**LAB FILE**

OF

# ARTIFICIAL INTELLIGENCE

**BACHELOR OF TECHNOLOGY**

**in**

***Computer Science & Engineering (AI)***

|  |  |
|---|---|
| **SUBMITTED BY:** | **SUBMITTED TO:** |
| PRABHAKAR YADAV | Dr. RUPALI KHARE |
| (2000971520034) | (Assistant Professor) |

**DEPARTMENT**
**OF**

**COMPUTER SCIENCE AND ENGINEERING**

**GALGOTIAS COLLEGE OF ENGINEERING**

**ANDTECHNOLOGY, GREATER NOIDA**

Session 2022-23

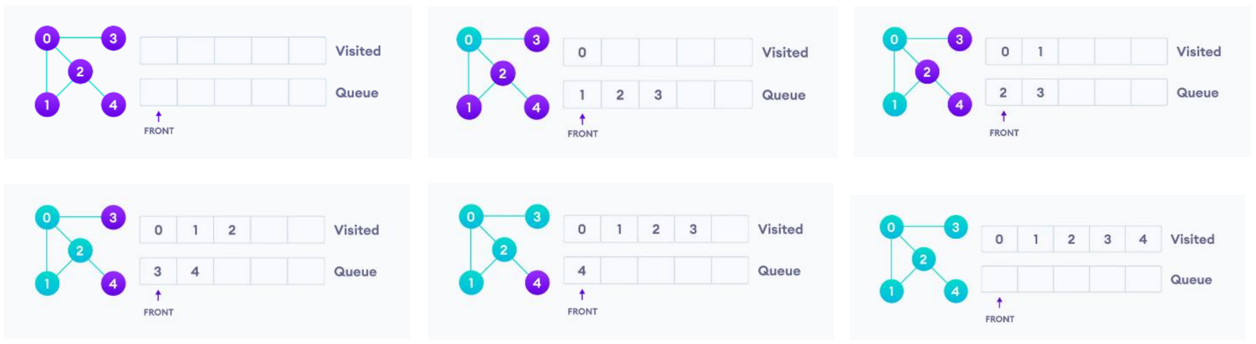
# Index

# PROGRAM-1

**Aim:** Write a python program to implement Breadth First Search (BFS) Traversal

**Logic:** Breadth-first search (BFS) is an algorithm for searching a tree data structure for a node that satisfies a given property. It starts at the tree root and explores all nodes at the present depth prior to moving on to the nodes at the next depth level. Extra memory, usually a queue, is needed to keep track of the child nodes that were encountered but not yet explored.

**Example:**



## Algorithm:

Step 1: Define a Queue of size total number of vertices in the graph.
Step 2: Select any vertex as starting point for traversal. Visit that vertex and insert it into the Queue.
Step 3: Visit all the non-visited adjacent vertices of the vertex which is at front of the Queue and insert them into the Queue.
Step 4: When there is no new vertex to be visited from the vertex which is at front of the Queue then delete that vertex.
Step 5: Repeat steps 3 and 4 until queue becomes empty.
Step 6: When queue becomes empty, then produce final spanning tree by removing unused edges from the graph

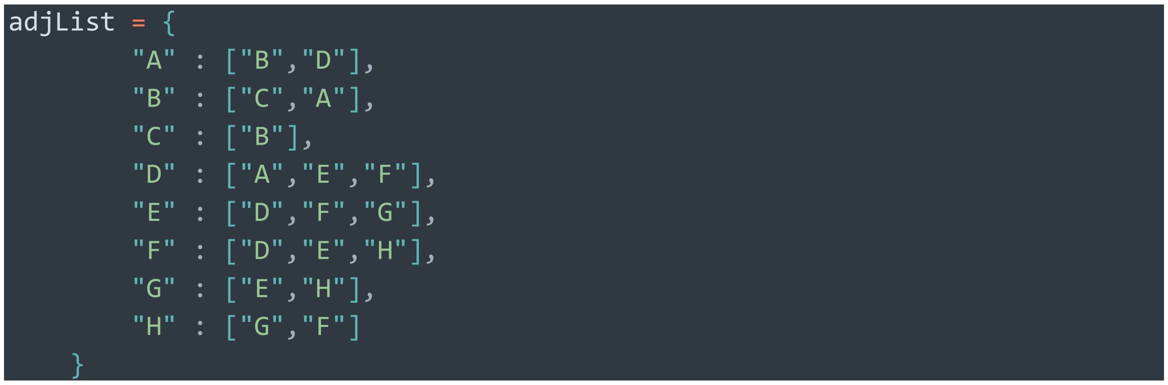
## Implementation:

```python
from queue import Queue

def initializeValues(visited,level,parent,queue):
    for node in adjList.keys():
        visited[node] = False
        parent[node] = None
        level[node] = -1
    s="A"
    visited[s] = True
    level[s] = 0
    queue.put(s)
```

```python
def bfsTraversal(adjList,parent,level,queue,bfsTraversalOutput):
    while not queue.empty():
        u = queue.get()
        bfsTraversalOutput.append(u)
        for v in adjList[u]:
            if not visited[v]:
                visited[v] = True
                parent[v] = u
                level[v] = level[u] + 1
                queue.put(v)

def printValues(bfsTraversalOutput):
    print("Output : ",end="")
    for node in bfsTraversalOutput:
        print(node,end=" ")
    print()

if __name__ == '__main__':
    adjList = {
        "A" : ["B","D"],
        "B" : ["C","A"],
        "C" : ["B"],
        "D" : ["A","E","F"],
        "E" : ["D","F","G"],
        "F" : ["D","E","H"],
        "G" : ["E","H"],
        "H" : ["G","F"]
    }
    visited = {}
    level = {}
    parent = {}
    bfsTraversalOutput=[]

    queue = Queue()
    initializeValues(visited,level,parent,queue)
    bfsTraversal(adjList,parent,level,queue,bfsTraversalOutput)
    printValues(bfsTraversalOutput)
```

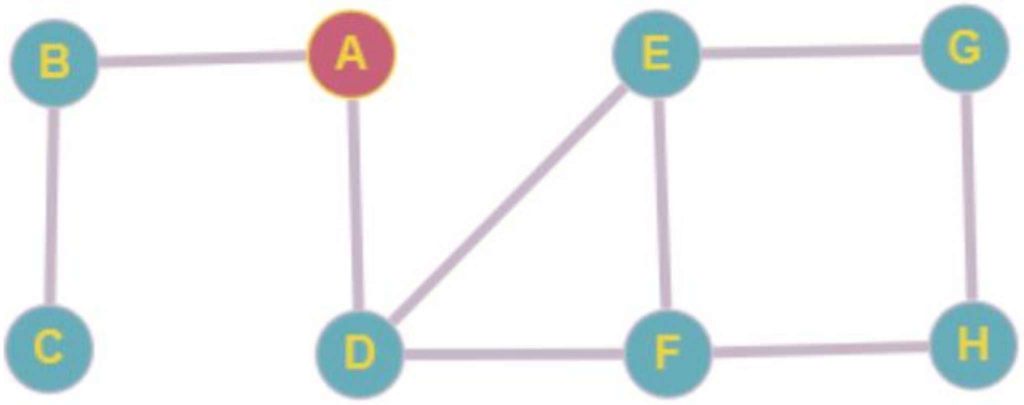
**Input:**

Code:

