|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Exp  . no |  | Experiment Name | Date of performance | R1 (3) | R2 (3) | R3 (3) | R4 (3) | R5 (3) | Total Marks (15) | Signature |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |

# Experiment - 8

**Aim:** Write a program to implement Hangman game.

## **Program:**

import random

def choose\_word():

words = ["doctor", "banana", "brain", "grape", "pear","monday","earth"]

return random.choice(words)

def display\_word(word, guessed\_letters):

display = ""

for letter in word:

if letter in guessed\_letters:

display += letter

else:

display += "\_"

return display

def hangman():

word = choose\_word()

guessed\_letters = []

attempts = 6

print("Welcome to Hangman!")

print(display\_word(word, guessed\_letters))

while attempts > 0:

guess = input("Guess a letter: ").lower()

if guess in guessed\_letters:

print("You've already guessed that letter.")

elif guess in word:

print("Correct guess!")

guessed\_letters.append(guess)

print(display\_word(word, guessed\_letters))

if "\_" not in display\_word(word, guessed\_letters):

print("Congratulations! You've guessed the word:", word)

break

else:

print("Incorrect guess!")

attempts -= 1

print("Attempts remaining:", attempts)

print(display\_word(word, guessed\_letters))

if attempts == 0:

print("Sorry, you've run out of attempts. The word was:", word)

hangman()

