Veermata Jijabai Technological Institute, Mumbai 400019

Experiment No.: 05

Aim: Text Processing at the Lowest Level, Text Processing with Unicode, Regular Expressions for Detecting Word Patterns, Useful Applications of Regular Expressions, Normalizing Text, Regular Expressions.

Name: Kiran K Patil

Enrolment No.: 211070904

Branch: Computer Engineering

Batch: D

Theory:

Text Processing at the Lowest Level:

1) Character Encoding and Decoding:

- **Character Encoding:** Text is stored as a sequence of bytes, and character encoding defines the mapping between these bytes and characters. Common character encodings include ASCII, UTF-8, and UTF-16. Choosing the appropriate encoding is crucial for correctly interpreting and manipulating text data.

- **Character Manipulation:** At the lowest level, you may need to deal with individual characters, inspecting their binary representations, and performing operations like counting occurrences, extracting substrings, or checking character properties.

Text Processing with Unicode:

1) Unicode Representation:

- **Code Points:** Unicode assigns a unique code point to each character. This standardization facilitates the representation of characters from diverse writing systems. Understanding code points is fundamental for working with Unicode.

- **UTF Encodings**: Unicode Transformation Formats (UTF-8, UTF-16, UTF-32) are methods for encoding Unicode characters into byte sequences. UTF-8 is widely used for its efficiency and compatibility.

2) Handling Multilingual Text:

- **Multilingual Support:** Unicode enables the handling of multilingual text seamlessly. This is essential for applications dealing with data in various languages and scripts.

Regular Expressions for Detecting Word Patterns:

1) Basics of Regular Expressions:

- **Pattern Matching**: Regular expressions are patterns used for matching character combinations in strings. They can include literal characters, metacharacters for more complex patterns, and quantifiers for repetition.

- **Character Classes:** Define sets of characters. For example, [aeiou] matches any vowel.

2) Applications of Regular Expressions:

- **Email Validation:** Using a regular expression to validate email addresses by checking if they follow a specified pattern.

- **Extracting Information:** Regular expressions can be employed to extract specific information from unstructured text, such as extracting phone numbers or dates.

- **Tokenization:** Breaking down text into tokens, which can be words, phrases, or sentences. Regular expressions can define the criteria for token boundaries.

Useful Applications of Regular Expressions:

1) Tokenization and Text Analysis:

- **Data Cleaning:** Regular expressions play a role in cleaning and preprocessing text data by removing unwanted characters, formatting inconsistencies, or noise.

- **Pattern-Based Analysis:** Identifying patterns in text that may be indicative of specific information, such as mentions of hashtags or URLs in social media data.

Normalizing Text:

1) Techniques for Text Normalization:

- **Lowercasing:** Converting all characters to lowercase to ensure consistent matching and analysis.

- **Stemming and Lemmatization:** Reducing words to their base or root form to handle variations in word forms.

- **Removing Stopwords:** Eliminating common words that do not contribute significant meaning.

2) Importance of Text Normalization:

- **Consistency:** Normalizing text ensures consistency in representations, reducing the impact of variations on downstream NLP tasks.

- **Improved Analysis:** Normalized text is often easier to analyze and model, especially when dealing with large and diverse datasets.

Regular Expressions:

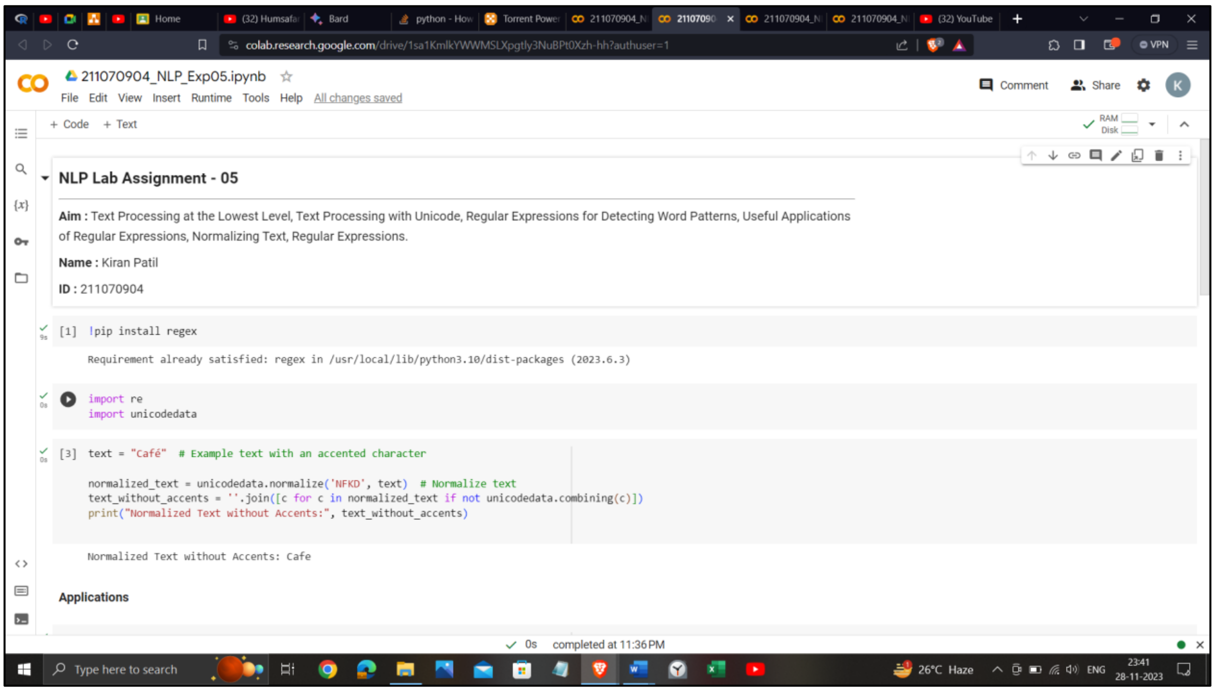
1) Advanced Regular Expression Concepts:

- **Capturing Groups:** Using parentheses to capture portions of a matched pattern for further analysis.

- **Quantifiers:** Specifying the number of occurrences of a character or group, such as * for zero or more occurrences.

- **Backreferences:** Referring back to previously captured groups within the regular expression.

Implementation:



211070904_NLP_Exp05.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text

NLP Lab Assignment - 05

Aim : Text Processing at the Lowest Level, Text Processing with Unicode, Regular Expressions for Detecting Word Patterns, Useful Applications of Regular Expressions, Normalizing Text, Regular Expressions.

Name : Kiran Patil

ID : 211070904

```
[1] !pip install regex
```

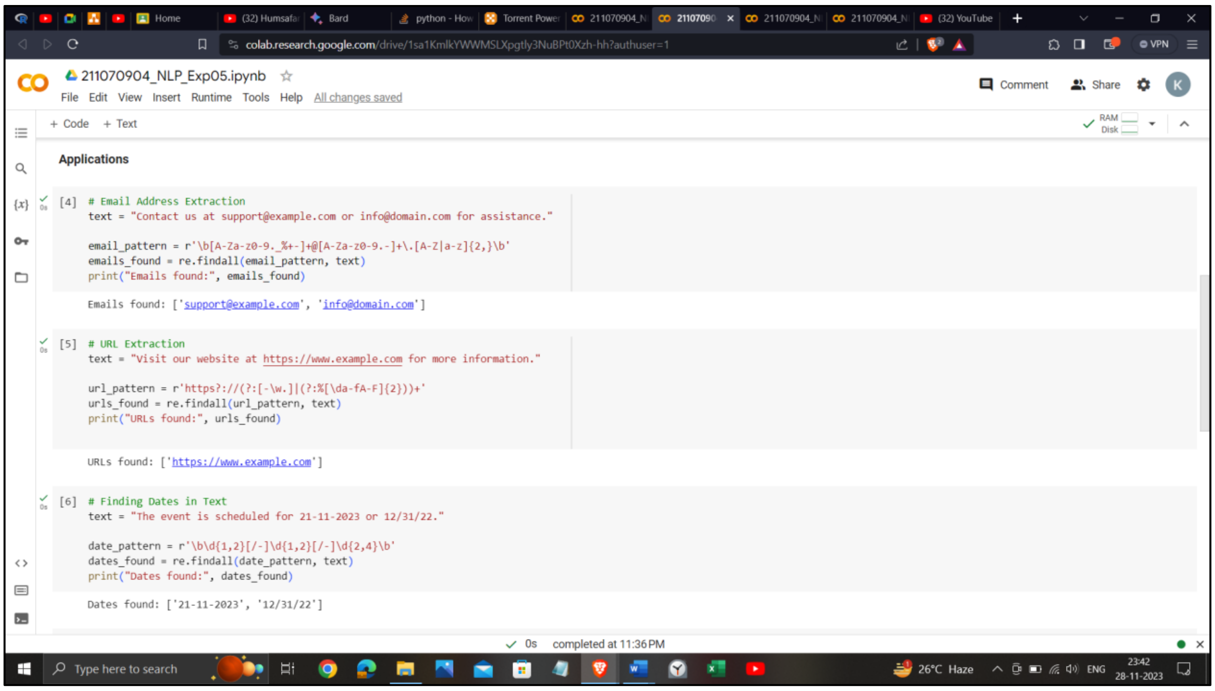
```
Requirement already satisfied: regex in /usr/local/lib/python3.10/dist-packages (2023.6.3)
```

```
import re
import unicodedata
```

```
[3] text = "Café"  # Example text with an accented character

    normalized_text = unicodedata.normalize('NFKD', text)  # Normalize text
    text_without_accents = ''.join([c for c in normalized_text if not unicodedata.combining(c)])
    print("Normalized Text without Accents:", text_without_accents)
```

```
Normalized Text without Accents: Cafe
```

