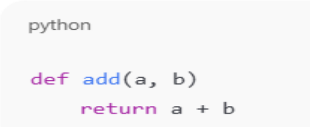
ASSIGNMENT-7.3

TASK-1:

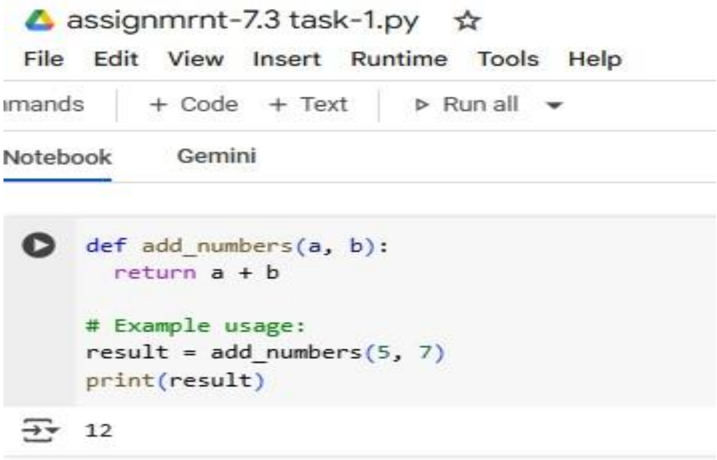


```python
def add(a, b)
    return a + b
```

Prompt: Identify the error in this code.

Code with Output:

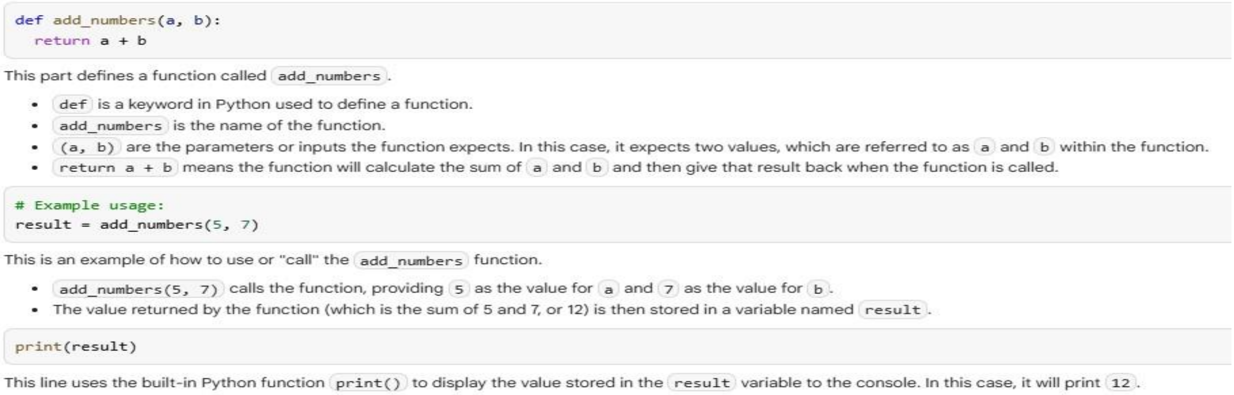
With GEMINI COPILOT:



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

# Example usage:
result = add_numbers(5, 7)
print(result)
```

```
12
```

Explanation:



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

This part defines a function called `add_numbers`.

- `def` is a keyword in Python used to define a function.
- `add_numbers` is the name of the function.
- `(a, b)` are the parameters or inputs the function expects. In this case, it expects two values, which are referred to as `a` and `b` within the function.
- `return a + b` means the function will calculate the sum of `a` and `b` and then give that result back when the function is called.

```python
# Example usage:
result = add_numbers(5, 7)
```

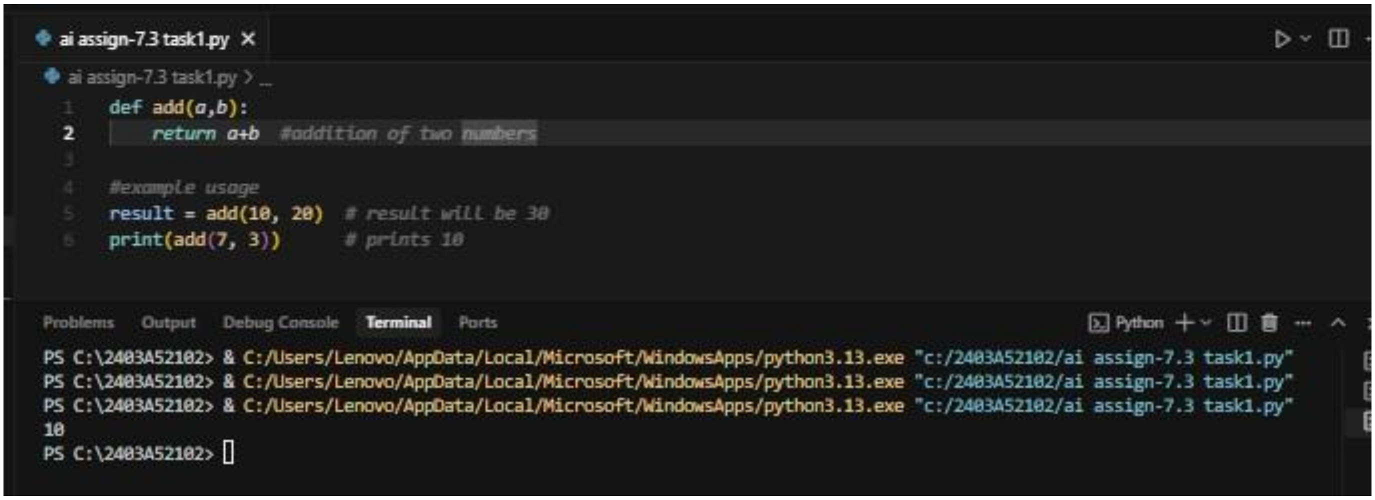
This is an example of how to use or "call" the `add_numbers` function.

- `add_numbers(5, 7)` calls the function, providing `5` as the value for `a` and `7` as the value for `b`.
- The value returned by the function (which is the sum of 5 and 7, or 12) is then stored in a variable named `result`.

