

# AIAC LAB

## LAB 7.5

2303A51802  
Billa Spurthi  
Batch 29

### Task1:

The screenshot shows the VS Code interface with the following details:

- EXPLORER:** Shows a tree view of files under "AI ASSISTANT CODING" for "Lab 7.5". "Task1.py" is selected.
- CODE EDITOR:** Displays the content of Task1.py:

```
Lab7.5 > Task1.py > ...
1  ## Task1 - 7.5
2  # Analyze the following Python function and identify any logical issue related to default arguments.
3  def add_item(item, items=None):
4      if items is None:
5          items = []
6      items.append(item)
7      return items
8
9  print(add_item(1))
10 print(add_item(2))
11
12 # Explanation
13 # The function `add item` uses a default argument `items` which is set to `None`.
14 # This is a common practice to avoid mutable default arguments like lists.
15 # If we were to use `items=[]` directly as a default argument,
16 # it would be shared across all calls to the function, leading to unexpected
17 # behavior where items from previous calls would accumulate in the list.
18 # By checking if `items` is `None` and then initializing it to an empty list,
19 # we ensure that each call to the function gets its own separate list,
20 # thus avoiding the logical issue of shared mutable default arguments.
```
- TERMINAL:** Shows the command line output for running Task1.py in a terminal window.

### Task2:

The screenshot shows the VS Code interface with the following details:

- EXPLORER:** Shows a tree view of files under "AI ASSISTANT CODING" for "Lab 7.5". "Task2.py" is selected.
- CODE EDITOR:** Displays the content of Task2.py:

```
Lab7.5 > Task2.py > ...
1  #Task2 7.5
2  # Analyze the following Python function. It compares floating-point values and produces an unexpected result.
3  def check_sum():
4      return abs((0.1 + 0.2) - 0.3) < 1e-9
5
6  print(check_sum())
7
8 #Explanation: The issue arises due to the way floating-point arithmetic works in Python
9 # (and in general in computing). The expression 0.1 + 0.2 does not exactly equal 0.3 due
10 # to precision limitations of floating-point representation.
11 # To fix this, we can use a small tolerance value (like 1e-9)
12 # to check if the two values are close enough, rather than checking for exact equality.
13
14
15
```
- TERMINAL:** Shows the command line output for running Task2.py in a terminal window.

## Task3:

The screenshot shows the VS Code interface with the terminal tab selected. The terminal window displays the following Python code and its analysis:

```
Lab7.5 > Task3.py > ...
1 # Task3 7.5
2 # Analyze the following Python function. It uses recursion but causes an infinite recursion error.
3 def countdown(n):
4     print(n)
5     if n == 0:
6         return
7     return countdown(n-1)
8
9 countdown(5)
10 #Explanation:
11 # The function `countdown` is designed to print numbers from `n` down to 0.
12 # However, if the input `n` is negative, the function will keep calling itself
13 # with decreasing values of `n`, leading to an infinite recursion error.
14 # To fix this, we can add a base case to handle negative values of `n`
15 # and prevent the function from calling itself indefinitely.
16
17
18
```

The terminal also shows the command used to run the code and the output of the program execution.

## Task4:

The screenshot shows the VS Code interface with the terminal tab selected. The terminal window displays the following Python code and its analysis:

```
Lab7.5 > Task4.py > ...
1 #Task4 7.5
2 # Analyze the following Python function. It raises an error due to accessing a non-existing dictionary key.
3 def get_value():
4     data = {"a": 1, "b": 2}
5     return data.get("c", "Key not found")
6
7 print(get_value())
8 #Explanation:
9 # The function `get_value` attempts to access the key "c" in the dictionary `data`, which does not exist.
10 # This would normally raise a KeyError if we tried to access it directly (e.g., `data["c"]`).
11 # However, by using the `get` method of the dictionary,
12 # we can provide a default value ("Key not found") that will be
13 # returned if the key is not found in the dictionary.
14
15
16
17
```

The terminal also shows the command used to run the code and the output of the program execution, which includes an error message.