Applications



211070904_NLP_Exp05.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text

Applications

```
[4] # Email Address Extraction
    text = "Contact us at support@example.com or info@domain.com for assistance."

    email_pattern = r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b'
    emails_found = re.findall(email_pattern, text)
    print("Emails found:", emails_found)
```

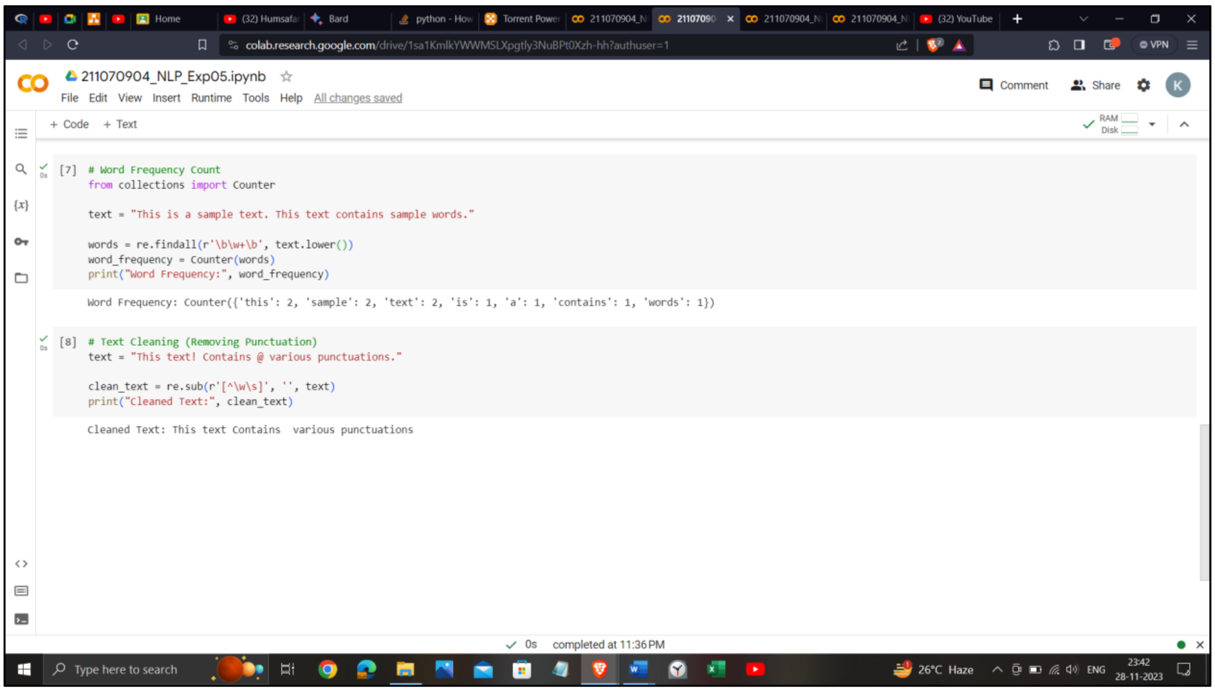
```
Emails found: ['support@example.com', 'info@domain.com']
```

```
[5] # URL Extraction
    text = "Visit our website at https://www.example.com for more information."

    url_pattern = r'https?://(?:[-\w.]|(?:%[\da-fA-F]{2}))+'
    urls_found = re.findall(url_pattern, text)
    print("URLs found:", urls_found)
```

```
URLs found: ['https://www.example.com']
```

```
[6] # Finding Dates in Text
    text = "The event is scheduled for 21-11-2023 or 12/31/22."

    date_pattern = r'\b\d{1,2}[/-]\d{1,2}[/-]\d{2,4}\b'
    dates_found = re.findall(date_pattern, text)
    print("Dates found:", dates_found)
```

```
Dates found: ['21-11-2023', '12/31/22']
```


colab.research.google.com/drive/1sa1KmlkYWWMSLXpgtly3NuBPt0Xzh-hh?authuser=1
211070904_NLP_Exp05.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
0s completed at 11:36 PM

```
[7] # Word Frequency Count
    from collections import Counter

    text = "This is a sample text. This text contains sample words."

    words = re.findall(r'\b\w+\b', text.lower())
    word_frequency = Counter(words)
    print("Word Frequency:", word_frequency)

    Word Frequency: Counter({'this': 2, 'sample': 2, 'text': 2, 'is': 1, 'a': 1, 'contains': 1, 'words': 1})
```

```
[8] # Text Cleaning (Removing Punctuation)
    text = "This text! Contains @ various punctuations."

    clean_text = re.sub(r'[^\w\s]', '', text)
    print("Cleaned Text:", clean_text)

    Cleaned Text: This text Contains  various punctuations
```

https://colab.research.google.com/drive/1sa1KmlkYWWMSLXpgtly3NuBPt0Xzh-hh?usp=sharing

Conclusion:

This experiment aims to provide a comprehensive understanding of text processing techniques, with a particular emphasis on Unicode handling, regular expressions, and text normalization. The findings will contribute to the development of more effective and robust text processing pipelines for applications such as information retrieval and natural language processing.