```python
print(result)
```

This line uses the built-in Python function `print()` to display the value stored in the `result` variable to the console. In this case, it will print `12`.

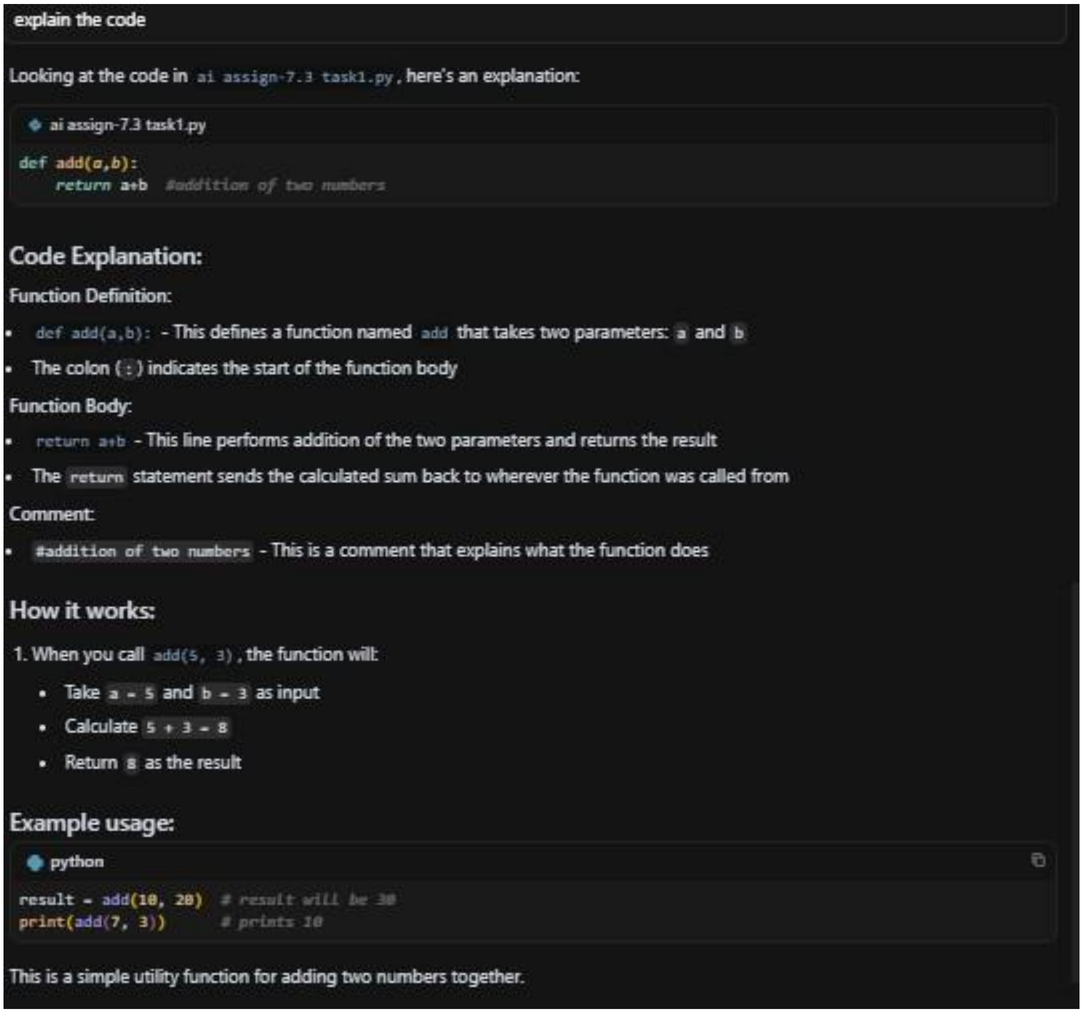
With Cursor AI:



```python
def add(a,b):
    return a+b  #addition of two numbers

#example usage
result = add(10, 20)  # result will be 30
print(add(7, 3))      # prints 10
```

```
PS C:\2403A52102> & C:/Users/Lenovo/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/2403A52102/ai assign-7.3 task1.py"
PS C:\2403A52102> & C:/Users/Lenovo/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/2403A52102/ai assign-7.3 task1.py"
PS C:\2403A52102> & C:/Users/Lenovo/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/2403A52102/ai assign-7.3 task1.py"
10
PS C:\2403A52102> []
```

Explanation:



explain the code

Looking at the code in `ai assign-7.3 task1.py`, here's an explanation:

```python
def add(a,b):
    return a+b  #addition of two numbers
```

Code Explanation:

Function Definition:

- `def add(a,b):` - This defines a function named `add` that takes two parameters: `a` and `b`
- The colon (`:`) indicates the start of the function body

Function Body:

- `return a+b` - This line performs addition of the two parameters and returns the result
- The `return` statement sends the calculated sum back to wherever the function was called from

Comment:

- `#addition of two numbers` - This is a comment that explains what the function does

How it works:

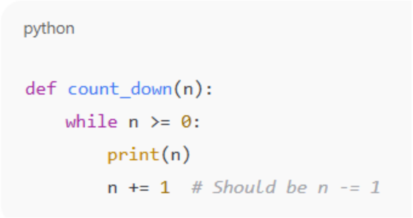
1. When you call `add(5, 3)`, the function will:
    - Take `a = 5` and `b = 3` as input
    - Calculate `5 + 3 = 8`
    - Return `8` as the result

Example usage:

```python
result = add(10, 20)  # result will be 30
print(add(7, 3))      # prints 10
```

This is a simple utility function for adding two numbers together.

TASK-2:

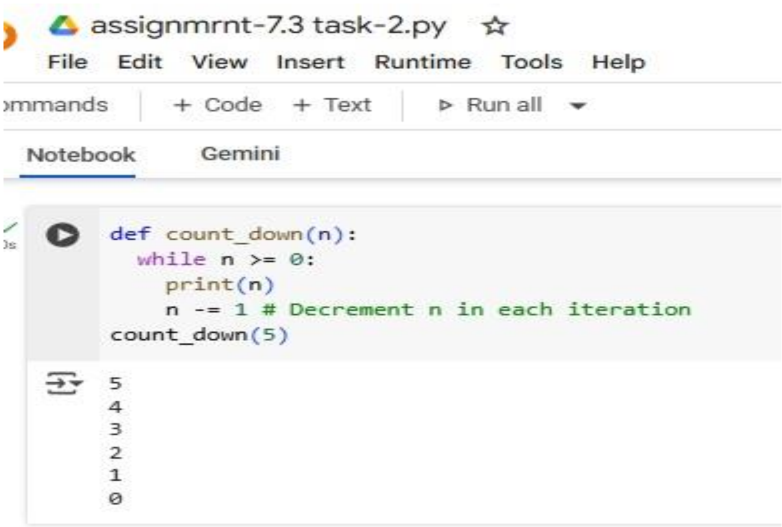


