

Assignment 7.3 Ai Assisted Coding

Htno:2303a51288

Btno:05

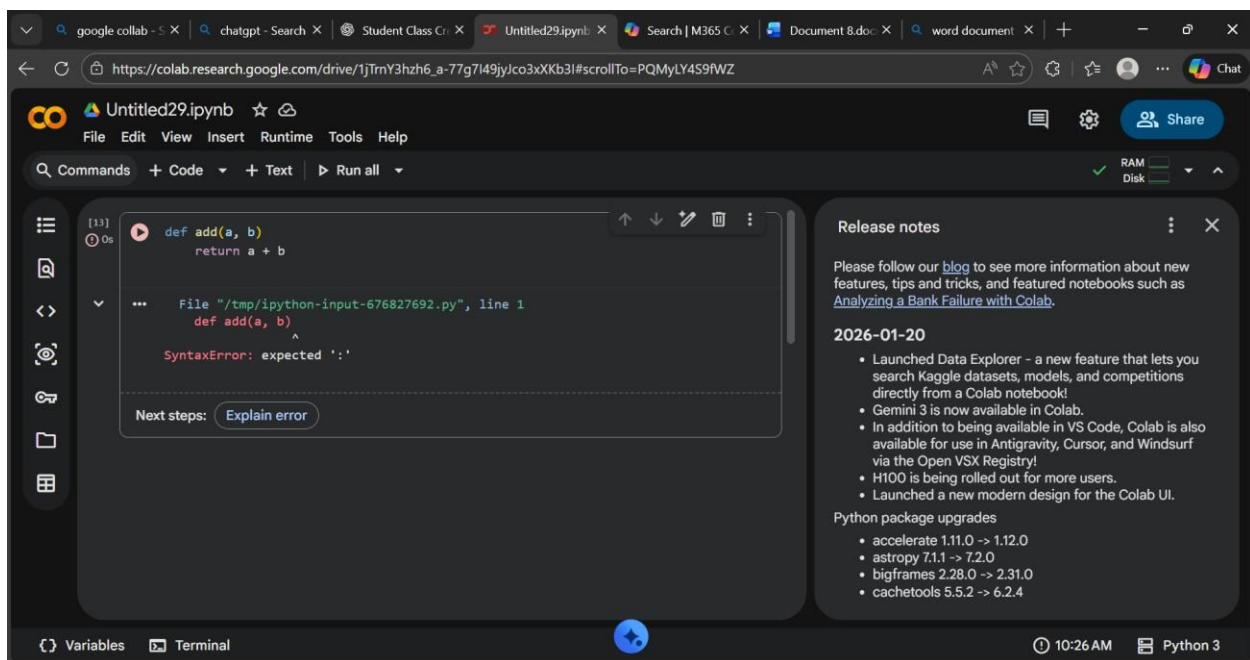
Task 1: Fixing Syntax Errors

Prompt: The following Python function has a syntax error. Identify the issue and correct it. Also explain what the syntax error is.

