3.1

import re

def match\_expression\_before\_linebreak(text, pattern):

# Regular expression to match the pattern at the start of the string before a line break

regex = r"^" + pattern + r"(?!.\*\n)" # Match at the beginning before line break

match = re.match(regex, text)

if match:

return match.group(0)

else:

return "No match found"

# Example usage:

text = "Hello world!\nThis is a second line."

pattern = r"Hello" # Pattern to match

result = match\_expression\_before\_linebreak(text, pattern)

print(result) # Output: "Hello"

(ii)

import re

def match\_expression\_before\_linebreak(text, pattern):

# Regular expression to match the pattern before the line break at the end of the string

regex = r"(?<=.\*" + pattern + r").\*(?=\n|$)" # Match pattern at the end before the line break

match = re.search(regex, text)

if match:

return match.group(0)

else:

return "No match found"

# Example usage:

text = "This is an example text\nAnother line here\nEnd of the line with pattern"

pattern = r"line" # Pattern to match

result = match\_expression\_before\_linebreak(text, pattern)

print(result) # Output: "line with pattern"

(iii)

import re

def match\_exactly\_one\_char(pattern, text):

# Using regex to match the pattern in the string

match = re.match(pattern, text)

if match:

return match.group(0) # Return the matched part of the string

else:

return "No match found"

# Example usage:

pattern = r"a.b" # Pattern to match: "a" followed by any one character, then "b"

text = "an" # The string to check

result = match\_exactly\_one\_char(pattern, text)

print(result) # Output: "No match found"

# But, let's modify to match "ab" properly

text = "ab" # The string to check

result = match\_exactly\_one\_char(pattern, text)

print(result) # Output: "ab"

3.2

import re

# Sample text to demonstrate regex functions

text = """

Natural Language Processing (NLP) is a field of artificial intelligence

that focuses on the interaction between computers and human language.

It includes tasks like text classification, sentiment analysis, and named entity recognition (NER).

"""

# 1. Finding all words starting with capital letters (e.g., proper nouns, named entities)

capital\_words = re.findall(r'\b[A-Z][a-z]\*\b', text)

print("Capitalized words (Proper Nouns):", capital\_words)

# 2. Extracting email addresses (if present)

# Let's add an email to the text for demonstration.

text\_with\_email = text + " Contact us at support@nlp.com for more information."

emails = re.findall(r'[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}', text\_with\_email)

print("Extracted Emails:", emails)

# 3. Tokenizing the text into words (splitting text into words)

words = re.findall(r'\b\w+\b', text)

print("\nTokenized words:", words)

# 4. Removing punctuation (cleaning the text)

cleaned\_text = re.sub(r'[^\w\s]', '', text)

print("\nText without punctuation:", cleaned\_text)

# 5. Replacing specific words (e.g., replace 'NLP' with 'Natural Language Processing')

replaced\_text = re.sub(r'\bNLP\b', 'Natural Language Processing', text)

print("\nText after replacing 'NLP' with 'Natural Language Processing':", replaced\_text)

# 6. Finding sentences (split text by periods to identify sentences)

sentences = re.split(r'(?<=\.)\s+', text)

print("\nSentences split by period:", sentences)

# 7. Extracting words that are part of a specific pattern (e.g., words containing 'ai')

ai\_words = re.findall(r'\b\w\*ai\w\*\b', text)

print("\nWords containing 'ai':", ai\_words)

# 8. Matching phone numbers (example with a format like '123-456-7890')

phone\_text = "Call us at 123-456-7890 or 987-654-3210."

phone\_numbers = re.findall(r'\d{3}-\d{3}-\d{4}', phone\_text)

print("\nExtracted Phone Numbers:", phone\_numbers)

# 9. Using re.match to check if a string starts with a pattern

match\_start = re.match(r'^Natural', text)

if match\_start:

print("\nMatch at the start of the text:", match\_start.group())

else:

print("\nNo match at the start of the text.")

# 10. Using re.search to find the first occurrence of a pattern

search\_result = re.search(r'processing', text)

if search\_result:

print("\nFirst occurrence of 'processing' found at position:", search\_result.start())

**3.3**

import re

import nltk

from nltk.corpus import stopwords

from nltk.stem import PorterStemmer

# Ensure you have the necessary resources downloaded

nltk.download('stopwords')

# Sample text

text = """

Natural Language Processing (NLP) is a field of artificial intelligence

that focuses on the interaction between computers and human language.

It includes tasks like text classification, sentiment analysis, and named entity recognition (NER).

"""

# 1. Convert to lowercase

def to\_lowercase(text):

return text.lower()

# 2. Remove punctuation

def remove\_punctuation(text):

return re.sub(r'[^\w\s]', '', text)

# 3. Remove special characters and numbers

def remove\_special\_chars\_and\_numbers(text):

return re.sub(r'[^a-zA-Z\s]', '', text)

# 4. Tokenize the text into words

def tokenize\_text(text):

return re.findall(r'\b\w+\b', text)

# 5. Remove stop words

def remove\_stopwords(tokens):

stop\_words = set(stopwords.words('english'))

return [word for word in tokens if word not in stop\_words]

# 6. Stemming (Optional)

def stem\_words(tokens):

stemmer = PorterStemmer()

return [stemmer.stem(word) for word in tokens]

# Preprocessing pipeline

def preprocess\_text(text):

# Step 1: Convert to lowercase

text = to\_lowercase(text)

# Step 2: Remove punctuation

text = remove\_punctuation(text)

# Step 3: Remove special characters and numbers

text = remove\_special\_chars\_and\_numbers(text)

# Step 4: Tokenize the text

tokens = tokenize\_text(text)

# Step 5: Remove stop words

tokens = remove\_stopwords(tokens)

# Step 6: Apply stemming (optional)

tokens = stem\_words(tokens)

return tokens

# Demonstrating the preprocessing

preprocessed\_text = preprocess\_text(text)

print("Original Text:")

print(text)

print("\nPreprocessed Tokens:")

print(preprocessed\_text)

output:

Original Text:

Natural Language Processing (NLP) is a field of artificial intelligence

that focuses on the interaction between computers and human language.