## Task5

The screenshot shows the VS Code interface with the Explorer sidebar open, displaying files under 'AI ASSISTANT CODING'. The file 'Task5.py' is selected and highlighted in blue. The main editor area contains the following Python code:

```
1 # Analyze the following Python function. The loop inside the function runs infinitely.
2 # Bug: Infinite loop
3 def loop_example():
4     i = 0
5     while i < 5:
6         print(i)
7         i += 1
8
9 loop_example()
10
11
12
13 #Explanation:
14 # The function `loop_example` is designed to print numbers from 0 to 4.
15 # However, if the variable `i` is not incremented properly,
16 # it will lead to an infinite loop. In this case, the line `i += 1`
17 # is correctly incrementing `i`, so the loop will terminate after printing 0 to 4.
18 # If we were to remove or comment out the line `i += 1`, the loop would run indefinitely, printing 0 repeatedly.
19
```

The terminal below shows the output of running the code:

```
PS C:\Users\SPURTHI\OneDrive\Desktop\Ai Assistant Coding> c;; cd 'c:\Users\SPURTHI\OneDrive\Desktop\Ai Assistant Coding'; & 'c:\Users\SPURTHI\OneDrive\Desktop\Ai Assistant Coding\Lab7.5\Task5.py'
0
1
2
3
4
```

## Task6:

The screenshot shows the VS Code interface with the Explorer sidebar open, displaying files under 'AI ASSISTANT CODING'. The file 'Task6.py' is selected and highlighted in blue. The main editor area contains the following Python code:

```
1 # Analyze the following Python code. It raises an error due to incorrect tuple unpacking.
2 # Identify why the error occurs and provide a corrected version of the code. If there are extra values in the tuple, demonstrate how to handle them.
3 a, b, c = (1, 2, 3)
4 print(a, b, c)
5 #Explanation: The error occurs because the number of variables on the left side of the assignment does not match the number of values in the tuple.
6 # To handle extra values in a tuple, we can use the * operator to capture them:
7 a, b, *rest = (1, 2, 3, 4, 5)
8 print(a, b) # output: 1 2
```

The terminal below shows the output of running the code:

```
PS C:\Users\SPURTHI\OneDrive\Desktop\Ai Assistant Coding> c;; cd 'c:\Users\SPURTHI\OneDrive\Desktop\Ai Assistant Coding'; & 'c:\Python314\python.exe' 'c:\Users\SPURTHI\OneDrive\Desktop\Ai Assistant Coding\Lab7.5\Task6.py'
1 2 3
1 2
```

## Task7:

The screenshot shows the VS Code interface with the following details:

- Left Sidebar:** A tree view titled "AI ASSISTANT CODING" containing files like Lab6, Lab3.1, Lab5, and various TaskX.py files from Task1.py to Task15.py, along with log.txt, OddEven.py, Prime.py, String.py, and user\_activity.log.
- Top Bar:** Tabs for Task10.py, Task9.py, Task8.py, Task7.py (highlighted), Task6.py, Lab7.5, Task5.py, Lab7.5, and Task4.py.
- Code Editor:** The code for Task7.py is displayed:

```
Lab7.5 > Task7.py ...
1 # Analyze the following Python code. It raises an indentation error due to inconsistent spacing.
2 # Identify the issue and fix the indentation to follow Python standards
3
4 def func():
5     x = 5
6     y = 10
7     return x + y
8
9 print(func())
10
11 #explanation: The issue arises because the code uses inconsistent indentation
12 # (mixing tabs and spaces or using different numbers of spaces).
13 # To fix this, ensure that all lines of code within the function are
14 # indented with the same number of spaces (commonly 4 spaces) and avoid mixing tabs and spaces.
```
- Bottom Bar:** Buttons for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (highlighted), and PORTS.
- Terminal:** Output from the terminal window showing the command to run Task7.py and its execution.

---

## Task8:

The screenshot shows a VS Code interface with the following details:

- Left Sidebar:** Shows a tree view of files under "AI ASSISTANT CODING". Files listed include Lab6, Lab3.1, Lab5, Lab7.5, Task1.py, Task2.py, Task3.py, Task4.py, Task5.py, Task6.py, Task7.py, Task8.py (which is selected), Task9.py, Task10.py, Task11.py, Task12.py, Task13.py, Task14.py, Task15.py, log.txt, OddEven.py, Prime.py, String.py, and user\_activity.log.
- Top Bar:** Shows tabs for Task10.py, Task9.py, Task8.py (active), Task7.py, Task6.py, and Task5.py.
- Code Editor:** Displays the content of Task8.py:

```
1 # The following code raises an import error. Identify the issue and correct the module name.
2 import math
3 print(math.sqrt(16))
4 #Explanation: The import statement is correct,
5 # and the code should work without any issues.
6 # If you are encountering an import error,
7 # it may be due to a problem with your Python environment or
8 # the math module itself. Ensure that you have Python installed
9 # correctly and that there are no issues with your environment.
10
11
```
- Terminal:** Shows the command line output for Task8.py:

```
PS C:\Users\SPURTHI\OneDrive\Desktop\Ai Assistant Coding> c; cd 'c:\Users\SPURTHI\OneDrive\Desktop\Ai Assistant Coding' & python Task8.py
4.0
PS C:\Users\SPURTHI\OneDrive\Desktop\Ai Assistant Coding>
```
- Bottom Navigation:** Includes buttons for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (active), and PORTS.

