

## ASSIGNMENT - 3.4

2303A51355

Batch-10

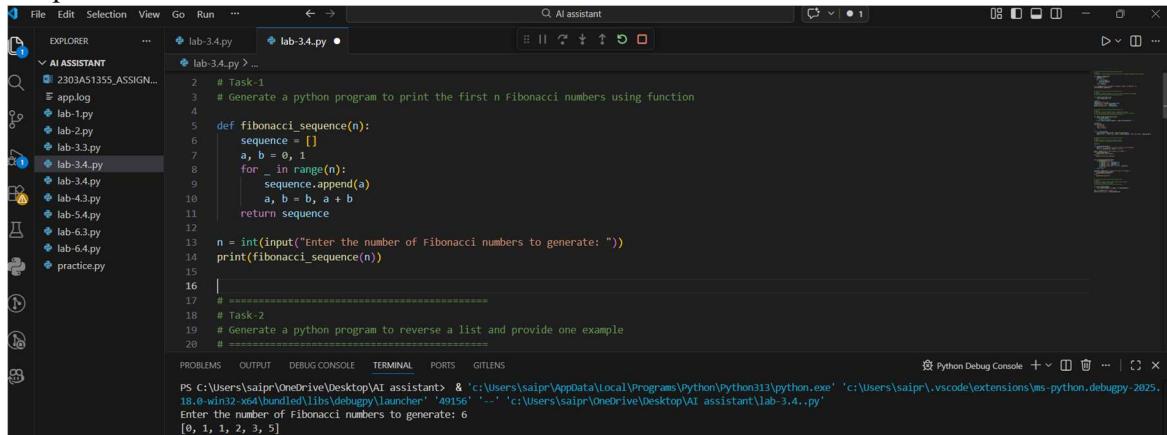
### Task-1

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 :



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

- Explorer View:** Shows files in the workspace, including AI ASSISTANT, 2303A51355\_ASSIGN..., app.log, lab-1.py, lab-2.py, lab-3.py, lab-3.4.py (the active file), lab-4.py, lab-4.3.py, lab-5.py, lab-6.py, lab-6.3.py, lab-6.4.py, practice.py.
- Code Editor:** Displays the Python code for generating Fibonacci numbers.
- Terminal:** Shows the command used to run the script and the resulting output.
- Output:** Shows the printed list of Fibonacci numbers [0, 1, 1, 2, 3, 5].

```
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
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 # =====
```

```
PS C:\Users\saipr\Desktop\AI assistant & "c:\Users\saipr\AppData\Local\Programs\Python\Python313\python.exe" "c:\Users\saipr\.vscode\extensions\ms-python.python-2025.18.0-win32-x64\bundled\libs\debugpy\launcher" "49156" "--" "c:\Users\saipr\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 :

