

AI ASSISTED CODING

T . ANANYA

2303A51128

BATCH – 03

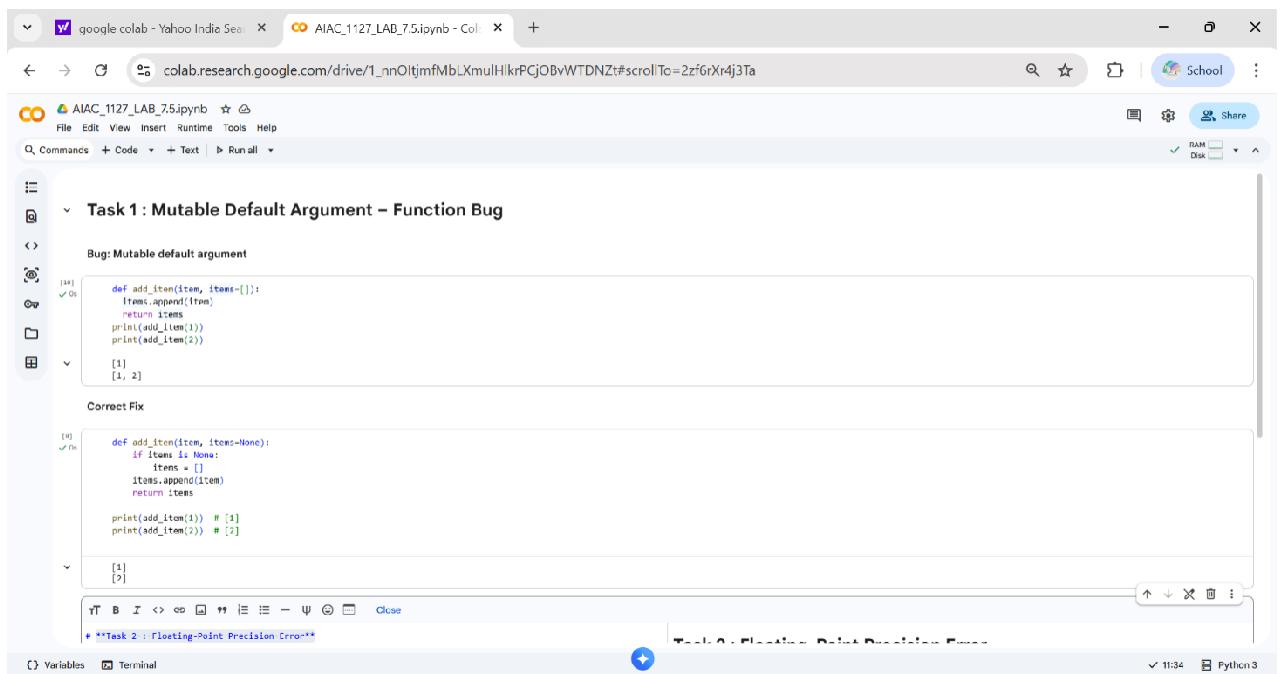
06 – 02 – 2026

ASSIGNMENT – 7.5

Lab 7: Error Debugging with AI: Systematic approaches to finding and fixing bugs

TASK - 01 : Mutable Default Argument – Function Bug

ERROR AND FIXED CODE:



The screenshot shows a Google Colab notebook titled "AIAC_1127_LAB_7.5.ipynb". The code cell contains the following Python code:

```
def add_item(item, items=[]):
    items.append(item)
    return items
print(add_item(1))
print(add_item(2))
```

The output shows the original code's bug: both print statements result in [1, 2]. Below the code, a "Correct Fix" section shows the modified code:

```
def add_item(item, items=None):
    if items is None:
        items = []
    items.append(item)
    return items

print(add_item(1)) # [1]
print(add_item(2)) # [2]
```

The output of the correct fix shows the expected behavior: the first print statement is [1] and the second is [2].

Explanation : Using None instead of a mutable default argument creates a new list on every function call and avoids shared data issues.

Task 2: Floating-Point Precision Error

ERROR AND FIXED CODE:

The screenshot shows a Google Colab notebook titled "AIAC_1127_LAB_7.5.ipynb". The code cell [11] contains:`def check_sum():
 return (0.1 + 0.2) == 0.3
print(check_sum())`

The output is "False". The code cell [12] contains:`def check_sum():
 return abs((0.1 + 0.2) - 0.3) < 1e-9
print(check_sum())`

The output is "... True".

Explanation: Floating-point values are compared using a tolerance (or `math.isclose`) instead of direct equality to handle precision errors.