**OUTPUT:**

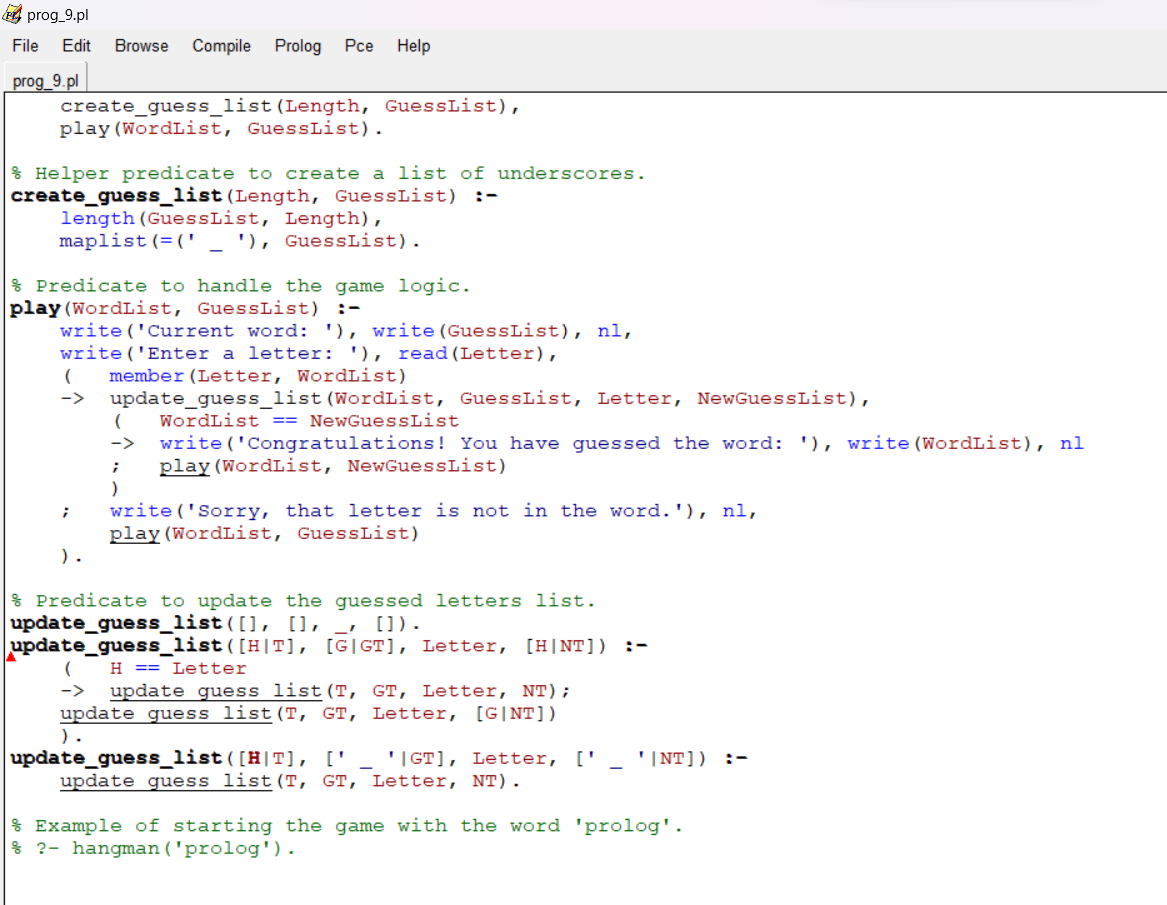


**Experiment – 9**

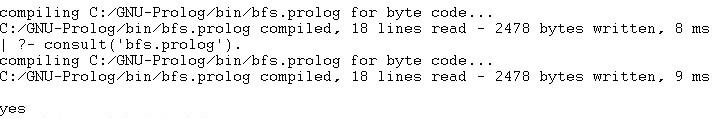
**Aim:**  Write a program to implement Hangman game.

**Source Code:**

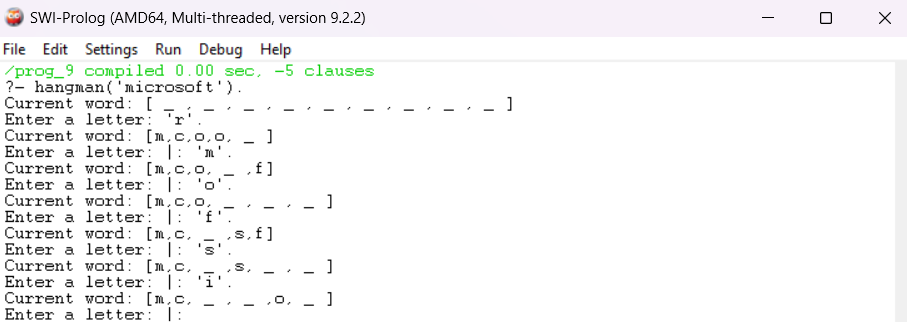
1. Create text file.



1. Open the prolog console and use the ‘consult’ command to compile the file and use it as a knowledge base.



**OUTPUT**

****

# Experiment - 10

**Aim:** Write a program to implement the 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()

# Define a function to check if the game is over

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"

# Check for a tie

elif "-" not in board:

return "tie"

# Game is not over

else:

return "play"

# Define the main game loop

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:

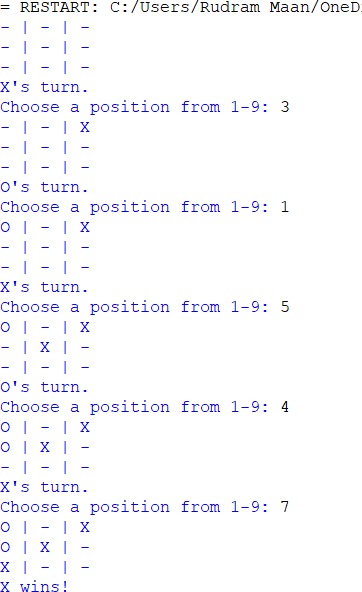
# Switch to the other player

current\_player = "O" if current\_player == "X" else "X"

# Start the game

play\_game()

**OUTPUT:**



# 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

from nltk.tokenize import word\_tokenize

nltk.download('stopwords')

example\_sent = """This is a sample sentence, showing off the stop words filtration."""

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

word\_tokens = word\_tokenize(example\_sent)

# Method 1: List comprehension

filtered\_sentence = [w for w in word\_tokens if not w.lower() in stop\_words]

# Method 2: Using a loop

filtered\_sentence\_loop = []

for w in word\_tokens:

if w.lower() not in stop\_words:

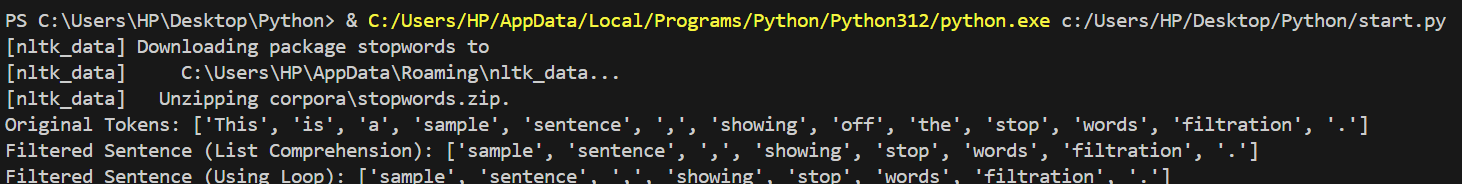
filtered\_sentence\_loop.append(w)

print("Original Tokens:", word\_tokens)

print("Filtered Sentence (List Comprehension):", filtered\_sentence)

print("Filtered Sentence (Using Loop):", filtered\_sentence\_loop)

**OUTPUT:**



# Experiment - 12

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

## **Theory:**

**Stemming** is the process of producing morphological variants of a root/base word. Stemming programs are commonly referred to as stemming algorithms or stemmers. A stemming algorithm reduces the words “chocolates”, “chocolatey”, and “choco” to the root word, “chocolate” and “retrieval”, “retrieved”, “retrieves” reduce to the stem “retrieve”.

## **Program:**

import nltk

from nltk.stem import PorterStemmer

from nltk.tokenize import word\_tokenize

nltk.download('punkt')

stemmer = PorterStemmer()

def stem\_sentence(sentence):

words = word\_tokenize(sentence)

stemmed\_words = [stemmer.stem(word) for word in words]

stemmed\_sentence = ' '.join(stemmed\_words)

return stemmed\_sentence

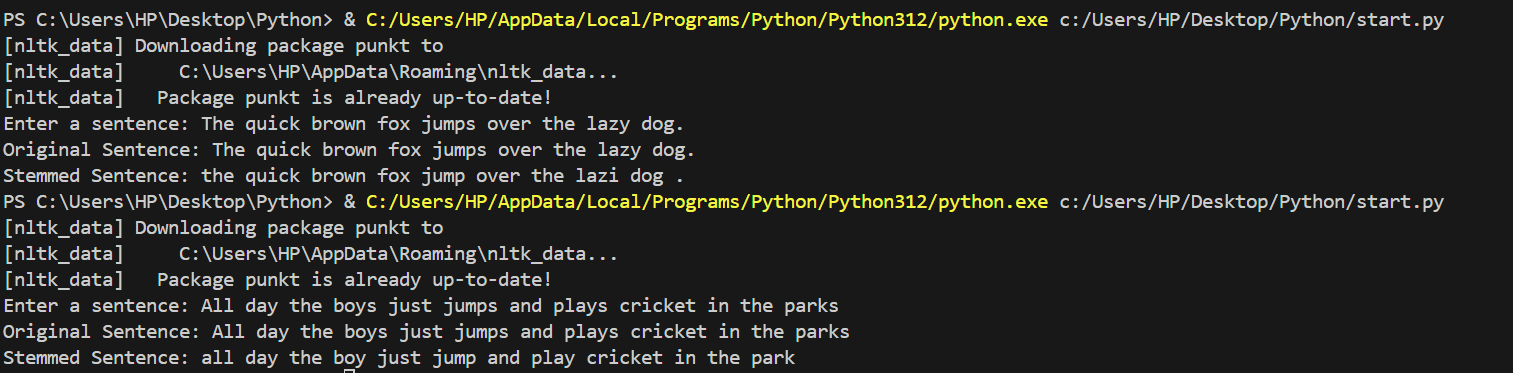
sentence = input("Enter a sentence: ")

stemmed\_sentence = stem\_sentence(sentence)

print("Original Sentence:", sentence)

print("Stemmed Sentence:", stemmed\_sentence)

**Output:**



# Experiment - 13

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

## **Program:**

import nltk

from nltk.tokenize import word\_tokenize from nltk import pos\_tag

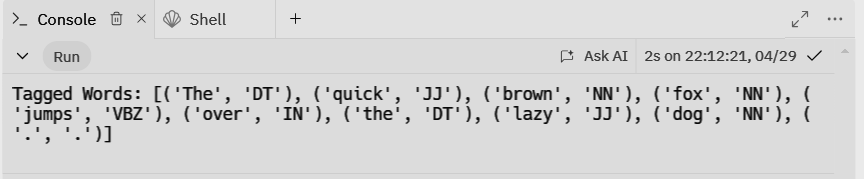
def pos\_tagging(sentence):

words = word\_tokenize(sentence) tagged\_words = pos\_tag(words) return tagged\_words

sentence = "The quick brown fox jumps over the lazy dog." tagged\_words = pos\_tagging(sentence)

print("Tagged Words:", tagged\_words)