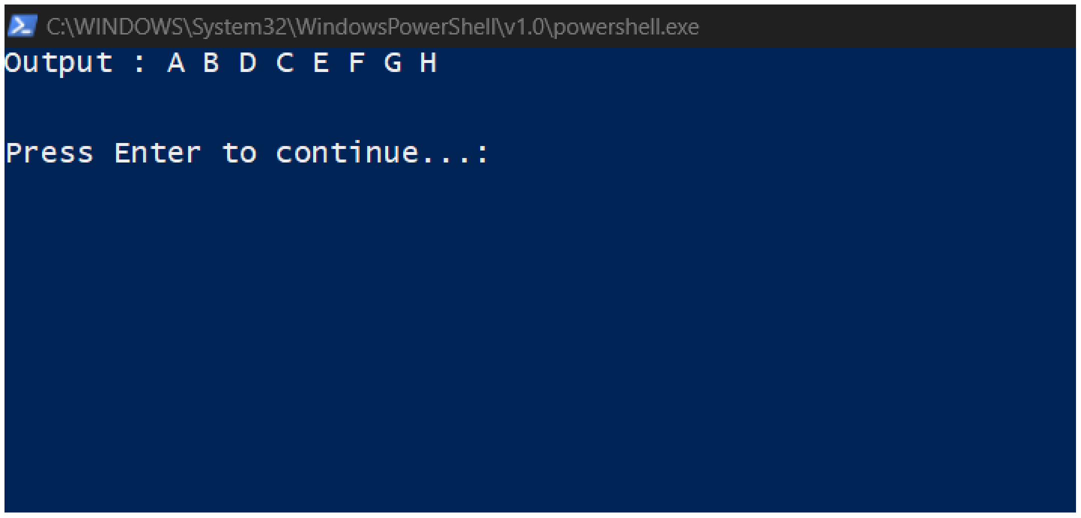


```
adjList = {
        "A" : ["B","D"],
        "B" : ["C","A"],
        "C" : ["B"],
        "D" : ["A","E","F"],
        "E" : ["D","F","G"],
        "F" : ["D","E","H"],
        "G" : ["E","H"],
        "H" : ["G","F"]
    }
```

Graphically:



**Output:**



```
Output : A B D C E F G H


Press Enter to continue...:
```

# PROGRAM-2

**Aim:** Write a program in python to implement Water Jug problem.

**Logic:** Problem contains m liter jug and a n liter jug where 0 < m < n. Both the jugs are initially empty. The jugs don't have markings to allow measuring smaller quantities. You have to use the jugs to measure d liters of water where d < n. Determine the minimum no of operations to be performed to obtain d liters of water in one of jug.
The operations that can be perform are:

1. Empty a Jug
2. Fill a Jug
3. Pour water from one jug to the other until one of the jugs is either empty or full.

**Algorithm:**

```
1. (x,y) Fill 4 gallon jug x = m
2. (x,y) Fill 3 gallon jug y=n
3. (x,y) Empty 4 gallon jug on ground x = 0
4. (x,y) Empty 3 gallon jug on ground y = 0
5. (x,y) Pour water from 4 gallon jug into 3 gallon jug until the 3 gallon jug is full
   t = n-y
   y = n
   x -= t
6. (x,y) Pour water from 3 gallon jug into 4 gallon jug until the 4 gallon jug is full
   t =m-x
   x = m
   y -= t
7. (x,y) Pour all water from 4 gallon jug into 3 gallon jug until 4 gallon becomes empty
8. (x,y) Pour all water from 3 gallon jug into 4 gallon jug until 3 gallon becomes empty
```

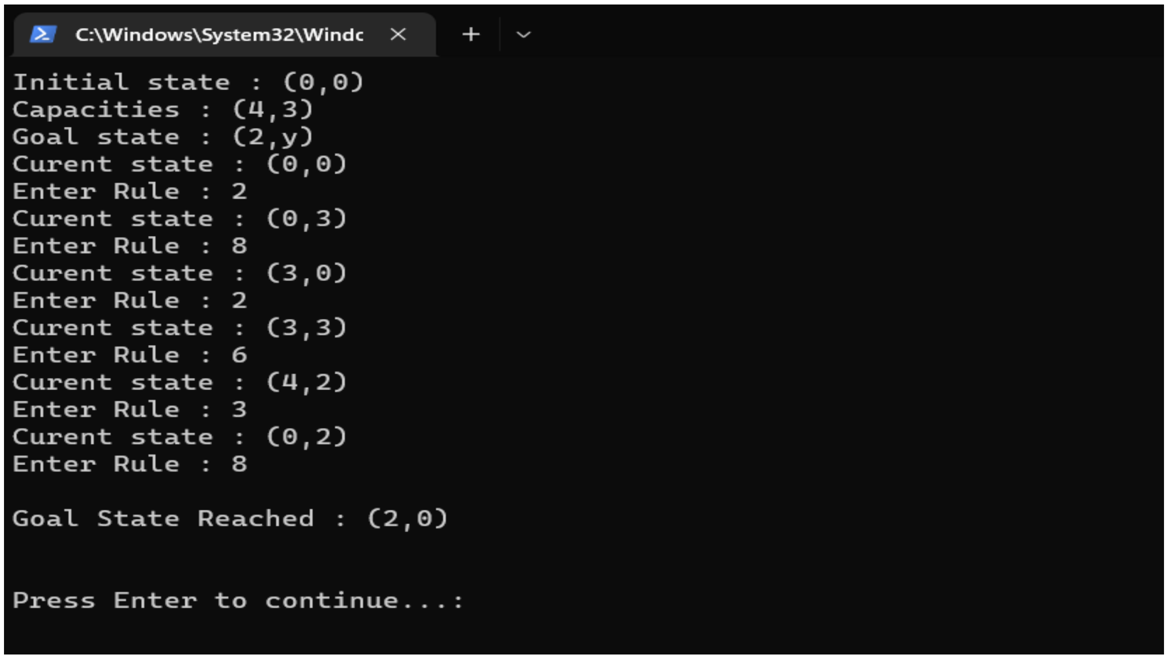
**Implementation:**

```python
x=0
y=0
m=4
n=3

print(f"Initial state : ({x},{y})")
print(f"Capacities : ({m},{n})")
print(f"Goal state : (2,y)")

while x != 2:
    print(f"Curent state : ({x},{y})")

    r = int(input("Enter Rule : "))
    if r == 1:
        x = m
    elif r == 2:
        y = n
    elif r == 3:
        x = 0
    elif r == 4:
        y = 0
    elif r == 5:
        t = n-y
        y = n
        x -= t
    elif r == 6:
        t = m-x
        x = m
        y -= t
    elif r == 7:
        y += x
        x = 0
    elif r == 8:
        x += y
        y = 0

print(f"\nGoal State Reached : ({x},{y})")
```

**Input:**

2 8 2 6 3 8

**Output:**



```
C:\Windows\System32\Windc
Initial state : (0,0)
Capacities : (4,3)
Goal state : (2,y)
Curent state : (0,0)
Enter Rule : 2
Curent state : (0,3)
Enter Rule : 8
Curent state : (3,0)
Enter Rule : 2
Curent state : (3,3)
Enter Rule : 6
Curent state : (4,2)
Enter Rule : 3
Curent state : (0,2)
Enter Rule : 8

Goal State Reached : (2,0)


Press Enter to continue...:
```