```python
def count_down(n):
    while n >= 0:
        print(n)
        n += 1  # Should be n -= 1
```

Prompt: Identify the error in this code.

Code with Output:

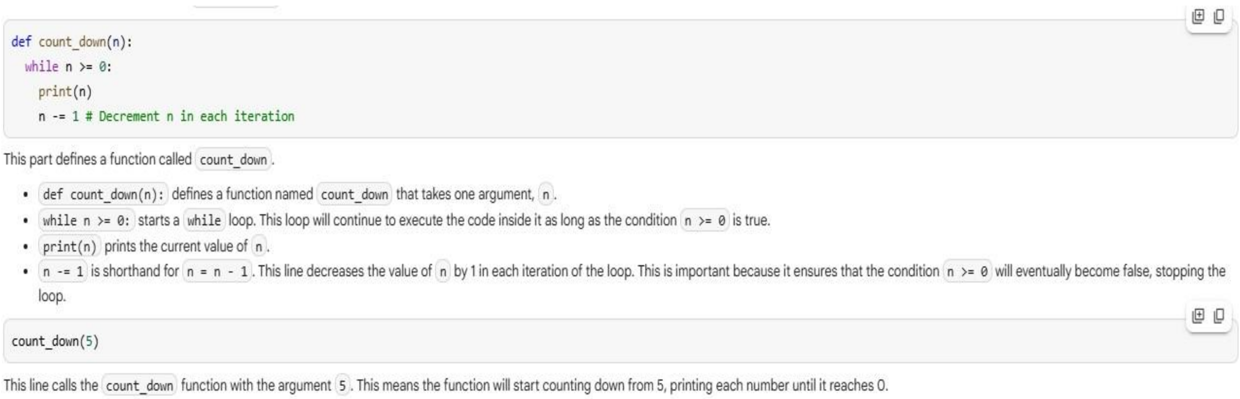
WITH GEMINI COPILOT:



```python
def count_down(n):
  while n >= 0:
    print(n)
    n -= 1 # Decrement n in each iteration
count_down(5)
```

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

Explanation:



```python
def count_down(n):
  while n >= 0:
    print(n)
    n -= 1 # Decrement n in each iteration
```

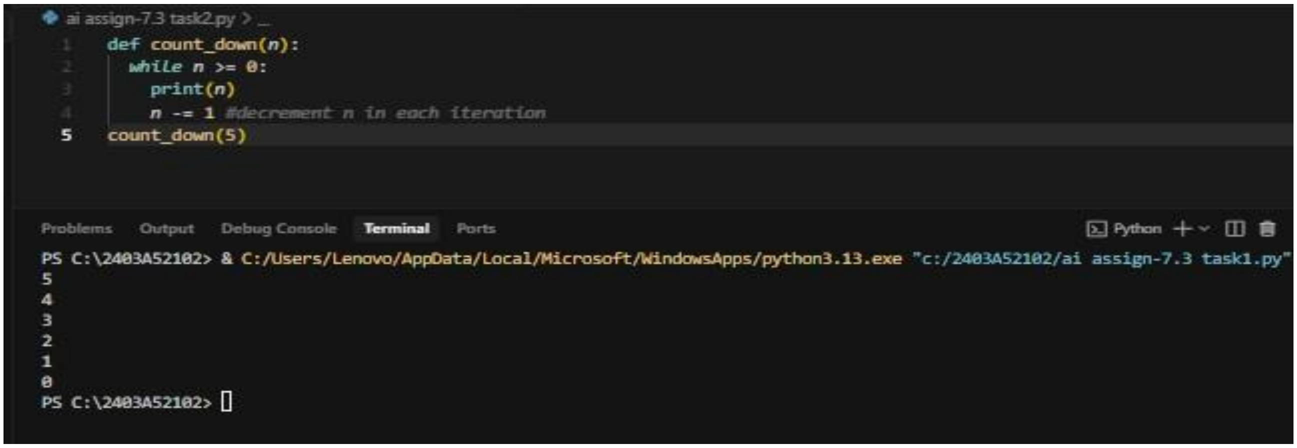
This part defines a function called `count_down`.

- `def count_down(n):` defines a function named `count_down` that takes one argument, `n`.
- `while n >= 0:` starts a `while` loop. This loop will continue to execute the code inside it as long as the condition `n >= 0` is true.
- `print(n)` prints the current value of `n`.
- `n -= 1` is shorthand for `n = n - 1`. This line decreases the value of `n` by 1 in each iteration of the loop. This is important because it ensures that the condition `n >= 0` will eventually become false, stopping the loop.

```python
count_down(5)
```

This line calls the `count_down` function with the argument `5`. This means the function will start counting down from 5, printing each number until it reaches 0.

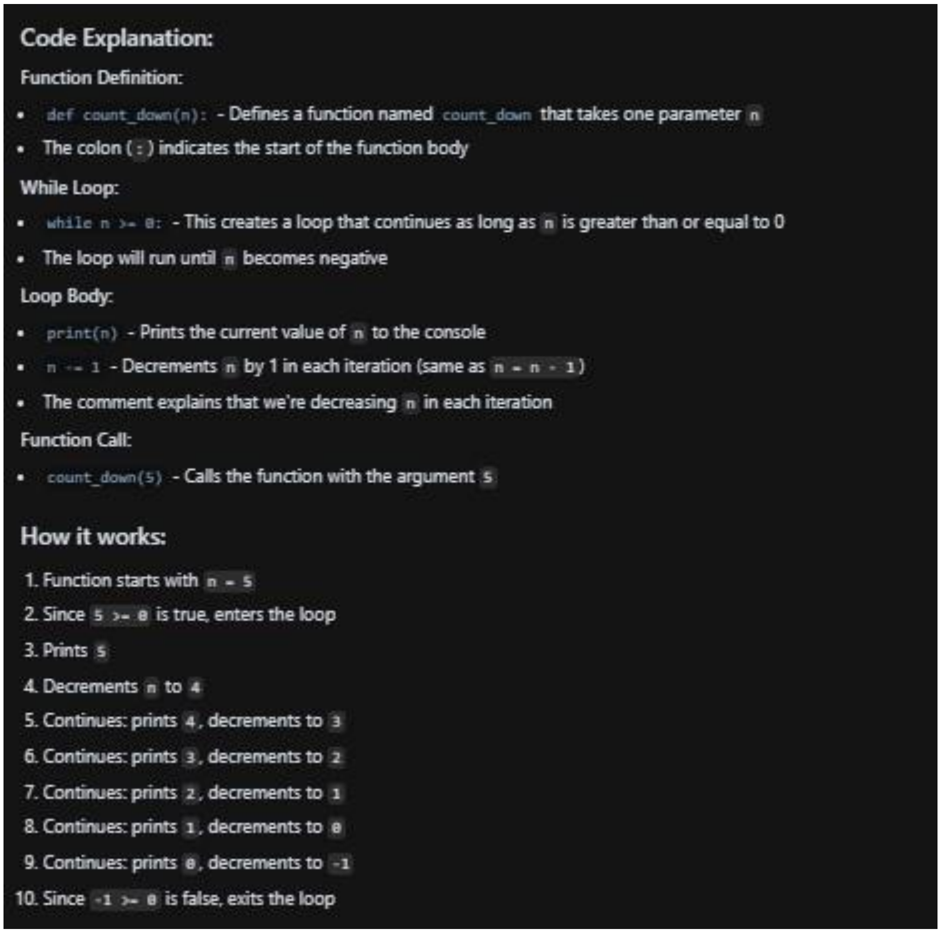
WITH CURSOR AI:



```python
def count_down(n):
  while n >= 0:
    print(n)
    n -= 1 #decrement n in each iteration
count_down(5)
```

