

AI ASSISTED CODING

SUMANTH POLAM

2303A51121

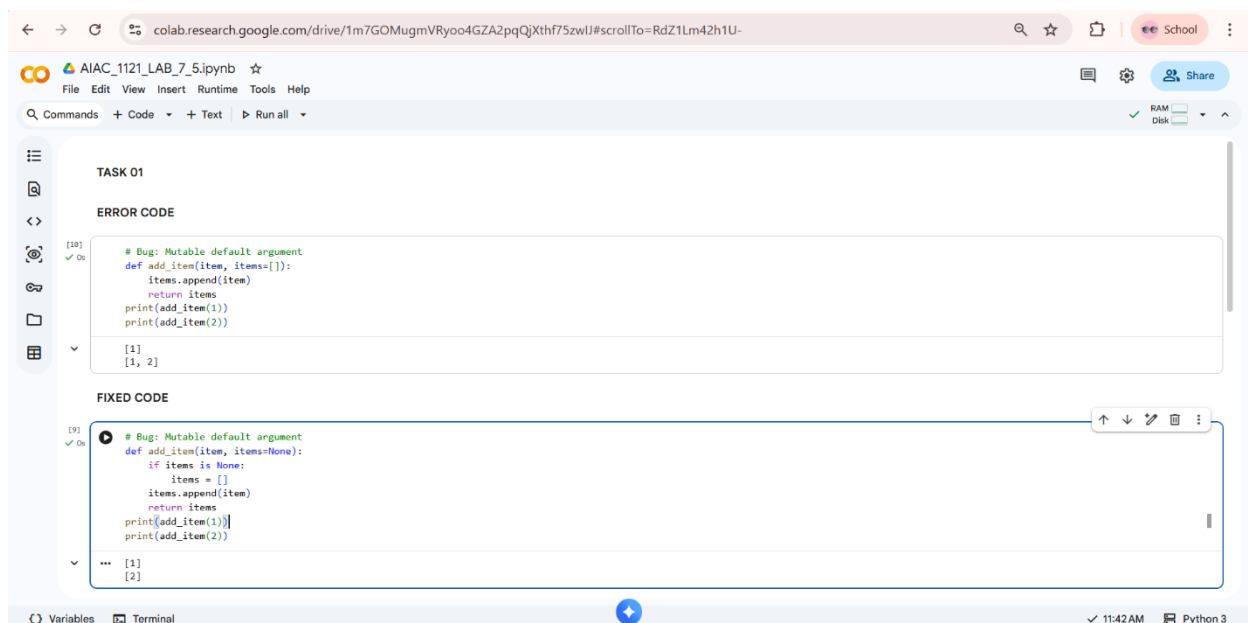
BATCH – 03

06 – 02 – 2026

ASSIGNMENT – 7.5

Lab – 07 : Error Debugging with AI : Systematic Approaches to finding and fixing bugs.

Task – 01 : Mutable Default Argument – Function Bug.



The screenshot shows a Google Colab notebook interface. The browser address bar displays a URL from colab.research.google.com. The notebook is titled 'AIAC_1121_LAB_7_5.ipynb'. The left sidebar contains icons for file explorer, code editor, and other tools. The main area is divided into two sections: 'TASK 01' and 'ERROR CODE'. The 'ERROR CODE' section shows a Python function `add_item` with a mutable default argument `items=[]`. The function appends an item to the list and returns it. The output shows the list `[1, 2]` after two calls. Below this, the 'FIXED CODE' section shows the same function but with a mutable default argument `items=None`. The function checks if `items` is `None` and initializes it to an empty list before appending the item. The output shows the list `[1, 2]` after two calls. The bottom status bar indicates 'Variables', 'Terminal', and 'Python 3'.

```
TASK 01

ERROR CODE

[120] ✓ On
# Bug: Mutable default argument
def add_item(item, items=[]):
    items.append(item)
    return items
print(add_item(1))
print(add_item(2))

[1]
[1, 2]

FIXED CODE

[19] ✓ On
# Bug: Mutable default argument
def add_item(item, items=None):
    if items is None:
        items = []
    items.append(item)
    return items
print(add_item(1))
print(add_item(2))

... [1]
[2]
```

Explanation : The above error occurs because the above list items are created only once and it is Reused.

