#### **Name: Javeria Manzoor**

**Roll no: BSAI-3C-201**

**Submitted to: Sir Rasikh**

**Lab task:04**

**Subject:Artificial Intelligence**

### **Part 1: Luhn Check Algorithm (Credit Card Validation)**

### **How it works:**

### The code takes a credit card number as a string. Example: "4532015112830366" It reverses the number and converts each character into an integer: digits = [int(d) for d in card\_number[::-1]]

### → So the digits are processed from right to left, as required by the Luhn algorithm. It loops through each digit with its index: for i, d in enumerate(digits):

### If the index i is odd, the digit is doubled.

### If doubling gives a number greater than 9, subtract 9 from it. (This is equivalent to adding the digits together, e.g. 12 → 1+2 = 3 → 12−9 = 3)

### All processed digits are added together to get a total.

### Finally: return total % 10 == 0

### If the total ends in 0, the card number is VALID according to the Luhn formula.

### **Why it works**

### The Luhn algorithm is used globally to verify if a credit or debit card number is entered correctly.

### It helps catch typing errors or invalid card numbers before processing.

### It doesn’t verify if the card actually exists — just if it’s structurally valid.

### 

### 

## **Part 2: Remove Punctuations from a String**

### **How it works:**

### The function remove\_punctuations() creates a translation table using: translator = str.maketrans('', '', string.punctuation)

### string.punctuation contains all common punctuation marks like ! , . ? - ' " etc. Then it removes all these punctuation marks using: text.translate(translator)

### The result is a clean string without punctuation: Original: Hello!!!, he said --- and went.Cleaned: Hello he said and went

### 

### 

### **Why it works:**

### Removing punctuation is a text preprocessing step in Natural Language Processing (NLP), data cleaning, and text analysis. It makes text simpler and cleaner for further processing (like counting words, training models, etc.).

### 

## Sorting Words in a Sentence Alphabetically

### 🔹 How it Works

### **Function Definition** def sort\_sentence(sentence: str) -> str:

### 

### The function sort\_sentence() takes a **sentence** as input.

### The type hint str means the parameter and return value are both strings.

### 

### **Split the Sentence into Words** words = sentence.split()

### 

### The split() function breaks the sentence into a list of **individual words**, using spaces as separators. Example: "Python is a Powerful and Easy to Learn Language." becomes ['Python', 'is', 'a', 'Powerful', 'and', 'Easy', 'to', 'Learn', 'Language.']

### 

### **Sort the Words Alphabetically (Ignoring Case)** words.sort(key=str.lower)

### The sort() method arranges the list **alphabetically**.

### The key=str.lower ensures sorting is **case-insensitive**, so “Python” and “powerful” are compared fairly (both treated as lowercase during sorting).

### Example after sorting: ['a', 'and', 'Easy', 'is', 'Language.', 'Learn', 'Powerful', 'Python', 'to']

### **Join the Words Back into a Sentence** return " ".join(words)

### " ".join(words) combines all words into a single string separated by spaces. The sorted sentence is then returned.

### 

### **Print the Results** print("Original Sentence:", sentence)

### print("Alphabetically Sorted:", sorted\_sentence)

### Displays both the original and the alphabetically sorted version.

### 

### **Why it Works:**

### **Purpose:** To rearrange words alphabetically for easier reading or text analysis.

### **Reason for key=str.lower:** Ensures that uppercase and lowercase words are treated the same (so sorting is accurate).

### **Use Case:** This method is useful in **natural language processing**, **word organization**, or when you want to alphabetically display words in a document or paragraph.



### 

### 

### 

### 