```
File Edit Selection View Go Run ... ⏪ ⏩ AI assistant
EXPLORER lab-3.4.py
AI ASSISTANT 2303A5135_ASSIGN...
app.log
lab-1.py
lab-2.py
lab-3.3.py
lab-3.4.py
lab-3.4.py
lab-4.3.py
lab-5.4.py
lab-6.3.py
lab-6.4.py
practice.py
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS
Python Debug Console + ×
PS C:\Users\saipr\Desktop\AI assistant & 'c:\Users\saipr\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\saipr\vscode\extensions\ms-python.vscode-pyright-2025.18.0-win32-x64\bundle\libs\debug\launcher' "52740" "-l" "c:\Users\saipr\Desktop\AI assistant\lab-3.4.py"
Original List: [1, 2, 3]
Reversed List: [3, 2, 1]
```

## 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

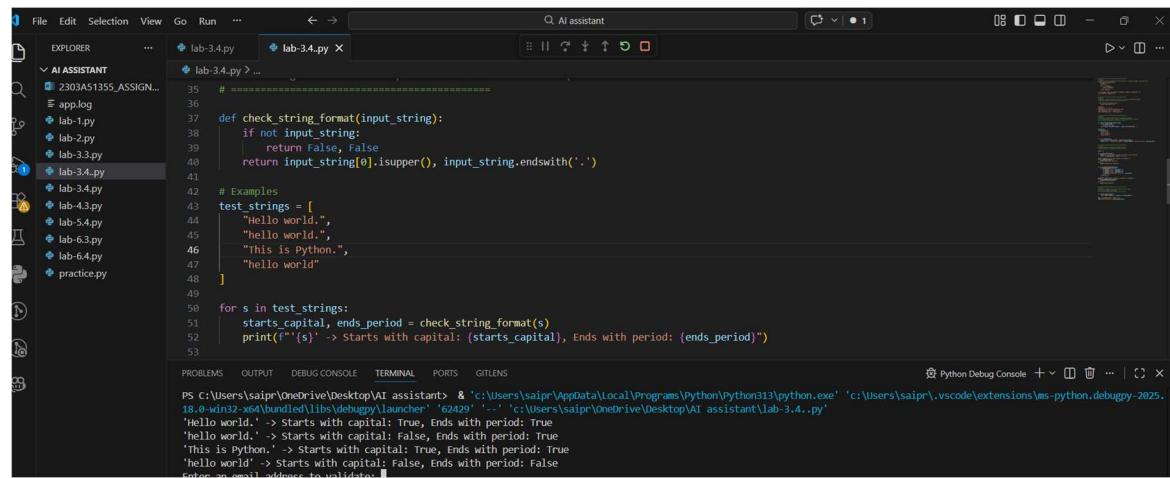
Prompt : generate a Python program with 2-3 examples of how to check if a string starts with a capital letter and ends with a period using a function.

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 the VS Code interface with the code editor open to 'lab-3.4.py'. The code implements the function 'check\_string\_format' to check if a string starts with a capital letter and ends with a period. It then iterates over a list of test strings, printing each string along with its start capital status and end period status. The terminal below shows the execution of the script and the resulting output. The output includes the strings 'Hello world.', 'hello world.', 'Hello world', 'This is a test.', and 'Hello world', followed by their analysis: 'Hello world.' -> Starts with capital: True, Ends with period: True; 'hello world.' -> Starts with capital: False, Ends with period: True; 'Hello world' -> Starts with capital: True, Ends with period: True; 'This is a test.' -> Starts with capital: False, Ends with period: False; and 'Hello world' -> Starts with capital: False, Ends with period: False.

```
PS C:\Users\saipr\OneDrive\Desktop\AI assistant> & "C:\Users\saipr\AppData\Local\Programs\Python\Python313\python.exe" "c:\Users\saipr\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
'Hello world' -> Starts with capital: True, Ends with period: True
'This is a test.' -> Starts with capital: False, Ends with period: False
'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 :

```

File Edit Selection View Go Run ... AI assistant
EXPLORER lab-3.4.py lab-3.4.py ...
2303A5135_ASSIGN...
app.log
lab-1.py
lab-2.py
lab-3.py
lab-3.4.py
lab-4.py
lab-4.3.py
lab-5.4.py
lab-6.3.py
lab-6.4.py
practice.py
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS
PS C:\Users\saipr\OneDrive\Desktop\AI assistant> 8 'c:\Users\saipr\AppData\Local\Programs\Python\Python311\python.exe' 'c:\Users\saipr\.vscode\extensions\ms-python.python.debug-2025.18.0-win2.x64\bundles\l1bs\debug\launcher' '56717' ... 'c:\Users\saipr\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 code editor interface with the following details:

- File Explorer:** Shows files in the "AI ASSISTANT" folder, including 2303A51355\_ASSIGN..., app.log, lab-1.py, lab-2.py, lab-3.py, lab-3.4.py (selected), lab-4.py, lab-4.3.py, lab-5.py, lab-6.3.py, lab-6.4.py, and practice.py.
- Code Editor:** Displays the content of lab-3.4.py:

```
# Generate a python program with function that returns
# the sum of digits of a number
def sum_of_digits(number):
    return sum(int(digit) for digit in str(abs(number)))
num = int(input("Enter a number: "))
print("Sum of digits:", sum_of_digits(num))
```
- Terminal:** Shows the command run in the terminal: PS C:\Users\saipr\OneDrive\Desktop\AI assistant> & 'C:\Users\saipr\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\saipr\vscode\hon\_debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '52544' '--' 'c:\Users\saipr\OneDrive\Desktop\AI assistant\lab-3.4.py'
- Output:** Displays the execution results:

```
pattern = r'[a-zA-Z0-9_%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
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.