

# ASSIGNMENT-7.5

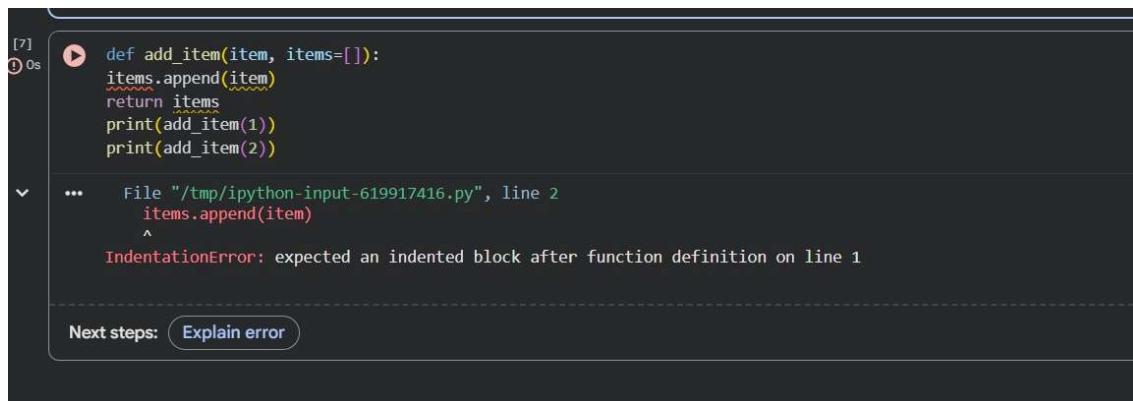
RISHAAK

2303A51125

BATCH NO: 03

## TASK 1: Mutable Default Argument – Function Bug

### ACTUAL CODE:



The screenshot shows a code editor window with Python code. The code defines a function `add_item` that appends an item to a mutable default argument `items`. The code is as follows:

```
[7] ① 0s  def add_item(item, items=[]):
        items.append(item)
        return items
    print(add_item(1))
    print(add_item(2))

...   File "/tmp/ipython-input-619917416.py", line 2
        items.append(item)
        ^
IndentationError: expected an indented block after function definition on line 1
```

A tooltip at the bottom left says "Next steps: Explain error".

**Prompt:** Analyze the Python function where a mutable default argument causes shared state between function calls. Fix the bug so each call uses a new list.

### CORRECTED CODE:

The screenshot shows a Jupyter Notebook interface with a dark theme. The title bar says "AIAC\_ASS.7.5\_1125.ipynb". The 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 code cell [1] contains the following Python code:

```
[1] 0s
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 issue occurs because a mutable object (list) is used as a default argument. In Python, default arguments are created once and reused across function calls, which leads to unexpected shared data. Each function call modifies the same list, causing incorrect results.

### TASK 2: Floating-Point Precision Error

#### ACTUAL CODE:

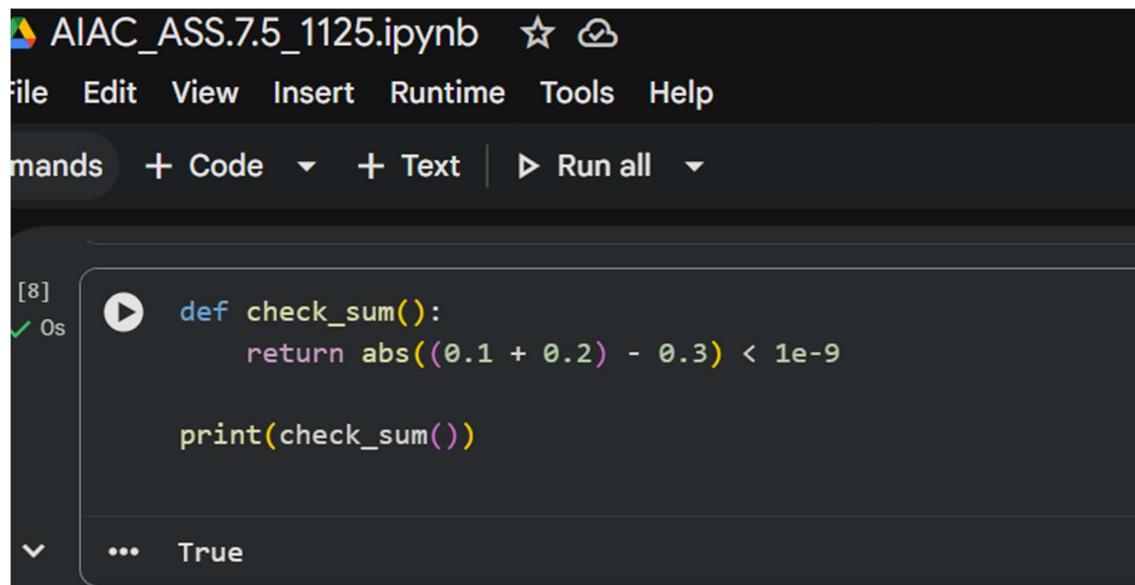
```
[7] ① 0s ⏎ def add_item(item, items=[]):
    items.append(item)
    return items
    print(add_item(1))
    print(add_item(2))

