AI ASSISTANT CODING ASSIGNMENT-3.4

Name: D.Divya sri

HT. No: 2303A51323

Batch: 19

Task 1: Zero-shot Prompt – Fibonacci Series Generator

Prompt: # Generate a Python function to print the first N Fibonacci numbers.

Code:

ass5.3.py > ...

```python
#create a simple login system in python using username and password
username = "admin"
password = "password123"
u=input("Enter username: ")
p=input("Enter password: ")
if u==username and p==password:
    print("Login successful!")
else:
    print("Login failed. Incorrect username or password.")
import getpass
stored_username = "admin"
stored_password_hash="hashed_password_value"  # Replace with actual hashed password
username_input = input("Enter username: ")
password_input = getpass.getpass("Enter password: ")
if username==stored_username:
    print("password verification required")
else:
    print("Invalid username")
```

Result:

```
Enter the number of Fibonacci numbers to generate: 7
The first 7 Fibonacci numbers are: [0, 1, 1, 2, 3, 5, 8]
PS C:\Users\domma\Downloads\AAC>
```

Observation:
The AI-generated program efficiently generates the Fibonacci sequence using an iterative approach and returns the first *n* Fibonacci numbers in a list. This method avoids the overhead of recursion and ensures linear time complexity. However, the program does not handle invalid inputs such as negative numbers or non-integer values, which may affect its robustness in practical use.

Task 2: One-shot Prompt – List Reversal Function

Prompt: # Generate a python code o reverse a list of

numbers

Code:

```
#task2
#Generate a python code o reverse a list of numbers
def reverse_list(numbers):
    reversed_numbers = []
    for i in range(len(numbers)-1, -1, -1):
        reversed_numbers.append(numbers[i])
    return reversed_numbers
numbers = [int(x) for x in input("Enter numbers separated by spaces: ").split()]
print(f"Reversed list: {reverse_list(numbers)}")
```

Result:

```
Enter numbers separated by spaces: 1 2 3 4
Reversed list: [4, 3, 2, 1]
```

Observation:
The AI-generated program reverses a list of numbers using an explicit loop and index-based traversal from the last element to the first. This approach clearly demonstrates control over iteration and avoids built-in reversal methods. However, it uses additional memory for a new list and does not handle invalid or non-numeric input, which may limit its efficiency and robustness in real-world scenarios.

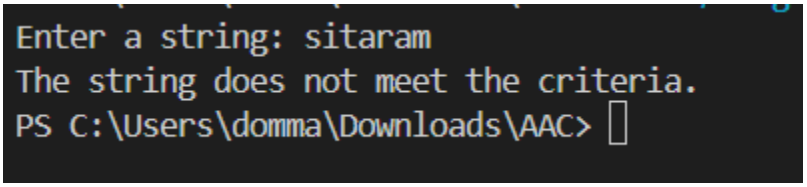
Task 3: Few-shot Prompt – String Pattern Matching

Prompt: # #Write a Python program that checks if a given string starts with a capital letter and ends with a period (.).

Code:

```
#Task3
#Write a Python program that checks if a given string starts with a capital
#  letter and ends with a period (.).
def check_string_format(s):
    if s and s[0].isupper() and s.endswith('.'):
        return True
    return False
input_string = input("Enter a string: ")
if check_string_format(input_string):
    print("The string starts with a capital letter and ends with a period.")
else:
    print("The string does not meet the criteria.")
```

Result:



Observation:

The AI-generated program correctly checks whether a given string starts with a capital letter and ends with a period using built-in string methods. It includes a safeguard against empty input and provides clear output based on the validation result. However, the program does not account for leading or trailing whitespace and does not validate whether the entire input forms a meaningful sentence, which may limit its accuracy in practical use.

Task 4: Zero-shot vs Few-shot – Email Validator

Prompt: #zero shot : Write a Python function to validate an email address.

#few shot : Write a Python function to validate an email address.

Examples:

Input: "user@gmail.com" → Output: Valid

Input: "usergmail.com" → Output: Invalid

Input: "user@com" → Output: Invalid

Input: "user@domain.co" → Output: Valid

Code:

```
#zero shot code
## Write a Python function to validate an email address.
def validate_email(email):
    if "@" in email and "." in email:
        return True
    return False
email = input("Enter an email address: ")
if validate_email(email):
    print("Valid email address.")
else:
    print("Invalid email address.")
#few shot code
## Write a Python function to validate an email address.
# Examples:
# Input: "user@gmail.com" → Output: Valid
# Input: "usergmail.com" → Output: Invalid
# Input: "user@com" → Output: Invalid
# Input: "user@domain.co" → Output: Valid
def validate_email(email):
    if "@" not in email:
        return False

    username, domain = email.split("@", 1)

    if not username or "." not in domain:
        return False

    if domain.startswith(".") or domain.endswith("."):
        return False

    return True
email = input("Enter an email address: ")
if validate_email(email):
    print("Valid email address.")
else:
    print("Invalid email address.")
```

Result:

```
Enter an email address: vijaya
Invalid email address.
Invalid email address.
Enter an email address: vijaya@gmail.com
Enter an email address: vijaya@gmail.com
Valid email address.
```

Observation

The zero-shot email validation program performs only basic checks by verifying the presence of the @ and . characters. While simple, this approach allows many invalid email formats to be incorrectly classified as valid, reducing reliability. In contrast, the few-shot version applies more structured validation by separating the username and domain, ensuring a valid domain format, and preventing common errors such as missing usernames or improper domain placement. This demonstrates that few-shot prompting produces more accurate and robust logic, making the generated code more suitable for real-world use.

Task 5: Prompt Tuning – Summing Digits of a Number

Prompt: # #Generic Task Prompt: Write a Python function to return the sum of digits of a number.

 # #Task + Input/Output Example:# Write a Python function to return the sum of digits of a number.

 # Examples:

 # Input: 123 → Output: 6

 # Input: 405 → Output: 9

 # Input: 0 → Output: 0

Code:

```python
#Task5:
#Generic Task Prompt: Write a Python function to return the sum of digits of a number.
def sum_of_digits(n):
    total = 0
    for digit in str(n):
        total += int(digit)
    return total
print(sum_of_digits(123))
#Task + Input/Output Example:# Write a Python function to return the sum of digits of a number.
# Examples:
# Input: 123 → Output: 6
# Input: 405 → Output: 9
# Input: 0 → Output: 0
def sum_of_digits(n):
    n = abs(n)
    total = 0
    while n > 0:
        total += n % 10
        n //= 10
    return total
print(sum_of_digits(2356))
```

Result:

```
6
16
PS C:\Users\domma\Downloads\AAC>
```

Observation:

The generic task prompt produces a simple implementation that converts the number to a string and sums its digits. While effective for positive integers, it does not explicitly handle negative values. In contrast, the prompt that includes input–output examples generates a more robust solution by using mathematical operations and handling negative numbers through absolute value conversion. This demonstrates that example-based prompting leads to cleaner, safer, and more reliable code.