|  |  |
| --- | --- |
| Project Title : | Lab Task 4 |
| Name : | **Aman Ali** |
| Roll No : | **195** |
| Class : | BSAI |
| Section : | **3C** |
| Submission Date : | September 18, 2025 |

Project Documentation

Project 1: Text Cleaner

**1. Code Explanation**

This program removes all **special characters** from a given string and only keeps **alphabets and spaces**.

* It loops through each character of the string.
* If the character is an alphabet or space → it is added to the new string.
* Otherwise → it is ignored.
* Finally, the cleaned string is printed.

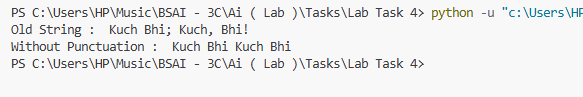
**2. How It Works**

* Input string is defined: "Kuch Bhi; Kuch, Bhi!"
* Program initializes new\_string as empty.
* For each character in the string:
  + If char.isalpha() or char == " " → add to new\_string.
  + Else skip it.
* Output is the cleaned version without ; , !.

**3. Why This Approach**

* Very simple implementation of **data cleaning**.
* Demonstrates string traversal and character filtering in Python.
* Useful in text preprocessing tasks like NLP.

**4. Sample Output**



Project 2: Bubble Sort Words and Characters

**1. Code Explanation**

This program sorts a string in two ways:

1. **By words** → Bubble Sort based on the first letter of each word.
2. **By characters** → Bubble Sort on all characters (ignoring spaces).

* Two separate bubble sort functions are defined:  
  • bubble\_sort\_words()  
  • bubble\_sort\_chars()
* Input text is split into words and also converted to a list of characters.
* Sorting is done using ASCII values (ord()).

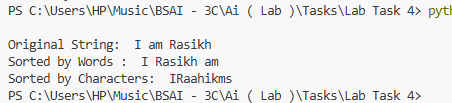
**2. How It Works**

* Input: "I am Rasikh"
* Step 1: Split into words → ["I", "am", "Rasikh"]
* Step 2: Sort words by first letter → "I Rasikh am"
* Step 3: Remove spaces, sort characters individually → "IRaaahikms"
* Print both results.

**3. Why This Approach**

* Bubble Sort is a **basic algorithm** → good for learning sorting.
* Works step by step by comparing adjacent elements.
* Demonstrates sorting both **words** and **characters**.

**4. Sample Output**



Project 3: Credit Card Validator

**1. Code Explanation**

This program validates a credit card number using the **Luhn Algorithm**.

* Checks if input contains only digits.
* Converts string to a list of integers.
* Removes last digit → x (check digit).
* Reverses the remaining digits.
* Doubles every second digit (even index).
* If any doubled digit > 9 → subtract 9.
* Computes sum of digits + check digit.
* If total is divisible by 10 → card is valid. Otherwise invalid.

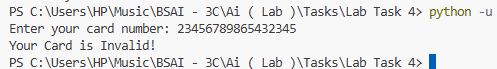
**2. How It Works**

* User enters a card number.
* Algorithm processes digits according to Luhn’s rules.
* Checks validity by divisibility rule.
* Prints result as **Valid** or **Invalid**.

**3. Why This Approach**

* The **Luhn algorithm** is widely used in real-life credit/debit card systems.
* Simple yet powerful example of **algorithmic problem-solving**.
* Demonstrates loops, conditionals, and list operations in Python.

**4. Sample Output**

****