...
File "/tmp/ipython-input-619917416.py", line 2
    items.append(item)
    ^
IndentationError: expected an indented block after function definition on line 1
```

Next steps: Explain error

**Prompt:** Identify why direct floating-point comparison fails in Python and correct the function using an appropriate tolerance-based comparison.

## CORRECTED CODE:



A screenshot of a Jupyter Notebook interface. The title bar shows "AIAC\_ASS.7.5\_1125.ipynb". The menu bar includes File, Edit, View, Insert, Runtime, Tools, and Help. Below the menu is a toolbar with buttons for Commands, Code, Text, Run all, and Cell. The code cell [8] contains the following Python code:

```
[8] ② 0s ⏎ def check_sum():
    return abs((0.1 + 0.2) - 0.3) < 1e-9

print(check_sum())

...
True
```

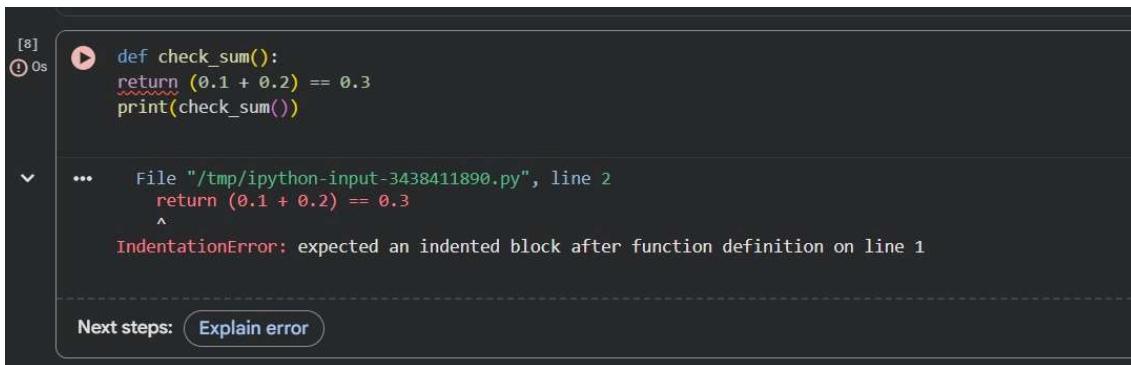
## Explanation:

Floating-point numbers are stored in binary form, which can cause small precision errors. Direct equality

comparison fails because the computed result is not exactly equal to the expected value. Using a tolerance-based comparison avoids this issue.