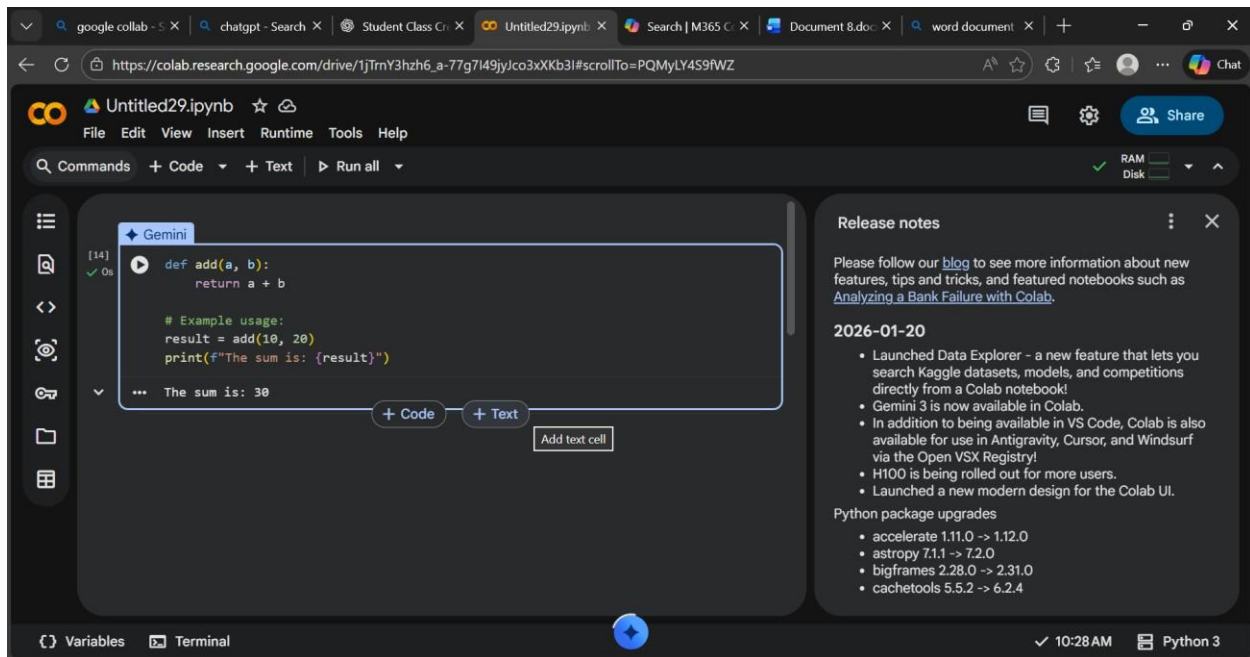
```
def add(a, b)
```

return a + b Input:

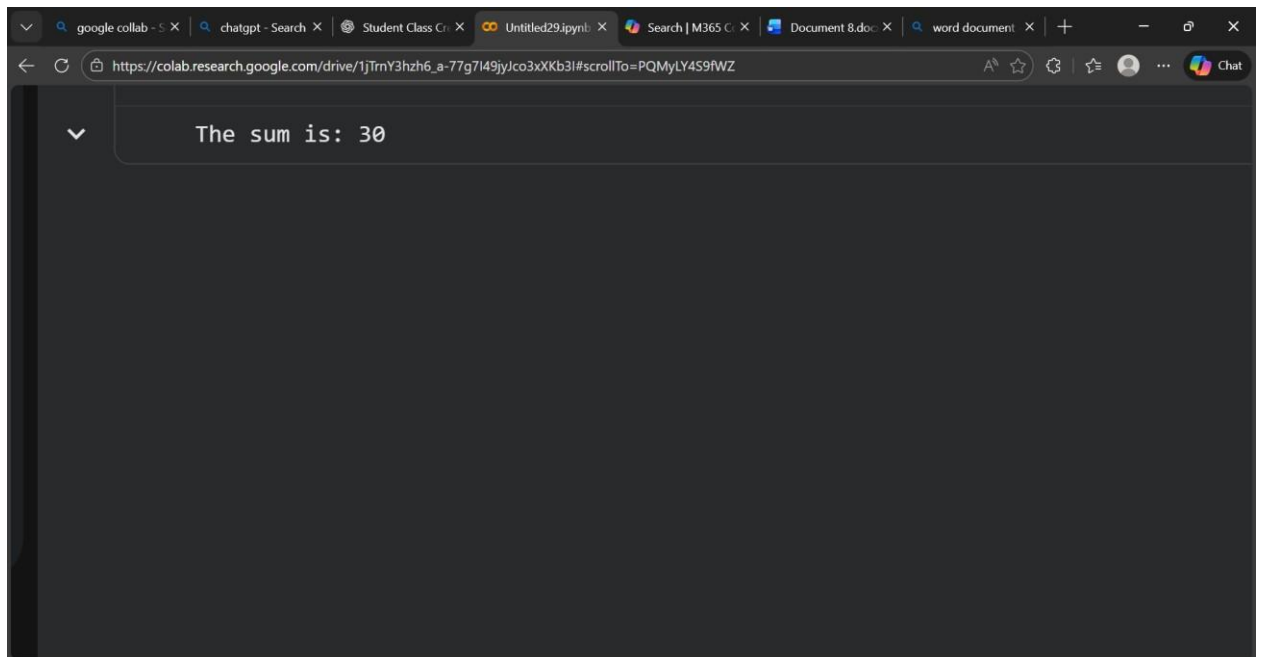
Bug Code:



2) corrected code:



Output:



Explanation:

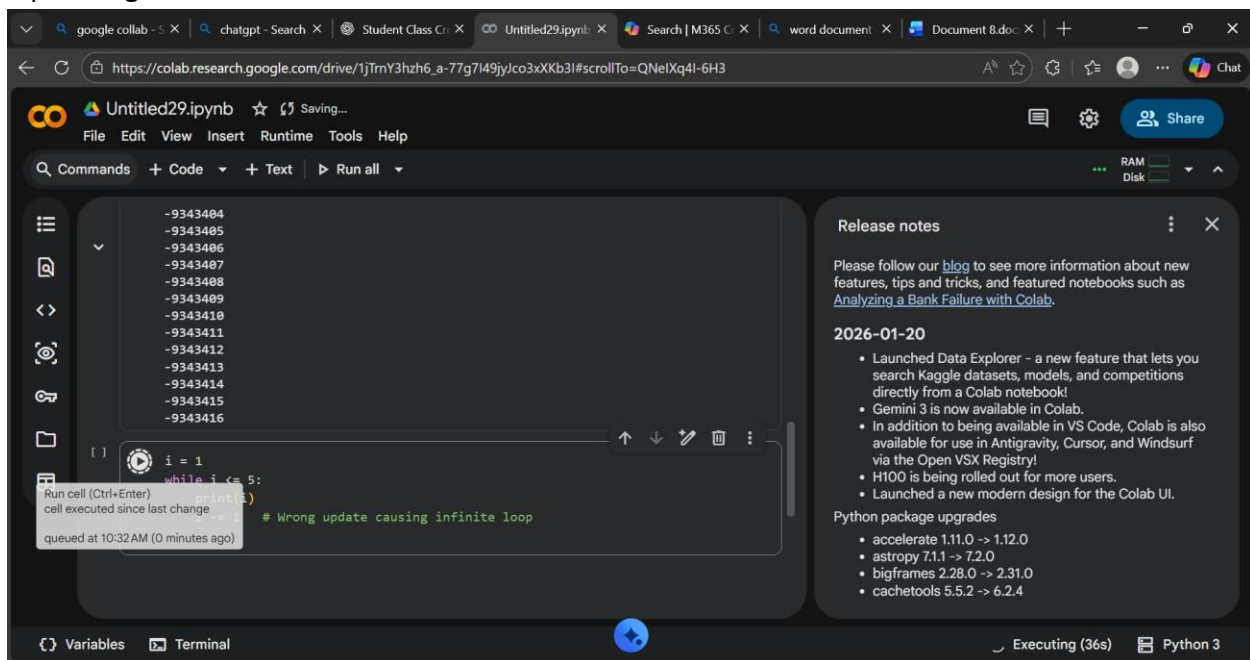
- In Python, a colon `:` is required after defining a function header.
- Without the colon, Python cannot recognize the start of the function block, causing a **SyntaxError**.
- AI correctly identified the missing colon and fixed the function definition.

Task 2: Debugging Logic Errors in Loops

Prompt: The following Python loop runs infinitely. Identify the logic error, correct the loop, and explain the issue.

```
i = 1 while i
<= 5:
    print(i)
i -= 1
```

Input: Bug code:



Corrected code:

The screenshot shows a Google Colab notebook interface. The browser tabs at the top include 'google collab', 'chatgpt - Search', 'Student Class Cr...', 'Untitled29.ipynb', 'Search | M365', 'word document', and 'Document 8.doc'. The notebook title is 'Untitled29.ipynb' with a 'Saving...' status. The menu bar includes 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help'. Below the menu is a toolbar with 'Commands', '+ Code', '+ Text', and 'Run all'. The code cell [16] contains the following Python code:

```
i = 1
while i <= 5:
    print(i)
    i += 1 # Corrected: increment i instead of decrementing

print("Loop finished.")
```

The output of the code cell shows the numbers 1, 2, and 3, followed by an ellipsis (...).

Output:

The screenshot shows the same Google Colab notebook interface, but the code cell is now empty. The output of the previous code cell is visible, showing the numbers 1, 2, 3, 4, and 5, followed by the text 'Loop finished.'.

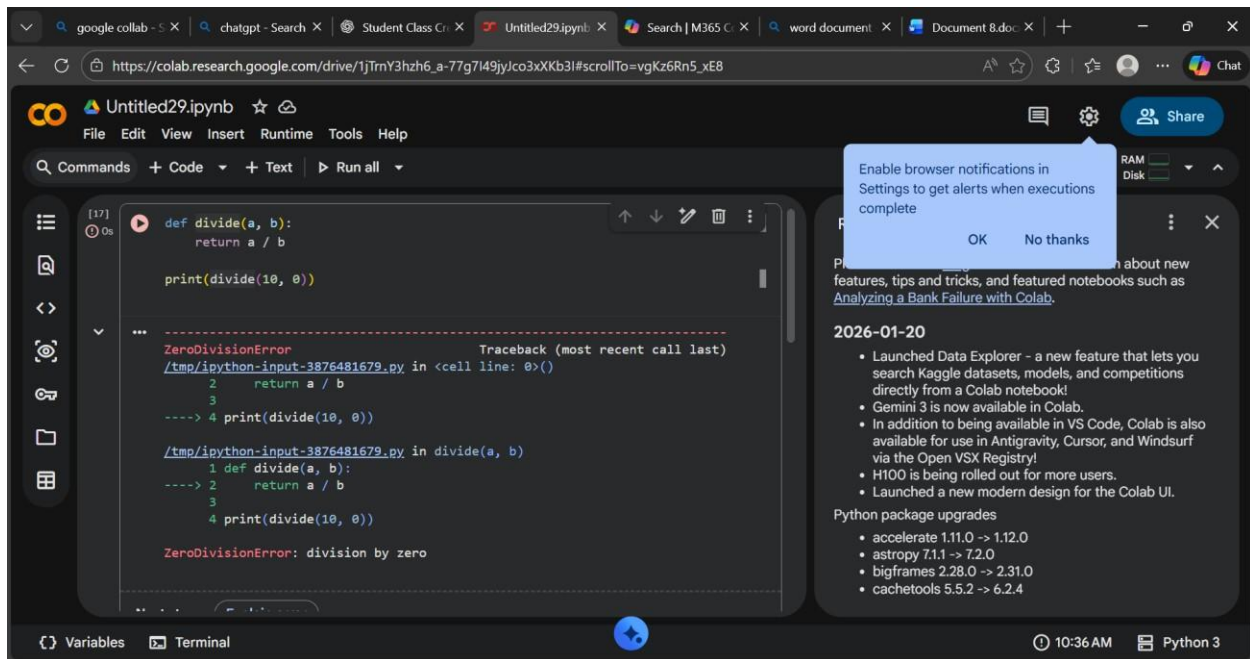
Explanation: The variable `i` was decreasing (`i -= 1`) while the condition required it to increase, causing an infinite loop.