# PROGRAM-3

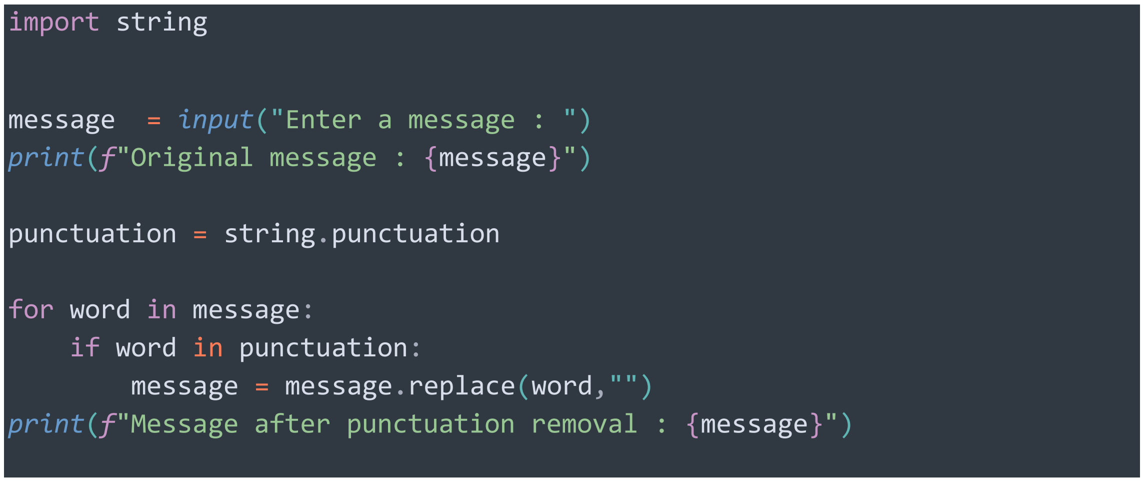
**Aim:** Write a program in python to remove punctuations from given string.

**Logic:** First check the input string if it contains punctuations, then we have to make it punctuation free. We will traverse over the string, and if punctuations are found we will remove them.

**Algorithm:**

1. Initialize the input string
2. Check if the character present in the string is punctuation or not
3. If a character is punctuation, then erase that character and decrement the index
4. Print the output string, which is punctuation free
5. End

**Implementation:**

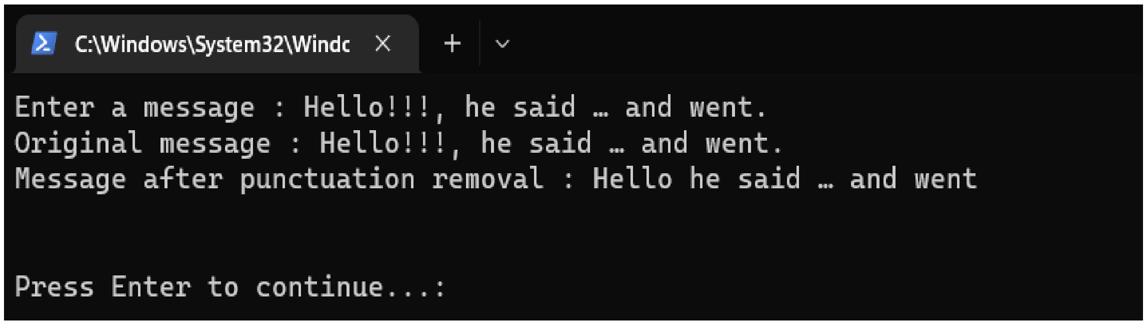


```python
import string


message  = input("Enter a message : ")
print(f"Original message : {message}")

punctuation = string.punctuation

for word in message:
    if word in punctuation:
        message = message.replace(word,"")
print(f"Message after punctuation removal : {message}")
```

**Input:**

Hello!!!, he said … and went.

**Output:**



```
Enter a message : Hello!!!, he said … and went.
Original message : Hello!!!, he said … and went.
Message after punctuation removal : Hello he said … and went


Press Enter to continue...:
```

# PROGRAM-4

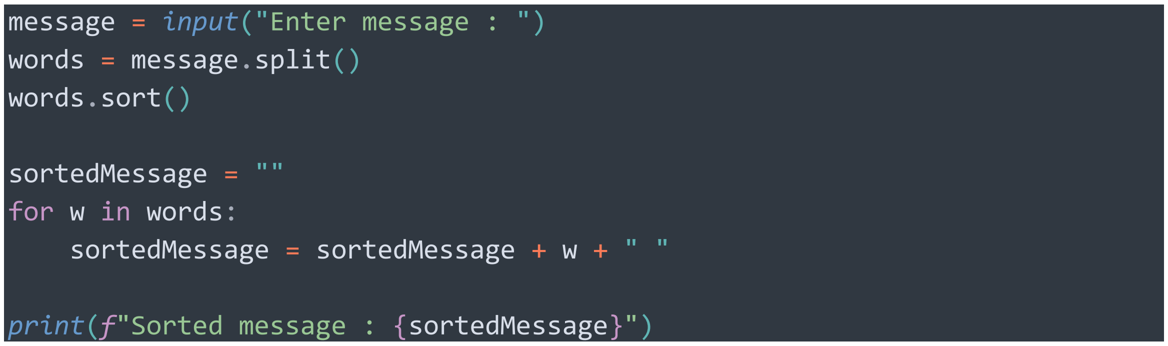
**Aim:** Write a program in python to sort words in a string in alphabetical order.

**Logic:** Strings in python are immutable, but python allows certain operation on string such as split() function to split the string in certain parts, then use sort function to alphabetically sort the string.

**Algorithm:**

1. Initialize input string
2. Split string on occurrence of spaces
3. Sort the sub-strings
4. Store sorted sub-string in a single string
5. Display the resultant string
6. Exit

**Implementation:**

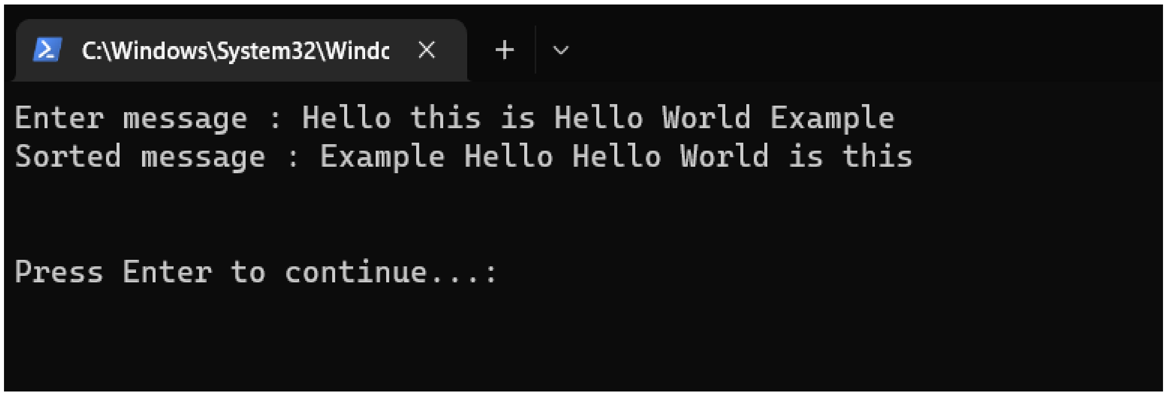


```python
message = input("Enter message : ")
words = message.split()
words.sort()

sortedMessage = ""
for w in words:
    sortedMessage = sortedMessage + w + " "

print(f"Sorted message : {sortedMessage}")
```

**Input:**

Hello this is Hello World Example

**Output:**



```
Enter message : Hello this is Hello World Example
Sorted message : Example Hello Hello World is this


Press Enter to continue...:
```

