Experiment - 10

Aim:

Write a programme to implement tic-tac-toe game.

Program:

```
# Set up the game board as a list

board = ["-", "-", "-",

         "-", "-", "-",

         "-", "-", "-"]


# Define a function to print the game board

def print_board():

print(board[0] + " | " + board[1] + " | " +
board[2]) print(board[3] + " | " + board[4] + " | "
+ board[5]) print(board[6] + " | " + board[7] + " |
" + board[8])


# Define a function to handle a player's turn

def take_turn(player):

    print(player + "'s turn.")

    position = input("Choose a position from 1-9: ")

    while position not in ["1", "2", "3", "4", "5", "6", "7", "8","9"]:

        position = input("Invalid input. Choose a position from 1-9: ")

    position = int(position) - 1
    while board[position] != "-":
        position = int(input("Position already taken. Choose a different position:"))-1
    board[position] = player
    print_board()


def check_game_over(): # Check for a win

    if (board[0] == board[1] == board[2] != "-") or \

       (board[3] == board[4] == board[5] != "-") or \

       (board[6] == board[7] == board[8] != "-") or \

       (board[0] == board[3] == board[6] != "-") or \

       (board[1] == board[4] == board[7] != "-") or \
```

```
            (board[2] == board[5] == board[8] != "-") or \
            (board[0] == board[4] == board[8] != "-") or \
            (board[2] == board[4] == board[6] != "-"):
        return "win"
    elif "-" not in board:
        return "tie"
    else:
        return "play"

def play_game():
    print_board()
    current_player = "X"
    game_over = False
    while not game_over:
        take_turn(current_player)
        game_result = check_game_over()
        if game_result == "win":
            print(current_player + " wins!")
            game_over = True
        elif game_result == "tie":
            print("It's a tie!")
            game_over = True
        else:
            # Switch to the other player
            current_player = "O" if current_player == "X" else "X"

# Start the game play_game()
```

Output:

```
X | X | -
- | O | -
- | - | -
O's turn.
Choose a position from 1-9: 9
X | X | -
- | O | -
- | - | O
X's turn.
Choose a position from 1-9: 3
X | X | X
- | O | -
- | - | O
X wins!
```

Experiment - 11

Aim:

Write a program to remove stop words for a given passage from a text file using NLTK

Program:

```
import nltk from nltk.corpus import stopwords

nltk.download('stopwords')

stop_words = set(stopwords.words('english'))

def remove_stopwords(text):
    # Tokenize the text
    words = nltk.word_tokenize(text)
    # Remove stopwords
    filtered_words = [word for word in words if word.lower() not in stop_words]
    # Join the words back into a string
    filtered_text = ' '.join(filtered_words) return
    filtered_text

def main():
    file_path = input("Enter the file path: ")
    try:
        with open(file_path, 'r') as file:
            passage = file.read()
            cleaned_passage = remove_stopwords(passage)
            print("Passage without stop words:")
            print(cleaned_passage)
    except FileNotFoundError:
        print("File not found. Please check the file path.")

if__name____== "_main_":
    main()
```

Output:

```
['This', 'is', 'a', 'sample', 'sentence', ',', 'showing', 'off', 'the', 'stop',
'words', 'filtration', '.']
['This', 'sample', 'sentence', ',', 'showing', 'stop', 'words', 'filtration', '.
']
```

Experiment - 12

Aim:

Write a program to implement stemming for a given sentence using NLTK.

Program:

```
import nltk from nltk.stem import PorterStemmer

# Download NLTK resources (punkt tokenizer and porter stemmer)
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')

# Create a PorterStemmer object
stemmer = PorterStemmer()

def stem_sentence(sentence):
    words = nltk.word_tokenize(sentence)
    # Stem each word
    stemmed_words = [stemmer.stem(word) for word in words]
    # Join the stemmed words back into a sentence
    stemmed_sentence = ' '.join(stemmed_words) return
    stemmed_sentence

def main():
    sentence = input("Enter a sentence: ")
    stemmed_sentence = stem_sentence(sentence)
    print("Stemmed sentence:")
    print(stemmed_sentence)

if__name_____== "_main_":
    main()
```

Output:

```
Original Sentence: He plays football and runs quickly.
Stemmed Sentence: he play footbal and run quickly .
 Original Sentence: He plays football and runs quickly.
Stemmed Sentence: he play footbal and run quickly .
```

Experiment - 13

Aim:

Write a program to POS (part of speech) tagging for the give sentence using NLTK.