## TASK 3: Recursion Error – Missing Base Case

### ACTUAL CODE:



The screenshot shows a code editor interface with the following code in a file named `/tmp/ipython-input-3438411890.py`:

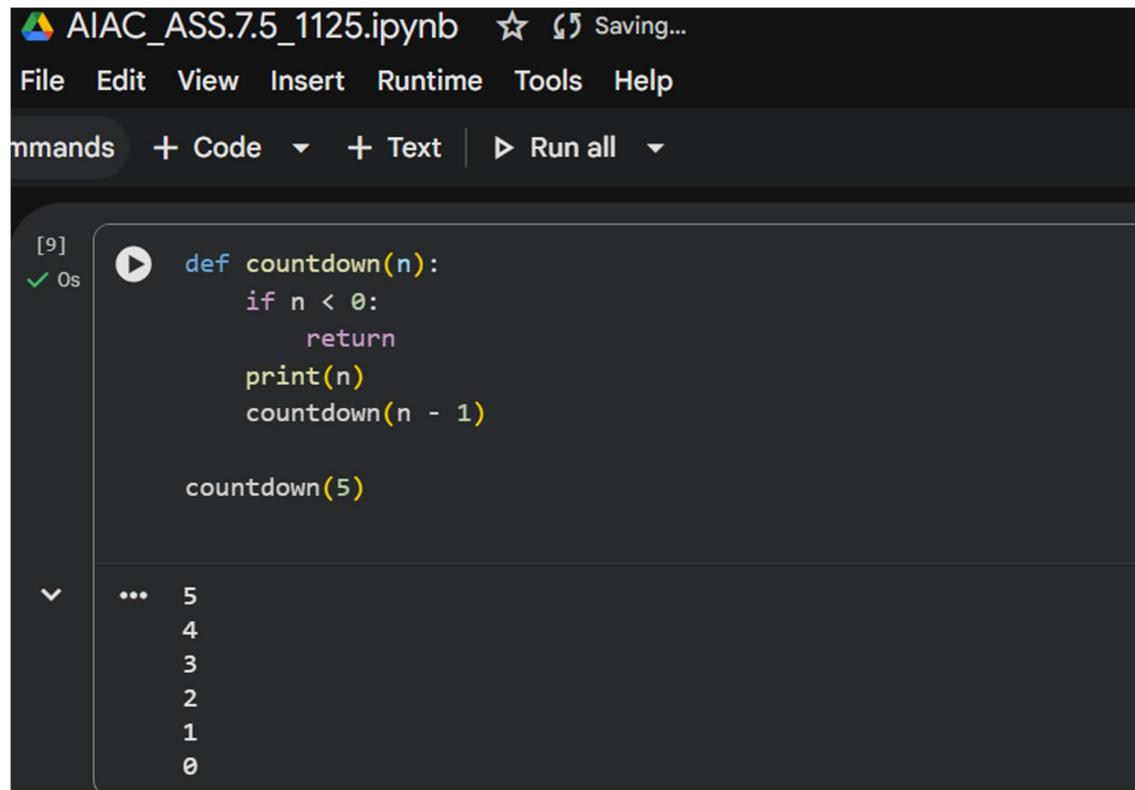
```
[8] ① 0s
  def check_sum():
    return (0.1 + 0.2) == 0.3
    print(check_sum())

...   File "/tmp/ipython-input-3438411890.py", line 2
        return (0.1 + 0.2) == 0.3
        ^
IndentationError: expected an indented block after function definition on line 1
```

Below the code, there is a button labeled "Next steps: Explain error".

**Prompt:** Debug the recursive function that runs infinitely due to a missing base case. Add a proper stopping condition

### CORRECTED CODE:



```
[9] ✓ 0s
    def countdown(n):
        if n < 0:
            return
        print(n)
        countdown(n - 1)

    countdown(5)

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

**Explanation:** The recursive function lacks a base case, so it keeps calling itself indefinitely. This leads to infinite recursion and eventually a runtime error. Adding a proper stopping condition ensures safe termination.

