

ASSIGNMENT - 3.4

2303A51355

Batch-10

Task-1

Prompt: generate a Python program to print the first n Fibonacci numbers using a function without giving an example

Code:

```
def fibonacci_sequence(n):  
    sequence = []    a, b = 0, 1  
    for _ in range(n):    sequence.append(a)    a, b = b, a + b  
    return sequence # Example usage n = int(input("Enter the number of  
Fibonacci numbers to generate: ")) fib_sequence =  
fibonacci_sequence(n) print(f"The first {n} Fibonacci numbers are:  
{fib_sequence}")
```

Output :

```
2 # Task-1  
3 # Generate a python program to print the first n Fibonacci numbers using function  
4  
5 def fibonacci_sequence(n):  
6     sequence = []  
7     a, b = 0, 1  
8     for _ in range(n):  
9         sequence.append(a)  
10        a, b = b, a + b  
11    return sequence  
12  
13 n = int(input("Enter the number of Fibonacci numbers to generate: "))  
14 print(fibonacci_sequence(n))  
15  
16 |  
17  
18 # Task-2  
19 # Generate a python program to reverse a list and provide one example  
20  
=====
```

Python Debug Console

```
PS C:\Users\sai\r\OneDrive\Desktop\AI assistant> & 'c:\Users\sai\r\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\sai\r\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\lib\debugpy\launcher' '49156' '-' 'c:\Users\sai\r\OneDrive\Desktop\AI assistant\lab-3.4..py'  
Enter the number of Fibonacci numbers to generate: 6  
[0, 1, 1, 2, 3, 5]
```

Code Analysis :

- ☐ The function fibonacci_sequence(n) generates Fibonacci numbers iteratively.
- ☐ Variables a and b store the previous two Fibonacci values.
- ☐ A for loop runs n times to generate required numbers.
- ☐ Each generated number is stored in a list for easy return.

- Function-based approach improves reusability and clarity.

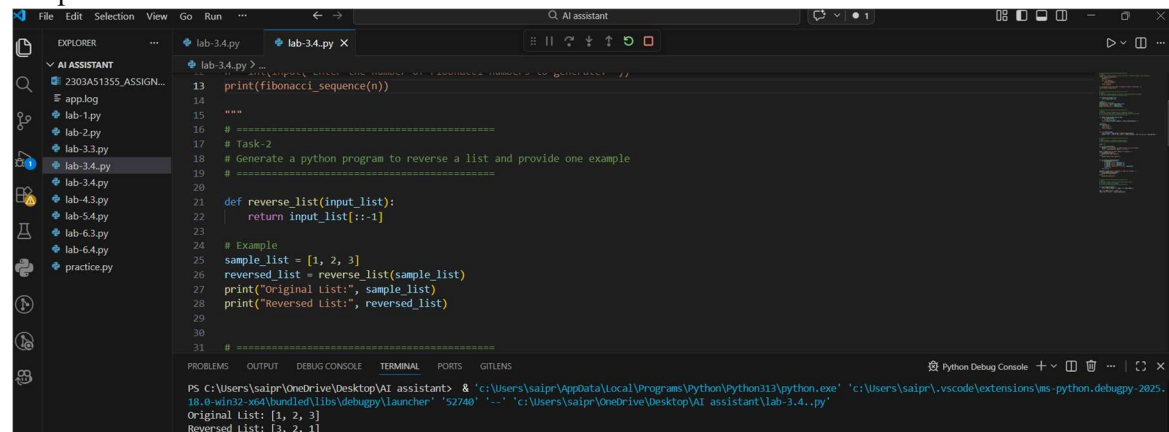
Task-2

Prompt: generate a Python program to reverse a list and provide one example

Code:

```
def reverse_list(input_list):  
    return input_list[::-1]  
  
# Example usage sample_list = [1, 2, 3]  
reversed_list = reverse_list(sample_list)  
print(f"Original List: {sample_list}")  
print(f"Reversed List: {reversed_list}")
```

Output :

The screenshot shows a Visual Studio Code editor window. The Explorer sidebar on the left lists several files, including 'lab-3.4.py' which is currently selected and open in the main editor. The code in the editor is a Python script that defines a function 'reverse_list' and uses it to reverse a sample list [1, 2, 3]. The output of the script is displayed in the 'TERMINAL' panel at the bottom, showing 'Original List: [1, 2, 3]' and 'Reversed List: [3, 2, 1]'. The interface is dark-themed, and the AI Assistant sidebar is visible on the left.

Code Analysis :

- The function reverse_list() accepts a list as input.
- Python slicing [::-1] is used for efficient reversal.
- No additional loop or memory-intensive operations are required.
- Original list remains unchanged, ensuring data safety.
- Function allows reuse for any list input.

