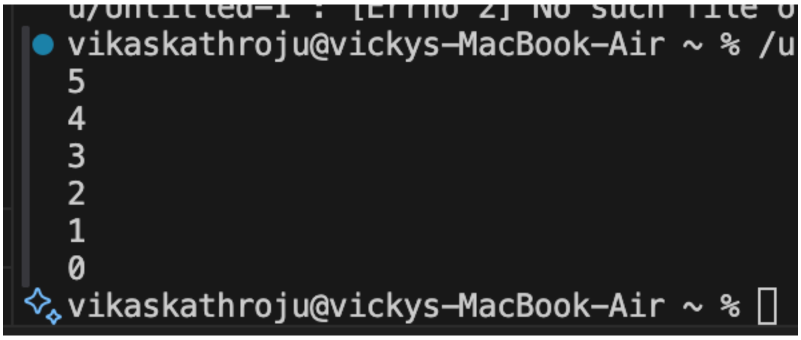
Since the task is complete, here are some further things you might want to explore:

1. **Count down from a different number:** Call `count_down()` with a different integer argument.
2. **Modify the `count_down` function:** Change the function to count down to a specific number other than 0.
3. **Explore other loop types:** Implement a similar countdown using a `for` loop instead of a `while` loop.

EXPECTED CODE AND OUTPUT PROVIDED BY GITHUB COPILOT :



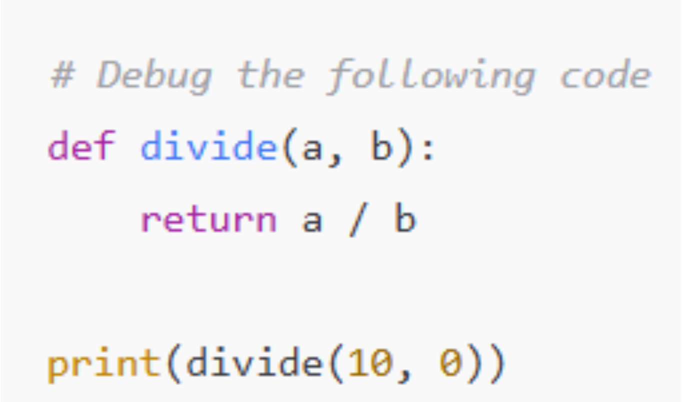
```python
def count_down(n):
    while n >= 0:
        print(n)
        n -= 1

# Count down from 5
count_down(5)
```



```
u/Untitled-1 : [Errno 2] No such file o
vikaskathroju@vickys-MacBook-Air ~ % /u
5
4
3
2
1
0
vikaskathroju@vickys-MacBook-Air ~ %
```

