1. Implement a Conflation algorithm to generate a document representative of a text file.

```
import re
import nltk

class ConflationAlgorithm:
    def __init__(self, stop_words=None):
        self.stop_words = stop_words or set()
        self.stemmer = nltk.stem.PorterStemmer()

    def conflate(self, text):
        # Tokenize the text.
        tokens = re.split(r'\s+', text)

        # Remove stop words.
        tokens = [token for token in tokens if token not in self.stop_words]

        # Stem the words.
        tokens = [self.stemmer.stem(token) for token in tokens]

        # Identify synonyms.
        synonyms = {}
        for token in tokens:
            synonyms.setdefault(token, set()).add(token)

            # TODO: Use a wordnet or statistical method to identify synonyms.

        # Group the words into classes.
        classes = []
        for token in tokens:
            class_found = False
            for class_ in classes:
                if token in class_:
                    class_.add(token)
                    class_found = True
                    break

            if not class_found:
                classes.append(set([token]))

        # Generate the document representative.
        document_representative = []
        for class_ in classes:
            document_representative.append(max(class_, key=len))

        return document_representative

# Example usage:

conflation_algorithm = ConflationAlgorithm()
document_representative = conflation_algorithm.conflate('This is a sample text file used for testing the conflation alg
    It contains various sentences and words that will be combined with other text files. \
    The goal is to create a representative document that captures the essential information from all the input files. \
    Please note that this is just a sample, and in a real-world scenario, you would use actual text data relevant \
    to your project or analysis.')

print(document_representative)
```

```
['thi', 'is', 'a', 'sampl', 'text', 'file', 'use', 'for', 'test', 'the', 'conflat', 'algorithm.', 'it', 'contain', 'variou', 'senter
```

3. Implement a program for retrieval of documents using inverted files.

```
import os
import re
from collections import defaultdict

# Function to tokenize and preprocess text
def preprocess_text(text):
    text = text.lower()  # Convert to lowercase
    tokens = re.findall(r'\w+', text)  # Tokenize
    return tokens

# Function to build an inverted index from a collection of documents
def build_inverted_index(documents):
    inverted_index = defaultdict(list)

    for doc_id, document in enumerate(documents):
        tokens = preprocess_text(document)
        for term in tokens:
            inverted_index[term].append(doc_id)

    return inverted_index

# Function to retrieve documents containing a query term
def retrieve_documents(query, inverted_index, documents):
    query = preprocess_text(query)
    results = set()

    for term in query:
        if term in inverted_index:
            results.update(inverted_index[term])

    if not results:
        print("No documents found for the query.")
        return

    print("Documents containing the query term:")
    for doc_id in results:
        print(f"Document {doc_id + 1}:\n{documents[doc_id]}\n")

# Sample collection of documents
documents = [
    "This is a sample document about inverted files.",
    "Inverted files are commonly used in information retrieval systems.",
    "Information retrieval is the process of obtaining information from a large dataset.",
    "The inverted index is a crucial data structure for document retrieval.",
]

# Build the inverted index
inverted_index = build_inverted_index(documents)

# User input for the query term
query_term = input("Enter a query term: ")

# Retrieve and print documents containing the query term
retrieve_documents(query_term, inverted_index, documents)
```

```
Enter a query term: for
Documents containing the query term:
Document 4:
The inverted index is a crucial data structure for document retrieval.
```

1. Implement a program to calculate precision and recall for sample input. (Answer set A, Query q1, Relevant documents to query q1- Rq1)

```
def calculate_precision_and_recall(relevant_documents, retrieved_documents):
  precision = len(relevant_documents & retrieved_documents) / len(retrieved_documents)
  recall = len(relevant_documents & retrieved_documents) / len(relevant_documents)
  return precision, recall

# Sample input:
relevant_documents = {"1", "2", "3"}
retrieved_documents = {"1", "2", "4"}

# Calculate precision and recall.
precision, recall = calculate_precision_and_recall(relevant_documents, retrieved_documents)

# Print the results.
print("Precision:", precision)
print("Recall:", recall)
```

```
Precision: 0.6666666666666666
Recall: 0.6666666666666666
```

2. Write a program to calculate the harmonic mean (F-measure) and E-measure for the above example.

```
def calculate_f_measure(precision, recall, beta=1):
  f_measure = (1 + beta**2) * (precision * recall) / ((beta**2 * precision) + recall)
  return f_measure


def calculate_e_measure(precision, recall, beta=1):
  e_measure = (1 + beta**2) * (precision * recall) / (precision + (beta**2 * recall))
  return e_measure


# Sample input:
precision = 0.6666666666666666
recall = 0.6666666666666666

# Calculate F-measure and E-measure.
f_measure = calculate_f_measure(precision, recall)
e_measure = calculate_e_measure(precision, recall)

# Print the results.
print("F-measure:", f_measure)
print("E-measure:", e_measure)
```

```
F-measure: 0.6666666666666666
E-measure: 0.6666666666666666
```

1. Build the web crawler to pull product information and links from an e-commerce website. (Python)

```
!pip install scrapy
```

```
import scrapy

class ProductItem(scrapy.Item):
    title = scrapy.Field()
    price = scrapy.Field()
    link = scrapy.Field()


class EcomSpider(scrapy.Spider):
    name = 'ecom'
    allowed_domains = ['https://www.amazon.in']  # Replace with the e-commerce website's domain
    start_urls = ['https://www.amazon.in/Dell-Inspiron-i5-1335U-Processor-Comfortview/dp/B0CCSQYP9K/ref=sxin_14_hcs-la·

    def parse(self, response):
        for product_selector in response.css('.product'):
            item = ProductItem()
            item['title'] = product_selector.css('.product-title::text').get()
            item['price'] = product_selector.css('.product-price::text').get()
            item['link'] = product_selector.css('a.product-link::attr(href)').get()
            yield item

        # Follow pagination links
        next_page = response.css('a.next::attr(href)').get()
        if next_page:
            yield response.follow(next_page, self.parse)
```

2. Write a program to find the live weather report (temperature, wind speed, description, and weather) of a given city. (Python).

```
import requests

# Replace 'YOUR_API_KEY' with your OpenWeatherMap API key
API_KEY = '12eb1f20e78790086e53caa1a4a1eb3b'
base_url = f'https://api.openweathermap.org/data/2.5/weather?q=pune&appid={API_KEY}'

def get_weather(city):

    try:
        response = requests.get(base_url)
        data = response.json()

        if response.status_code == 200:
            # Extract weather information
            temperature = data['main']['temp']
            wind_speed = data['wind']['speed']
            description = data['weather'][0]['description']
            weather = data['weather'][0]['main']

            print(f"Weather in {city}:")
            print(f"Temperature: {temperature}°C")
            print(f"Wind Speed: {wind_speed} m/s")
            print(f"Description: {description}")
            print(f"Weather: {weather}")
        else:
            print(f"Error: Unable to retrieve weather data for {city}")

    except Exception as e:
        print(f"An error occurred: {str(e)}")

if __name__ == '__main__':
    city = input("Enter the city name: ")
    get_weather(city)
```

```
Enter the city name: pune
Weather in pune:
Temperature: 301.78°C
Wind Speed: 3.04 m/s
Description: overcast clouds
Weather: Clouds
```