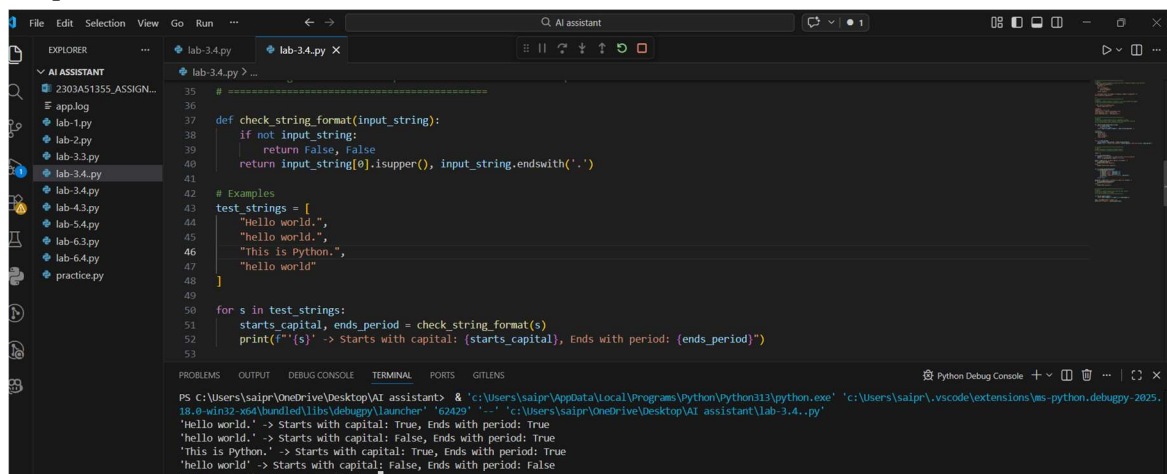
Task-3

Prompt : generate a Python program with 2-3 examples of how to check if a string starts with a capital letter and ends with a period using a function.

Code :

```
def check_string_format(input_string):  
    starts_with_capital = input_string[0].isupper() if input_string else False  
    ends_with_period = input_string.endswith('.') if input_string else False  
    return starts_with_capital, ends_with_period  
  
# Example usage  
test_strings = [  
    "Hello world.",  
    "hello world.",  
    "Hello world",  
    "This is a test."  
]  
for s in  
test_strings:  
    starts_capital, ends_period = check_string_format(s)  
    print(f"String: '{s}' | Starts with capital: {starts_capital} | Ends with period:  
{ends_period}")
```

Output :



The screenshot shows a VS Code editor with a file explorer on the left containing several Python files. The main editor window displays a Python script named 'lab-3.4.py'. The script defines a function 'check_string_format' that takes an input string and returns a tuple of two boolean values: 'starts_with_capital' and 'ends_with_period'. It then uses a loop to iterate over a list of test strings and prints the results for each. The terminal at the bottom shows the output of the script, which matches the expected results from the code analysis.

```
PS C:\Users\sai\r\OneDrive\Desktop\AI assistant> & 'c:\Users\sai\r\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\sai\r\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundle\libs\debugpy\launcher' '62429' -> 'c:\Users\sai\r\OneDrive\Desktop\AI assistant\lab-3.4..py'  
'Hello world.' -> Starts with capital: True, Ends with period: True  
'hello world.' -> Starts with capital: False, Ends with period: True  
'This is Python.' -> Starts with capital: True, Ends with period: True  
'hello world' -> Starts with capital: False, Ends with period: False  
Enter an email address to validate:
```

Code Analysis :

- ☐ The function checks both starting and ending conditions of a string.
- ☐ isupper() verifies whether the first character is capitalized.
- ☐ endswith('.') confirms proper sentence termination.
- ☐ Handles empty strings safely using conditional checks.
- ☐ Returns multiple Boolean values for detailed validation.

Task-4

Prompt: **generate a code for Email Validator**

Code: import re def

is_valid_email(email):

```
# Define a regex pattern for validating an Email
pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
return re.match(pattern, email) is not None
if __name__ == "__main__":
    email = input("Enter an email address to validate: ")
    if is_valid_email(email):
```

```
    print(f"The email address '{email}' is valid.")
```

else:

```
    print(f"The email address '{email}' is not valid.")
```

Password Strength Checker def

is_strong_password(password):

```
# A strong password has at least 8 characters, contains uppercase, lowercase, digit, and
special character
if (len(password) >= 8 and re.search(r'[A-Z]', password) and
re.search(r'[a-z]', password) and re.search(r'[0-9]', password) and
re.search(r'[!@#$%^&*().,?":{}|<>]', password)):
```

```
    return True
    return
```

```
False
if __name__ ==
```

```
"__main__":
```

```

password = input("Enter a password to check its strength:
") if is_strong_password(password):    print("The
password is strong.") else:
    print("The password is weak.")

