ASSIGNMENT - 3.4

2303A51355

Batch-10

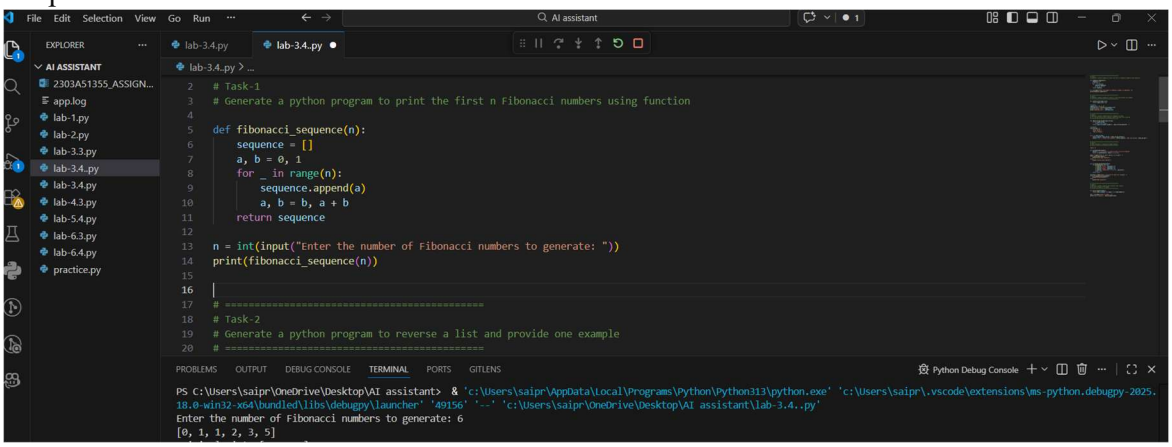
Task-1

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 :



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.

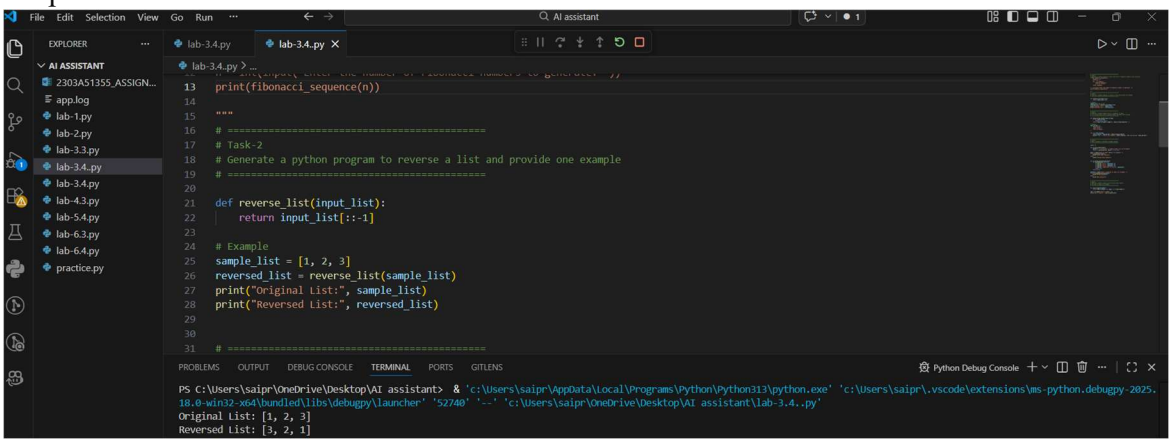
Task-2

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 :



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.

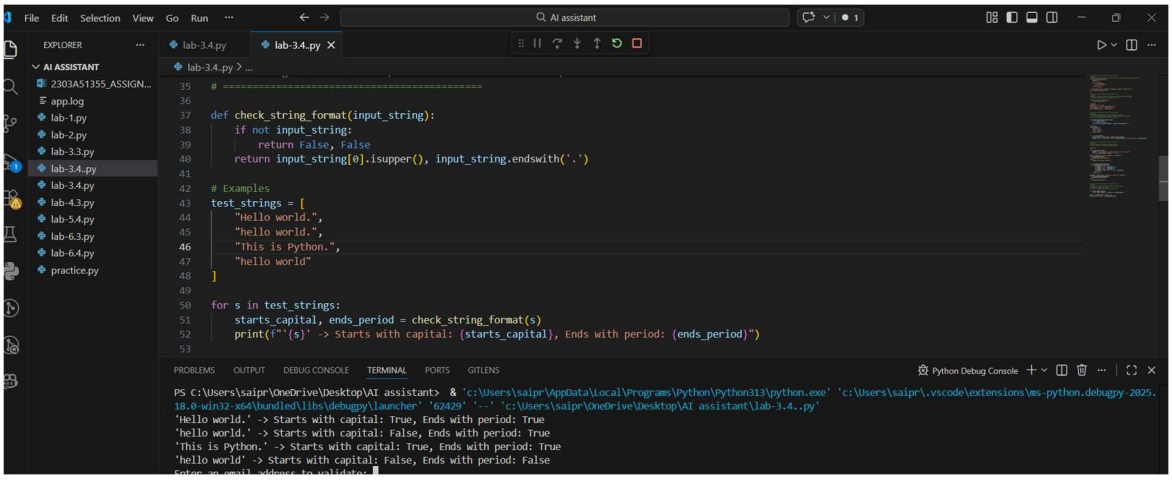
Task-3

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 :