# PROGRAM-5

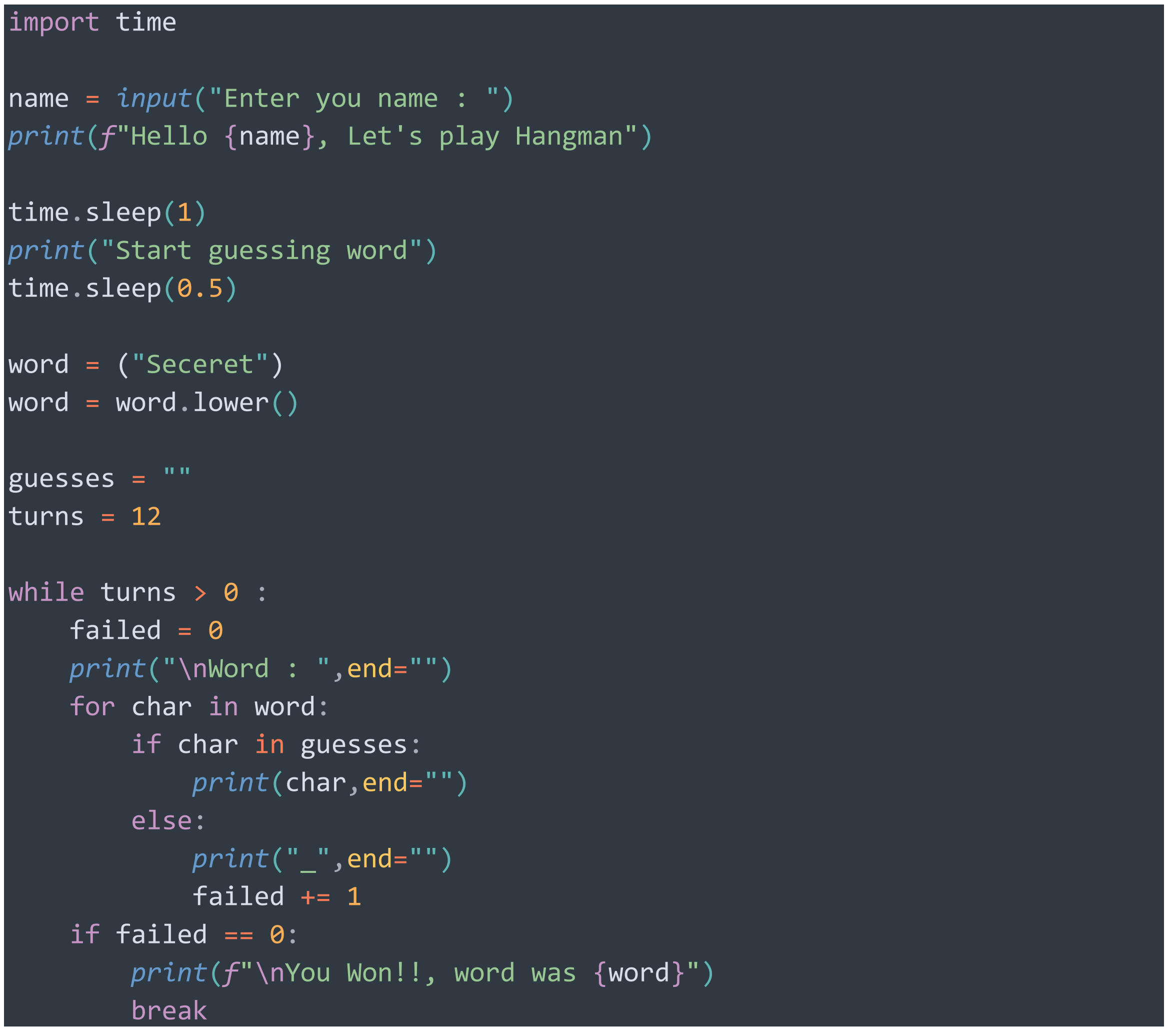
**Aim:** Write a program in python to implement Hangman game.

**Logic:** Hangman is a classic word guessing game. User should guess words correctly by entering alphabets of user choice. Let input as an alphabet from user and make match with original word, if it matches then user guesses remaining alphabets. If user finds all characters he will win otherwise lose.

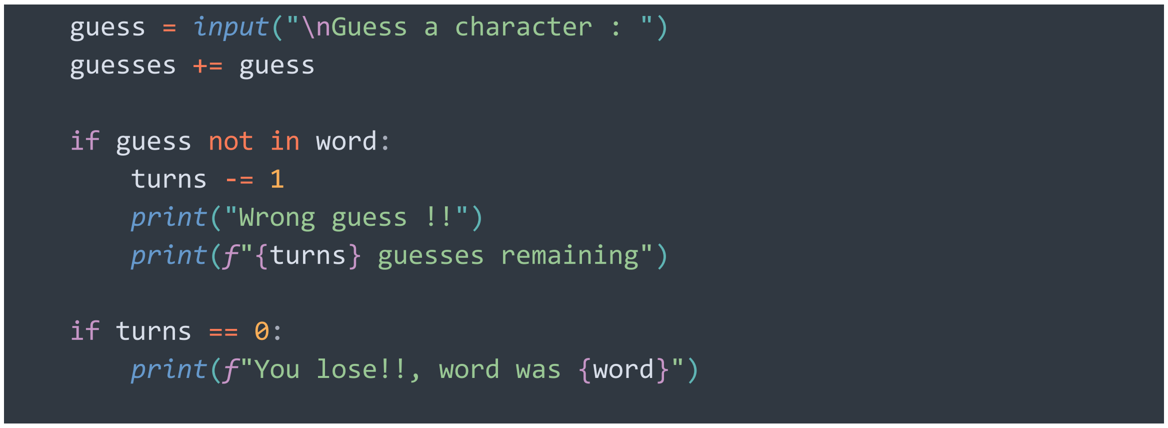
**Algorithm:**

1. Import the random module
2. Define function to welcome user
3. Let player name using a string method to capture users name in sentence case
4. Create a decision-making process to check that usure only enter alphabet and not number
5. Generate random word for the user to guess
6. Initialize the number of attempts
7. Keep checking if current word is present of not until attempts != 0

**Implementation:**



```python
import time

name = input("Enter you name : ")
print(f"Hello {name}, Let's play Hangman")

time.sleep(1)
print("Start guessing word")
time.sleep(0.5)

word = ("Seceret")
word = word.lower()

guesses = ""
turns = 12

while turns > 0 :
    failed = 0
    print("\nWord : ",end="")
    for char in word:
        if char in guesses:
            print(char,end="")
        else:
            print("_",end="")
            failed += 1
    if failed == 0:
        print(f"\nYou Won!!, word was {word}")
        break
```