Changing it to `i += 1` allows the loop to reach the stopping condition and terminate correctly.

Task 3: Handling Runtime Errors (Division by Zero)

Prompt: This Python code causes a runtime error. Identify the problem, fix it using `tryexcept`, and explain the issue. `def divide(a, b): return a / b print(divide(10, 0))`

Input:Bug Code



The screenshot shows a Google Colab notebook titled 'Untitled29.ipynb'. The code cell [17] contains a function `divide(a, b)` that returns `a / b`. Below the function definition, there is a call to `print(divide(10, 0))`. The output shows a `ZeroDivisionError: division by zero` traceback. A notification bubble in the top right corner says 'Enable browser notifications in Settings to get alerts when executions complete'. The right sidebar shows a list of updates for January 20, 2026, including the launch of Data Explorer and Gemini 3.

```
[17] def divide(a, b):
      return a / b

      print(divide(10, 0))

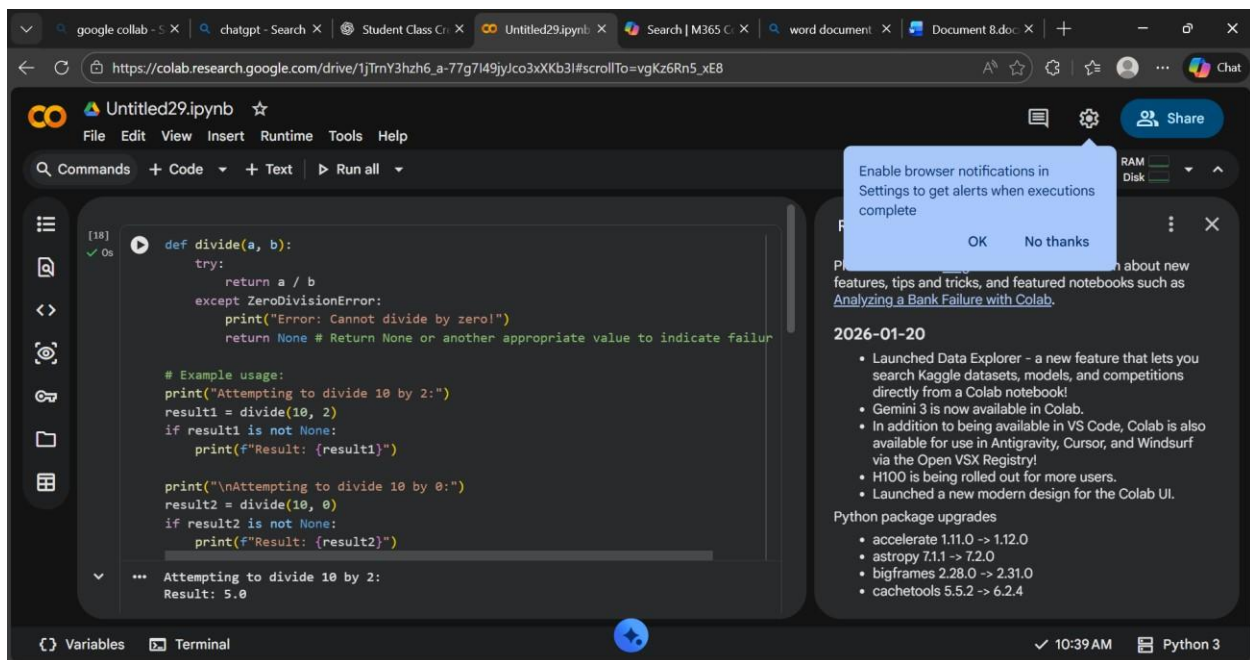
      ...

      ZeroDivisionError                                Traceback (most recent call last)
      /tmp/ipython-input-3876481679.py in <cell line: 0>()
          2     return a / b
          3
      ----> 4 print(divide(10, 0))

      /tmp/ipython-input-3876481679.py in divide(a, b)
          1 def divide(a, b):
      ----> 2     return a / b
          3
          4 print(divide(10, 0))

      ZeroDivisionError: division by zero
```