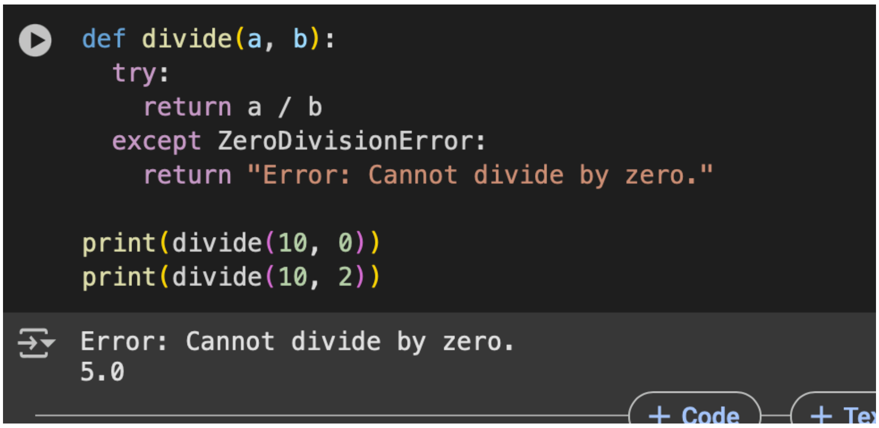
Task Description 3

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



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

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

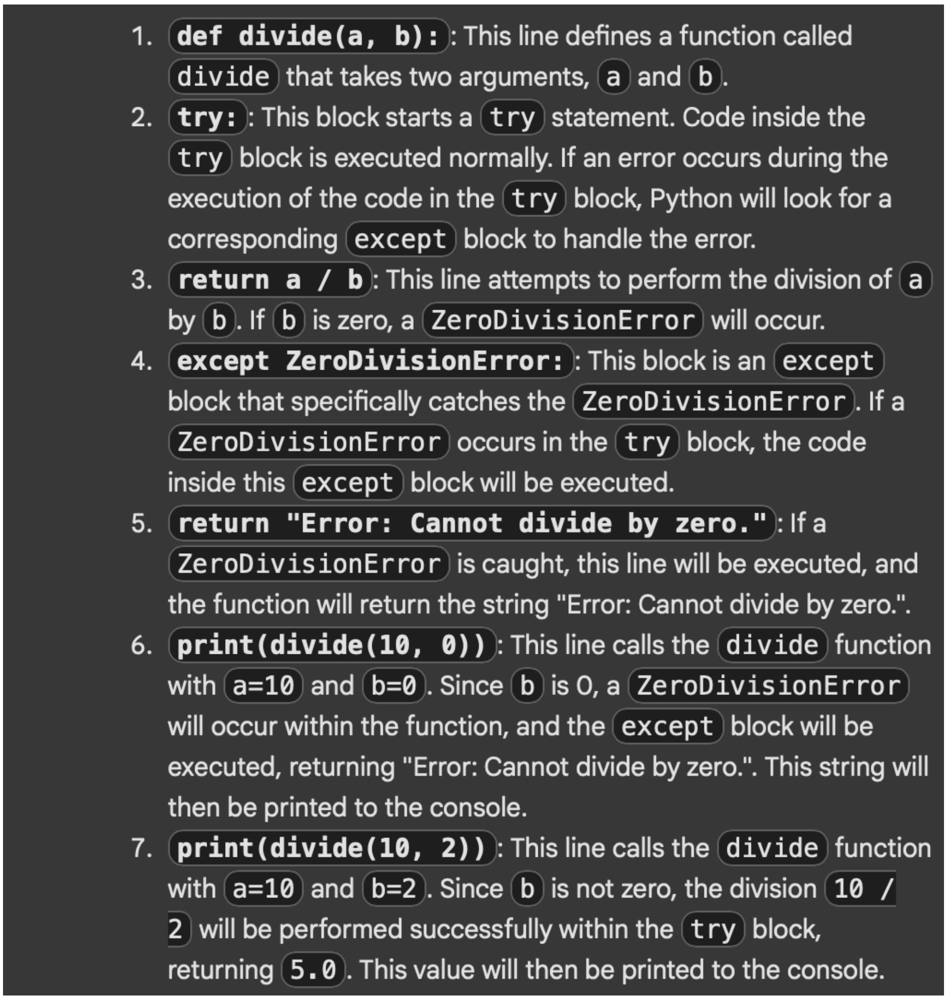
EXPECTED CODE AND OUTPUT GIVEN BY THE GOOGLE COLAB :



```
def divide(a, b):
  try:
    return a / b
  except ZeroDivisionError:
    return "Error: Cannot divide by zero."

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

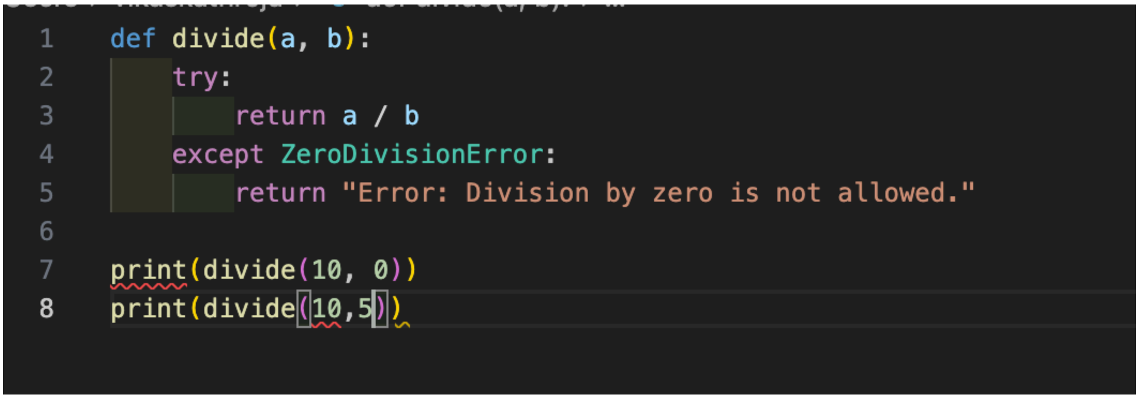
```
Error: Cannot divide by zero.
5.0
```

explanation fo the code given by the google colab :

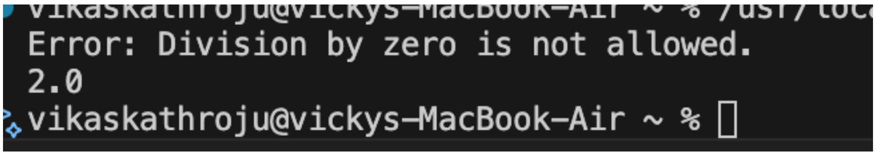


1. **`def divide(a, b):`**: This line defines a function called `divide` that takes two arguments, `a` and `b`.
2. **`try:`**: This block starts a `try` statement. Code inside the `try` block is executed normally. If an error occurs during the execution of the code in the `try` block, Python will look for a corresponding `except` block to handle the error.
3. **`return a / b`**: This line attempts to perform the division of `a` by `b`. If `b` is zero, a `ZeroDivisionError` will occur.