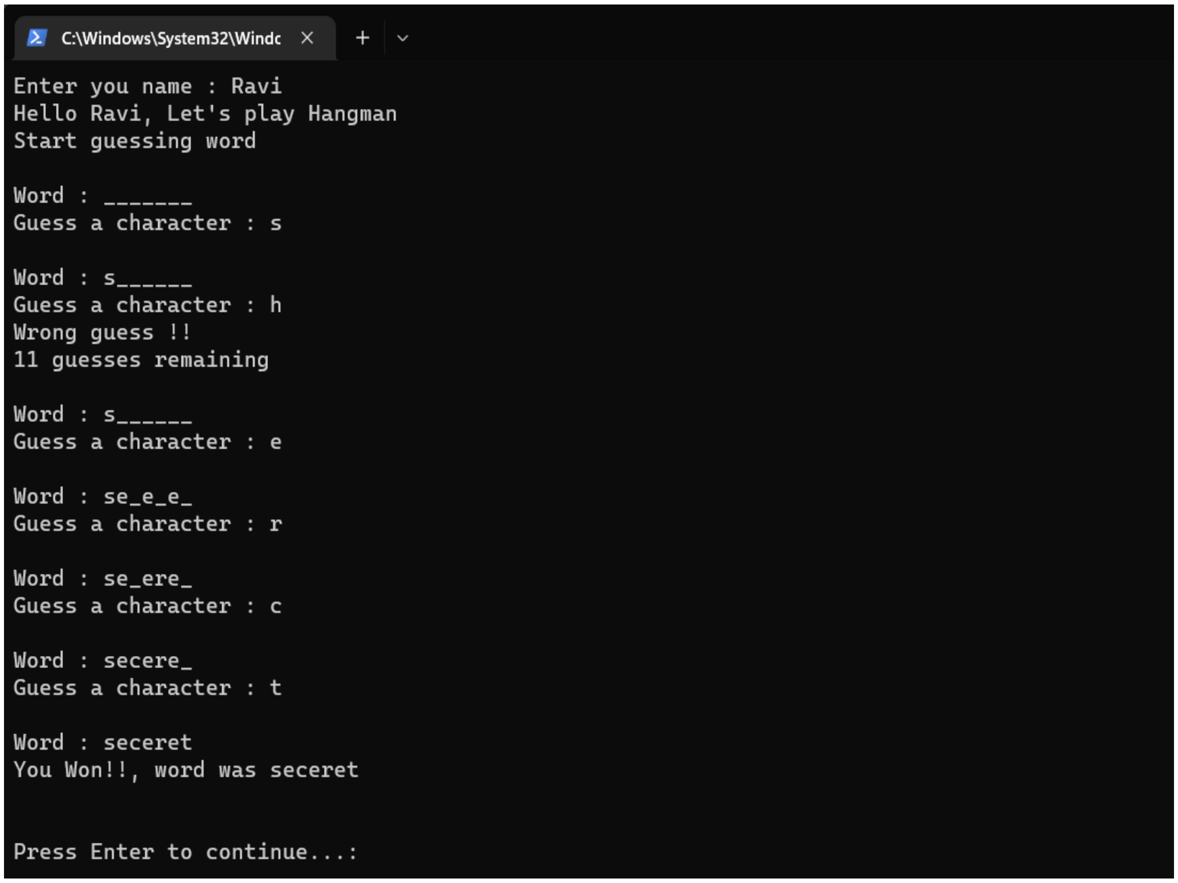
```python
    guess = input("\nGuess a character : ")
    guesses += guess

    if guess not in word:
        turns -= 1
        print("Wrong guess !!")
        print(f"{turns} guesses remaining")

    if turns == 0:
        print(f"You lose!!, word was {word}")
```

**Input:**

Ravi

s h e r c t

**Output:**



```
Enter you name : Ravi
Hello Ravi, Let's play Hangman
Start guessing word

Word : _______
Guess a character : s

Word : s______
Guess a character : h
Wrong guess !!
11 guesses remaining

Word : s______
Guess a character : e

Word : se_e_e_
Guess a character : r

Word : se_ere_
Guess a character : c

Word : secere_
Guess a character : t

Word : seceret
You Won!!, word was seceret


Press Enter to continue...:
```

# PROGRAM-6

**Aim:** Write a program in python to implement tic-tac-toe.

**Logic:**

1. It is a two-player game
2. There are two characters ‘X’ and ‘O’
3. The game board consists of a 3x3 grid
4. Players who succeed in placing 3 same chars in horizontal, vertical, diagonal row will win the game

**Algorithm:**

1. Create a design of tic-tac-toe
2. Store information using data structure
3. Handle player input
4. Update the cell occupied according to current player
5. Check win or draw
6. Switch current player
7. Enter player name
8. Store information
9. Design a scorecard
10. Handle and assign player choice
11. Update scoreboard

**Implementation:**

```python
import os
import time

board = ['#',' ',' ',' ',' ',' ',' ',' ',' ',' ']

def displayBoard(board):
    print(f"       |       |       ")
    print(f"   {board[7]}   |   {board[8]}   |   {board[9]}   ")
    print(f"_______|_______|_______")
    print(f"       |       |       ")
    print(f"   {board[4]}   |   {board[5]}   |   {board[6]}   ")
    print(f"_______|_______|_______")
    print(f"       |       |       ")
    print(f"   {board[1]}   |   {board[2]}   |   {board[3]}   ")
    print(f"       |       |       ")

def isWinner(board,mark):
    if(board[1] == mark and board[2] == mark and board[3] == mark) or \
      (board[4] == mark and board[5] == mark and board[6] == mark) or \
      (board[7] == mark and board[8] == mark and board[9] == mark) or \
      (board[1] == mark and board[4] == mark and board[7] == mark) or \
      (board[2] == mark and board[5] == mark and board[8] == mark) or \
      (board[3] == mark and board[6] == mark and board[9] == mark) or \
      (board[1] == mark and board[5] == mark and board[9] == mark) or \
      (board[3] == mark and board[5] == mark and board[7] == mark):
      return True
    return False

def isBoardFull(board):
    if ' ' in board:
        return False
    return True

def validInput(board,player,mark):
    while True:
        choice = int(input(f"{player} enter choice : "))
        if board[choice] == ' ':
            board[choice] = mark
            return
        print("Wrong Choice!!")

def playerInput(board,player,mark,gameOver):
    validInput(board,player,mark)

    if isWinner(board,mark):
        os.system("cls")
        displayBoard(board)
        print(f"{player} won the game!!")
        gameOver[0] = True


playerX = input("Player X Enter your name : ")
playerO = input("Player O Enter your name : ")
playerXChance = True
gameOver = [False]

while not gameOver[0]:
    os.system("cls")
    displayBoard(board)
    if playerXChance:
        playerInput(board,playerX,'X',gameOver)
        playerXChance = not playerXChance
    else:
        playerInput(board,playerO,'O',gameOver)
        playerXChance = not playerXChance

    if isBoardFull(board) and not gameOver[0]:
        os.system("cls")
        displayBoard(board)
        print("Draw!!")
        gameOver[0] = True
```