Program:

```
import nltk

# Download NLTK resources (punkt tokenizer and averaged perceptron tagger)
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')

def pos_tag_sentence(sentence):
    words = nltk.word_tokenize(sentence)
    words = nltk.pos_tag(words)
    return tagged_words
def main():
    sentence = input("Enter a sentence: ")
    tagged_sentence = pos_tag_sentence(sentence)
    print("POS tagged sentence:")
    print(tagged_sentence)
if__name____== "_main_":
    main()
```

Output:

```
Tagged Words: [('The', 'DT'), ('quick', 'JJ'), ('brown', 'NN'), ('fox', 'NN'), (
'jumps', 'VBZ'), ('over', 'IN'), ('the', 'DT'), ('lazy', 'JJ'), ('dog', 'NN'), (
'.', '.')]
```

Experiment - 14

Aim:

Write a program to implement Lemmatization using NLTK.

Program:

```
import nltk from nltk.stem
import WordNetLemmatizer

# Download NLTK resources (WordNet)
nltk.download('wordnet')

# Create a WordNetLemmatizer object
lemmatizer = WordNetLemmatizer()

def lemmatize_sentence(sentence):
    # Tokenize the sentence
    words = nltk.word_tokenize(sentence)
    # Lemmatize each word
    lemmatized_words = [lemmatizer.lemmatize(word) for word in words]
    # Join the lemmatized words back into a sentence
    lemmatized_sentence = ' '.join(lemmatized_words) return
    lemmatized_sentence

def main():
    sentence = input("Enter a sentence: ")
    lemmatized_sentence = lemmatize_sentence(sentence)
    print("Lemmatized sentence:")
    print(lemmatized_sentence)

if__name_____== "_main_":
    main()
```

Output:

```
Original Sentence: The quick brown foxes are jumping over the lazy dogs
Lemmatized Sentence: The quick brown fox are jumping over the lazy dog
```

Experiment - 15

Aim:

Write a programme to implement tic-tac-toe game.

Program:

```
import nltk from nltk.tokenize
import word_tokenize from nltk.corpus
import stopwords from nltk.stem
import WordNetLemmatizer from sklearn.feature_extraction.text
import TfidfVectorizer from sklearn.naive_bayes
import MultinomialNB from sklearn.pipeline import Pipeline

# Download NLTK resources (WordNet and stopwords)
nltk.download('wordnet') nltk.download('stopwords')

# Create a WordNetLemmatizer object lemmatizer =
WordNetLemmatizer()

# Define a function to preprocess text def
preprocess_text(text):
# Tokenize the text
tokens = word_tokenize(text)
# Remove stopwords
    stopwords_list = set(stopwords.words('english')) filtered_tokens = [word for word in
    tokens if word.lower() not in stopwords_list]
    # Lemmatize tokens
    lemmatized_tokens = [lemmatizer.lemmatize(word) for word in filtered_tokens]
    return ' '.join(lemmatized_tokens)
def classify_text(text):
    preprocessed_text = preprocess_text(text)
    classifier = Pipeline([
        ('tfidf', TfidfVectorizer()),
        ('clf', MultinomialNB()),
    ])
    # Sample training data
    X_train = ["I love this movie", "This movie is terrible", "This film is great"]
    Y_train = ["positive", "negative", "positive"]
```

```
    # Train the classifier
    classifier.fit(X_train, Y_train)
    # Predict the class of the input text
    predicted_class = classifier.predict([preprocessed_text])[0]
    return predicted_class
def main():
    sentence = input("Enter a sentence to classify: ")
    predicted_class = classify_text(sentence)
    print("Predicted class:", predicted_class)

if __name__ == "__main__":
    main()
```

Output:

```
Sentence: This movie was absolutely fantastic! I highly recommend it.
Sentiment: positive
```