4. **`except ZeroDivisionError:`**: This block is an `except` block that specifically catches the `ZeroDivisionError`. If a `ZeroDivisionError` occurs in the `try` block, the code inside this `except` block will be executed.
5. **`return "Error: Cannot divide by zero."`**: If a `ZeroDivisionError` is caught, this line will be executed, and the function will return the string "Error: Cannot divide by zero.".
6. **`print(divide(10, 0))`**: This line calls the `divide` function with `a=10` and `b=0`. Since `b` is 0, a `ZeroDivisionError` will occur within the function, and the `except` block will be executed, returning "Error: Cannot divide by zero.". This string will then be printed to the console.
7. **`print(divide(10, 2))`**: This line calls the `divide` function with `a=10` and `b=2`. Since `b` is not zero, the division `10 / 2` will be performed successfully within the `try` block, returning `5.0`. This value will then be printed to the console.

expected code and output given by the github copilot :



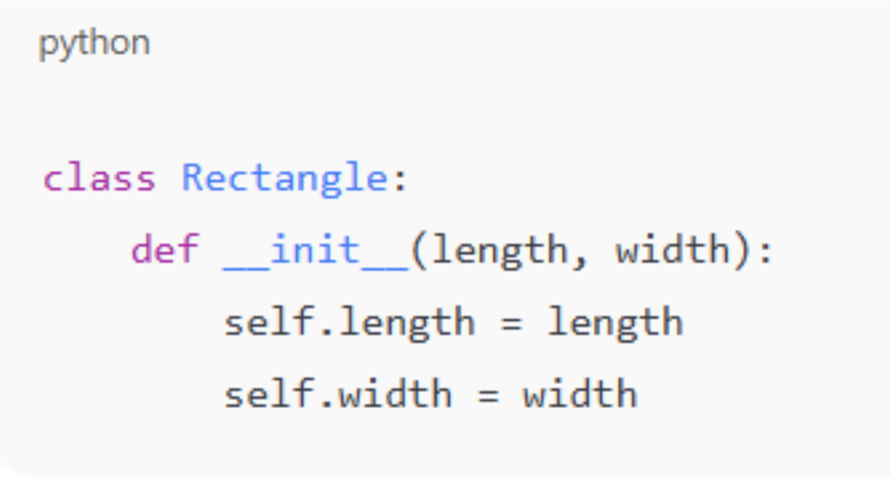
```python
def divide(a, b):
    try:
        return a / b
    except ZeroDivisionError:
        return "Error: Division by zero is not allowed."