**Input:**

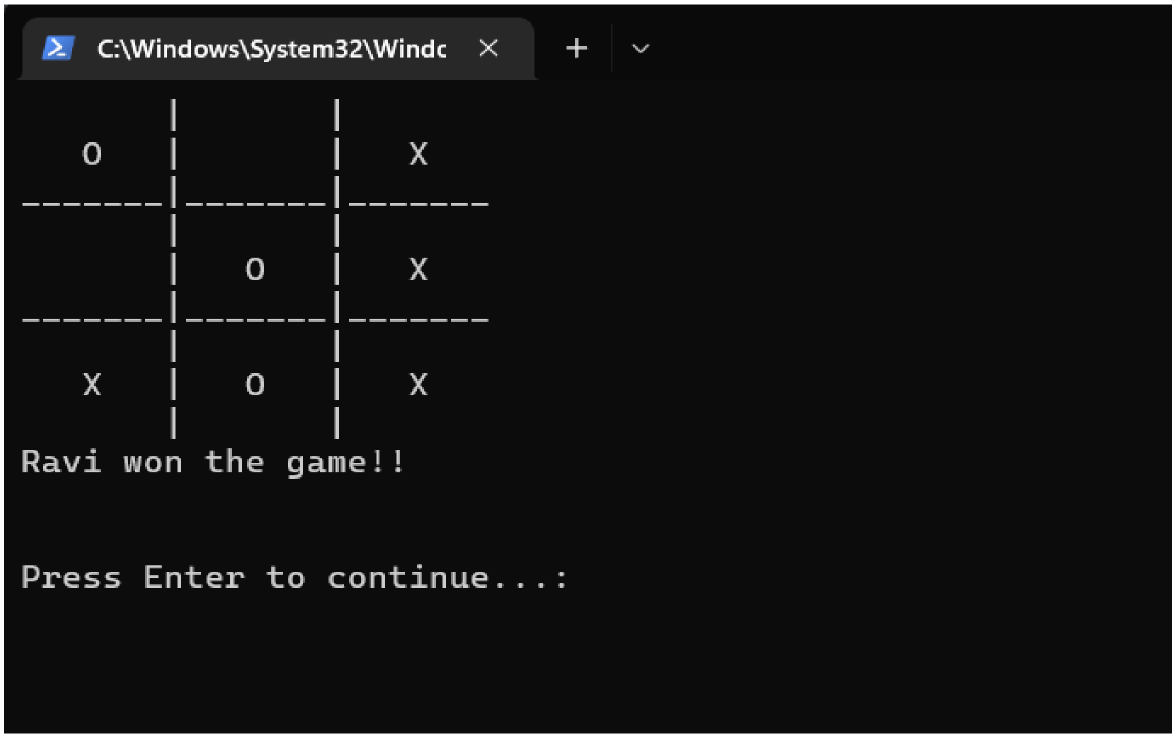
Ravi

Rahul

Ravi:  1 9 3 6

Rahul: 5 7 2

**Output:**



```
        |       |
   O    |       |   X
________|_______|_______
        |       |
        |   O   |   X
________|_______|_______
        |       |
   X    |   O   |   X
        |       |
Ravi won the game!!


Press Enter to continue...:
```

# PROGRAM-7

**Aim:** Write a program in python to remove stop-words for a given passage from text-file using NLTK

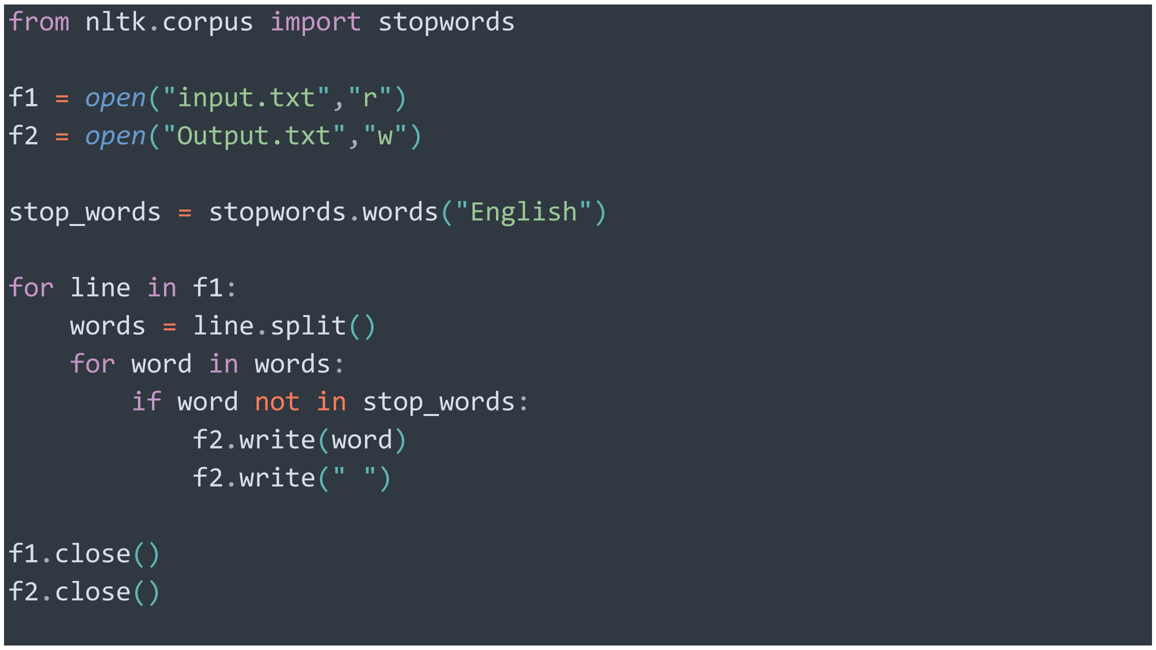
**Logic:** A stop-word is a commonly used word such as the, a, am, in that a search engine has been programmed to ignore, both when indexing entries for searches and when retrieving them as the result of search query. We do not want these words to take space in databox so we remove them.

NLTK in python has a list of stop-words stored in 16 different languages

**Algorithm:**

1. Import NLTK library
2. Import stop-words from NLTK
3. Open source and destination files
4. Check all words in source file. If word is a stop word remove it, else write it into destination file

**Implementation:**

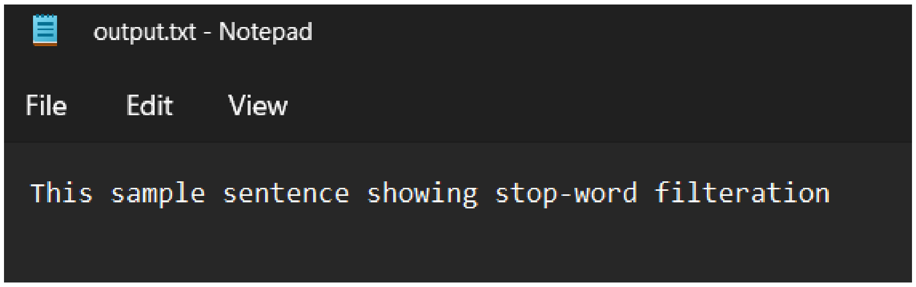


```python
from nltk.corpus import stopwords

f1 = open("input.txt","r")
f2 = open("Output.txt","w")

stop_words = stopwords.words("English")

for line in f1:
    words = line.split()
    for word in words:
        if word not in stop_words:
            f2.write(word)
            f2.write(" ")

f1.close()
f2.close()
```

**Input:**

Input.txt: This is a sample sentence showing off the stop-word filteration

**Output:**



```
This sample sentence showing stop-word filteration
```

# PROGRAM-8

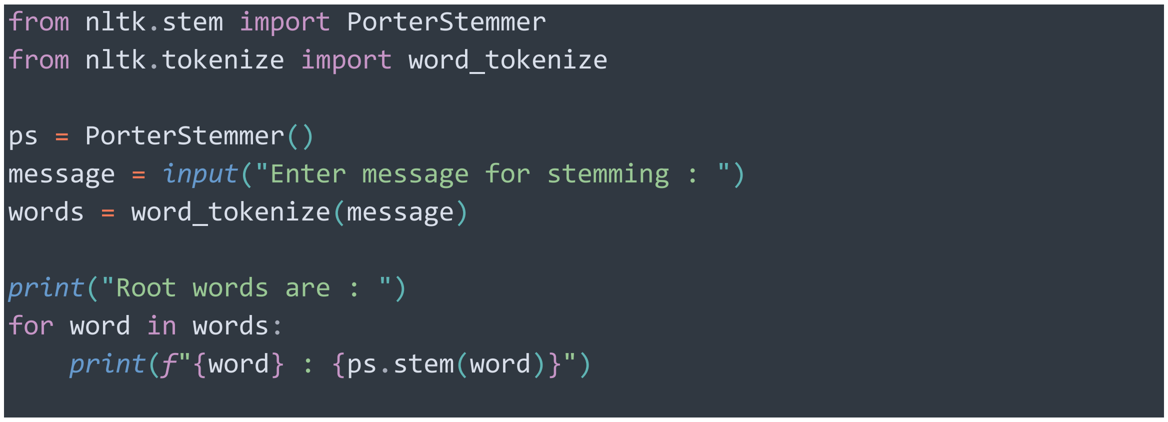
**Aim:** Write a program in python to implement stemming for a given sentence using NLTK

**Logic:** Stemming is the process of producing morphological variants of a root/base word. Stemming programs are commonly referred to as stemming algorithms and stemmers. It reduces the given words into the root word.