Corrected Code:



The screenshot shows the same Google Colab notebook with the corrected code. The code cell [18] now includes a try-except block to handle the `ZeroDivisionError`. The output shows the function being called with `divide(10, 2)` and `divide(10, 0)`, and the results being printed. A notification bubble in the top right corner says 'Enable browser notifications in Settings to get alerts when executions complete'. The right sidebar shows the same list of updates for January 20, 2026.

```
[18] def divide(a, b):
      try:
          return a / b
      except ZeroDivisionError:
          print("Error: Cannot divide by zero!")
          return None # Return None or another appropriate value to indicate failure

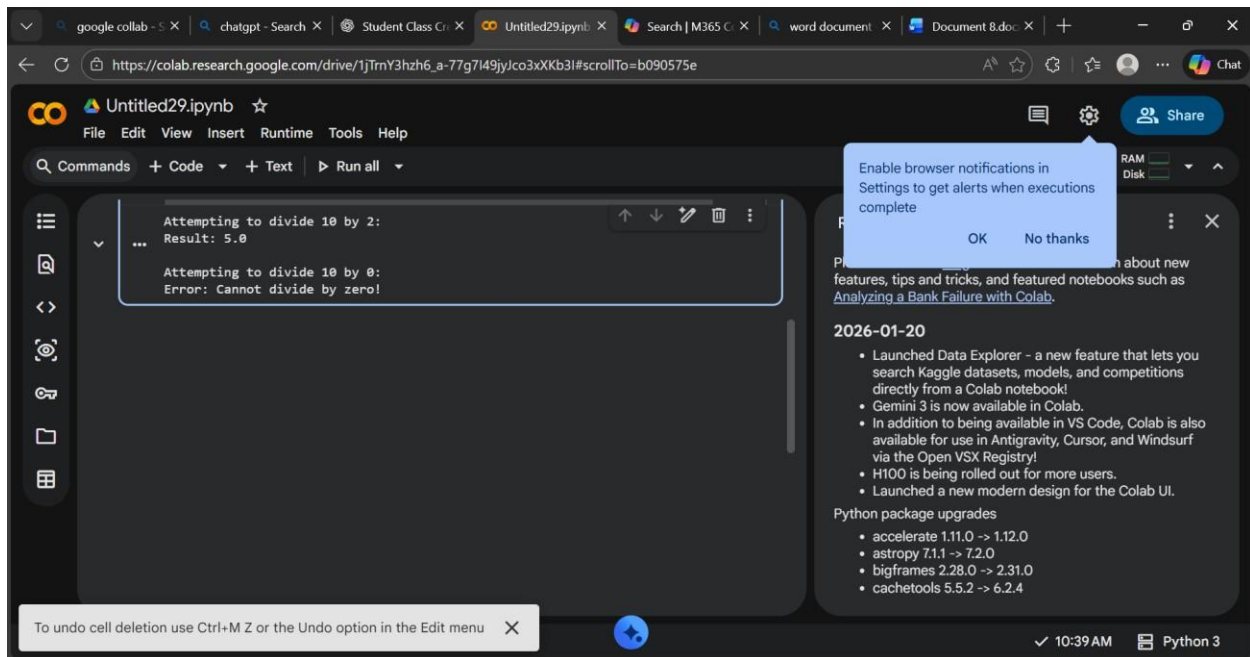
      # Example usage:
      print("Attempting to divide 10 by 2:")
      result1 = divide(10, 2)
      if result1 is not None:
          print(f"Result: {result1}")

      print("\nAttempting to divide 10 by 0:")
      result2 = divide(10, 0)
      if result2 is not None:
          print(f"Result: {result2}")

      ...

      Attempting to divide 10 by 2:
      Result: 5.0
```

Output:



Explanation: the program crashes because division by zero is not allowed in Python, causing a `ZeroDivisionError`.

