

Assignment-7-3

Task 1:

I have a Python function with a syntax error. The function add(a, b) is written without a colon. First detect what the error is. Then correct the function definition. Explain clearly what the mistake is and why it causes a syntax error. Give the correct code and a simple explanation of the fix.

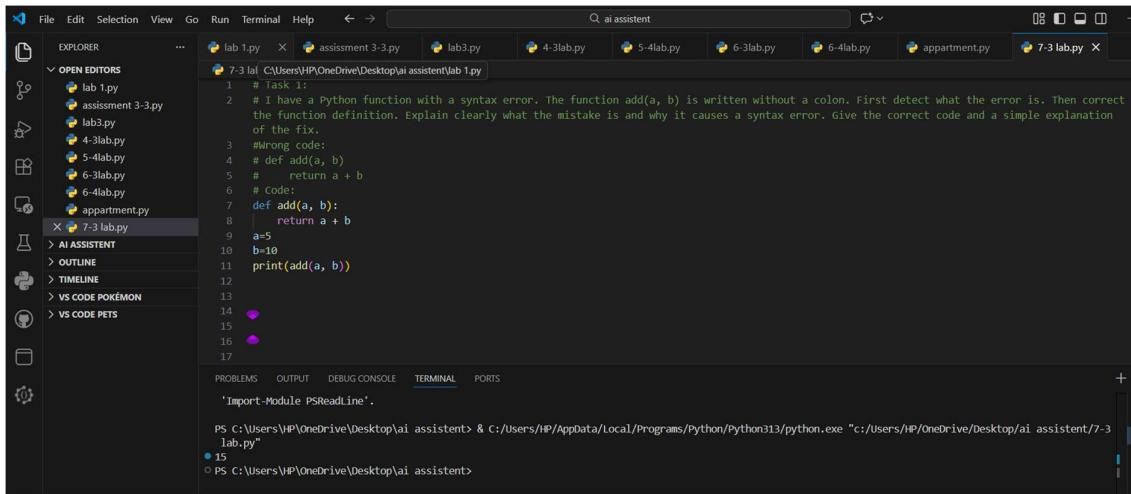
Wrong code:

```
def add(a, b)
    return a + b
```

Code:

```
def add(a, b):
    return a + b
a=5
b=10
print(add(a, b))
```

Output:



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

- File Explorer:** Shows files like lab 1.py, assessment 3-3.py, lab3.py, 4-3lab.py, 5-4lab.py, 6-3lab.py, 6-4lab.py, apartment.py, and 7-3 lab.py.
- Terminal:** The active tab shows the content of the 7-3 lab.py file:

```
# Task 1:
# I have a Python function with a syntax error. The function add(a, b) is written without a colon. First detect what the error is. Then correct the function definition. Explain clearly what the mistake is and why it causes a syntax error. Give the correct code and a simple explanation of the fix.

#Wrong code:
# def add(a, b)
#     return a + b
# Code:
def add(a, b):
    return a + b
a=5
b=10
print(add(a, b))
```
- Output:** The terminal also displays the command run in the terminal tab:

```
'Import-Module PSReadLine'.

PS C:\Users\HP\OneDrive\Desktop\ai assistent> & C:/Users/HP/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/HP/OneDrive/Desktop/ai assistent/7-3 lab.py"
● 15
○ PS C:\Users\HP\OneDrive\Desktop\ai assistent>
```

Analysis:

By adding the colon at the end of the function header, we correctly indicate that the indented block of code that follows is the body of the function. This allows Python to parse the function definition correctly and execute the code without any syntax errors.

Task 2:

I have a Python loop that runs infinitely due to a logic mistake. Find the error, fix it, and explain simply why the loop was infinite and how the correction solves it.