print(divide(10, 0))
print(divide(10,5))
```



```
vikaskathroju@vickys-MacBook-Air ~ % /usr/loc
Error: Division by zero is not allowed.
2.0
vikaskathroju@vickys-MacBook-Air ~ % 
```

Task Description 4

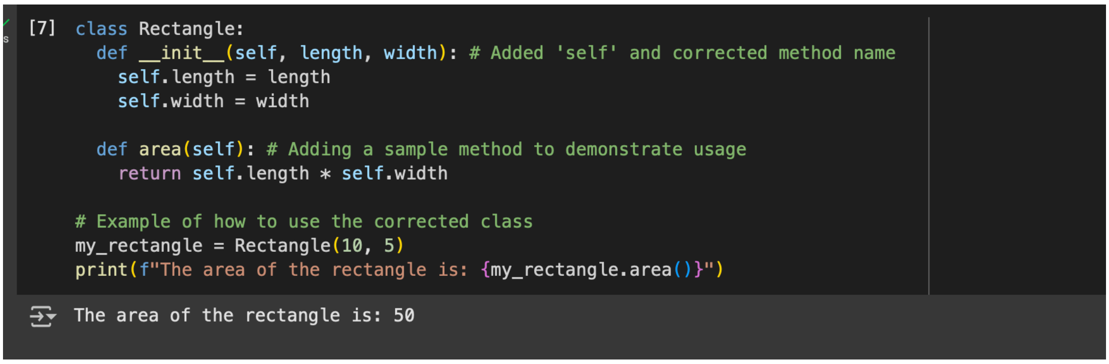
- **Provide a faulty class definition (missing self in parameters). Let AI fix it**



python

