

ASSIGNMENT-7.3

HT NO:2303A51585

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](#)

The screenshot shows the Gemini AI interface. At the top, there's a code editor window with the following content:

```
▶ Gemini
● def add(a,b)
  □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 ':'
```

Below the code editor, there's a button labeled "Next steps: [Explain error](#)". A modal window titled "0 Please explain this error:" is open, containing the following information:

- A small icon with a question mark and the text "SyntaxError: expected ':'".
- An explanatory message: "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!"
- Buttons for "Accept & Run", "Accept", and "Cancel".

At the bottom of the modal, there's a text input field asking "What can I help you build?", a plus sign button, and a "Gemini 2.5 Flash" dropdown menu. A small note at the bottom says "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

```

❶ 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

```

The screenshot shows the Gemini AI code editor interface. On the left, there is a code editor window with the following Python code:

```

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

```

The line `n+=1` is highlighted with a red background, indicating a potential error. A tooltip window appears over this line with the following text:

➊ : 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

At the bottom of the tooltip, there are three buttons: **Accept & Run**, **Accept**, and **Cancel**.

Below the code editor, a list of numbers is shown:

```

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

```

At the bottom right of the editor window, it says "Gemini 2.5 Flash".

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](#)

The screenshot shows the Gemini AI interface. On the left, there is a code editor window with Python code. The code defines a function `divide` that checks if `b` is zero and prints an error message if it is. It then returns `None` if `b` is zero or the result of `a/b` otherwise. A call to `divide(10, 0)` is shown, resulting in a `ZeroDivisionError`. Below the code editor, a button says "Next steps: Explain error".

In the center, a modal window titled "Please explain this error:" provides information about the error. It says "ZeroDivisionError: division by zero" and suggests adding a try-except block to handle the error. Buttons for "Accept & Run", "Accept", and "Cancel" are available.

At the bottom right of the modal, there is a text input field asking "What can I help you build?" and a "Gemini 2.5 Flash" button.

2. AI Fixed Code and Output

```
▶ def divide(a,b):
    if b == 0:
        print("Error: Cannot divide by zero!")
        return None
    return a/b
print(divide(10,0))
print(divide(10,2))
```

... Error: Cannot divide by zero!
None
5.0

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 Definition Errors

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

The screenshot shows the Gemini AI interface. On the left, there's a code editor window titled "Gemini" containing Python code for a Rectangle class. The code has several errors underlined with yellow squiggly lines: the class definition is missing a colon at the end, and the __init__ method is missing the self parameter. A green completion suggestion box is open over the code, showing a corrected version of the __init__ method with the self parameter added. To the right of the code editor is a large, semi-transparent AI assistant window. It contains a task card with the following text:
① 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...
Below this, there are three buttons: "Accept & Run", "Accept", and "Cancel". At the bottom of the AI window, there's a text input field asking "What can I help you build?" and a "Gemini 2.5 Flash" button.

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:

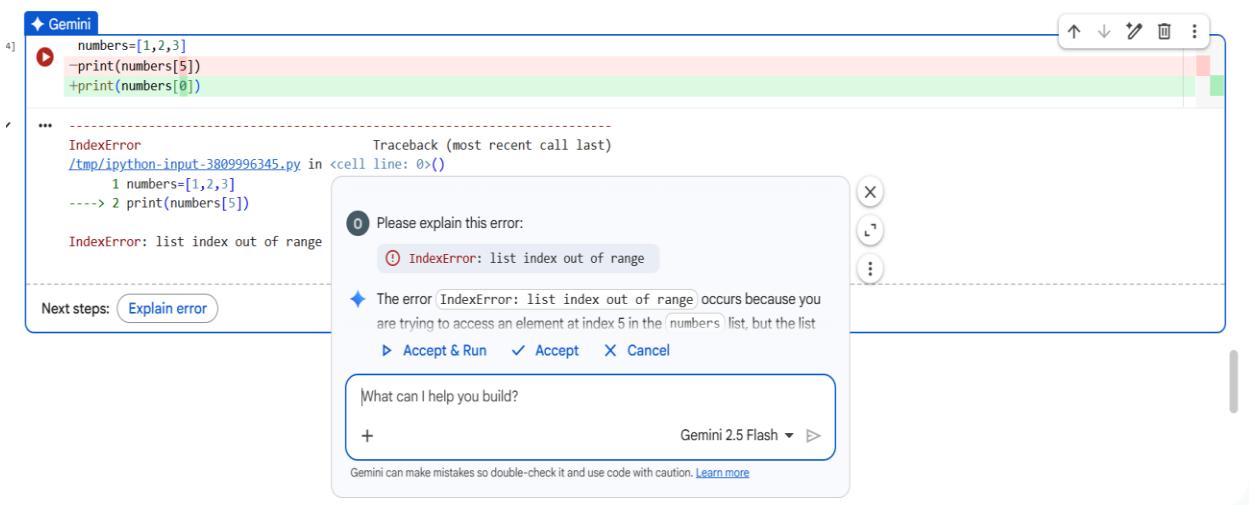
1. Code with Index Errors

▶ 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](#)



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.