```
PS C:\2403A52102> & C:/Users/Lenovo/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/2403A52102/ai assign-7.3 task1.py"
5
4
3
2
1
0
PS C:\2403A52102> 
```

Explanation:



Code Explanation:

Function Definition:

- `def count_down(n):` - Defines a function named `count_down` that takes one parameter `n`
- The colon (`:`) indicates the start of the function body

While Loop:

- `while n >= 0:` - This creates a loop that continues as long as `n` is greater than or equal to 0
- The loop will run until `n` becomes negative

Loop Body:

- `print(n)` - Prints the current value of `n` to the console
- `n -= 1` - Decrements `n` by 1 in each iteration (same as `n = n - 1`)
- The comment explains that we're decreasing `n` in each iteration

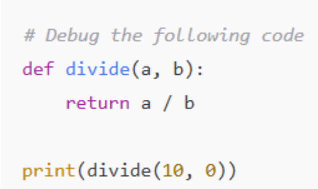
Function Call:

- `count_down(5)` - Calls the function with the argument `5`

How it works:

1. Function starts with `n = 5`
2. Since `5 >= 0` is true, enters the loop
3. Prints `5`
4. Decrements `n` to `4`
5. Continues: prints `4`, decrements to `3`
6. Continues: prints `3`, decrements to `2`
7. Continues: prints `2`, decrements to `1`
8. Continues: prints `1`, decrements to `0`
9. Continues: prints `0`, decrements to `-1`
10. Since `-1 >= 0` is false, exits the loop

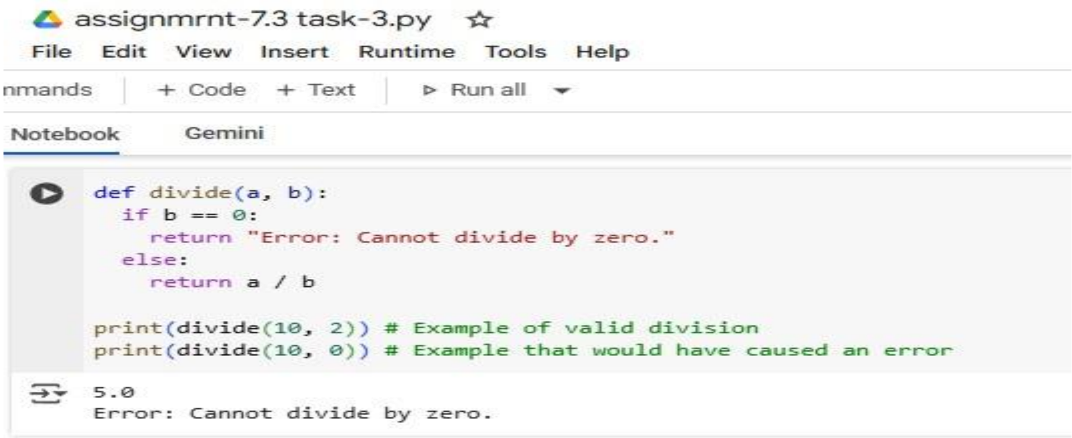
TASK-3:



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

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

Prompt: Identify the error this code.

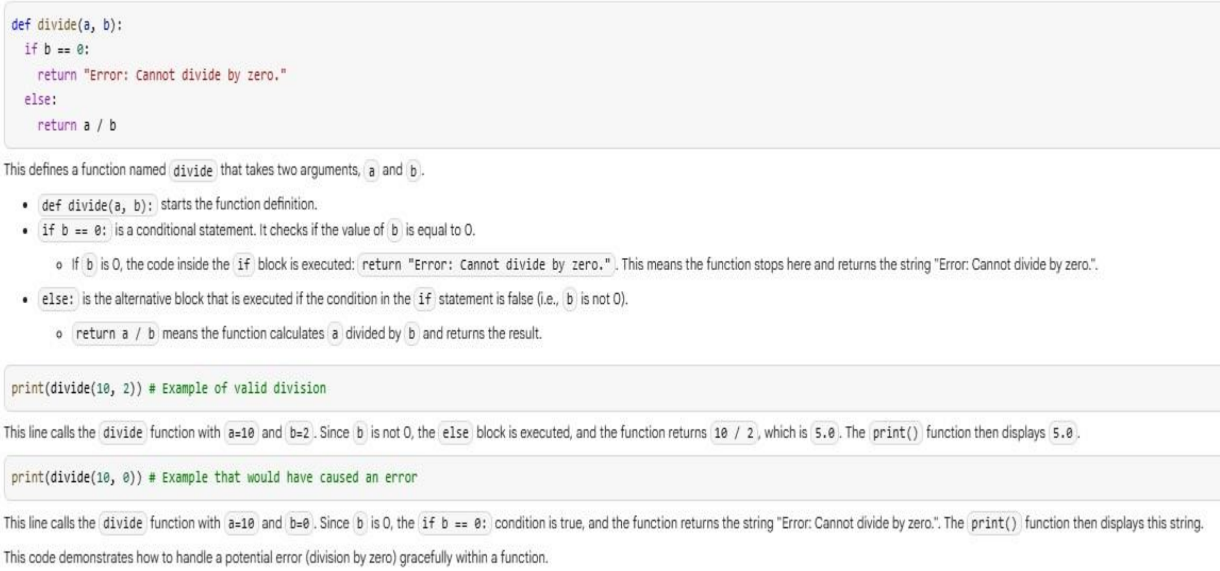
WITH GEMINI COPILOT:



```
def divide(a, b):
  if b == 0:
    return "Error: Cannot divide by zero."
  else:
    return a / b

print(divide(10, 2)) # Example of valid division
print(divide(10, 0)) # Example that would have caused an error
```

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

Explanation:



```
def divide(a, b):
  if b == 0:
    return "Error: Cannot divide by zero."
  else:
    return a / b
```

This defines a function named `divide` that takes two arguments, `a` and `b`.

- `def divide(a, b):` starts the function definition.
- `if b == 0:` is a conditional statement. It checks if the value of `b` is equal to 0.
  - If `b` is 0, the code inside the `if` block is executed: `return "Error: Cannot divide by zero."`. This means the function stops here and returns the string "Error: Cannot divide by zero.".
- `else:` is the alternative block that is executed if the condition in the `if` statement is false (i.e., `b` is not 0).
  - `return a / b` means the function calculates `a` divided by `b` and returns the result.

```
print(divide(10, 2)) # Example of valid division
```

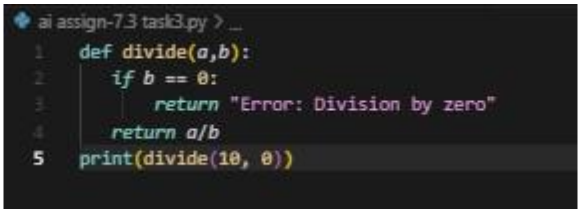
This line calls the `divide` function with `a=10` and `b=2`. Since `b` is not 0, the `else` block is executed, and the function returns `10 / 2`, which is `5.0`. The `print()` function then displays `5.0`.

