

# Assignment -7.3

Name: Shaik Nishaaj

2303A51613

Btach:06

## Ask 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:

The screenshot shows a Google Colab notebook titled "Untitled29.ipynb". In cell [13], there is a syntax error in the code:

```
def add(a, b)
    return a + b
... File "/tmp/ipython-input-676827692.py", line 1
        def add(a, b)
                           ^
SyntaxError: expected ':'
```

A tooltip "Next steps: Explain error" is visible below the error message. To the right of the notebook area, there is a "Release notes" sidebar for January 2026, which includes sections for "New features", "Gemini 3", "VS Code", "Antigravity", "Cursor", "Windsurf", "H100", and "Python package upgrades". The sidebar also mentions the launch of Data Explorer and updates to accelerate, astropy, bigframes, and cachetools.

2) corrected code:

The screenshot shows a Google Colab notebook titled "Untitled29.ipynb". In the code cell, there is a function definition:

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

# Example usage:
result = add(10, 20)
print(f"The sum is: {result}")

... The sum is: 30
```

Below the code cell are buttons for "+ Code" and "+ Text". To the right of the code cell is a "Release notes" sidebar. The sidebar header says "Release notes" and includes a link to a blog post. The main content of the sidebar is dated "2026-01-20" and lists several updates:

- Launched Data Explorer - a new feature that lets you search Kaggle datasets, models, and competitions directly from a Colab notebook!
- Gemini 3 is now available in Colab.
- In addition to being available in VS Code, Colab is also available for use in Antigravity, Cursor, and Windsurf via the Open VSX Registry!
- H100 is being rolled out for more users.
- Launched a new modern design for the Colab UI.

Below this, under "Python package upgrades", there is a list:

- accelerate 1.11.0 → 1.12.0
- astropy 7.1.1 → 7.2.0
- bigframes 2.28.0 → 2.31.0
- cachetools 5.5.2 → 6.2.4

At the bottom of the sidebar, it says "10:28 AM" and "Python 3".

Output:

The screenshot shows the output of the code execution. The text "The sum is: 30" is displayed in a code cell, indicating that the AI has corrected the missing colon in the function definition.

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:

The screenshot shows a Google Colab notebook titled "Untitled29.ipynb". In the code cell, there is an infinite loop:

```
i = 1
while i <= 5:
    print(i)
    i -= 1 # Wrong update causing infinite loop
```

The output of the cell shows the numbers 1 through 5 being printed sequentially. A tooltip from the Colab UI highlights the line `i -= 1` with the note "# Wrong update causing infinite loop". The Colab interface includes a sidebar with a list of recent notebooks and a "Release notes" panel on the right side.

Corrected code:

A screenshot of a Google Colab notebook titled "Untitled29.ipynb". The code cell contains the following Python script:

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

print("Loop finished.")
```

The output pane shows the results of the execution:

```
1
2
3
4
5
Loop finished.
```

Output:

A screenshot of a Google Colab notebook showing the output of the previous code execution. The output pane displays the results:

```
1
2
3
4
5
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

A screenshot of a Google Colab notebook titled "Untitled29.ipynb". The code defines a simple division function:

```
def divide(a, b):
    return a / b

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

The output shows a `ZeroDivisionError`:

```
ZeroDivisionError: division by zero
```

A tooltip message is displayed in the top right corner: "Enable browser notifications in Settings to get alerts when executions complete".

## Corrected Code:

A screenshot of a Google Colab notebook titled "Untitled29.ipynb". The code now includes exception handling to catch `ZeroDivisionError`:

```
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}")
```