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

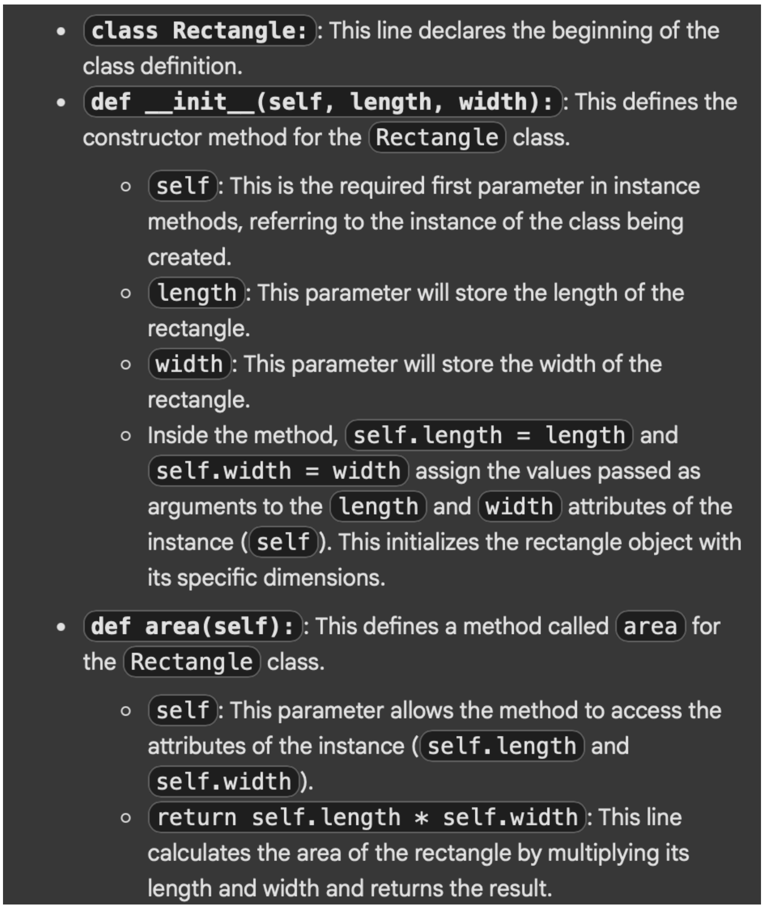
expected code and output provided by the google colab :



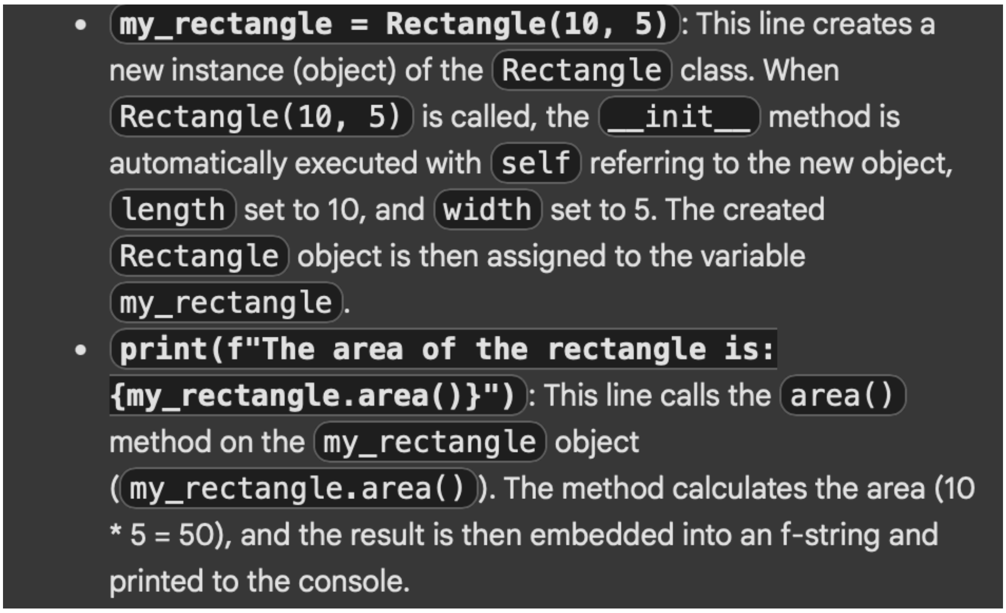
```python
[7] class Rectangle:
      def __init__(self, length, width): # Added 'self' and corrected method name
        self.length = length
        self.width = width