```
print(divide(10, 0)) # Example that would have caused an error
```

This line calls the `divide` function with `a=10` and `b=0`. Since `b` is 0, the `if b == 0:` condition is true, and the function returns the string "Error: Cannot divide by zero.". The `print()` function then displays this string.

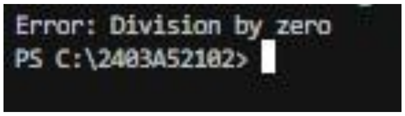
This code demonstrates how to handle a potential error (division by zero) gracefully within a function.

WITH CURSOR AI:



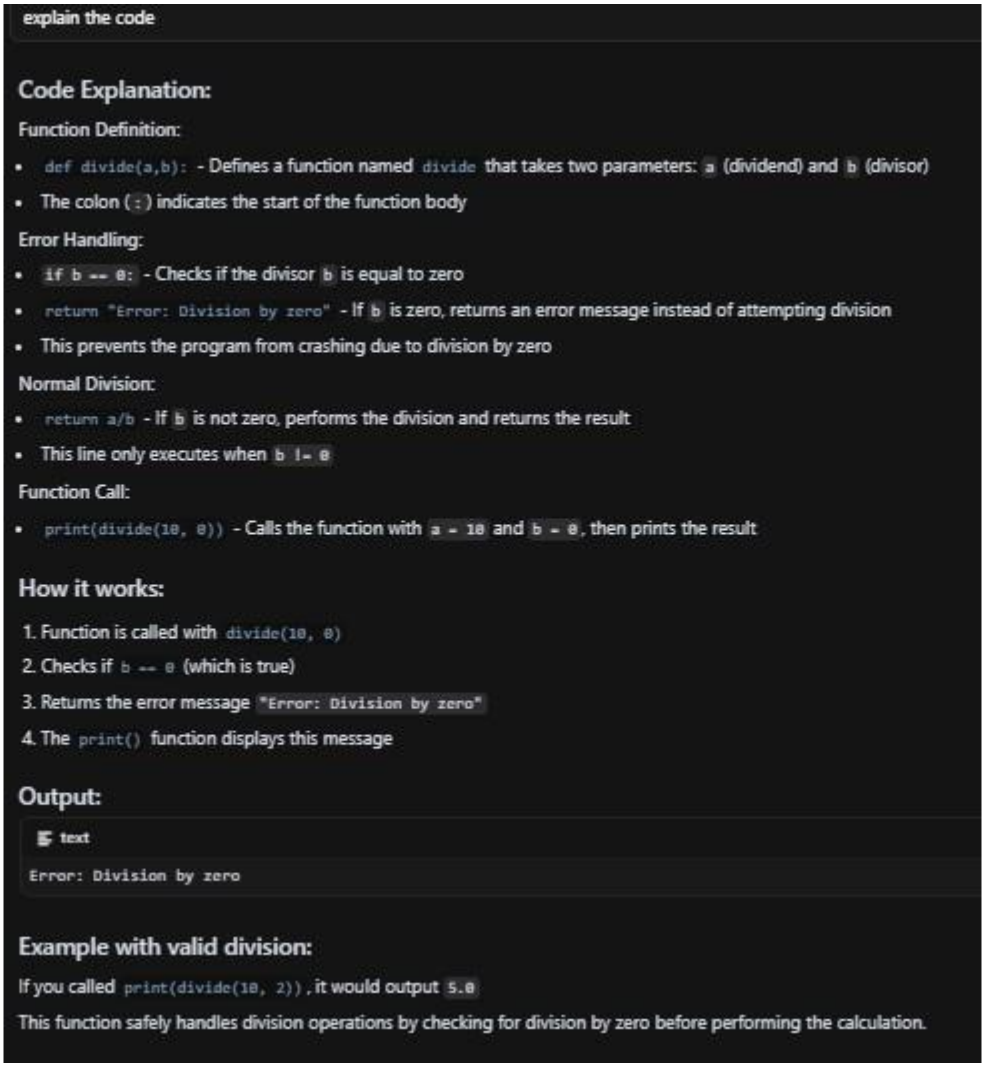
```python
def divide(a,b):
    if b == 0:
        return "Error: Division by zero"
    return a/b
print(divide(10, 0))
```

Output:



```
Error: Division by zero
PS C:\2403A52102>
```

Explanation:



explain the code

Code Explanation:

Function Definition:

- `def divide(a,b):` - Defines a function named `divide` that takes two parameters: `a` (dividend) and `b` (divisor)
- The colon (`:`) indicates the start of the function body

Error Handling:

- `if b == 0:` - Checks if the divisor `b` is equal to zero
- `return "Error: Division by zero"` - If `b` is zero, returns an error message instead of attempting division
- This prevents the program from crashing due to division by zero

Normal Division:

- `return a/b` - If `b` is not zero, performs the division and returns the result
- This line only executes when `b != 0`

Function Call:

- `print(divide(10, 0))` - Calls the function with `a = 10` and `b = 0`, then prints the result

How it works:

1. Function is called with `divide(10, 0)`
2. Checks if `b == 0` (which is true)
3. Returns the error message `"Error: Division by zero"`
4. The `print()` function displays this message

Output:

text

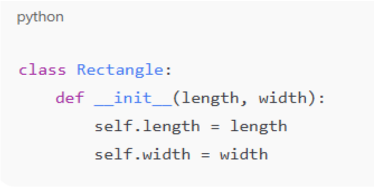
```
Error: Division by zero
```

Example with valid division:

If you called `print(divide(10, 2))`, it would output `5.0`

This function safely handles division operations by checking for division by zero before performing the calculation.

TASK-4:

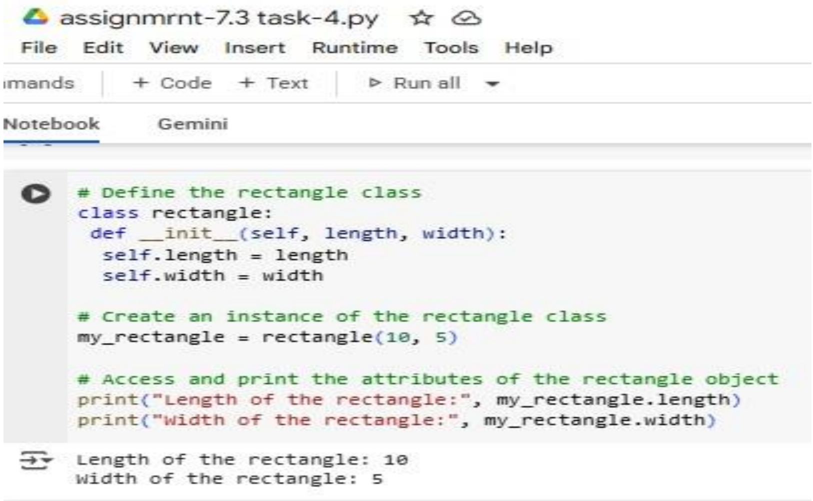