#### **TASK 4:** Dictionary Key Error

**ACTUAL CODE:**

```
[6] ⚡ 0s ⚡ def countdown(n):
    print(n)
    return countdown(n-1)
countdown(5)

...
File "/tmp/ipython-input-782688475.py", line 3
    return countdown(n-1)
          ^
IndentationError: unexpected indent
```

Next steps: [Explain error](#)

**Prompt:** Fix the function that raises a KeyError when accessing a non-existing dictionary key by using safe access or error handling.

## CORRECTED CODE:

```
AIAC_ASS.7.5_1125.ipynb ⚡ Saving...
File Edit View Insert Runtime Tools Help
Commands + Code + Text ▶ Run all ▾

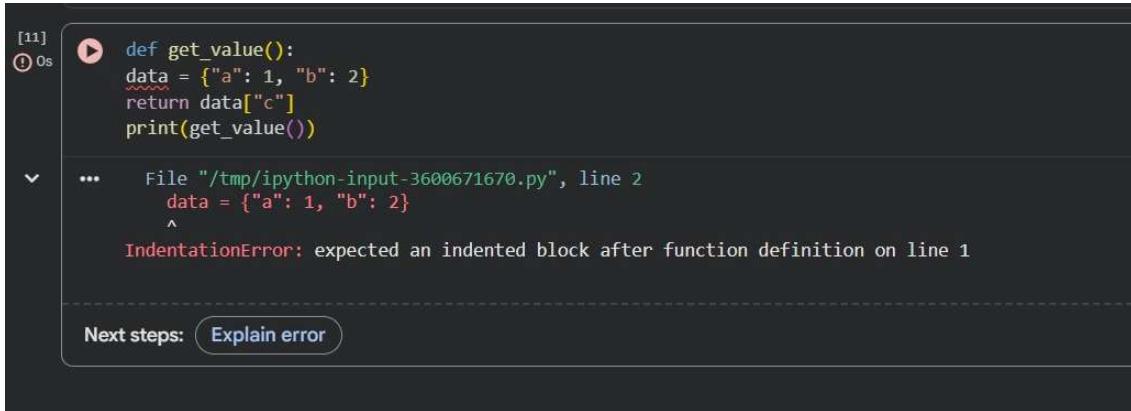
[10] ✓ 0s ⚡ def get_value():
    data = {"a": 1, "b": 2}
    return data.get("c", "Key not found")

print(get_value())
...
... Key not found
```

**Explanation:** Accessing a key that does not exist in a dictionary raises a `KeyError`. Using safe access methods or handling missing keys prevents the program from crashing.

## TASK 5: Infinite Loop – Wrong Condition

### ACTUAL CODE:



The screenshot shows a code editor interface with a dark theme. A code editor window displays the following Python code:

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

...   File "/tmp/ipython-input-3600671670.py", line 2
        data = {"a": 1, "b": 2}
        ^
IndentationError: expected an indented block after function definition on line 1
```

A tooltip box is overlaid on the code, pointing to the line `return data["c"]`. The tooltip contains the text:

Next steps: Explain error

**Prompt:** Detect and correct the infinite loop caused by an incorrect loop condition so the loop terminates properly.

### CORRECTED CODE:

```
[11] 0s
▶ def loop_example():
    i = 0
    while i < 5:
        print(i)
        i += 1

    loop_example()

▼ ... 0
   1
   2
   3
   4
```

**Explanation:** The loop condition is correct, but the loop variable is never updated. This causes the loop to run endlessly. Incrementing the loop variable allows proper termination.

## TASK 6: Unpacking Error – Wrong Variables

### ACTUAL CODE:

The screenshot shows a code editor interface with a dark theme. A code cell at index [12] contains the following Python code:

```
[12] ① 0s def loop_example():
    i = 0
    while i < 5:
        print(i)

