

ASSIGNMENT - 3.4

2303A51525

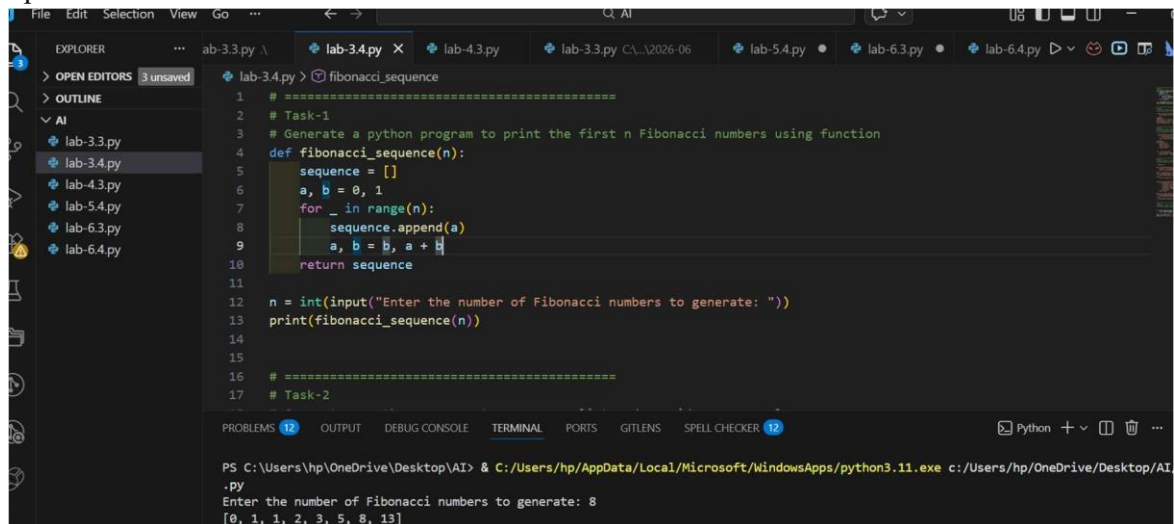
Batch-10

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 :

A screenshot of a Python IDE interface. The top part shows the code editor with a file named 'lab-3.4.py' containing the Fibonacci sequence function and its usage. The bottom part shows the terminal output where the user has entered '8' and the program has printed the first 8 Fibonacci numbers: '[0, 1, 1, 2, 3, 5, 8, 13]'.

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

PS C:\Users\hp\OneDrive\Desktop\AI> & C:/Users/hp/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/hp/OneDrive/Desktop/AI/.py
Enter the number of Fibonacci numbers to generate: 8
[0, 1, 1, 2, 3, 5, 8, 13]

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 image shows a VS Code editor window with the following details:

- Menu Bar:** File, Edit, Selection, View, Go, AI, Search, Run and Debug, Window, Help.
- Explorer Sidebar:**
 - Open Editors: 3 unsaved
 - Outline: AI
 - Files: lab-3.3.py, lab-3.4.py (selected), lab-4.3.py, lab-5.4.py, lab-6.3.py, lab-6.4.py
- Main Editor:**
 - File: lab-3.4.py
 - Code:

```

16 # =====
17 # Task-2
18 # Generate a python program to reverse a list and provide one example
19 # =====
20
21 def reverse_list(input_list):
22     return input_list[::-1]
23
24 # Example
25 sample_list = [5,8,9]
26 reversed_list = reverse_list(sample_list)
27 print("Original List:", sample_list)
28 print("Reversed List:", reversed_list)
29
30
31 # =====
32 # Task-3

```
- Terminal:**
 - PS C:\Users\hp\OneDrive\Desktop\AI> & C:/Users/hp/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/hp/OneDrive/Desktop/AI/lab-3.4.py
 - Original List: [5, 8, 9]
 - Reversed List: [9, 8, 5]

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 :

```

41
42 # Examples
43 test_strings = [
44     "Hello world.",
45     "hello world.",
46     "This is Python.",
47     "hello world"
48 ]
49
50 for s in test_strings:
51     starts_capital, ends_period = check_string_format(s)
52     print(f'{s}' -> Starts with capital: {starts_capital}, Ends with period: {ends_period}")
53
54
55 # =====
56 # Task-4
57 # Email Validator & Password Strength Checker