```python
class Rectangle:
    def __init__(length, width):
        self.length = length
        self.width = width
```

Prompt: Identify the error in this code.

Code with output:

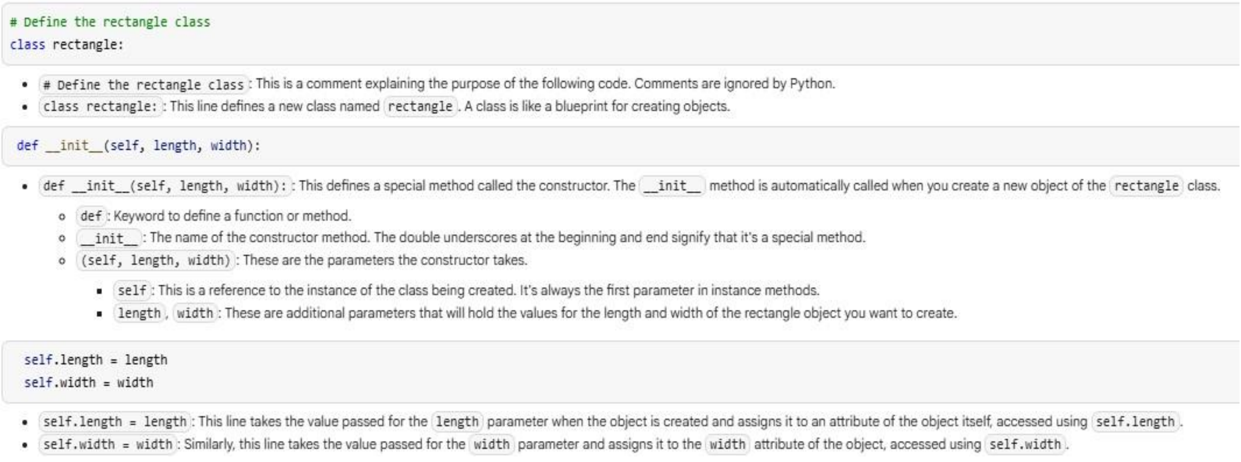
WITH GEMINI COPILOT:



```python
# Define the rectangle class
class rectangle:
 def __init__(self, length, width):
  self.length = length
  self.width = width

# Create an instance of the rectangle class
my_rectangle = rectangle(10, 5)

# Access and print the attributes of the rectangle object
print("Length of the rectangle:", my_rectangle.length)
print("Width of the rectangle:", my_rectangle.width)
```

```
Length of the rectangle: 10
Width of the rectangle: 5
```

Explanation:



```python
# Define the rectangle class
class rectangle:
```

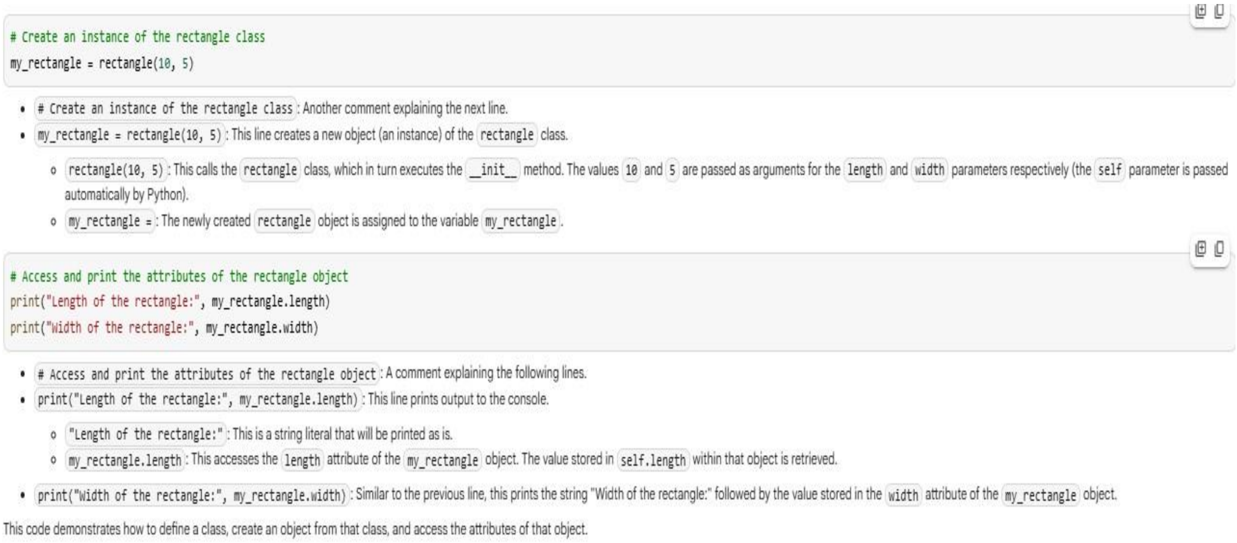
- `# Define the rectangle class` : This is a comment explaining the purpose of the following code. Comments are ignored by Python.
- `class rectangle:` : This line defines a new class named `rectangle` . A class is like a blueprint for creating objects.

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

- `def __init__(self, length, width):` : This defines a special method called the constructor. The `__init__` method is automatically called when you create a new object of the `rectangle` class.
  - `def` : Keyword to define a function or method.
  - `__init__` : The name of the constructor method. The double underscores at the beginning and end signify that it's a special method.
  - `(self, length, width)` : These are the parameters the constructor takes.
    - `self` : This is a reference to the instance of the class being created. It's always the first parameter in instance methods.
    - `length` , `width` : These are additional parameters that will hold the values for the length and width of the rectangle object you want to create.

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

- `self.length = length` : This line takes the value passed for the `length` parameter when the object is created and assigns it to an attribute of the object itself, accessed using `self.length` .
- `self.width = width` : Similarly, this line takes the value passed for the `width` parameter and assigns it to the `width` attribute of the object, accessed using `self.width` .



```
# Create an instance of the rectangle class
my_rectangle = rectangle(10, 5)
```

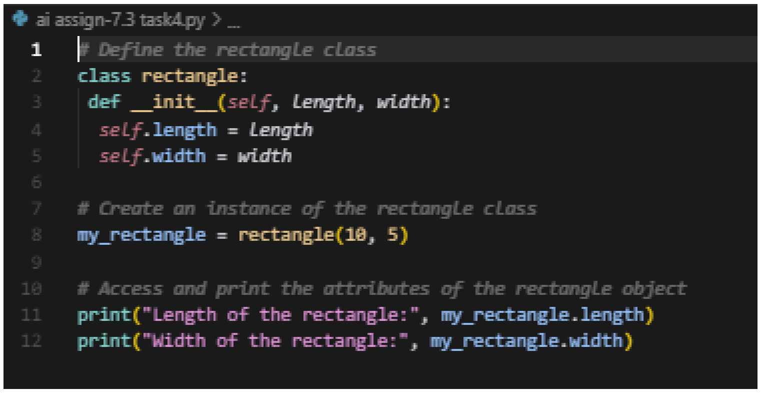
- `# Create an instance of the rectangle class` : Another comment explaining the next line.
- `my_rectangle = rectangle(10, 5)` : This line creates a new object (an instance) of the `rectangle` class.
    - `rectangle(10, 5)` : This calls the `rectangle` class, which in turn executes the `__init__` method. The values `10` and `5` are passed as arguments for the `length` and `width` parameters respectively (the `self` parameter is passed automatically by Python).
    - `my_rectangle =` : The newly created `rectangle` object is assigned to the variable `my_rectangle`.