## Task10:

The screenshot shows the VS Code interface with the Explorer sidebar open, displaying files under 'AI ASSISTANT CODING'. The file 'Task10.py' is selected and highlighted in the sidebar. The main editor area contains the following Python code:

```
Lab7.5 > Task10.py > ...
1 # Identify the missing variable definitions and fix the function using parameters. Add assert test cases.
2 def calculate_area(length, width):
3     return length * width
4
5 # Assertions
6 assert calculate_area(5, 4) == 20
7 assert calculate_area(3, 3) == 9
8 assert calculate_area(10, 2) == 20
9
10 print("All tests passed.")
11
12 #explanation: The original function likely had missing variable definitions
13 # for 'length' and 'width', which would cause an error when trying to calculate
14 # the area. By adding these as parameters to the function, we can pass the necessary
15 # values when calling the function. The assert statements are used to test that the
16 # function returns the expected results for given inputs.
17
```

The terminal tab at the bottom shows the command line and its output:

```
PS C:\Users\SPURTHI\OneDrive\Desktop\Ai Assistant Coding> c:; cd 'c:\Users\SPURTHI\OneDrive\Desktop\Ai Assistant Coding'; & 'c:\pythonon.exe' 'c:\Users\SPURTHI\.vscode\extensions\ms-python.debugpy-2025.19.2026012701-win32-x64\bundled\libs\debugpy\launcher' '--rs\SPURTHI\OneDrive\Desktop\Ai Assistant Coding\Lab7.5\Task10.py'
All tests passed.
○ PS C:\Users\SPURTHI\OneDrive\Desktop\Ai Assistant Coding>
```

## Task11:

The screenshot shows the VS Code interface with the Explorer sidebar open, displaying files under 'AI ASSISTANT CODING'. The file 'Task11.py' is selected and highlighted in the sidebar. The main editor area contains the following Python code:

```
Lab7.5 > Task11.py > ...
1 # Explain why adding int and string fails and fix using type conversion.
2
3 # Adding an integer and a string fails because Python does not allow implicit type conversion between these types.
4 # To fix this, we must explicitly convert one of the values to the same type as the other.
5
6 # Example of failure:
7 # result = 5 + "10" # This raises a TypeError
8
9 # Corrected version:
10 result = 5 + int("10") # Convert string "10" to integer 10
11 print(result) # Output: 15
12 #explanation: In the corrected version, we use the int()
13 # function to convert the string "10" into an integer before
14 # performing the addition. This allows us to successfully add
15 # the two values together without any errors.
```

The terminal tab at the bottom shows the command line and its output:

```
PS C:\Users\SPURTHI\OneDrive\Desktop\Ai Assistant Coding> c:; cd 'c:\Users\SPURTHI\OneDrive\Desktop\Ai Assistant Coding'; & 'c:\pythonon.exe' 'c:\Users\SPURTHI\.vscode\extensions\ms-python.debugpy-2025.19.2026012701-win32-x64\bundled\libs\debugpy\launcher' '--rs\SPURTHI\OneDrive\Desktop\Ai Assistant Coding\Lab7.5\Task11.py'
15
○ PS C:\Users\SPURTHI\OneDrive\Desktop\Ai Assistant Coding>
```

## Task12:

The screenshot shows the VS Code interface with the Explorer sidebar open, displaying files under 'AI ASSISTANT CODING'. The file 'Task12.py' is selected and highlighted in the sidebar. The main editor area contains the following Python code:

```
Lab7.5 > Task12.py > ...
1 # Fix the type mismatch between string and list.
2 def combine():
3     return "Numbers: " + str([1, 2, 3])
4
5 assert combine() == "Numbers: [1, 2, 3]"
6 assert "Numbers: " + str([4, 5]) == "Numbers: [4, 5]"
7 assert isinstance(combine(), str)
8
9 print("All tests passed.")
10
11 #explanation: The issue arises because we are trying to
12 # concatenate a string with a list, which is not allowed in
13 # Python. To fix this, we can convert the list to a string using
14 # the str() function before concatenating it with the other string.
15 # This way, we can successfully combine the two without any type mismatch errors.
16
```