      def area(self): # Adding a sample method to demonstrate usage
        return self.length * self.width

    # Example of how to use the corrected class
    my_rectangle = Rectangle(10, 5)
    print(f"The area of the rectangle is: {my_rectangle.area()}")
```

```
The area of the rectangle is: 50
```

explanation of the code provide by the google colab :

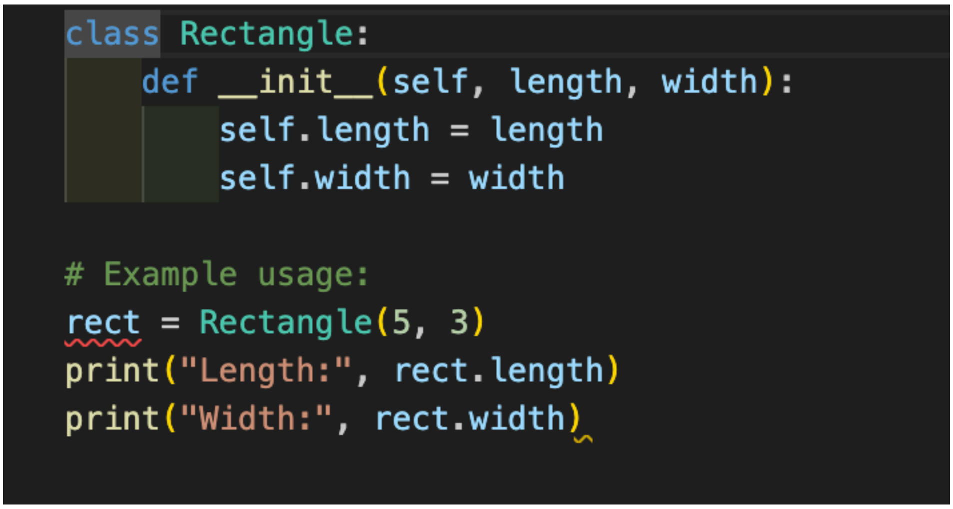


- **`class Rectangle:`**: This line declares the beginning of the class definition.
- **`def __init__(self, length, width):`**: This defines the constructor method for the `Rectangle` class.
    - `self`: This is the required first parameter in instance methods, referring to the instance of the class being created.
    - `length`: This parameter will store the length of the rectangle.
    - `width`: This parameter will store the width of the rectangle.
    - Inside the method, `self.length = length` and `self.width = width` assign the values passed as arguments to the `length` and `width` attributes of the instance (`self`). This initializes the rectangle object with its specific dimensions.
- **`def area(self):`**: This defines a method called `area` for the `Rectangle` class.
    - `self`: This parameter allows the method to access the attributes of the instance (`self.length` and `self.width`).
    - `return self.length * self.width`: This line calculates the area of the rectangle by multiplying its length and width and returns the result.

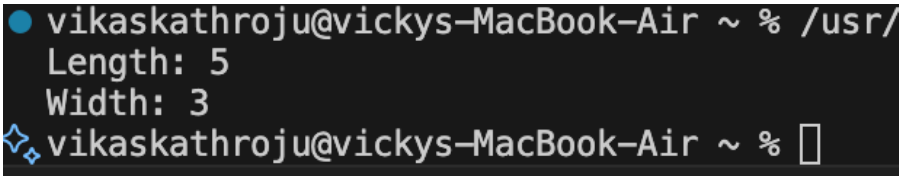


- **`my_rectangle = Rectangle(10, 5)`**: This line creates a new instance (object) of the `Rectangle` class. When `Rectangle(10, 5)` is called, the `__init__` method is automatically executed with `self` referring to the new object, `length` set to 10, and `width` set to 5. The created `Rectangle` object is then assigned to the variable `my_rectangle`.
- **`print(f"The area of the rectangle is: {my_rectangle.area()}")`**: This line calls the `area()` method on the `my_rectangle` object (`my_rectangle.area()`). The method calculates the area (10 * 5 = 50), and the result is then embedded into an f-string and printed to the console.

expected code and output given by github copilot :



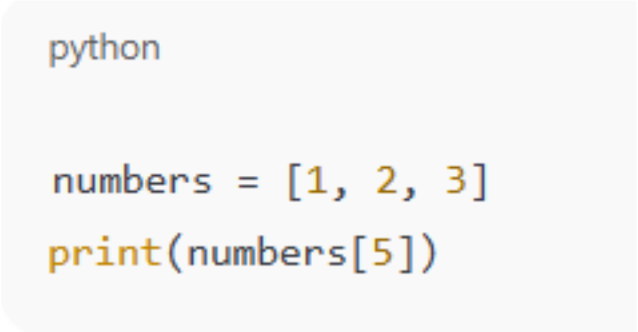
```
class Rectangle:
    def __init__(self, length, width):
        self.length = length
        self.width = width