```
# Access and print the attributes of the rectangle object
print("Length of the rectangle:", my_rectangle.length)
print("Width of the rectangle:", my_rectangle.width)
```

- `# Access and print the attributes of the rectangle object` : A comment explaining the following lines.
- `print("Length of the rectangle:", my_rectangle.length)` : This line prints output to the console.
    - `"Length of the rectangle:"` : This is a string literal that will be printed as is.
    - `my_rectangle.length` : This accesses the `length` attribute of the `my_rectangle` object. The value stored in `self.length` within that object is retrieved.
- `print("Width of the rectangle:", my_rectangle.width)` : Similar to the previous line, this prints the string "Width of the rectangle:" followed by the value stored in the `width` attribute of the `my_rectangle` object.

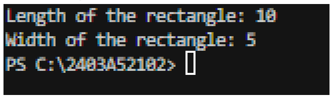
This code demonstrates how to define a class, create an object from that class, and access the attributes of that object.

WITH CURSOR AI:



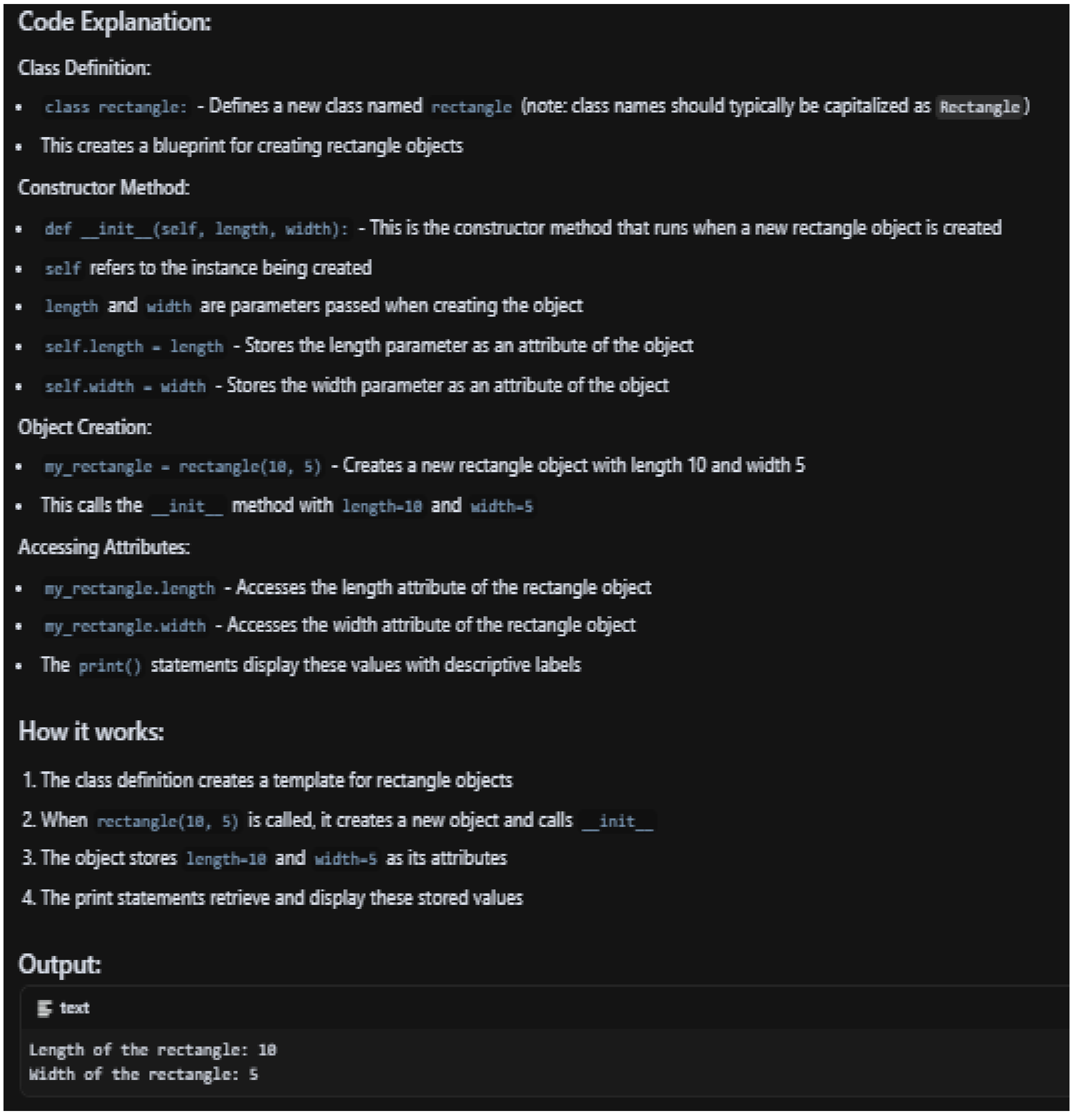
```python
# Define the rectangle class
class rectangle:
 def __init__(self, length, width):
  self.length = length
  self.width = width

# Create an instance of the rectangle class
my_rectangle = rectangle(10, 5)

# Access and print the attributes of the rectangle object
print("Length of the rectangle:", my_rectangle.length)
print("Width of the rectangle:", my_rectangle.width)
```

Output:



```
Length of the rectangle: 10
Width of the rectangle: 5
PS C:\2403A52102>
```

Explanation:



Code Explanation:

Class Definition:

- `class rectangle:` - Defines a new class named `rectangle` (note: class names should typically be capitalized as `Rectangle`)
- This creates a blueprint for creating rectangle objects

Constructor Method:

- `def __init__(self, length, width):` - This is the constructor method that runs when a new rectangle object is created
- `self` refers to the instance being created
- `length` and `width` are parameters passed when creating the object
- `self.length = length` - Stores the length parameter as an attribute of the object
- `self.width = width` - Stores the width parameter as an attribute of the object

Object Creation:

- `my_rectangle = rectangle(10, 5)` - Creates a new rectangle object with length 10 and width 5
- This calls the `__init__` method with `length=10` and `width=5`

Accessing Attributes:

- `my_rectangle.length` - Accesses the length attribute of the rectangle object
- `my_rectangle.width` - Accesses the width attribute of the rectangle object
- The `print()` statements display these values with descriptive labels

How it works:

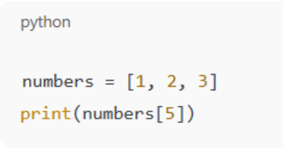
1. The class definition creates a template for rectangle objects
2. When `rectangle(10, 5)` is called, it creates a new object and calls `__init__`
3. The object stores `length=10` and `width=5` as its attributes
4. The print statements retrieve and display these stored values

Output:

text