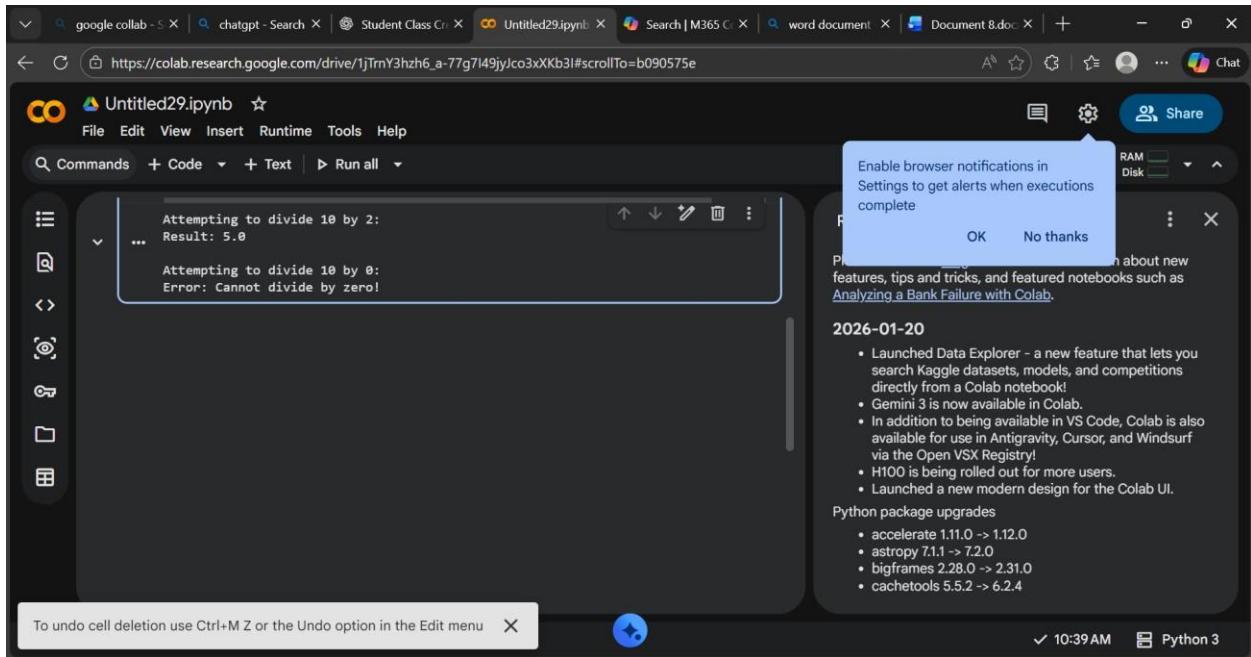
The output shows the results of running the corrected code:

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

Attempting to divide 10 by 0:
Error: Cannot divide by zero!
```

A tooltip message is displayed in the top right corner: "Enable browser notifications in Settings to get alerts when executions complete".

## 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

A screenshot of a Google Colab notebook titled "Untitled29.ipynb". The code cell contains:

```
[28] class Student:  
    def __init__(name, roll):  
        name = name  
        roll = roll
```

A tooltip is displayed in the top right corner, showing a message about browser notifications and a link to "Analyzing a Bank Failure with Colab".

Corrected code:

A screenshot of a Google Colab notebook titled "Untitled29.ipynb". The code cell now contains the corrected code:

```
[21] class Student:  
    # Corrected constructor: 'self' is the first parameter  
    def __init__(self, name, roll):  
        self.name = name # Assign 'name' to the instance's 'name' attribute  
        self.roll = roll # Assign 'roll' to the instance's 'roll' attribute  
  
    def display_student_info(self):  
        print(f"Student Name: {self.name}, Roll Number: {self.roll}")  
  
    # Example usage:  
student1 = Student("Alice", 101)  
student1.display_student_info()  
  
student2 = Student("Bob", 102)  
student2.display_student_info()
```

The output of the code is shown below the code cell.

Output:

```
class Student:
    def __init__(self, name, roll):
        self.name = name
        self.roll = roll

student = Student("Alice", 101)
```

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

The screenshot shows a Google Colab interface. In cell [22], there is an error message:

```
numbers = [10, 20, 30]
print(numbers[5]) # Invalid index

...
IndexError: list index out of range
```

A tooltip provides a link to "Explain error". On the right side of the screen, a notification window is open, asking if the user wants to enable browser notifications. It also displays a news item about Data Explorer and Python package upgrades.

Corrected Code:

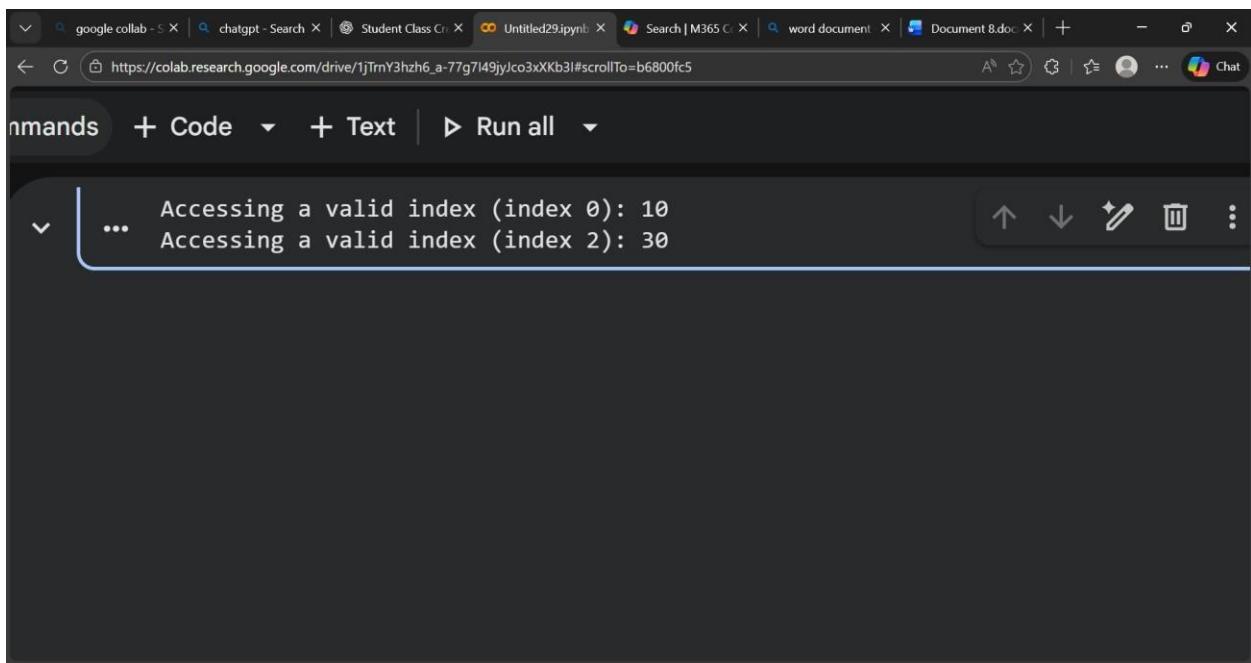
The screenshot shows a Google Colab interface. In cell [23], the code has been corrected to handle index errors safely:

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

# Attempt to access an element safely using try-except
try:
    print(f"Attempting to access index 5: {numbers[5]}")
except IndexError:
    print("Error: Index out of bounds! The list does not have an element at this index.")

# Example of valid access:
print(f"\nAccessing a valid index (index 0): {numbers[0]}")
print(f"Accessing a valid index (index 2): {numbers[2]}")
```

Output:



The screenshot shows a Jupyter Notebook interface in Google Colab. At the top, there are several tabs: "google collab - S X", "chatgpt - Search X", "Student Class Cr X", "Untitled29.ipynb X", "Search | M365 C X", "word document X", "Document 8.doc X", and a "+" button. Below the tabs, the URL is https://colab.research.google.com/drive/1jTrnY3hzh6\_a-77g7I49yJco3xXKb3l#scrollTo=b6800fc5. The toolbar includes "Commands", "+ Code", "+ Text", and "Run all". The main area displays two lines of code output: "Accessing a valid index (index 0): 10" and "Accessing a valid index (index 2): 30". To the right of the output, there are edit icons: up, down, edit, delete, and more.

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.