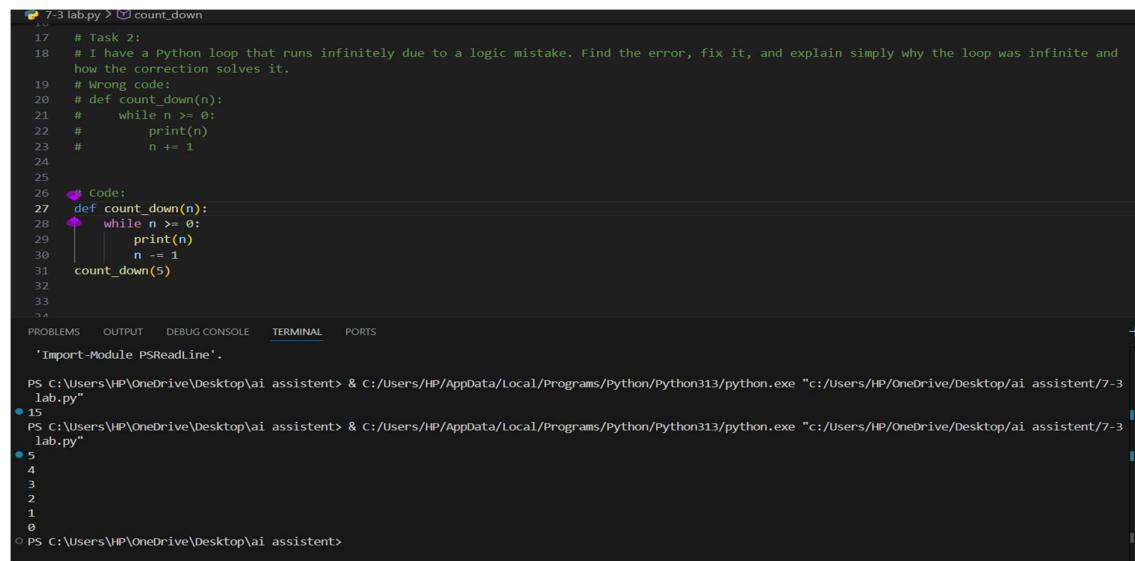
Wrong code:

```
def count_down(n):
    while n >= 0:
        print(n)
        n += 1
```

Code:

```
def count_down(n):
    while n >= 0:
        print(n)
        n -= 1
count_down(5)
```

Output:



```
7-3 lab.py > count_down
17 # Task 2:
18 # I have a Python loop that runs infinitely due to a logic mistake. Find the error, fix it, and explain simply why the loop was infinite and how the correction solves it.
19 # Wrong code:
20 # def count_down(n):
21 #     while n >= 0:
22 #         print(n)
23 #         n += 1
24
25
26 # Code:
27 def count_down(n):
28     while n >= 0:
29         print(n)
30         n -= 1
31 count_down(5)
32
33
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

'Import-Module PSReadLine'.

PS C:\Users\HP\OneDrive\Desktop\ai assistent> & c:/Users/HP/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/HP/OneDrive/Desktop/ai assistent/7-3 lab.py"

● 15

PS C:\Users\HP\OneDrive\Desktop\ai assistent> & c:/Users/HP/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/HP/OneDrive/Desktop/ai assistent/7-3 lab.py"

● 5
4
3
2
1
0

○ PS C:\Users\HP\OneDrive\Desktop\ai assistent>

Analysis:

The original code was incrementing n ($n += 1$) instead of decrementing it ($n -= 1$). This caused the value of n to increase indefinitely, resulting in an infinite loop. By changing it to $n -= 1$, we ensure that n decreases with each iteration, eventually reaching a point where n is less than 0, which will terminate the loop.

Task 3:

I have a Python function that performs division but it crashes at runtime. Identify the runtime error, fix it using try-except, and explain simply how the error is handled.

Wrong code:

```
def divide(a, b):
    return a / b
```

Code:

```
def divide(a, b):
    try:
        return a / b
    except ZeroDivisionError:
        return "Error: Cannot divide by zero."
a = 10
b = 0
print(divide(a, b))
```

Output:

```
7-3 lab.py > divide
41 # Task 3:
42 # I have a Python function that performs division but it crashes at runtime. Identify the runtime error, fix it using try-except, and explain
43 # simply how the error is handled.
44 # Wrong code:
45 # def divide(a, b):
46 #     return a / b
47
48 # Code:
49 def divide(a, b):
50     try:
51         return a / b
52     except ZeroDivisionError:
53         return "Error: Cannot divide by zero."
54 a = 10
55 b = 0
56 print(divide(a, b))
57
58
59
60
61
62 # Analysis:
63 # The mistake was assuming that the code would run because the condition was not checked. It's important to handle errors and prevent them from crashing the program.

