

ASSIGNMENT-7.3

HT NO:2303A51961

Batch No:28

Course:AI Assisted Coding

Task 1: Fixing Syntax Errors

Prompt: Detect and fix the syntax error in the following Python function.

Code:

1. Code with Errors





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

... File `"/tmp/ipython-input-2057910517.py"`, line 1

```
def add(a,b)
      ^
SyntaxError: expected ':'
```

Next steps: [Explain error](#)

 Gemini




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


... File `"/tmp/ipython-input-2057910517.py"`, line 1

```
def add(a,b)
      ^
SyntaxError: expected ':'
```

Next steps: [Explain error](#)



0 Please explain this error:

 `SyntaxError: expected ':'`

 The error `SyntaxError: expected ':'` means that a colon is missing at the end of your function definition (`def add(a,b)`). Python requires a colon after the parameters of a function definition. I've fixed this for you.

[Accept & Run](#) [Accept](#) [Cancel](#)

What can I help you build?

 Gemini 2.5 Flash 

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

2.AI Fixed Code and Output:

```
def add(a,b):  
    return a+b  
print(add(1,2))
```

... 3

Justification:

- Python function definitions must end with a colon (:).
- Missing the colon causes a SyntaxError during compilation.
- The interpreter cannot identify the start of the function body.
- Due to this error, the program does not execute at all.
- AI detects such errors by checking Python grammar rules.
- Adding the colon resolves the syntax issue completely.

Task2: Debugging Logic Errors in Loops

Prompt: Identify why the following loop runs infinitely and fix the logic.

Code:

1. Code with logical Errors

```
def count_down(n):
    while n>=0:
        print(n)
        n+=1
    print(count_down(4))
```

```
... 12062986
12062987
12062988
12062989
12062990
12062991
12062992
12062993
12062994
12062995
12062996
```

Gemini

```
def count_down(n):
    while n>=0:
        print(n)
        n+=1
    print(count_down(4))
```

...

31391758
31391759
31391760
31391761
31391762
31391763
31391764
31391765
31391766
31391767
31391768
31391769
31391770
31391771
31391772
31391773

0
: Identify why the following loop runs infinitely and fix the logic.

The loop in your `count_down` function runs infinitely because `n` is being incremented (`n+=1`) instead of decremented. This means `n` will always be greater than or equal to 0. To fix this, we need to change `n+=1` to `n-=1` so

Accept & Run
Accept
Cancel

What can I help you build?

+
Gemini 2.5 Flash

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

2.AI Fixed Code and Output

```
def count_down(n):  
    while n>=0:  
        print(n)  
        n-=1  
    print(count_down(4))
```

```
... 4  
    3  
    2  
    1  
    0  
    None
```

Justification:

The loop condition depends on the variable n.

- The variable n is not updated inside the loop body.
- This causes the condition to remain true indefinitely.
- As a result, the loop runs infinitely.
- AI identifies the missing increment statement.
- Adding proper increment logic fixes the infinite loop.

Task 3: Handling Runtime Errors (Division by Zero)

Prompt: Find the runtime error in the division function and handle it safely

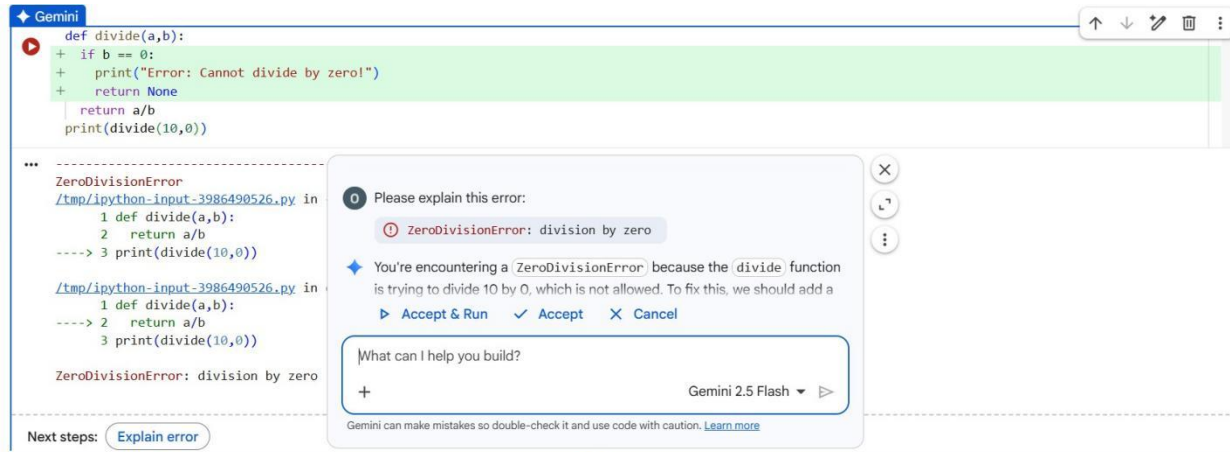
Code:

1. Code with Runtime Errors

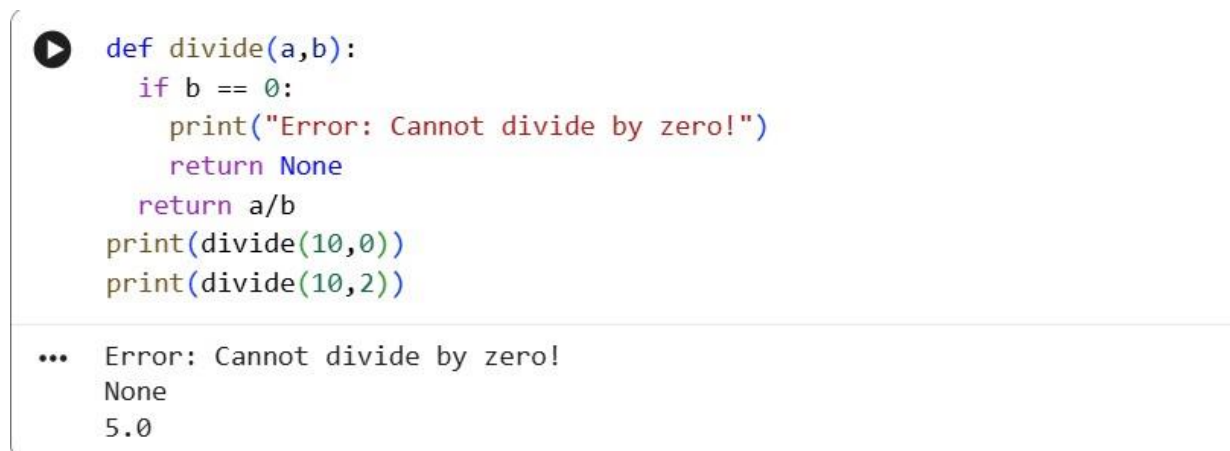
```
def divide(a,b):  
    return a/b  
print(divide(10,0))
```

```
... -----  
ZeroDivisionError                                Traceback (most recent call last)  
  /tmp/ipython-input-3986490526.py in <cell line: 0>()  
    1 def divide(a,b):  
    2     return a/b  
----> 3 print(divide(10,0))  
  
  /tmp/ipython-input-3986490526.py in divide(a, b)  
    1 def divide(a,b):  
----> 2     return a/b  
    3 print(divide(10,0))  
  
ZeroDivisionError: division by zero
```

Next steps: [Explain error](#)



2.AI Fixed Code and Output



Justification:

- Dividing a number by zero causes a runtime error.
- Python raises a ZeroDivisionError during execution.
- Without handling, the program crashes immediately.
- AI detects this risk by analyzing execution flow.
- A try-except block prevents program termination.
- Proper error handling ensures safe execution.

Task 4: Debugging Class Definition Errors

Prompt: Detect the issue in the class constructor and correct it.

Code:

1.Code with Defination Errors

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

The screenshot shows a code editor with a Python class definition for a Rectangle. The class has an `__init__` method that takes `length` and `width` as arguments but is missing the `self` parameter. A Gemini AI chat window is open, displaying the prompt: "Detect the issue in the class constructor and correct it." The AI response identifies the error: "The issue in the Rectangle class constructor is that the __init__ method is missing the self parameter. In Python, instance methods, including constructors, must always have self as their first argument to access the object's attributes." The AI provides two options: "Accept & Run" and "Accept". Below the chat window, there is a text input field with the placeholder "What can I help you build?" and a "Gemini 2.5 Flash" model selector.

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

0 Detect the issue in the class constructor and correct it.

◆ The issue in the `Rectangle` class constructor is that the `__init__` method is missing the `self` parameter. In Python, instance methods, including constructors, must always have `self` as their first argument to access the object's attributes. *Please ensure that the definition is correct.*

▶ Accept & Run ✓ Accept ✕ Cancel

What can I help you build?

+ Gemini 2.5 Flash ▶

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

2.AI Fixed Code and Output

[16]

✓ 0s

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

my_rectangle = Rectangle(5, 10)
print(f"Rectangle length: {my_rectangle.length}")
print(f"Rectangle width: {my_rectangle.width}")
```

✓ ... Rectangle length: 5
Rectangle width: 10

Justification:

- self represents the current object in a class.
- It is mandatory in the __init__() method.
- Missing self prevents instance variable creation.
- This leads to incorrect object initialization.
- AI detects structural issues in class definitions.
- Adding self enables proper object behavior.

Task 5: Resolving Index Errors in Lists

Prompt: Identify and fix the index error in the list access code.

Code:

1. Code with Index Errors

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

```
... -----
IndexError                                Traceback (most recent call last)
/tmp/ipython-input-3809996345.py in <cell line: 0>()
      1 numbers=[1,2,3]
----> 2 print(numbers[5])

IndexError: list index out of range
```

Next steps: [Explain error](#)

The screenshot shows the Gemini AI interface. At the top, a code cell contains the following Python code:

```
numbers=[1,2,3]
print(numbers[5])
print(numbers[0])
```

The code cell shows a red error bar for the first two lines and a green bar for the third line. Below the code cell, the error message is displayed:

```
... -----
IndexError                                Traceback (most recent call last)
/tmp/ipython-input-3809996345.py in <cell line: 0>()
      1 numbers=[1,2,3]
----> 2 print(numbers[5])

IndexError: list index out of range
```

Below the error message, there is a "Next steps:" section with a button labeled "Explain error". A modal window is open, displaying the error message and an explanation:

0 Please explain this error:

⚠️ `IndexError: list index out of range`

◆ The error `IndexError: list index out of range` occurs because you are trying to access an element at index 5 in the `numbers` list, but the list

▶ Accept & Run ✓ Accept ✕ Cancel

What can I help you build?

+ Gemini 2.5 Flash ▶

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

2. AI Fixed Code and Output

◆ Gemini

```
▶ numbers=[1,2,3]  
print(numbers[0])
```

... 1

Justification:

- Python lists have fixed index boundaries.
- Accessing an invalid index causes an `IndexError`.
- Such errors occur at runtime.
- AI detects index misuse by checking list size.
- Bounds checking or exception handling prevents crashes.
- Safe access improves program stability.