

## Assignment-7.5

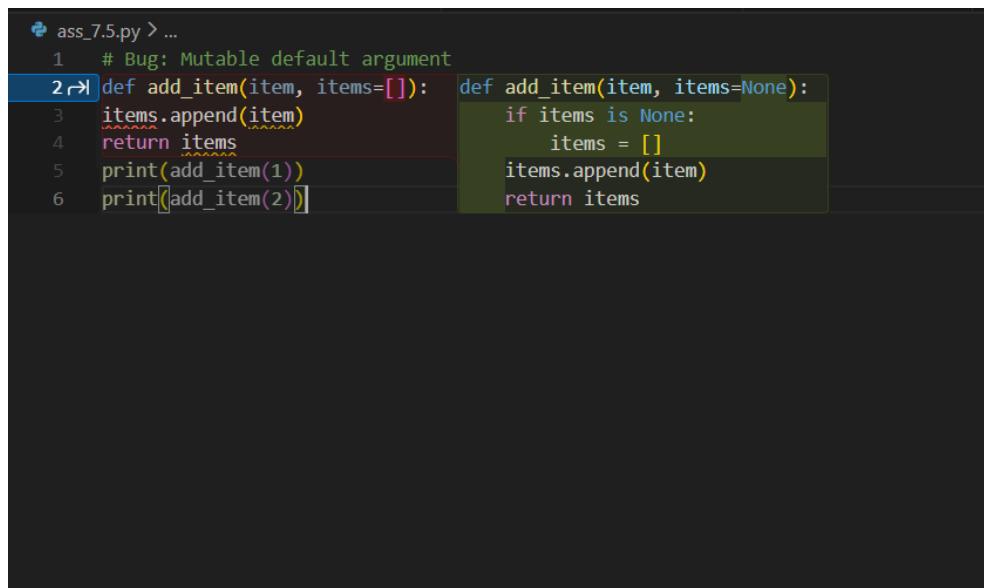
NAME:N.PAUL BENJAMIN

HNO:2303A51116

BATCH:03

1)INPUT:

```
ass_7.5.py > ...
1  # Bug: Mutable default argument
2 ↵ def add_item(item, items=[]):
3      items.append(item)
4      return items
5      print(add_item(1))
6      print(add_item(2))
```

A screenshot of a code editor window titled "ass\_7.5.py > ...". The code is written in Python. It defines a function "add\_item" that takes two parameters: "item" and "items". The "items" parameter is assigned a mutable default argument of an empty list `[]`. Inside the function, the "item" is appended to "items", and then "items" is returned. Two calls to the function are made: `print(add\_item(1))` and `print(add\_item(2))`. The code uses color-coded syntax highlighting where keywords like "def", "if", and "return" are in blue, variable names like "item" and "items" are in green, and strings are in red. A cursor is visible at the start of the second line of code.

```
ass_7.5.py > add_item
1 # Bug: Mutable default argument
2 def add_item(item, items=None):
3     if items is None:
4         items = []
5     items.append(item)
6     return items
7 print(add_item(1))
8 print(add_item(2))
```

OUTPUT:

```
ai_coding/ass_7.5.py
[1]
[2]
```

2) INPUT:

```
1 # Bug: Floating point precision issue
2 def check_sum():
3 → 3 return (0.1 + 0.2) == 0.3
4     return abs((0.1 + 0.2) - 0.3) < 1e-10 # Use a small tolerance for floating point comparison
5
6
7
8
9
```

```
# Bug: Floating point precision issue
def check_sum():
    return abs((0.1 + 0.2) - 0.3) < 1e-10 # Use a small tolerance for floating point comparison
print(check_sum())
```

OUTPUT:

```
True
```

3)INPUT:

```
ass_7.5.py > ...
3    print(n)
r→ 4 if n == 0:
      return
4 5 return countdown(n-1)
5 5 countdown(5)
```

```
# Bug: No base case
def countdown(n):
    if n == 0:
        return
    print(n)
    countdown(n-1)
countdown(5)
```

OUTPUT:

```
5
4
3
2
1
```

4)INPUT:

```
⚡ ass_7.5.py > ...
1  def get_value():
2      data = {"a": 1, "b": 2}
3
4 →| return data["c"]
5      return data.get("c", "Key not found")
6
7  print(get_value())
```

```
⚡ ass_7.5.py > ⚡ get_value
1  # Bug: Accessing non-existing key
2  def get_value():
3      data = {"a": 1, "b": 2}
4      return data.get("c", "Key not found")
5
6  print(get_value())
```

OUTPUT:

```
| Key not found
```

5) INPUT:

```
ass_7.5.py > [i]
1 # Bug: Infinite loop
2 def loop_example():
3     i = 0
4     while i < 5:
5         print(i)
→       i += 1
```

```
Go Run Terminal Help ← → Q Ai codin
Welcome # AI-Generated Logic Without Modularizat.py ass_3.2.py
ass-7.5.py > ...
1 def loop_example():
2     i = 0
3     while i < 5:
4         print(i)
5         i += 1    # Increment added
6
7 loop_example()
8 |
```

OUTPUT:

```
0
1
2
3
4
```

6) INPUT:

# Bug: Wrong unpacking

a, b = (1, 2, 3)

Expected Output: Correct unpacking or using \_ for extra values.

A screenshot of a code editor interface. The top bar includes 'Go', 'Run', 'Terminal', and 'Help' buttons, along with navigation arrows and a search bar containing 'Ai cod'. Below the bar, there are tabs for 'Welcome', '# AI-Generated Logic Without Modularizat.py', and 'ass\_3.2.py'. The main area shows a file named 'ass-7.5.py' with the following content:

```
ass-7.5.py > ...
1 a, b, _ = (1, 2, 3)
2 print(a, b)
3 |
```

OUTPUT:

A terminal window showing the output of the Python code. The output is '1 2', where '1' is blue and '2' is white.

7) INPUT:

```
# Bug: Mixed indentation
def func():
    x = 5
    y = 10
    return x+y
```

Expected Output : Consistent indentation applied.

```
Go Run Terminal Help ← → Search Ai cod
Welcome # AI-Generated Logic Without Modulariz...
ass-7.5.py > ...
1 def func():
2     x = 5
3     y = 10
4     return x + y
5
6 print(func())
7
```

OUTPUT:

```
15
```

8)INPUT:

```
# Bug: Wrong import
import maths
print(maths.sqrt(16))
```

Expected Output: Corrected to import math

```
Go Run Terminal Help ← → Search Ai
Welcome # AI-Generated Logic Without Modulariz...
ass-7.5.py
1 import math
2
3 print(math.sqrt(16))
4
```

OUTPUT:

4.0