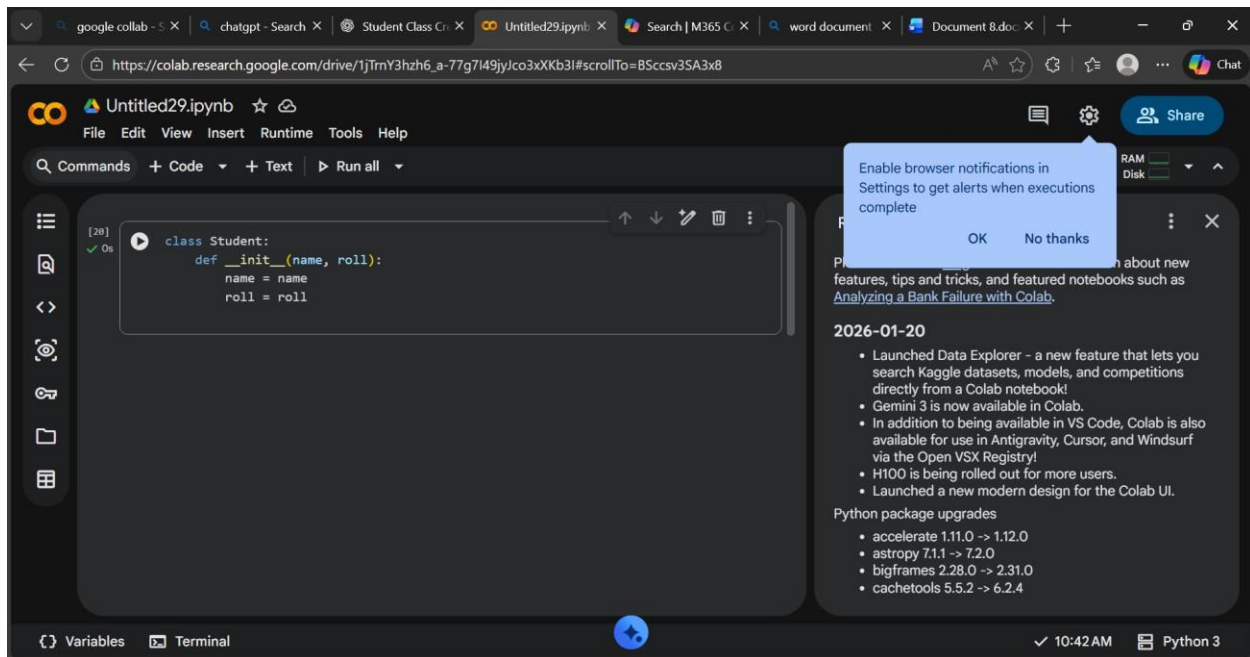
Using `try-except` prevents the crash and safely handles the error.

Task 4: Debugging Class Definition Errors

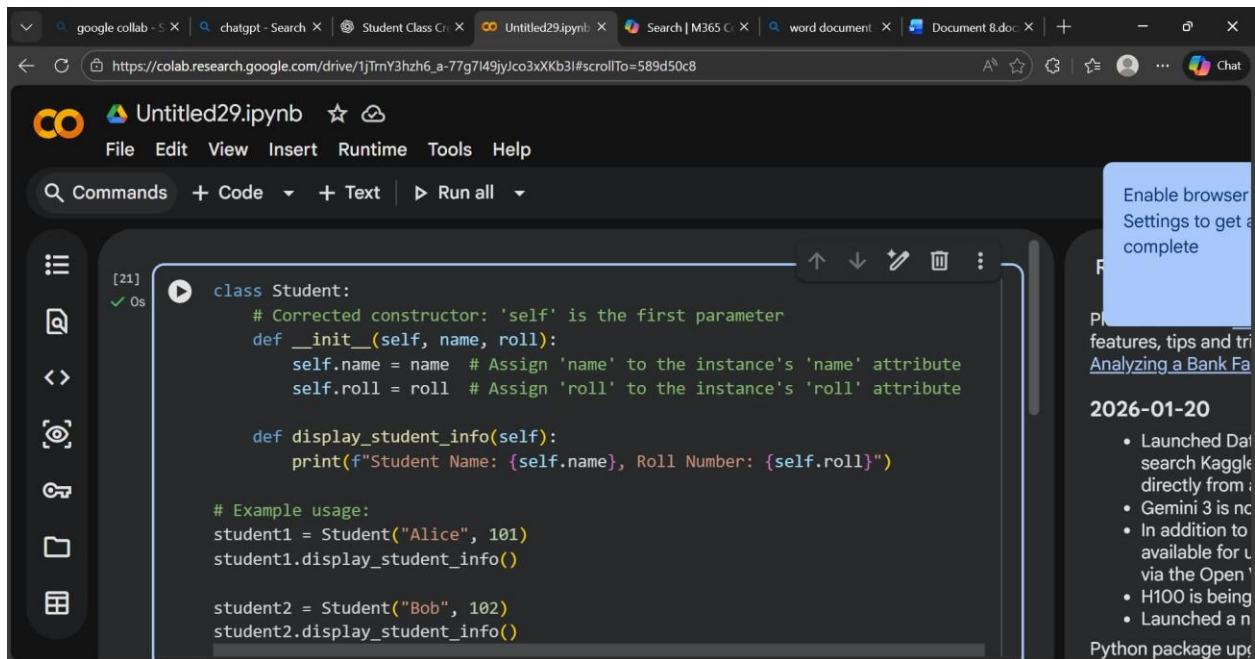
Prompt: The following Python class has an error in the constructor. Identify the issue, correct the class definition, and explain why the fix is needed.

```
class Student: def init(name, roll): name = name roll = roll
```