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\HP\OneDrive\Desktop\ai assistant> & C:/Users/HP/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/HP/
● lab.py"
Error: Cannot divide by zero.
PS C:\Users\HP\OneDrive\Desktop\ai assistant>
```

Analysis:

The runtime error occurs when b is zero, as division by zero is undefined in mathematics and causes a `ZeroDivisionError` in Python. By using a try-except block, we can catch this specific error and return a user-friendly message instead of crashing the program. This way, if the user tries to divide by zero, they will receive an informative message rather than an unhandled exception.

Task 4:

I have a Python class where the constructor is defined incorrectly. Find the error in the `__init__()` method, fix it by adding the missing parameter, and explain simply why `self` is required.

Wrong code:

```
class Rectangle:
```

```
    def __init__(length, width):  
        self.length = length  
        self.width = width
```

Code:

```
class Rectangle:
```

```
    def __init__(self, length, width):  
        self.length = length  
        self.width = width  
  
object1 = Rectangle(5, 10)  
print(f"Length: {object1.length}, Width: {object1.width}")
```

Output:

```
68 # Task 4:
69 # I have a Python class where the constructor is defined incorrectly. Find the error in the __init__() method, fix it by adding the missing
70 # parameter, and explain simply why self is required.
71 # Wrong code:
72 #     def __init__(length, width):
73 #         self.length = length
74 #         self.width = width
75
76
77 # Code:
78 class Rectangle:
79     def __init__(self, length, width):
80         self.length = length
81         self.width = width
82 object1 = Rectangle(5, 10)
83 print(f"Length: {object1.length}, Width: {object1.width}")
84
85
86
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\HP\OneDrive\Desktop\ai assistant>
PS C:\Users\HP\OneDrive\Desktop\ai assistant> & C:/Users/HP/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/HP/OneDrive/Desktop/ai assistant/7-lab.py"
Length: 5, Width: 10
```

Analysis:

The error in the original code is that the `__init__()` method is missing the '`self`' parameter, which is required to refer to the instance of the class being created. The '`self`' parameter allows us to access and modify the attributes of the instance. By adding '`self`' as the first parameter, we can properly initialize the `length` and `width` attributes for each instance of the `Rectangle` class.

Task 5:

I have a Python program that crashes due to accessing an invalid list index. Identify the error, fix it using safe access (bounds checking or try-except), and explain simply how the error is handled.

Wrong code:

```
numbers = [1, 2, 3]
print(numbers[5])
```

Code:

```
numbers = [1, 2, 3]
try:
    print(numbers[5])
except IndexError:
    print("Error: Index out of range. Please provide a valid index.")
```

Output:

The screenshot shows a code editor interface with a dark theme. On the left is a vertical scroll bar. The main area contains Python code with line numbers from 63 to 111. Lines 90-92 are a comment explaining the task. Lines 93-95 show a wrong code example where an invalid index is printed. Lines 97-111 show the correct code using a try-except block to handle the IndexError and print a user-friendly message. Below the code editor is a navigation bar with tabs: PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is selected), and PORTS. The terminal tab shows a command prompt in Windows style: PS C:\Users\HP\OneDrive\Desktop\ai assistent> & C:/Users/HP/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/HP/OneDrive/Desktop/ai assistent/7-3 lab.py". It then displays the output of the program, which is a single line: Error: Index out of range. Please provide a valid index.

```
63
90
91 # Task 5:
92 # I have a Python program that crashes due to accessing an invalid list index. Identify the error, fix it using safe access (bounds checking or
93 # try-except), and explain simply how the error is handled.
94 # numbers = [1, 2, 3]
95 # print(numbers[5])
96
97
98 # Code:
99 numbers = [1, 2, 3]
100 try:
101     print(numbers[5])
102 except IndexError:
103     print("Error: Index out of range. Please provide a valid index.")
104
105
106
107
108
109
110
111
```

TERMINAL

```
PS C:\Users\HP\OneDrive\Desktop\ai assistent> & C:/Users/HP/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/HP/OneDrive/Desktop/ai assistent/7-3 lab.py"
● Error: Index out of range. Please provide a valid index.
```

Analysis:

The error occurs because we are trying to access an index (5) that does not exist in the list 'numbers', which only has indices 0, 1, and 2. This results in an `IndexError`. By using a `try-except` block, we can catch this specific error and provide a user-friendly message instead of crashing the program. This way, if the user tries to access an invalid index, they will receive an informative message rather than an unhandled exception.