Task 3: Recursion Error – Missing Base Case

ERROR AND FIXED CODE :

The screenshot shows a Google Colab notebook titled "AIAC_1127_LAB_7.5.ipynb". The code cell [11] contains:`def countdown(n):
 print(n)
 return countdown(n-1)
Countdown(5)`

The output shows a stack overflow error with many recursive calls listed:`-992
-933
-934
-935
-936
-937
-938
-939
-940
-941
-942
-943
-944
-945
-946
-947
-948
-949
-950
-951
-952
-953
-954
-955
-956
-957
-958
-959
-960`

The screenshot shows a Google Colab notebook titled "AIAC_1127_LAB_7.5.ipynb". In the code editor, there is a recursive definition of a `countdown` function:

```

def countdown(n):
    if n < 0:
        return
    print(n)
    countdown(n - 1)

```

This results in a `RecursionError: maximum recursion depth exceeded`. Below the code, a "Correct Fix" section shows the addition of a base case:

```

def countdown(n):
    if n < 0:
        return
    print(n)
    countdown(n - 1)

countdown(5)

```

The output of this corrected code is:

```

5
4
3
2
1
0

```

Explanation: A base case is added to stop recursive calls and prevent infinite recursion.

Task 4: Dictionary Key Error

ERROR AND FIXED CODE:

The screenshot shows a Google Colab notebook titled "AIAC_1127_LAB_7.5.ipynb". In the code editor, there is a function `get_value` that tries to access a non-existing key in a dictionary:

```

def get_value():
    data = {"x": 1, "y": 2}
    return data["z"]
print(get_value())

```

This results in a `KeyError: 'z'`. Below the error, a "Correct Fix" section shows the addition of a check for the key's existence:

```

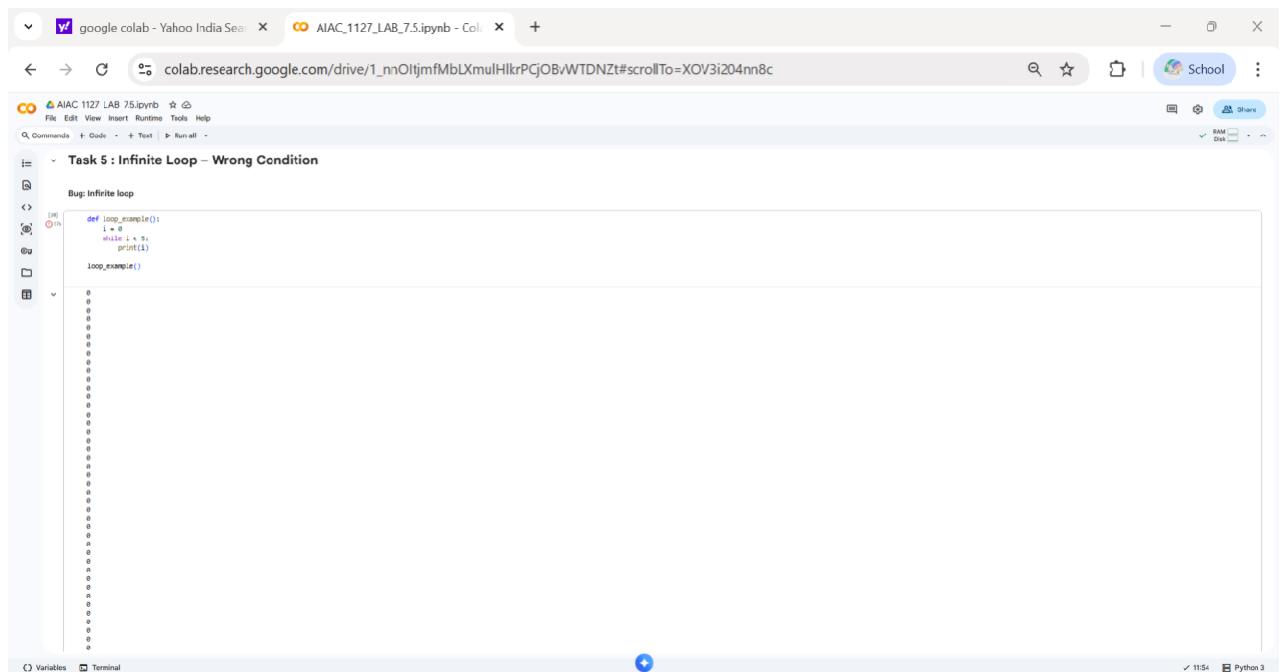
def get_value():
    data = {"x": 1, "y": 2}
    return data.get("z") # returns None if key not found
print(get_value())

```

Explanation: Using `dict.get()` or exception handling prevents `KeyError` when accessing missing dictionary keys.