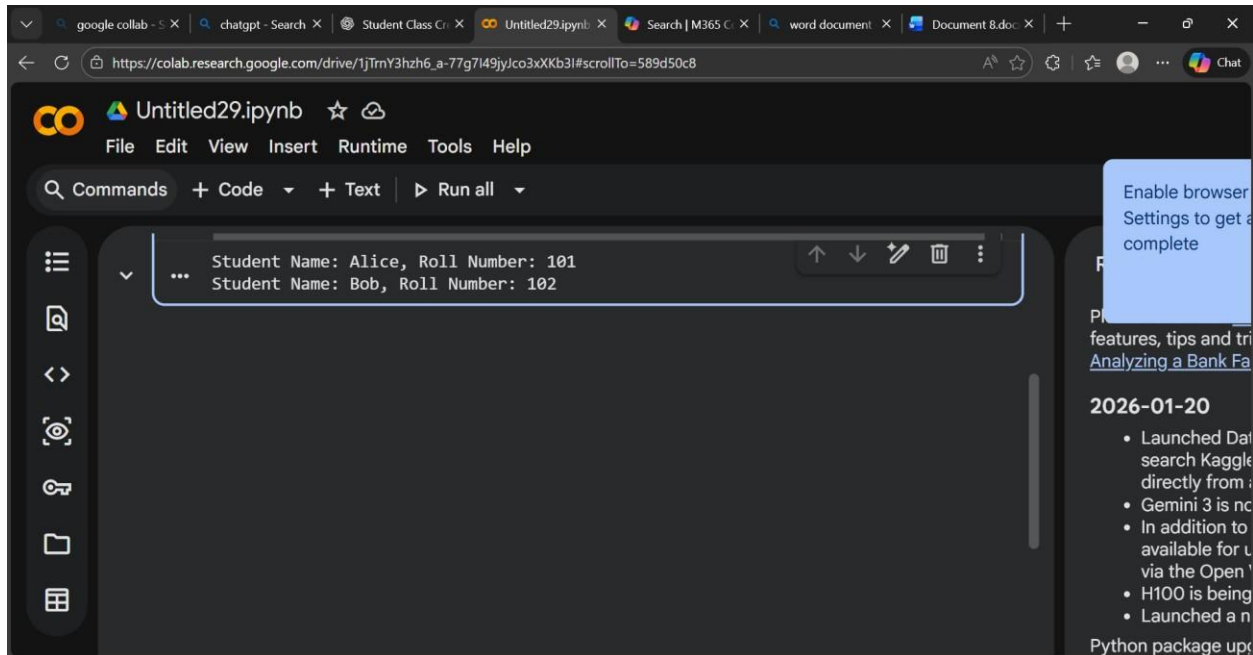
Input: Bug Code



Corrected code:



Output:



Explanation: The constructor was missing the `self` parameter, which is required to refer to the object instance.

Using `self.name` and `self.roll` stores values inside the object properly. Task 5:

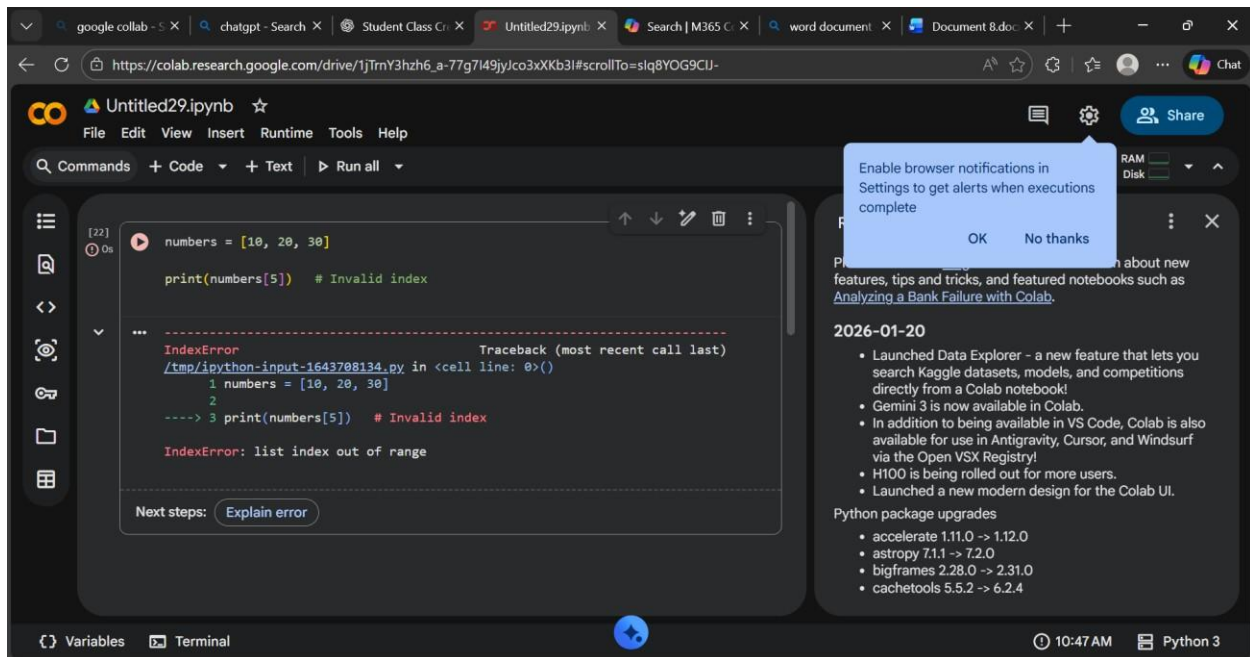
Resolving Index Errors in Lists

Prompt: This Python code causes an `IndexError`. Identify the issue, correct the code using safe access methods, and explain the problem.

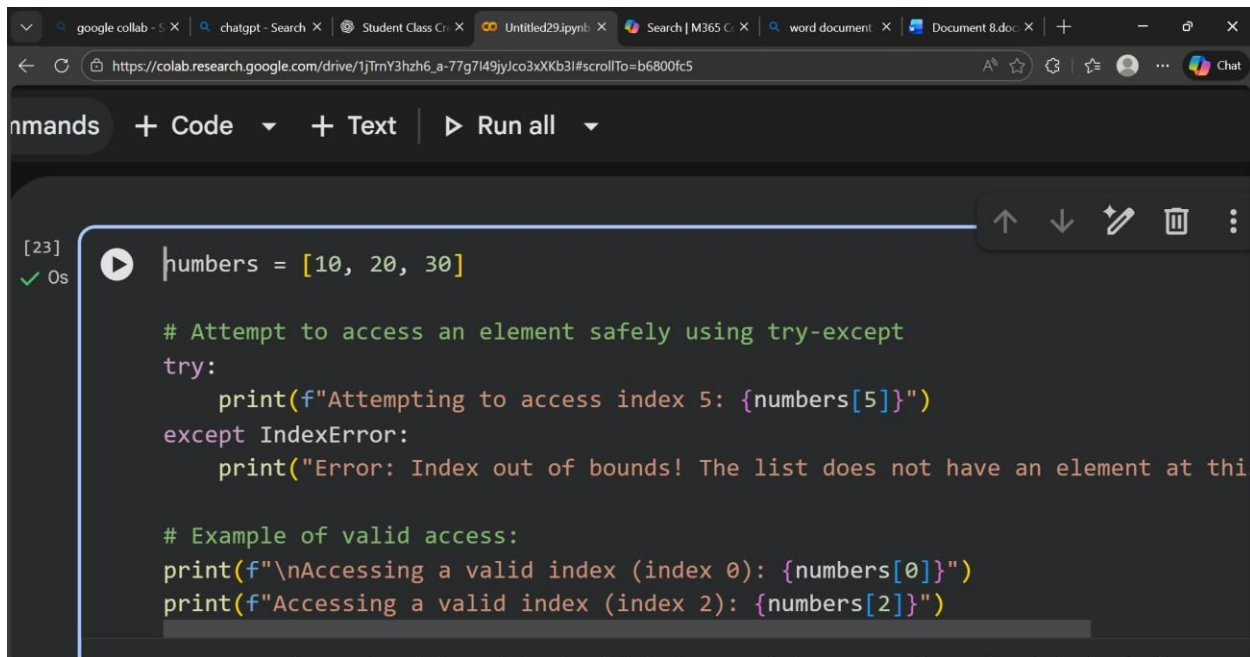
```
numbers = [10, 20, 30]
```

```
(numbers[5])
```

Input: Bug code



Corrected Code:



Output:

```
Accessing a valid index (index 0): 10
Accessing a valid index (index 2): 30
```

Explanation: The program tried to access an index that does not exist in the list, causing an `IndexError`.

Using `len()` to check bounds prevents the program from crashing.