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__":

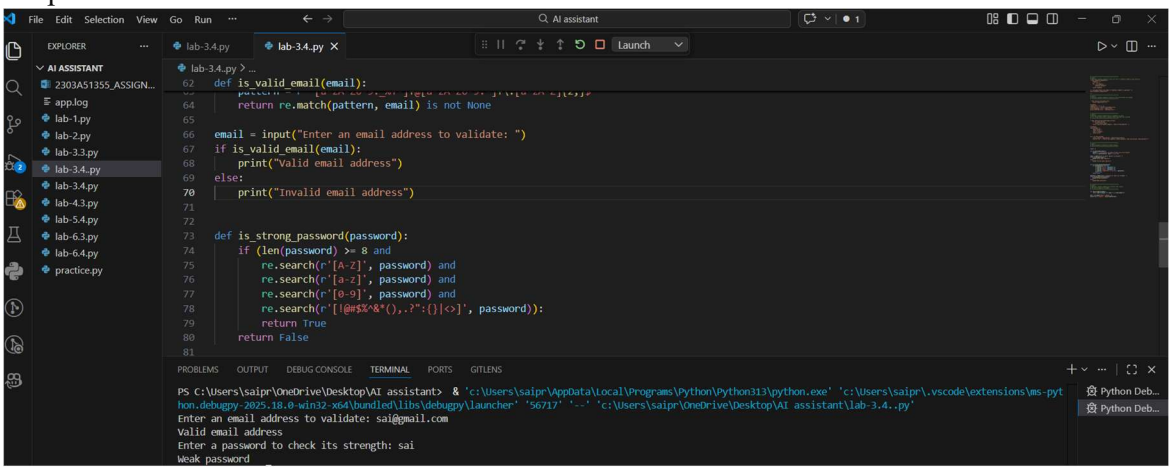
password = input("EnteQr a password to check its strength:

") if is_strong_password(password): print("The

password is strong.") else:

print("The password is weak.")

Output :



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):

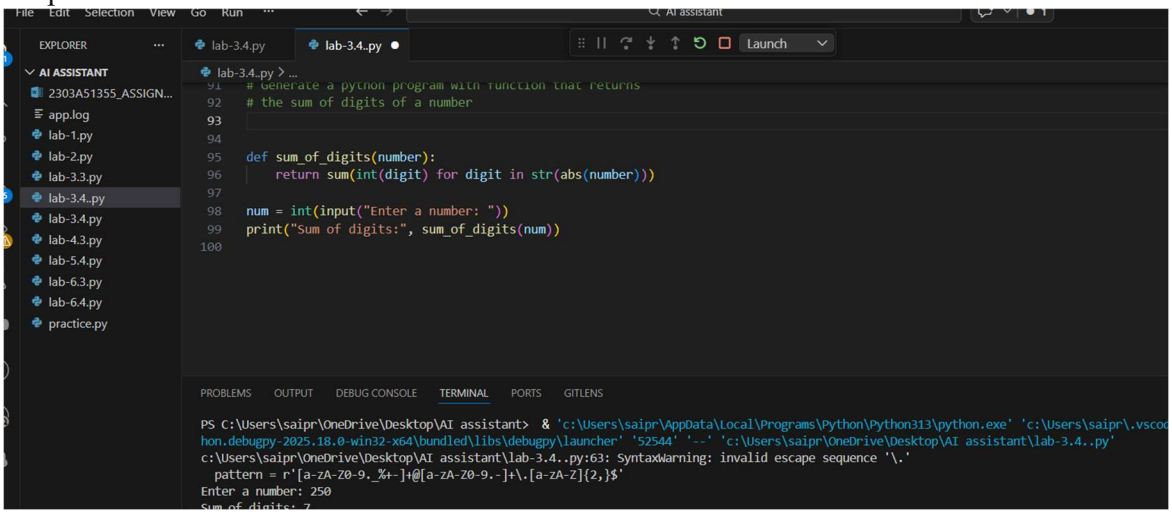
return sum(int(digit) for digit in str(abs(number)))

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 :



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.