Task – 02 : Floating – Point Precision Error.



The image shows a Google Colab notebook titled "AIAC_1121_LAB_7_5.ipynb". The notebook is divided into two sections: "TASK 02" and "ERROR CODE".

ERROR CODE:

```
[12] ✓ Os
# Bug: Floating point precision issue
def check_sum():
    return (0.1 + 0.2) == 0.3
print(check_sum())

False
```

FIXED CODE:

```
[13] ✓ Os
def check_sum():
    return abs((0.1 + 0.2) - 0.3) < 1e-9

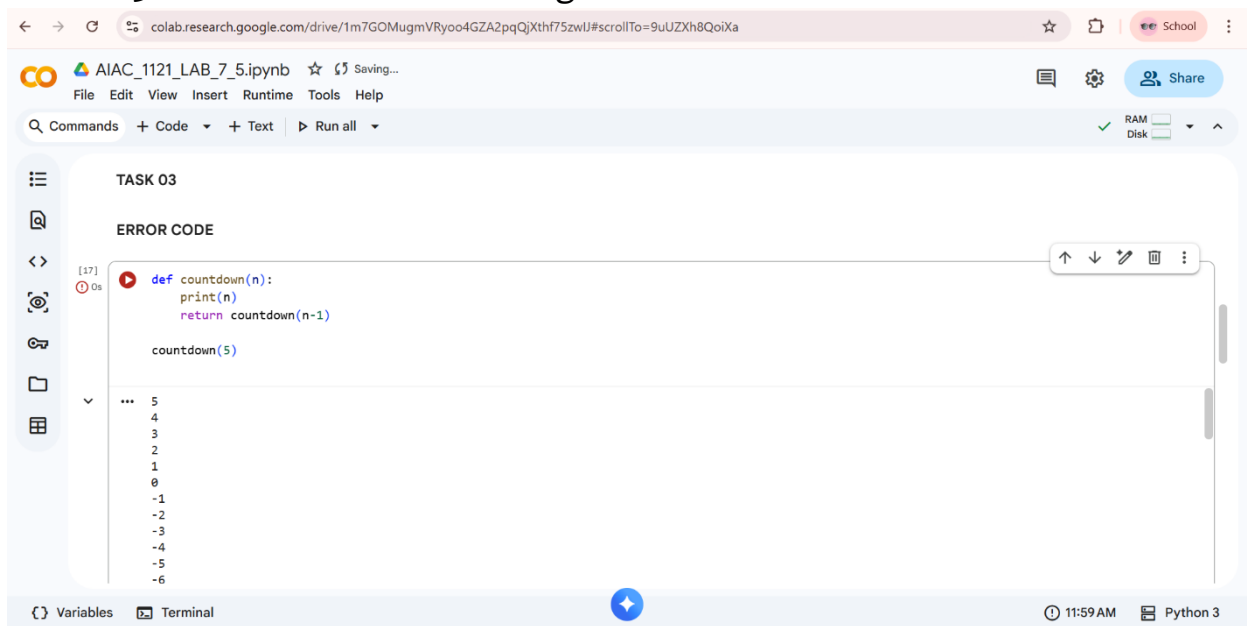
print(check_sum())

... True
```

The bottom of the notebook shows the status bar with "Variables", "Terminal", "11:50 AM", and "Python 3".

Explanation : The above error occurs because Float – Point Numbers cannot be Compared Directly. So, it is Fixed Using Tolerance.

Task – 03 : Recursion Error – Missing base Case.



The image shows a Google Colab notebook titled "AIAC_1121_LAB_7_5.ipynb". The notebook is divided into two sections: "TASK 03" and "ERROR CODE".