# Example usage:
rect = Rectangle(5, 3)
print("Length:", rect.length)
print("Width:", rect.width)
```



```
vikaskathroju@vickys-MacBook-Air ~ % /usr/
Length: 5
Width: 3
vikaskathroju@vickys-MacBook-Air ~ % 
```

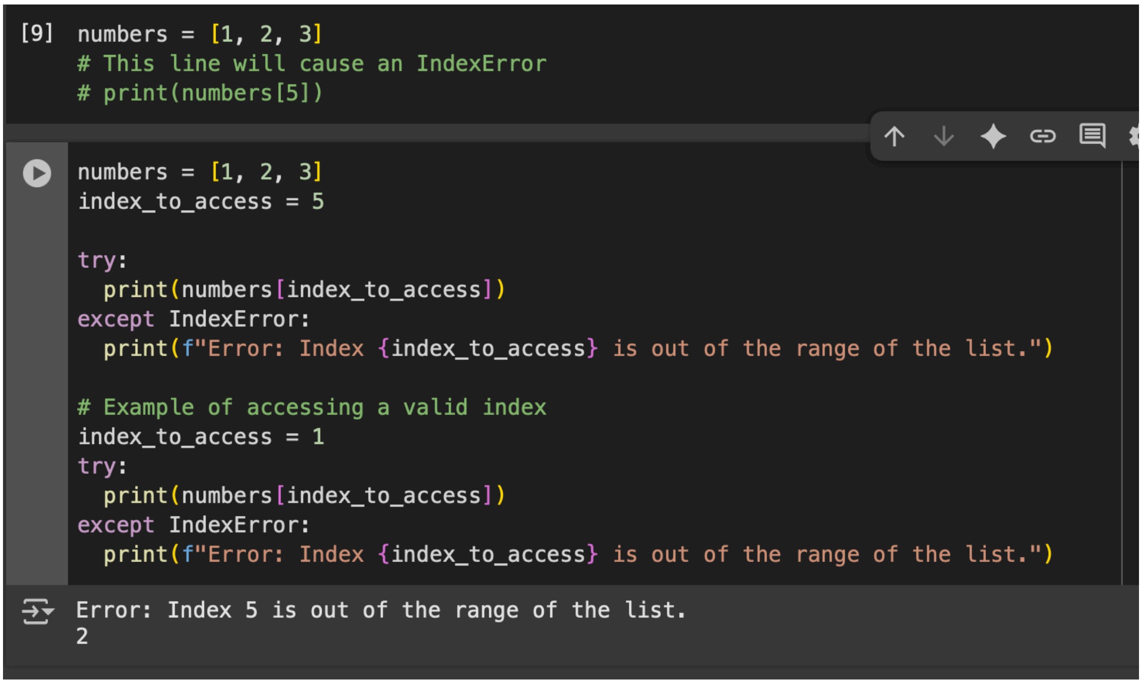
Task Description 5

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



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

expected code and output provided by google colab :

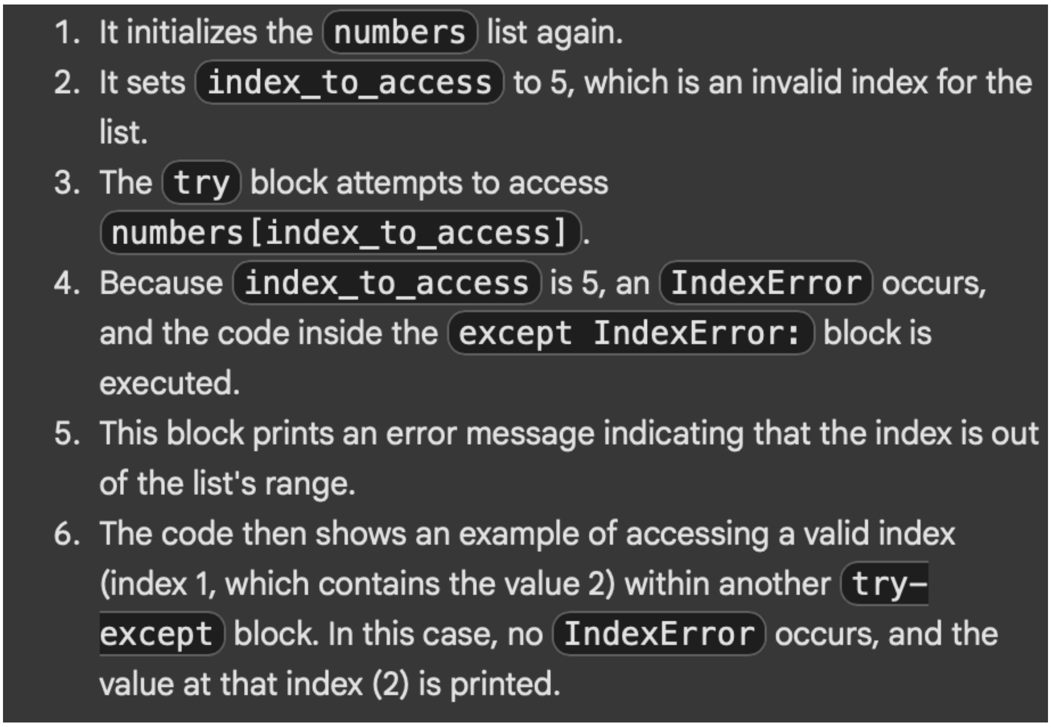