The terminal tab at the bottom shows the command line and its output:

```
PS C:\Users\SPURTHI\OneDrive\Desktop\Ai Assistant Coding> c:; cd 'c:\Users\SPURTHI\OneDrive\Desktop\Ai Assistant Coding'; & 'c:\pythonon.exe' 'c:\Users\SPURTHI\.vscode\extensions\ms-python.debugpy-2025.19.2026012701-win32-x64\bundled\libs\debugpy\launcher' '--rs\SPURTHI\OneDrive\Desktop\Ai Assistant Coding\Lab7.5\Task12.py'
All tests passed.
○ PS C:\Users\SPURTHI\OneDrive\Desktop\Ai Assistant Coding>
```

## Task13

The screenshot shows the VS Code interface with the Explorer sidebar on the left and the Editor tab on the right. The Explorer sidebar lists files under 'AI ASSISTANT CODING' and 'Lab7.5'. The file 'Task13.py' is selected and highlighted with a blue border. The Editor tab displays the following Python code:

```
Lab7.5 > Task13.py > ...
1 #Detect why string multiplication by float fails and fix it.
2 def repeat_text():
3     return "Hello" * int(2.5)
4
5 assert repeat_text() == "HelloHello"
6 assert "Hi" * int(3.0) == "HiHiHi"
7 assert isinstance(repeat_text(), str)
8
9 print("All tests passed.")
10
11
12 #explanation: The error occurs because you cannot multiply a string by a float.
13 # To fix this, we can convert the float to an integer using the int() function
14 # before multiplying the string. This way, we can successfully repeat the
15 # string the desired number of times without encountering a type error.
```

The terminal at the bottom shows the command line and the output 'All tests passed.'

## Task14:

The screenshot shows the VS Code interface with the Explorer sidebar on the left and the Editor tab on the right. The Explorer sidebar lists files under 'AI ASSISTANT CODING' and 'Lab7.5'. The file 'Task14.py' is selected and highlighted with a blue border. The Editor tab displays the following Python code:

```
Lab7.5 > Task14.py > ...
1 # Explain why NoneType cannot be added and fix by assigning default value.
2 def compute():
3     value = 0 # default value instead of None
4     return value + 10
5
6 assert compute() == 10
7 assert 5 + 10 == 15
8 assert isinstance(compute(), int)
9
10 print("All tests passed.")
11 #Explanation: The error occurs because NoneType cannot be added to an integer.
12 # To fix this, we can assign a default value (like 0) to the variable instead of None.
13 # This way, we can perform the addition without encountering a TypeError.
14
15
```

The terminal at the bottom shows the command line and the output 'All tests passed.'

## Task15

The screenshot shows a code editor interface with the following details:

- EXPLORER:** Shows a tree view of files under "AI ASSISTANT CODING". The "Task15.py" file is selected.
- EDITOR:** Displays the content of "Task15.py". The code defines a function "sum\_two\_numbers" that converts string inputs to integers and adds them. It includes assertions for various inputs and a final print statement.
- TERMINAL:** Shows the command-line output of running the code. It includes the path to the file, the command run, and the output "All tests passed."

```
Lab7.5 > Task15.py > ...
1 #Explain why input() returns string and fix using type conversion.
2 # The input() function in Python always returns a string because it is designed to read user input as text. If you want to use
3
4 def sum_two_numbers(a, b):
5     return int(a) + int(b)
6
7 assert sum_two_numbers("5", "3") == 8
8 assert sum_two_numbers("10", "2") == 12
9 assert sum_two_numbers("0", "7") == 7
10
11 print("All tests passed.")
12
13 #explanation: In the function `sum two numbers`, we use the int()
14 # function to convert the string inputs `a` and `b` into integers
15 # before performing the addition. This allows us to correctly sum
16 # the two numbers even though they were originally provided as strings.

PS C:\Users\SPURTHI\OneDrive\Desktop\Ai Assistant Coding> c:; cd 'c:\Users\SPURTHI\OneDrive\Desktop\Ai Assistant Coding'; & 'c:\Python314\python.exe' 'c:\Users\SPURTHI\OneDrive\Desktop\Ai Assistant Coding\Lab.5\Task15.py'
All tests passed.

PS C:\Users\SPURTHI\OneDrive\Desktop\Ai Assistant Coding>
```