ERROR CODE:

```
[17] Os
def countdown(n):
    print(n)
    return countdown(n-1)

countdown(5)

... 5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
```

The bottom of the notebook shows the status bar with "Variables", "Terminal", "11:59 AM", and "Python 3".

The screenshot shows a Google Colab notebook titled "AIAC_1121_LAB_7_5.ipynb". The initial code in the first cell is:

```
def countdown(n):
    return countdown(n-1)
countdown(5)
```

This code results in a `RecursionError: maximum recursion depth exceeded`. The notebook then shows the "FIXED CODE" in a second cell:

```
def countdown(n):
    if n <= 0: # Base case
        return
    print(n)
    countdown(n-1)
countdown(5)
```

Running the fixed code produces the output:

```
5
4
3
2
1
```

The bottom status bar indicates the runtime is at 11:59 AM using Python 3.

Explanation : The above error occurs because in the error code there is no stopping condition it has infinite loop. So, fixed it using loop condition.

Task – 04 : Dictionary key Errors.

The screenshot shows a Google Colab notebook titled "AIAC_1121_LAB_7_5.ipynb". The code in the first cell is:

```
def get_value():
    data = {"a": 1, "b": 2}
    return data["c"]
print(get_value())
```

This code results in a `KeyError` because the dictionary does not have a key 'c'. The notebook then shows the "ERROR CODE" in a second cell, which includes a traceback:

```
KeyError                                Traceback (most recent call last)
/tmp/ipython-input-1845996374.py in <cell line: 0>()
      3     return data["c"]
      4
----> 5 print(get_value())

/tmp/ipython-input-1845996374.py in get_value()
      1 def get_value():
      2     data = {"a": 1, "b": 2}
----> 3     return data["c"]
      4
```

The bottom status bar indicates the runtime is at 12:04 PM using Python 3.

The screenshot shows a Google Colab notebook titled 'AIAC_1121_LAB_7_5.ipynb'. The top toolbar includes 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help'. Below the toolbar is a 'Commands' bar with '+ Code', '+ Text', and 'Run all' buttons. The left sidebar contains icons for file explorer, search, and other tools. The main code cell shows a snippet of Python code that results in a `KeyError: 'c'` error. Below the error, a 'Next steps: Explain error' button is visible. The 'FIXED CODE' section shows the corrected code: a function `get_value()` that uses `data.get("c", "Key not found")` to safely access a dictionary value. The output of the fixed code is `Key not found`. The bottom status bar indicates 'Variables', 'Terminal', '12:04 PM', and 'Python 3'.

```
----> 3     return data["c"]
      4
      5 print(get_value())

KeyError: 'c'

Next steps: Explain error

FIXED CODE

[21] def get_value():
      data = {"a": 1, "b": 2}
      return data.get("c", "Key not found")

      print(get_value())

... Key not found
```

Explanation: The above Error occurs in the above code is key Error because it has accessing the key which is not existed. So, Fixed it using returning None value or Key Not Found Method.

Task – 05: Infinite Loop – Wrong Condition.

The screenshot shows a Google Colab notebook titled 'AIAC_1121_LAB_7_5.ipynb'. The top toolbar includes 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help'. Below the toolbar is a 'Commands' bar with '+ Code', '+ Text', and 'Run all' buttons. The left sidebar contains icons for file explorer, search, and other tools. The main code cell shows a snippet of Python code that results in an infinite loop error. Below the error, a 'Show hidden output' button is visible. The 'FIXED CODE' section shows the corrected code: a function `loop_example()` that increments the loop variable `i` by 1 in each iteration. The output of the fixed code is a list of numbers from 0 to 4. The bottom status bar indicates 'Variables', 'Terminal', '12:10 PM', and 'Python 3'.