...   File "/tmp/ipython-input-3417722996.py", line 2
        i = 0
              ^
IndentationError: expected an indented block after function definition on line 1
```

Below the code cell, there is a button labeled "Next steps: Explain error".

**Prompt:** Analyze the tuple unpacking error caused by mismatched variables and fix it using proper unpacking.

### CORRECTED CODE:

The screenshot shows a Jupyter Notebook interface with a dark theme. A code cell at index [12] contains the following Python code:

```
[12] ② 0s a, b, _ = (1, 2, 3)
      print(a, b)

...   1 2
```

**Explanation:** Tuple unpacking fails when the number of variables does not match the number of values. Correct unpacking or ignoring extra values resolves the error.

### TASK 7: Mixed Indentation – Tabs vs Spaces

#### ACTUAL CODE:

```
[14] ① Os
    def func():
        x = 5
        y = 10
        return x+y

    ...
    File "/tmp/ipython-input-1176682017.py", line 2
        x = 5
        ^
IndentationError: expected an indented block after function definition on line 1

Next steps: Explain error
```

**Prompt:** Correct the Python function that fails due to mixed or incorrect indentation by applying consistent indentation.

## CORRECTED CODE:

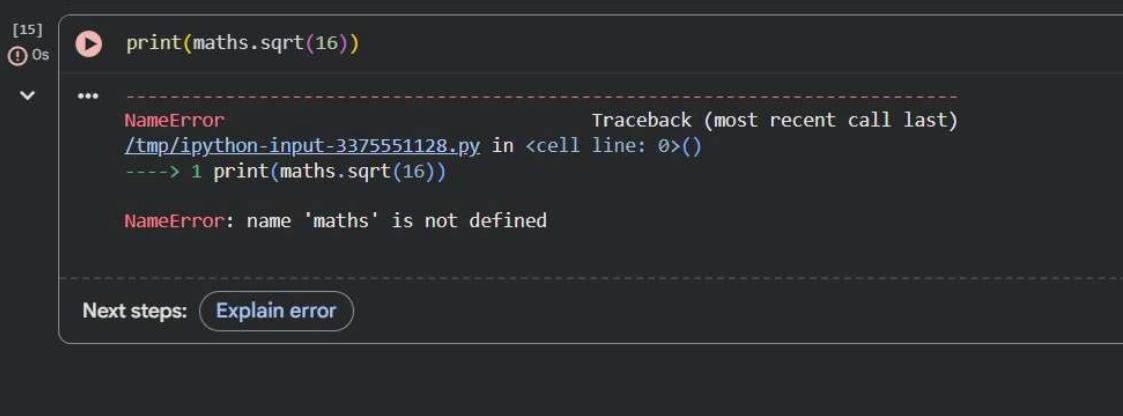
```
AIAC_ASS.7.5_1125.ipynb ☆ 🌐
File Edit View Insert Runtime Tools Help
Commands + Code ▾ + Text | ▶ Run all ▾
13] ② Os
    def func():
        x = 5
        y = 10
        return x + y

    print(func())
    ...
    15
```

**Explanation:** Python relies on indentation to define code blocks. Mixed or incorrect indentation causes syntax errors. Using consistent spacing fixes the issue

## TASK 8: Import Error – Wrong Module Usage

### ACTUAL CODE:



The screenshot shows a Jupyter Notebook cell with the following content:

```
[15] ① 0s
      print(maths.sqrt(16))

...  -----
NameError: name 'maths' is not defined
```

The cell has a status bar indicating [15] ① 0s. The code attempts to call `maths.sqrt(16)`. A `NameError` is raised because the module `maths` is not defined. The traceback shows the error occurred in the current cell at line 1.

**Prompt:** Correct the Python function that fails due to mixed or incorrect indentation by applying consistent indentation.

### CORRECTED CODE:

AIAC\_ASS.7.5\_1125.ipynb ☆

File Edit View Insert Runtime Tools Help

Cells + Code + Text ▶ Run all ▾

```
[1]: import math  
      print(math.sqrt(16))  
      ... 4.0
```

**Explanation:** The error occurs due to importing a non-existent module. Using the correct standard library module name resolves the import issue.