**Algorithm:**

1. Import PorterStemmer from nltk.stem
2. Import word-tokenize from nltk.tokenize
3. Input a sentence
4. Tokenize it
5. Print the root word of each word in the sentence

**Implementation:**

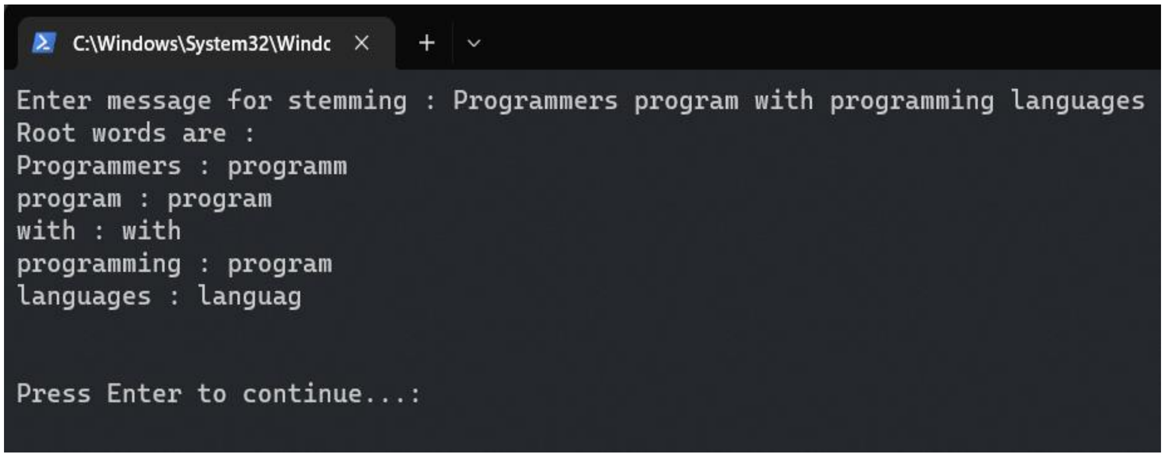


```python
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize

ps = PorterStemmer()
message = input("Enter message for stemming : ")
words = word_tokenize(message)

print("Root words are : ")
for word in words:
    print(f"{word} : {ps.stem(word)}")
```

**Input:**

Programmers program with programming languages

**Output:**



```
Enter message for stemming : Programmers program with programming languages
Root words are :
Programmers : programm
program : program
with : with
programming : program
languages : languag


Press Enter to continue...:
```

# PROGRAM-9

**Aim:** Write a program in python to POS (Parts of Speech) tagging for given sentence using NLTK

**Logic:** POS tagging is a process to mark up the words in text format for a particular part of a speech based on its definition and context. It is responsible for text reading in a language and assigning some specific token to each word. Also called grammatical tagging.

**Algorithm:**

1. Import NLTK module into code
2. Import stop-words module using nltk.Corpus
3. Import word_tokenize, sent_tokenize from nltk.tokenize
4. Input sentence
5. Use nltk.pos_tag to tag word in the sentence.
6. Print the tagged words

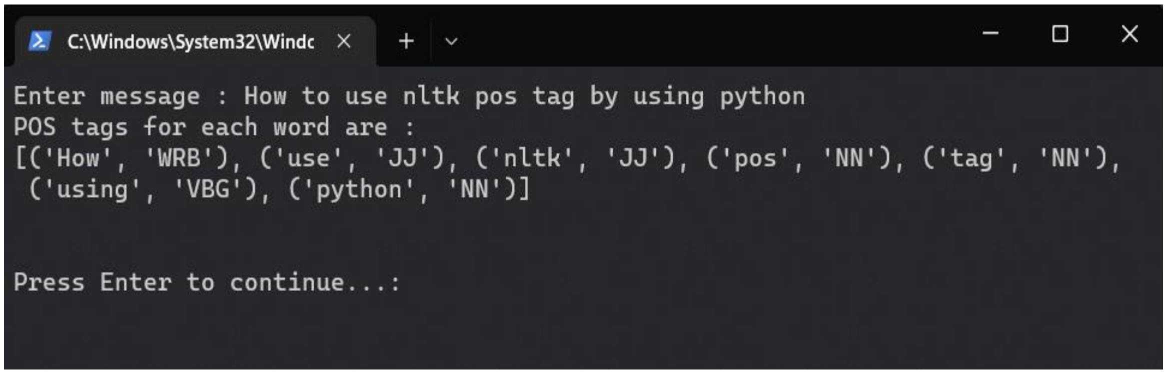
**Implementation:**

```python
from nltk.corpus import stopwords

from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.tag import pos_tag

stop_words = set(stopwords.words("English"))
message = input("Enter message : ")
tokens = sent_tokenize(message)

print("POS tags for each word are : ")
for token in tokens:
    word = word_tokenize(token)
    word = [w for w in word if not w in stop_words]
    tag = pos_tag(word)
    print(tag)
```

**Input:**

How to use nltk pos tag by using python

**Output:**



```
Enter message : How to use nltk pos tag by using python
POS tags for each word are :
[('How', 'WRB'), ('use', 'JJ'), ('nltk', 'JJ'), ('pos', 'NN'), ('tag', 'NN'),
 ('using', 'VBG'), ('python', 'NN')]


Press Enter to continue...:
```

# PROGRAM-10

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

**Logic:** Lemmatization is the process of grouping together the different reflected forms of a word so that can be analyzed as a single item. It is similar to stemming but it brings context to the word. So, it links word with similar meaning to one word.

**Algorithm:**

1. Import NLTK
2. From nltk.stem import WordnetLemmatizer
3. Import sentence from user
4. Remove punctuations from sentence
5. Print each lemmatized word with corresponding lemma

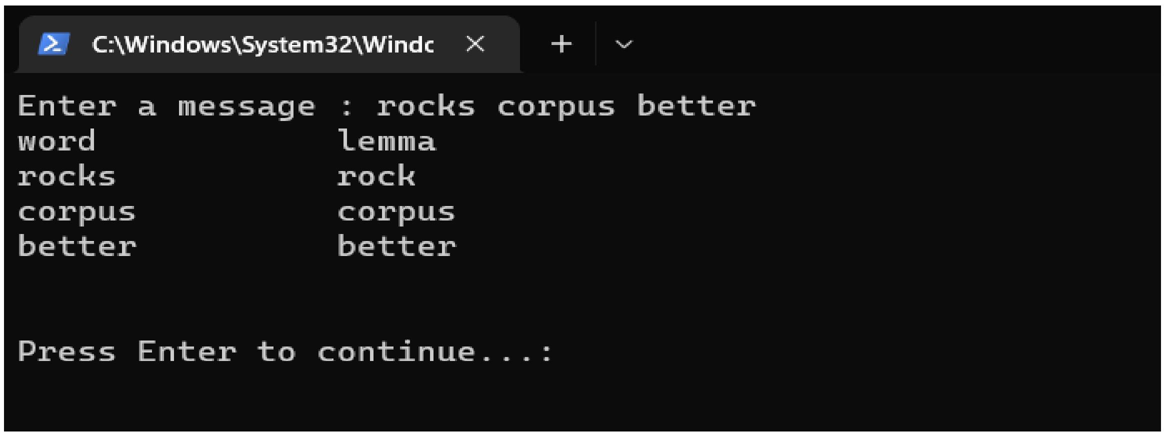
**Implementations:**

```python
from nltk.stem import WordNetLemmatizer

import string

lemmatizer = WordNetLemmatizer()

message = input("Enter a message : ")
punctuations = string.punctuation

for word in message:
    if word in punctuations:
        message.remove(word)

message = message.split()
print("word\t\tlemma")
for word in message:
    print(f"{word}\t\t{lemmatizer.lemmatize(word)}")
```

**Input:**

rocks corpus better

**Output:**



```
Enter a message : rocks corpus better
word                lemma
rocks               rock
corpus              corpus
better              better


Press Enter to continue...:
```

# PROGRAM-11

**Aim:** Write a program in python for text classification for the given sentence using NLTK

**Logic:** Text classification also known as text categorization is the process of sorting text into categories. For example, classification of customer feedback by topic, urgency, sentiment etc. NLTK is python library which can be used to perform text classification.