```

Output :

```

62 def is_valid_email(email):
63     pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
64     return re.match(pattern, email) is not None
65
66 email = input("Enter an email address to validate: ")
67 if is_valid_email(email):
68     print("Valid email address")
69 else:
70     print("Invalid email address")
71
72
73 def is_strong_password(password):
74     if (len(password) >= 8 and
75         re.search(r'[A-Z]', password) and
76         re.search(r'[a-z]', password) and
77         re.search(r'[0-9]', password) and
78         re.search(r'[@$%&*()!.,:~{}|<>]', password)):
79         return True
80     return False
81

```

Terminal Output:

```

PS C:\Users\sai\r\OneDrive\Desktop\AI assistant> & 'c:\Users\sai\r\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\sai\r\.vscode\extensions\ms-pyt
hon.debugpy-2025.18.0-win32-x64\bundle\libs\debugpy_launcher' '56717' '-' 'c:\Users\sai\r\OneDrive\Desktop\AI assistant\lab-3.4.py'
Enter an email address to validate: sai@gmail.com
Valid email address
Enter a password to check its strength: sai
Weak password

```

Code Analysis :

- ☐ Regular expressions (re) are used for pattern matching.
- ☐ Email validation ensures correct structure using a defined regex.
- ☐ Password checker verifies length, case, digits, and special characters.
- ☐ Separate functions improve modularity and readability.
- ☐ Enhances security by validating user credentials effectively.

Task 5

Prompt: generate a Python program with a function that returns the sum of the digits of a number

Code:

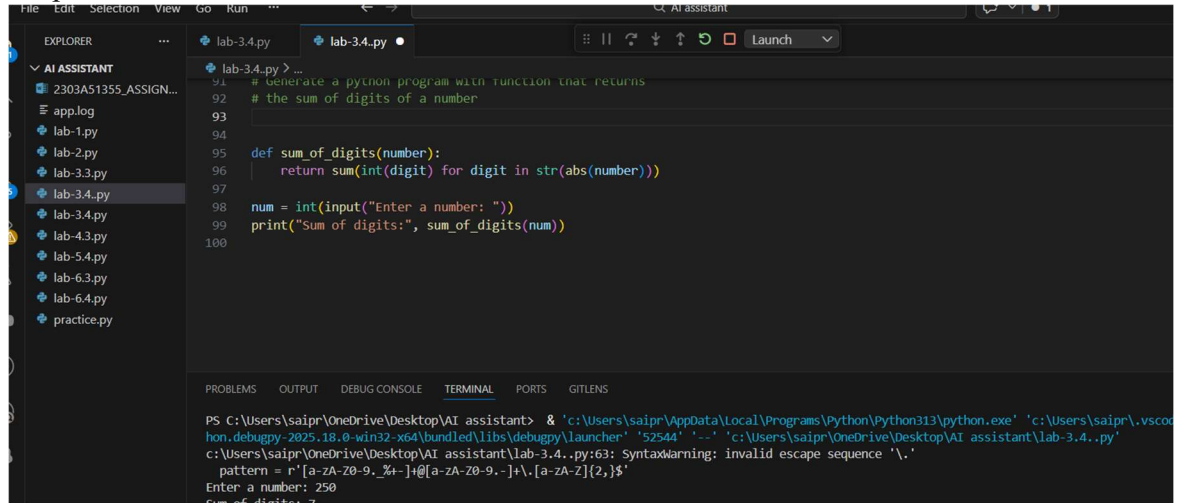
```

def sum_of_digits(number):
    return sum(int(digit) for digit in str(abs(number)))

# Example usage
num = int(input("Enter a number to calculate the sum of its digits: "))
result = sum_of_digits(num)
print(f"The sum of the digits of {num} is: {result}")

```

Output :



The screenshot shows a Visual Studio Code editor with a file explorer on the left containing several Python files. The main editor window displays a Python script named `lab-3.4.py`. The script defines a function `sum_of_digits` that takes a number, converts it to a string, iterates over each character, converts it back to an integer, and sums them. It then prompts the user for a number and prints the result. The terminal at the bottom shows the command to run the script, a syntax warning, and the execution output where the user entered 250 and the sum of digits is 7.

```
lab-3.4.py > ...
91 # Generate a python program with function that returns
92 # the sum of digits of a number
93
94
95 def sum_of_digits(number):
96     return sum(int(digit) for digit in str(abs(number)))
97
98 num = int(input("Enter a number: "))
99 print("Sum of digits:", sum_of_digits(num))
100
```

PS C:\Users\sai\OneDrive\Desktop\AI assistant> & 'c:\Users\sai\OneDrive\Desktop\AI assistant\lab-3.4.py'

Enter a number: 250

Sum of digits: 7

Code Analysis :

- ☐ The function converts the number into a string for easy digit access.
- ☐ `abs()` ensures correct handling of negative numbers.
- ☐ `int()` converts each character back to a digit.
- ☐ `sum()` efficiently adds all digits in one line.
- ☐ Function returns the result, supporting reuse in other programs.