

AI ASSISTED CODING

SHASHANK YELAGAM

2303A510i3

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 code editor interface with a sidebar on the left containing icons for Explorer, Search, Run and Debug, Source Control, Run and Test, and Extensions. The main editor area is titled 'TASK 01' and contains two sections: 'ERROR CODE' and 'FIXED 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. When called twice, it prints `[1]` and `[1, 2]`. The 'FIXED CODE' section shows the same function but with a mutable default argument `items=None`. Inside the function, it checks if `items is None` and initializes it as an empty list before appending the item. The output remains the same. The bottom status bar shows 'Variables', 'Terminal', a blue star icon, '11:42 AM', and 'Python 3'.

```
Commands + Code + Text Run all
```

TASK 01

ERROR CODE

```
[10] ✓ 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

```
[9] ✓ 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]
```

Variables Terminal 11:42 AM Python 3

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 screenshot shows a code editor interface with a sidebar on the left containing icons for file explorer, search, and other tools. The main area is titled 'TASK 02' and contains two sections: 'ERROR CODE' and 'FIXED CODE'. The 'ERROR CODE' section shows a Python function `check_sum()` that returns `(0.1 + 0.2) == 0.3`, which evaluates to `False`. The 'FIXED CODE' section shows the same function but with a tolerance check: `abs((0.1 + 0.2) - 0.3) < 1e-9`, which evaluates to `True`. The bottom status bar indicates 'Variables', 'Terminal', '11:50 AM', and 'Python 3'.

```
[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
```

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 screenshot shows a code editor interface with a sidebar on the left. The main area is titled 'TASK 03' and contains an 'ERROR CODE' section. It shows a Python function `countdown(n)` that prints `n` and returns `countdown(n-1)`. The function is called with `countdown(5)`. The output shows a sequence of numbers from 5 down to -6, indicating a recursion error because the base case is missing. The bottom status bar indicates 'Variables', 'Terminal', '11:59 AM', and 'Python 3'.

```
[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 screenshot shows a Jupyter Notebook interface. The top 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 main area displays a code cell with the following code:

```
----> 3     return countdown(n-1)
      4
      5     countdown(5)
...
RecursionError: maximum recursion depth exceeded
```

Below the error message, there is a button labeled 'Explain error'. The next section is titled 'FIXED CODE' and shows the corrected code:

```
[16]
def countdown(n):
    if n <= 0: # Base case
        return
    print(n)
    countdown(n-1)
countdown(5)
```

The output of the fixed code is shown below:

```
5
4
3
2
1
```

The bottom status bar shows 'Variables', 'Terminal', a blue star icon, '11:59 AM', and '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 Jupyter Notebook interface. The top 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 main area displays a code cell with the following code:

```
TASK 04
ERROR CODE
[22]
def get_value():
    data = {"a": 1, "b": 2}
    return data["c"]

print(get_value())
```

Below the code, there is a traceback for a `KeyError`:

```
...
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 shows 'Variables', 'Terminal', a blue star icon, '12:04 PM', and 'Python 3'.

The screenshot shows a code editor interface. At the top, there's a search bar and tabs for 'Commands', 'Code', 'Text', and 'Run all'. Below the tabs, the code editor displays a snippet of Python code with a `KeyError: 'c'` error. The code is as follows:

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

KeyError: 'c'
```

Below the error, there's a 'Next steps:' section with a button labeled 'Explain error'. Underneath, the 'FIXED CODE' section shows the corrected code:

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

      print(get_value())

... Key not found
```

At the bottom of the editor, there are tabs for 'Variables' and 'Terminal', a status bar showing '12:04 PM' and 'Python 3', and a blue circular icon.

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 code editor interface. At the top, there's a search bar and tabs for 'Commands', 'Code', 'Text', and 'Run all'. Below the tabs, the code editor displays a snippet of Python code with an infinite loop error. The code is as follows:

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

... Show hidden output
```

Below the error, there's a 'FIXED CODE' section showing the corrected code:

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

      0
      1
      2
      3
      4
```

At the bottom of the editor, there are tabs for 'Variables' and 'Terminal', a status bar showing '12:10 PM' and 'Python 3', and a blue circular icon.

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.

```
Commands + Code + Text Run all
```

TASK 06

ERROR CODE

```
[26] 0s
a, b = (1, 2, 3)

ValueError                                Traceback (most recent call last)
/tmp/ipynb-input-1105745977.py in <cell line: 0>()
----> 1 a, b = (1, 2, 3)

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

Next steps: [Explain error](#)

FIXED CODE

```
[27] 0s
a, b, _ = (1, 2, 3)
print(a, b)

... 1 2
```

Variables Terminal 12:15 PM 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.

```
Commands + Code + Text Run all
```

TASK 07

ERROR CODE

```
[28] 0s
def func():
    x = 5
    y = 10
    return x+y
print(func())

File ~/tmp/ipynb-input-2484591817.py, line 3
    y = 10
    ^
IndentationError: unexpected indent
```

Next steps: [Explain error](#)

FIXED CODE

```
[29] 0s
def func():
    x = 5
    y = 10
    return x + y
print(func())

... 15
```

Variables Terminal 12:19 PM Python 3

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

Task – 08: Import Error – Wrong Module Usage.

The image displays two screenshots of a Jupyter Notebook interface, likely from Google Colab, showing a Python error and its resolution.

Top Screenshot:

- TASK 08**
- ERROR CODE**
- Gemini** (AI assistant icon)
- Code Cell [30]:**

```
-import maths
-print(maths.sqrt(16))
+import math
+print(math.sqrt(16))
```
- 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'
```
- 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**

Bottom Screenshot:

- Code Cell [31]:**

```
import math
print(math.sqrt(16))
```
- Output:** 4.0
- Next steps:** [Explain error](#)
- FIXED ERROR**
- Toggle Gemini** (AI assistant icon)

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