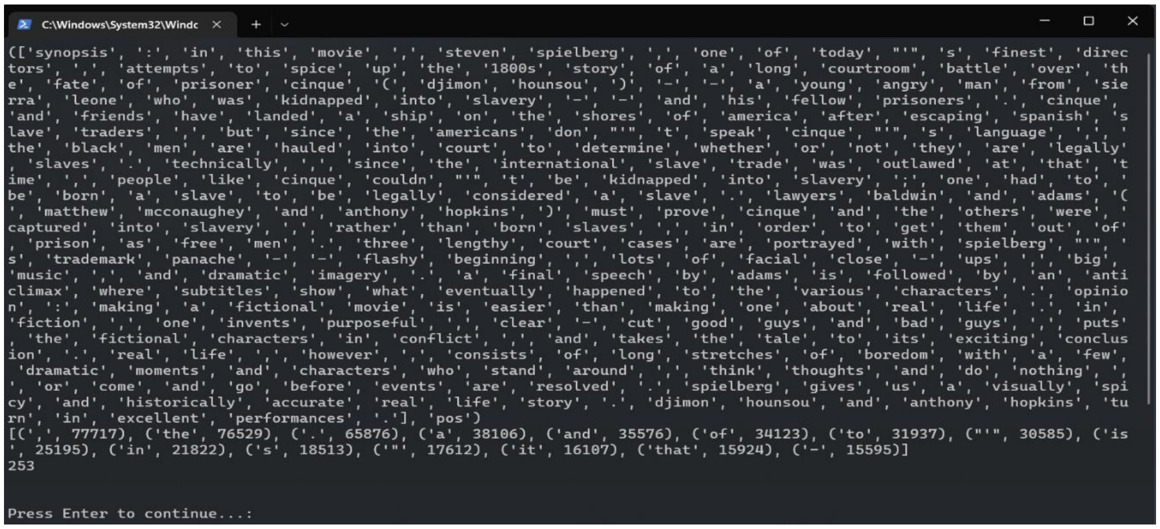
**Algorithm:**

1. Import NLTK
2. Import random library
3. Store all sentences
4. Store movie reviews in variable
5. Use nltk.FreqDist and most_common to print result

**Implementation:**

```python
import nltk
import random
from nltk.corpus import movie_reviews

documents = [(list(movie_reviews.words(fileid)), category)
             for category in movie_reviews.categories()
             for fileid in movie_reviews.fileids(category)]

random.shuffle(documents)
print(documents[1])

all_words = []
for w in movie_reviews.words():
    all_words.append(w.lower())

all_words = nltk.FreqDist(all_words)
print(all_words.most_common(15))
print(all_words["stupid"])
```

**Output:**



```
(['synopsis', ':', 'in', 'this', 'movie', ',', 'steven', 'spielberg', ',', 'one', 'of', 'today', "'", 's', 'finest', 'direc
tors', ',', 'attempts', 'to', 'spice', 'up', 'the', '1800s', 'story', 'of', 'a', 'long', 'courtroom', 'battle', 'over', 'th
e', 'fate', 'of', 'prisoner', 'cinque', '(', 'djimon', 'hounsou', ')', '-', '-', 'a', 'young', 'angry', 'man', 'from', 'sie
rra', 'leone', 'who', 'was', 'kidnapped', 'into', 'slavery', '-', '-', 'and', 'his', 'fellow', 'prisoners', '.', 'cinque',
'and', 'friends', 'have', 'landed', 'a', 'ship', 'on', 'the', 'shores', 'of', 'america', 'after', 'escaping', 'spanish', 's
lave', 'traders', ',', 'but', 'since', 'the', 'americans', 'don', "'", 't', 'speak', 'cinque', "'", 's', 'language', ',', '
the', 'black', 'men', 'are', 'hauled', 'into', 'court', 'to', 'determine', 'whether', 'or', 'not', 'they', 'are', 'legally'
, 'slaves', '.', 'technically', ',', 'since', 'the', 'international', 'slave', 'trade', 'was', 'outlawed', 'at', 'that', 't
ime', ',', 'people', 'like', 'cinque', 'couldn', "'", 't', 'be', 'kidnapped', 'into', 'slavery', ';', 'one', 'had', 'to', '
be', 'born', 'a', 'slave', 'to', 'be', 'legally', 'considered', 'a', 'slave', '.', 'lawyers', 'baldwin', 'and', 'adams', '(
', 'matthew', 'mcconaughey', 'and', 'anthony', 'hopkins', ')', 'must', 'prove', 'cinque', 'and', 'the', 'others', 'were', '
captured', 'into', 'slavery', ',', 'rather', 'than', 'born', 'slaves', ',', 'in', 'order', 'to', 'get', 'them', 'out', 'of'
, 'prison', 'as', 'free', 'men', '.', 'three', 'lengthy', 'court', 'cases', 'are', 'portrayed', 'with', 'spielberg', "'", '
s', 'trademark', 'panache', '-', '-', 'flashy', 'beginning', ',', 'lots', 'of', 'facial', 'close', '-', 'ups', ',', 'big',
'music', ',', 'and', 'dramatic', 'imagery', '.', 'a', 'final', 'speech', 'by', 'adams', 'is', 'followed', 'by', 'an', 'anti
climax', 'where', 'subtitles', 'show', 'what', 'eventually', 'happened', 'to', 'the', 'various', 'characters', '.', 'opinio
n', ':', 'making', 'a', 'fictional', 'movie', 'is', 'easier', 'than', 'making', 'one', 'about', 'real', 'life', '.', 'in',
'fiction', ',', 'one', 'invents', 'purposeful', ',', 'clear', '-', 'cut', 'good', 'guys', 'and', 'bad', 'guys', ',', 'puts'
, 'the', 'fictional', 'characters', 'in', 'conflict', ',', 'and', 'takes', 'the', 'tale', 'to', 'its', 'exciting', 'conclus
ion', '.', 'real', 'life', ',', 'however', ',', 'consists', 'of', 'long', 'stretches', 'of', 'boredom', 'with', 'a', 'few',
 'dramatic', 'moments', 'and', 'characters', 'who', 'stand', 'around', ',', 'think', 'thoughts', 'and', 'do', 'nothing', '
', 'or', 'come', 'and', 'go', 'before', 'events', 'are', 'resolved', '.', 'spielberg', 'gives', 'us', 'a', 'visually', 'spi
cy', 'and', 'historically', 'accurate', 'real', 'life', 'story', '.', 'djimon', 'hounsou', 'and', 'anthony', 'hopkins', 'tu
rn', 'in', 'excellent', 'performances', '.'], 'pos')
[(',', 77717), ('the', 76529), ('.', 65876), ('a', 38106), ('and', 35576), ('of', 34123), ('to', 31937), ("'", 30585), ('is
', 25195), ('in', 21822), ('s', 18513), ('"', 17612), ('it', 16107), ('that', 15924), ('-', 15595)]
253


Press Enter to continue...:
```