```
Length of the rectangle: 10
Width of the rectangle: 5
```

TASK-5:



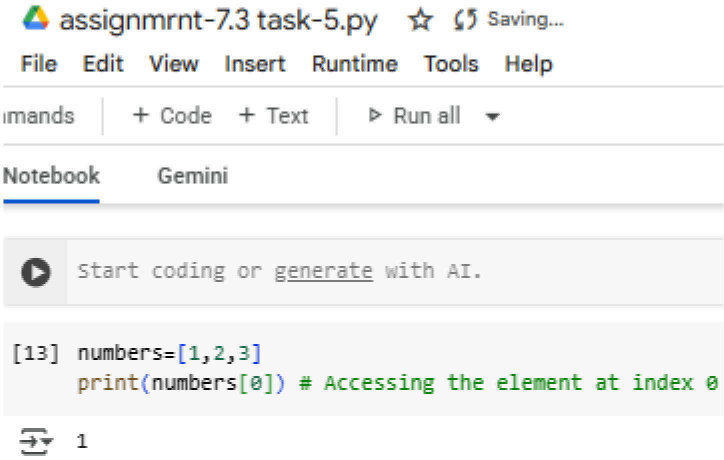
python

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

Prompt: Identify the error in this code.

Code with Output:

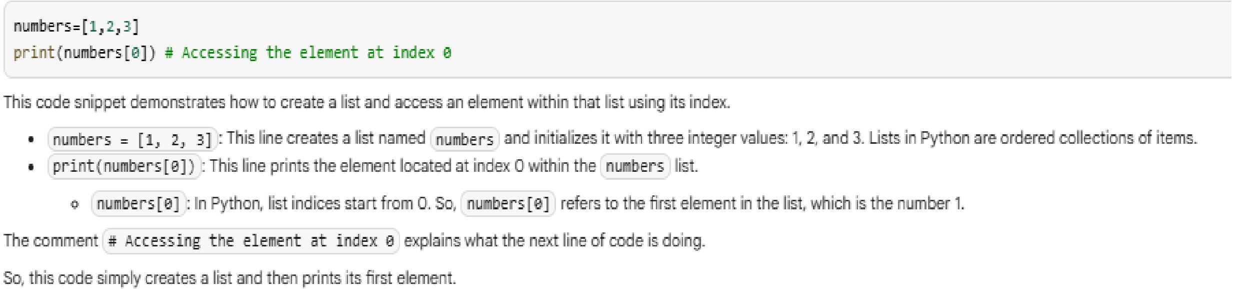
WITH GEMINI COPILOT:



```python
numbers=[1,2,3]
print(numbers[0]) # Accessing the element at index 0
```

```
1
```

Explanation:



```python
numbers=[1,2,3]
print(numbers[0]) # Accessing the element at index 0
```

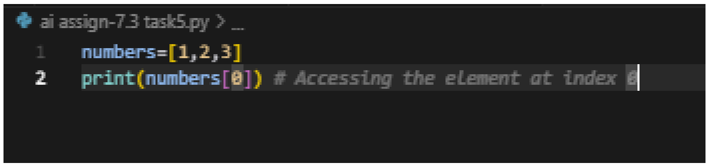
This code snippet demonstrates how to create a list and access an element within that list using its index.

- `numbers = [1, 2, 3]`: This line creates a list named `numbers` and initializes it with three integer values: 1, 2, and 3. Lists in Python are ordered collections of items.
- `print(numbers[0])`: This line prints the element located at index 0 within the `numbers` list.
    - `numbers[0]`: In Python, list indices start from 0. So, `numbers[0]` refers to the first element in the list, which is the number 1.

The comment `# Accessing the element at index 0` explains what the next line of code is doing.

So, this code simply creates a list and then prints its first element.

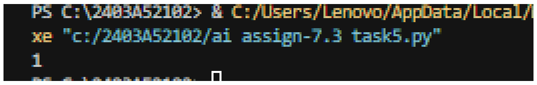
WITH CURSOR AI:



ai assign-7.3 task5.py > ...

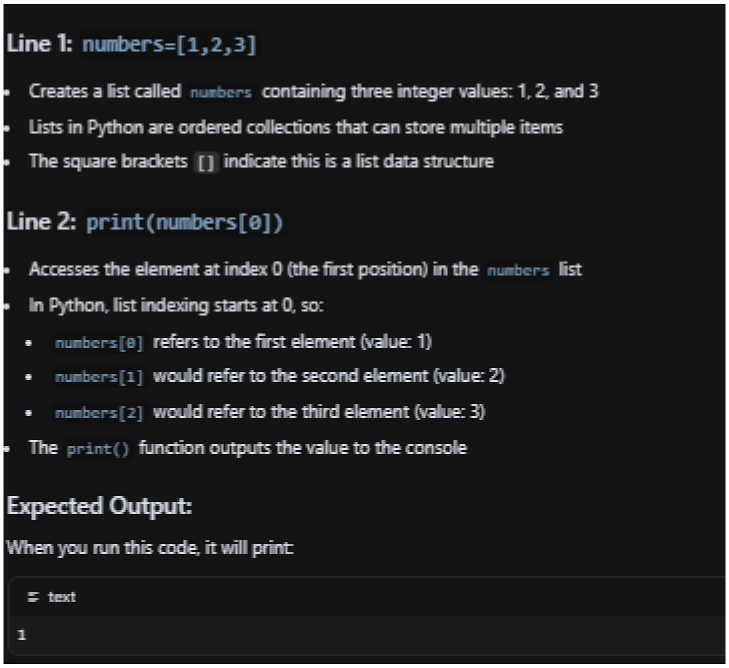
```
1   numbers=[1,2,3]
2   print(numbers[0]) # Accessing the element at index 0
```

Output:



```
PS C:\2403A52102> & C:/Users/Lenovo/AppData/Local/
xe "c:/2403A52102/ai assign-7.3 task5.py"
1
```

Explanation:



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

- Creates a list called `numbers` containing three integer values: 1, 2, and 3
- Lists in Python are ordered collections that can store multiple items
- The square brackets `[]` indicate this is a list data structure

Line 2: `print(numbers[0])`

- Accesses the element at index 0 (the first position) in the `numbers` list
- In Python, list indexing starts at 0, so:
  - `numbers[0]` refers to the first element (value: 1)
  - `numbers[1]` would refer to the second element (value: 2)
  - `numbers[2]` would refer to the third element (value: 3)
- The `print()` function outputs the value to the console

Expected Output:

When you run this code, it will print:

```text
1
```