Task 5: Infinite Loop – Wrong Condition

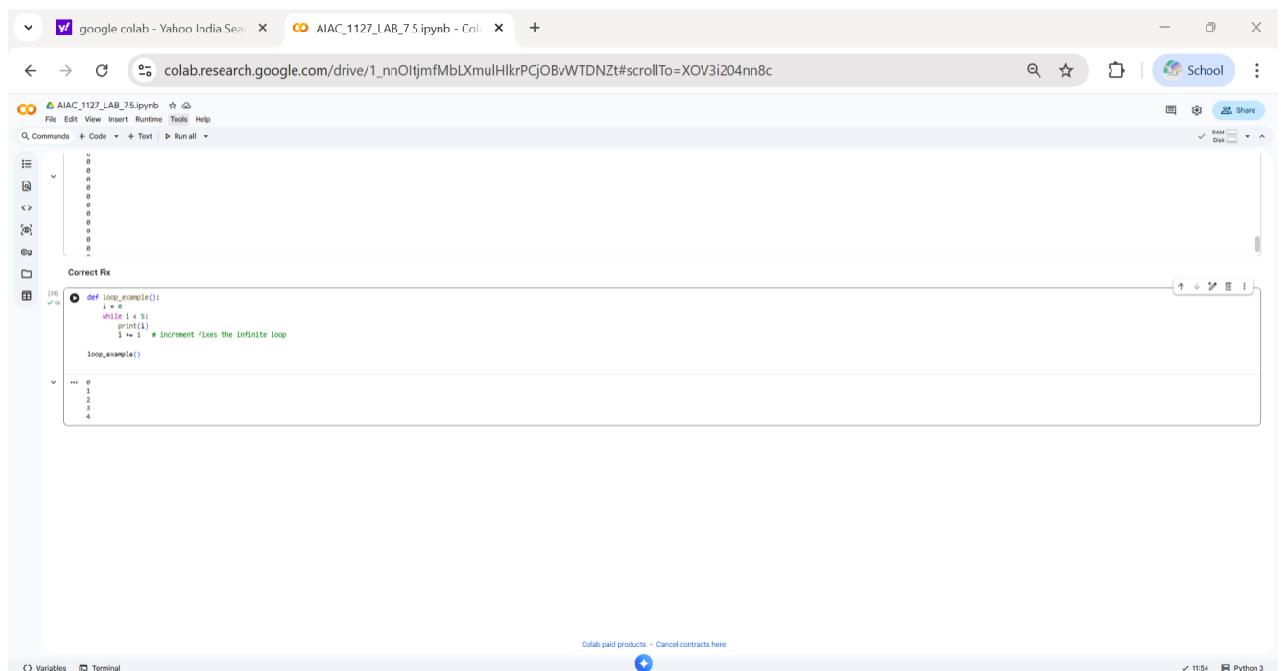
ERROR AND FIXED CODE:



The screenshot shows a Google Colab notebook titled "Task 5 : Infinite Loop – Wrong Condition". The code in the cell is:

```
def loop_example():
    i = 0
    while i < 5:
        print(i)
loop_example()
```

The output of the code is a continuous stream of zeros, indicating an infinite loop. The Colab interface includes tabs for "Variables" and "Terminal", and a status bar at the bottom right showing "11:54 Python 3".



The screenshot shows the same Google Colab notebook after a fix has been applied. The code in the cell is now:

```
def loop_example():
    i = 0
    while i < 5:
        print(i)
        i += 1 # increment fixes the infinite loop
loop_example()
```

The output of the code is the expected sequence of numbers: 0, 1, 2, 3, 4. The Colab interface includes tabs for "Variables" and "Terminal", and a status bar at the bottom right showing "11:54 Python 3".

Explanation: Incrementing the loop variable ensures the loop condition eventually becomes false.

TASK 6: Unpacking Error – Wrong Variables

ERROR AND FIXED CODE:

The screenshot shows a Google Colab notebook titled "AIAC_1127_LAB_7.5.ipynb". A section titled "Task 6 : Unpacking Error – Wrong Variables" contains the following code:

```
a, b = (1, 2, 3)
```

An error message is displayed:

```
ValueError
ValueError: too many values to unpack (expected 2)
```

The "Correct Fix" section shows the following code:

```
a, b, _ = (1, 2, 3)
print(a, b)
```

The output of the fixed code is:

```
1 2
```

Explanation: Correct unpacking is achieved by matching variable count or ignoring extra values using `_` or `*`.

Task 7: Mixed Indentation – Tabs vs Spaces

ERROR AND FIXED CODE:

The screenshot shows a Google Colab notebook titled "AIAC_1127_LAB_7.5.ipynb". The code in cell 24 is:`[24] def func():
 x = 5
 y = 10
 return x + y`

The output shows the error:`File "/tmp/ipython-Input-4084137757.py", line 2
 x = 5
 ^
IndentationError: expected an indented block after function definition on line 1`

The "Correct Fix" section shows the code with consistent space indentation:`[25] def func():
 x = 5
 y = 10
 return x + y

print(func())
... 15`

EXPLANATION: Consistent indentation using spaces fixes IndentationError and allows proper code execution.

Task 8: Import Error – Wrong Module Usage

ERROR AND FIXED CODE: Correcting the module name to math resolves the import error.

The screenshot shows a Google Colab notebook titled "AIAC_1127_LAB_7.5.ipynb". The code in cell 27 is:`[27] import maths
print(maths.sqrt(16))`

The output shows the error:`ModuleNotFoundError: No module named 'maths'

Traceback (most recent call last):
 File "/tmp/ipython-Input-1512532258.py", line 1, in <module>
 import maths
 ^
ModuleNotFoundError: No module named 'maths'

NOTE: If your import is failing due to a missing package, you can
manually install dependencies using either !pip or !apt.

To view examples of installing some common dependencies, click the
"Open Examples" button below.`

The "Correct Fix" section shows the code with the correct module name "math":`[28] import math
print(math.sqrt(16))
... 4.0`

Explanation: Correcting the module name to math resolves the import error.