```
[9] numbers = [1, 2, 3]
    # This line will cause an IndexError
    # print(numbers[5])
```

```python
numbers = [1, 2, 3]
index_to_access = 5

try:
  print(numbers[index_to_access])
except IndexError:
  print(f"Error: Index {index_to_access} is out of the range of the list.")

# Example of accessing a valid index
index_to_access = 1
try:
  print(numbers[index_to_access])
except IndexError:
  print(f"Error: Index {index_to_access} is out of the range of the list.")
```

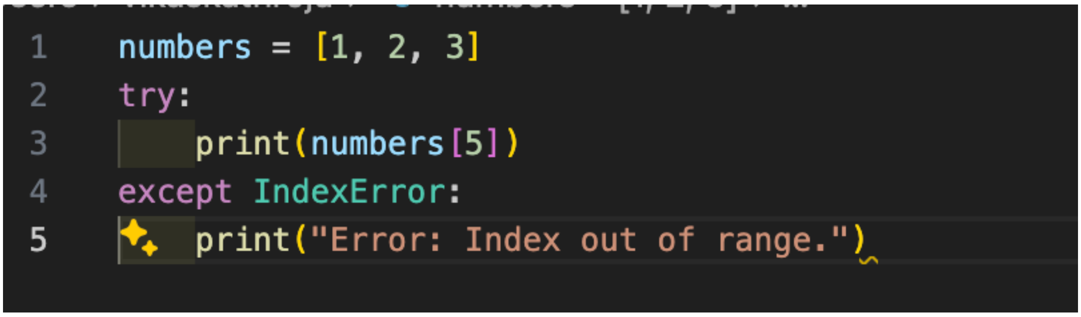
```
Error: Index 5 is out of the range of the list.
2
```

explanation of the code provided by the google colab :

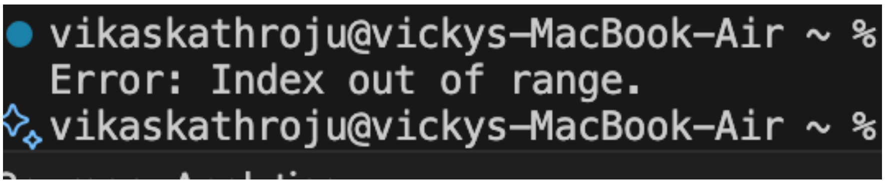


1. It initializes the `numbers` list again.
2. It sets `index_to_access` to 5, which is an invalid index for the list.
3. The `try` block attempts to access `numbers[index_to_access]`.
4. Because `index_to_access` is 5, an `IndexError` occurs, and the code inside the `except IndexError:` block is executed.
5. This block prints an error message indicating that the index is out of the list's range.
6. The code then shows an example of accessing a valid index (index 1, which contains the value 2) within another `try-except` block. In this case, no `IndexError` occurs, and the value at that index (2) is printed.

expected code and output provided by the github copilot :



```python
numbers = [1, 2, 3]
try:
    print(numbers[5])
except IndexError:
    print("Error: Index out of range.")
```



```
vikaskathroju@vickys-MacBook-Air ~ %
Error: Index out of range.
vikaskathroju@vickys-MacBook-Air ~ %
```