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

... Show hidden output

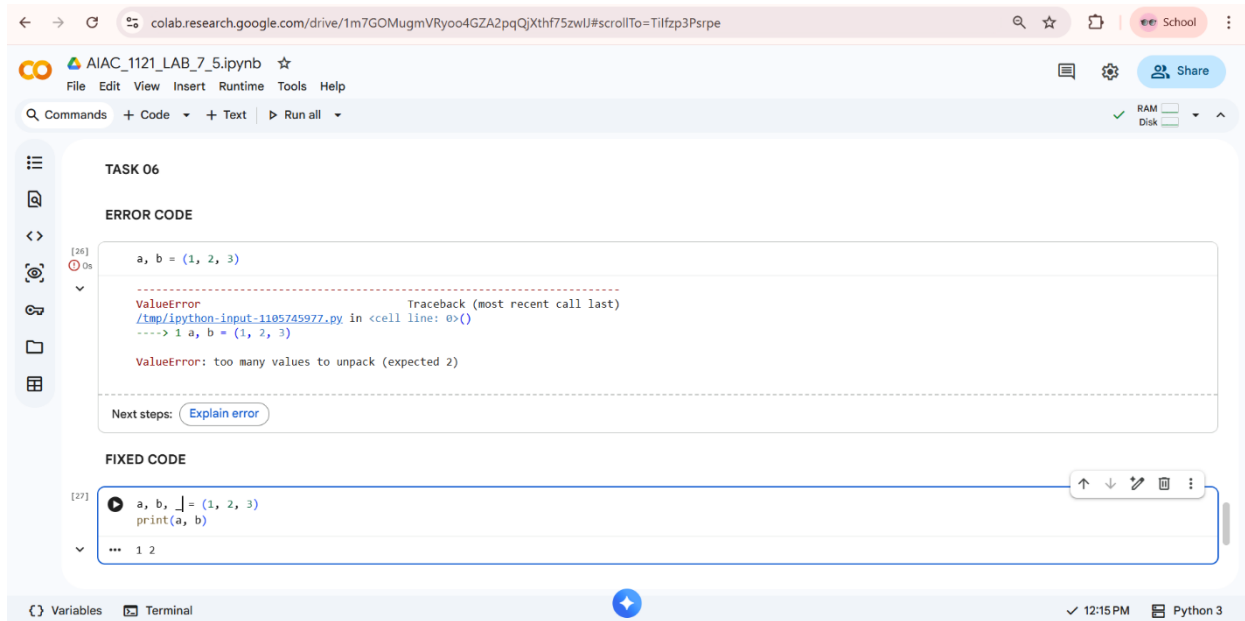
FIXED CODE

[25] def loop_example():
      i = 0
      while i < 5:
          print(i)
          i += 1 # Increment added
      loop_example()

      0
      1
      2
      3
      4
```

Explanation: The above Error occurs in the above code because in the above loop the “I” is never incremented in the loop variable. So, the code is fixed by incrementing the “I” in the loop.

Task – 06: Unpacking Error – Wrong variables.



The screenshot shows a Google Colab notebook titled "AIAC_1121_LAB_7_5.ipynb". The "TASK 06" section displays an "ERROR CODE" with a traceback for a `ValueError: too many values to unpack (expected 2)`. The error occurred in the following code cell:

```
[26] a, b = (1, 2, 3)
```

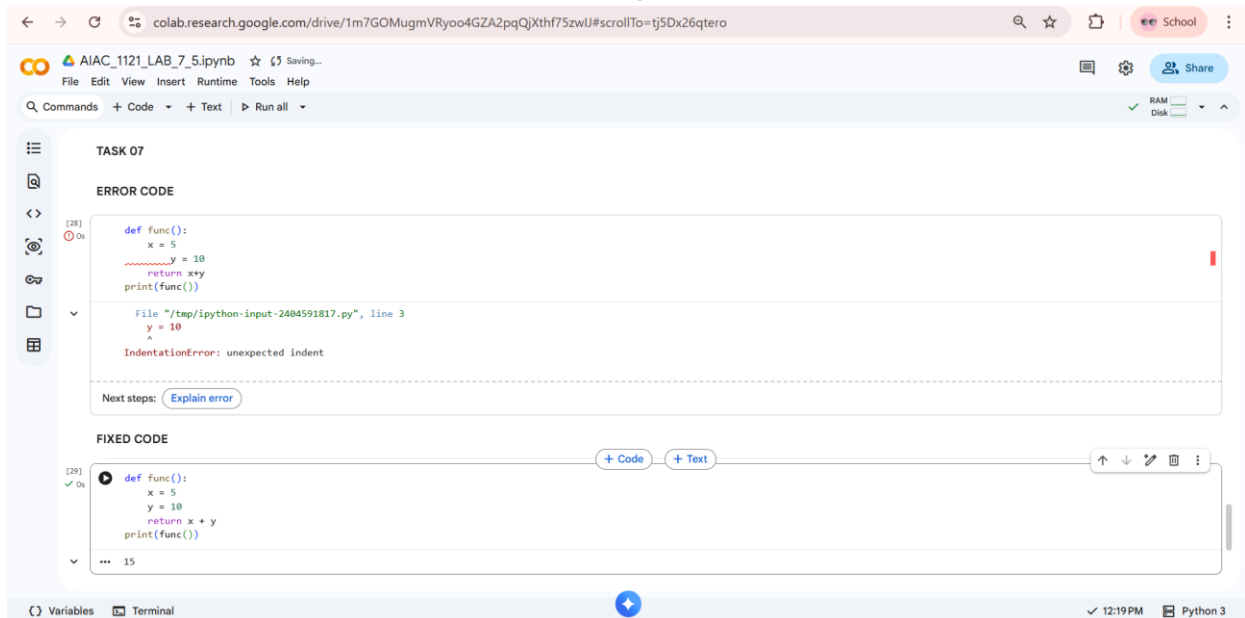
The "FIXED CODE" section shows the corrected code:

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

The output of the fixed code is `1 2`. The interface includes a left sidebar with navigation icons, a top menu bar, and a bottom status bar showing "Variables", "Terminal", and "Python 3".

Explanation: The above Error occurs in the above code because it has too many values to unpack. So, fixed the above code using packing method.

Task – 07: Mixed Indentation – Tabs vs Spaces.



The screenshot shows a Google Colab notebook titled "AIAC_1121_LAB_7_5.ipynb". The "TASK 07" section displays an "ERROR CODE" with a traceback for an `IndentationError: unexpected indent`. The error occurred in the following code cell:

```
[28] def func():
     x = 5
     y = 10
     return x+y
     print(func())
```

The "FIXED CODE" section shows the corrected code, where the indentation is consistent using spaces:

```
[29] def func():
     x = 5
     y = 10
     return x + y
     print(func())
```

The output of the fixed code is `15`. The interface includes a left sidebar with navigation icons, a top menu bar, and a bottom status bar showing "Variables", "Terminal", and "Python 3".

Explanation: The above error occurs in the above code because, python does not allow the mixing of tabs and spaces. So, it is fixed by consisting only 4 spaces systematically.

Task – o8: Import Error – Wrong Module Usage.

The first screenshot shows a Google Colab notebook titled 'AIAC_1121_LAB_7_5.ipynb'. The code cell contains the following Python code:

```
[30] -import maths
      -print(maths.sqrt(16))
      +import math
      +print(math.sqrt(16))
```

The code is highlighted in green, indicating it was executed. Below the code, a red error message is displayed:

```
ModuleNotFoundError: Traceback (most recent call last)
/tmp/ipython-input-1512532258.py in <cell line: 0>()
----> 1 import maths
      2 print(maths.sqrt(16))

ModuleNotFoundError: No module named 'maths'
```

Below the error message, there is a note: "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 second screenshot shows the same notebook after the error has been fixed. The code cell now contains:

```
[31] import math
      print(math.sqrt(16))
```

The code is highlighted in green, indicating it was executed. Below the code, the output is displayed:

```
4.0
```

Below the output, there is a button labeled "OPEN EXAMPLES".

Explanation: The above error occurs in the above code because the module name is Incorrect. So, fixed it by renaming the module name correctly.

THANK YOUU!!