**OUTPUT:**



# Experiment - 14

**Aim:** Write a program to implement Lemmatization using NLTK

## **Theory:**

Lemmatization is the process of reducing words to their base or root form, known as the lemma. Unlike stemming, which simply chops off the ends of words to create their base form, lemmatization takes into account the meaning of the word and its context in the sentence. This means that lemmatization produces valid words that are present in the dictionary.

Ex - "running" -> "run"

## **Program:**

import nltk

from nltk.stem import WordNetLemmatizer

from nltk.tokenize import word\_tokenize

nltk.download('wordnet')

lemmatizer = WordNetLemmatizer()

def lemmatize\_sentence(sentence):

words = word\_tokenize(sentence)

lemmatized\_words = [lemmatizer.lemmatize(word) for word in words]

lemmatized\_sentence = ' '.join(lemmatized\_words)

return lemmatized\_sentence

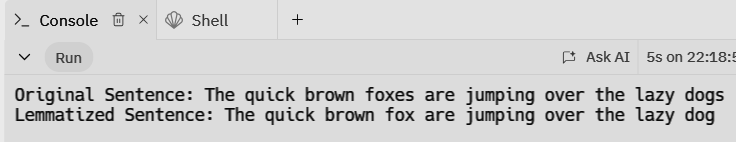
sentence = "The quick brown foxes are running over the lazy dogs"

lemmatized\_sentence = lemmatize\_sentence(sentence)

print("Original Sentence:", sentence)

print("Lemmatized Sentence:", lemmatized\_sentence)

**OUTPUT:**



**Experiment – 15**

**Aim:**  Write a program for Text Classification for the given sentence using NLTK.

**Source Code:**

import nltk

from nltk.tokenize import word\_tokenize

from nltk.corpus import stopwords

from nltk.stem import WordNetLemmatizer

from nltk.classify import NaiveBayesClassifier

import random

# Download necessary NLTK data

nltk.download('punkt')

nltk.download('stopwords')

nltk.download('wordnet')

def preprocess(sentence):

# Tokenize the sentence

tokens = word\_tokenize(sentence.lower())

# Remove stopwords

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

filtered\_tokens = [word for word in tokens if word not in stop\_words]

# Lemmatize tokens

lemmatizer = WordNetLemmatizer()

lemmatized\_tokens = [lemmatizer.lemmatize(word) for word in filtered\_tokens]

return lemmatized\_tokens

def extract\_features(words):

return dict([(word, True) for word in words])

def train\_classifier():

# Sample training data

training\_data = [

(preprocess("Natural language processing is a field of artificial intelligence."), "technology"),

(preprocess("Forests are home to diverse ecosystems."), "nature"),

(preprocess("Computers can understand and generate human language."), "technology"),

(preprocess("Mountains offer breathtaking views and fresh air."), "nature"),

(preprocess("Machine learning algorithms improve with more data."), "technology"),

(preprocess("Rivers provide water for plants and animals."), "nature")

]

# Extract features from training data

training\_features = [(extract\_features(tokens), category) for tokens, category in training\_data]

# Train Naive Bayes classifier

classifier = NaiveBayesClassifier.train(training\_features)

return classifier

def classify\_sentence(classifier, sentence):

tokens = preprocess(sentence)

features = extract\_features(tokens)

category = classifier.classify(features)

return category

def main():

classifier = train\_classifier()

test\_sentences = [

"The internet has revolutionized communication.",

"Birds migrate to warmer climates during winter.",

"Artificial intelligence is shaping the future of technology.",

"Forests play a crucial role in maintaining ecological balance.",

"Deep learning models require large datasets for training.",

"Oceans cover more than 70% of the Earth's surface.",

"Blockchain technology is transforming various industries.",

"Wildlife conservation is essential for biodiversity."

]

# Classify test sentences

for sentence in test\_sentences:

category = classify\_sentence(classifier, sentence)

print(f"Sentence: {sentence}")

print(f"Category: {category}")

print("-" \* 50)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Output:**

## 