```

```

PS C:\Users\hp\OneDrive\Desktop\AI> & C:/Users/hp/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/hp/OneDrive/Desktop/AI/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

```

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 :

```

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
82 password = input("Enter a password to check its strength: ")
83 if is_strong_password(password):
84     print("Strong password")
85 else:
86     print("Weak password")
87

```

PROBLEMS 12 OUTPUT DEBUG CONSOLE TERMINAL PORTS GIT LENS SPELL CHECKER 12

```

PS C:\Users\hp\OneDrive\Desktop\AI> & C:/Users/hp/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/hp/OneDrive/Desktop/
.py
Enter an email address to validate: vamshi@gmail.com
Valid email address
Enter a password to check its strength: sunny
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.

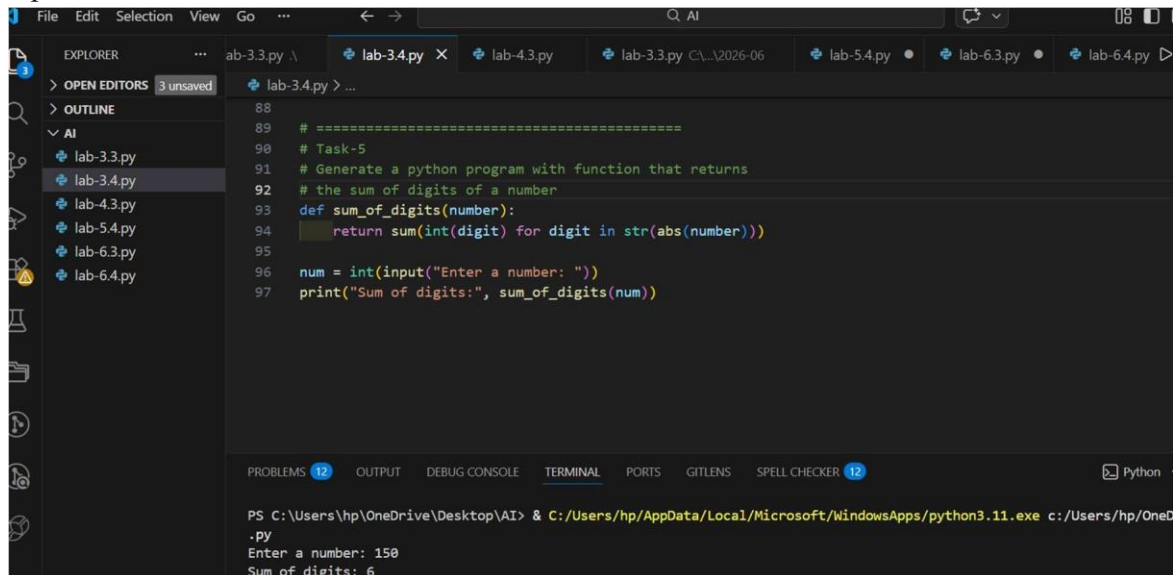
Task 5

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 :

A screenshot of a code editor interface, likely Visual Studio Code, showing a Python file named 'lab-3.4.py'. The code defines a function 'sum_of_digits' that takes a number and returns the sum of its digits. It then uses this function to calculate the sum of digits for a user input. The output console at the bottom shows the command prompt running the script, entering the number 150, and displaying the output 'Sum of digits: 6'.

```
88  
89 # =====  
90 # Task-5  
91 # Generate a python program with function that returns  
92 # the sum of digits of a number  
93 def sum_of_digits(number):  
94     return sum(int(digit) for digit in str(abs(number)))  
95  
96 num = int(input("Enter a number: "))  
97 print("Sum of digits:", sum_of_digits(num))
```

PROBLEMS 12 OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS SPELL CHECKER 12 Python

PS C:\Users\hp\OneDrive\Desktop\AI> & C:/Users/hp/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/hp/OneDrive/Desktop/AI/lab-3.4.py
Enter a number: 150
Sum of digits: 6

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.