It includes tasks like text classification, sentiment analysis, and named entity recognition (NER).

Preprocessed Tokens:

['natur', 'languag', 'process', 'field', 'artifici', 'intellig', 'focus', 'interact', 'comput', 'human', 'languag', 'includ', 'task', 'like', 'text', 'classif', 'sentiment', 'analysi', 'nam', 'enti', 'recognit']

**3.4**

import re

# Sample text

text = "Hello, this is a test string for regex anchors."

# 1. Using ^ anchor: Match the beginning of the string

def match\_start\_of\_string(text, pattern):

match = re.match(pattern, text)

if match:

return f"Match found: {match.group(0)}"

else:

return "No match at the beginning of the string."

# 2. Using $ anchor: Match the end of the string

def match\_end\_of\_string(text, pattern):

match = re.search(pattern, text)

if match and match.end() == len(text):

return f"Match found: {match.group(0)}"

else:

return "No match at the end of the string."

# 3. Match a word at the start of the string using ^ anchor

pattern\_start = r"^Hello"

result\_start = match\_start\_of\_string(text, pattern\_start)

print(result\_start) # Should print: Match found: Hello

# 4. Match a word at the end of the string using $ anchor

pattern\_end = r"anchors\.$" # Note that we escape the period since it has a special meaning

result\_end = match\_end\_of\_string(text, pattern\_end)

print(result\_end) # Should print: Match found: anchors.

# 5. Using ^ and $ together to check if the string matches exactly

def match\_entire\_string(text, pattern):

if re.fullmatch(pattern, text):

return f"Exact match found: {text}"

else:

return "No exact match."

# Check for an exact match with the whole string

pattern\_exact = r"^Hello, this is a test string for regex anchors\.$"

result\_exact = match\_entire\_string(text, pattern\_exact)

print(result\_exact) # Should print: Exact match found: Hello, this is a test string for regex anchors.

Output:

Match found: Hello

Match found: anchors.

Exact match found: Hello, this is a test string for regex anchors.

**3.5 Grouping in Regular Expressions**

import re

# Sample text

text = "John's email is john.doe@example.com and Jane's email is jane.smith@domain.org."

# 1. Basic Grouping: Extracting emails and their usernames using parentheses (grouping)

def extract\_emails(text):

# Group 1 captures the username, Group 2 captures the domain

pattern = r'(\w+\.\w+)@(\w+\.\w+\.\w+)' # Username and domain group

matches = re.findall(pattern, text)

if matches:

return matches

else:

return "No emails found."

# 2. Named Groups: Capture using named groups for better clarity

def extract\_named\_groups(text):

# Using named groups for email extraction

pattern = r'(?P<username>\w+\.\w+)@(?P<domain>\w+\.\w+\.\w+)'

matches = re.findall(pattern, text)

if matches:

return matches

else:

return "No emails found."

# 3. Referencing Groups: Extracting the part after '@' using groups

def extract\_after\_at\_symbol(text):

# Capture everything after '@' in Group 2

pattern = r'(\w+\.\w+)@(\w+\.\w+\.\w+)' # Group 1: username, Group 2: domain

matches = re.findall(pattern, text)

# Return the domain part (Group 2)

if matches:

domains = [match[1] for match in matches] # Extract the domain part (Group 2)

return domains

else:

return "No domains found."

# Demonstrating the grouping

emails = extract\_emails(text)

print("Extracted Emails (Username + Domain):", emails)

named\_groups = extract\_named\_groups(text)

print("\nExtracted Named Groups (Username + Domain):", named\_groups)

domains = extract\_after\_at\_symbol(text)

print("\nExtracted Domains:", domains)

# 4. Example with multiple groups: Extracting date in "dd/mm/yyyy" format

def extract\_date(text):

# Group 1 captures the day, Group 2 captures the month, Group 3 captures the year

pattern = r'(\d{2})/(\d{2})/(\d{4})' # Day/Month/Year

match = re.search(pattern, text)

if match:

# Accessing individual groups

day = match.group(1)

month = match.group(2)

year = match.group(3)

return f"Extracted Date: {day}-{month}-{year}"

else:

return "No date found."

# Sample text with a date

text\_with\_date = "Today's date is 17/03/2025."

date = extract\_date(text\_with\_date)

print("\nExtracted Date:", date)

**Output:**

Extracted Emails (Username + Domain): [('john.doe', 'example.com'), ('jane.smith', 'domain.org')]

Extracted Named Groups (Username + Domain): [('john.doe', 'example.com'), ('jane.smith', 'domain.org')]

Extracted Domains: ['example.com', 'domain.org']

Extracted Date: 17-03-2025