**List of Experiments**

**Experiment No. 1**

**Task:** Write a python program to implement Breadth First Search Traversal?

**Solution:**

graph = {

'5' : ['3','7'],

'3' : ['2', '4'],

'7' : ['8'],

'2' : [],

'4' : ['8'],

'8' : []

}

visited = [] *# List for visited nodes.*

queue = [] *#Initialize a queue*

**def** bfs(visited, graph, node): *#function for BFS*

visited.append(node)

queue.append(node)

**while** queue: *# Creating loop to visit each node*

m = queue.pop(0)

**print** (m, end = " ")

**for** neighbour **in** graph[m]:

**if** neighbour **not** **in** visited:

visited.append(neighbour)

queue.append(neighbour)

*# Driver Code*

**print**("Following is the Breadth-First Search")

bfs(visited, graph, '5') *# function calling*

**Experiment No. 2**

**Task:** Write a python program to implement Water Jug Problem?

**Solution:**

from collections import defaultdict

jug1, jug2, aim = 4, 3, 2

visited = defaultdict(lambda: False)

def waterJugSolver(amt1, amt2):

if (amt1 == aim and amt2 == 0) or (amt2 == aim and amt1 == 0):

print(amt1, amt2)

return True

if visited[(amt1, amt2)] == False:

print(amt1, amt2)

visited[(amt1, amt2)] = True

return (waterJugSolver(0, amt2) or

waterJugSolver(amt1, 0) or

waterJugSolver(jug1, amt2) or

waterJugSolver(amt1, jug2) or

waterJugSolver(amt1 + min(amt2, (jug1-amt1)),

amt2 - min(amt2, (jug1-amt1))) or

waterJugSolver(amt1 - min(amt1, (jug2-amt2)),

amt2 + min(amt1, (jug2-amt2))))

else:

return False

print("Steps: ")

waterJugSolver(0, 0)

**Experiment No. 3**

**Task:** Write a python program to remove punctuations from the given string

**Solution:**

# define punctuation

punctuations = '''!()-[]{;:'"\,<>./?@#$%^&\*\_~}'''

my\_str = "Hello!!!, he said ---and went."

# To take input from the user

# my\_str = input("Enter a string: ")

# remove punctuation from the string

no\_punct = ""

for char in my\_str:

if char not in punctuations:

no\_punct+=char

print(no\_punct)

**Experiment No. 4**

**Task:** Write a python program to sort the sentence in alphabetical order.

**Solution:**

my\_str = input("Enter a string: ")

# breakdown the string into a list of words

words = my\_str.split()

# sort the list

words.sort()

# display the sorted words

for word in words:

print(word, end=“ ”)

**Experiment No. 5**

**Task:** Write a program to implement Hangman game using python.

**Solution:**

import random

name = input("What is your name? ")

print("Good Luck ! ", name)

words = ['rainbow', 'computer', 'science', 'programming',

'python', 'mathematics', 'player', 'condition',

'reverse', 'water', 'board', 'geeks']

word = random.choice(words)

print("Guess the characters")

guesses = ''

# any number of turns can be used here

turns = 12

while turns > 0:

# counts the number of times a user fails

failed = 0

for char in word:

if char in guesses:

print(char, end=" ")

else:

print("\_")

print(char, end=" ")

failed += 1

if failed == 0:

print("You Win")

print("The word is: ", word)

break

print()

guess = input("guess a character:")

guesses += guess

if guess not in word:

turns -= 1

print("Wrong")

print("You have", + turns, 'more guesses')

if turns == 0:

print("You Loose")

**Experiment No. 6**

**Task:** Write a program to implement Tic-Tac-Toe game using python.

**Solution:**

import numpy as np

import random

from time import sleep

def create\_board():

return(np.array([[0, 0, 0],

[0, 0, 0],

[0, 0, 0]]))

def possibilities(board):

l = []

for i in range(len(board)):

for j in range(len(board)):

if board[i][j] == 0:

l.append((i, j))

return(l)

def random\_place(board, player):

selection = possibilities(board)

current\_loc = random.choice(selection)

board[current\_loc] = player

return(board)

def row\_win(board, player):

for x in range(len(board)):

win = True

for y in range(len(board)):

if board[x, y] != player:

win = False

continue

if win == True:

return(win)

return(win)

def col\_win(board, player):

for x in range(len(board)):

win = True

for y in range(len(board)):

if board[y][x] != player:

win = False

continue

if win == True:

return(win)

return(win)

def diag\_win(board, player):

win = True

y = 0

for x in range(len(board)):

if board[x, x] != player:

win = False

if win:

return win

win = True

if win:

for x in range(len(board)):

y = len(board) - 1 - x

if board[x, y] != player:

win = False

return win

def evaluate(board):

winner = 0

for player in [1, 2]:

if (row\_win(board, player) or

col\_win(board,player) or

diag\_win(board,player)):

winner = player

if np.all(board != 0) and winner == 0:

winner = -1

return winner

def play\_game():

board, winner, counter = create\_board(), 0, 1

print(board)

sleep(2)

while winner == 0:

for player in [1, 2]:

board = random\_place(board, player)

print("Board after " + str(counter) + " move")

print(board)

sleep(2)

counter += 1

winner = evaluate(board)

if winner != 0:

break

return(winner)

# Driver Code

print("Winner is: " + str(play\_game()))

**Alternate Solution:**

import random

class TicTacToe:

def \_\_init\_\_(self):

self.board = []

def create\_board(self):

for i in range(3):

row = []

for j in range(3):

row.append('-')

self.board.append(row)

def get\_random\_first\_player(self):

return random.randint(0, 1)

def fix\_spot(self, row, col, player):

self.board[row][col] = player

def is\_player\_win(self, player):

win = None

n = len(self.board)

# checking rows

for i in range(n):

win = True

for j in range(n):

if self.board[i][j] != player:

win = False

break

if win:

return win

# checking columns

for i in range(n):

win = True

for j in range(n):

if self.board[j][i] != player:

win = False

break

if win:

return win

# checking diagonals

win = True

for i in range(n):

if self.board[i][i] != player:

win = False

break

if win:

return win

win = True

for i in range(n):

if self.board[i][n - 1 - i] != player:

win = False

break

if win:

return win

return False

for row in self.board:

for item in row:

if item == '-':

return False

return True

def is\_board\_filled(self):

for row in self.board:

for item in row:

if item == '-':

return False

return True

def swap\_player\_turn(self, player):

return 'X' if player == 'O' else 'O'

def show\_board(self):

for row in self.board:

for item in row:

print(item, end=" ")

print()

def start(self):

self.create\_board()

player = 'X' if self.get\_random\_first\_player() == 1 else 'O'

while True:

print(f"Player {player} turn")

self.show\_board()

# taking user input

row, col = list(

map(int, input("Enter row and column numbers to fix spot: ").split()))

print()

# fixing the spot

self.fix\_spot(row - 1, col - 1, player)

# checking whether current player is won or not

if self.is\_player\_win(player):

print(f"Player {player} wins the game!")

break

# checking whether the game is draw or not

if self.is\_board\_filled():

print("Match Draw!")

break

# swapping the turn

player = self.swap\_player\_turn(player)

# showing the final view of board

print()

self.show\_board()

# starting the game

tic\_tac\_toe = TicTacToe()

tic\_tac\_toe.start()

**Experiment No. 7**

**Task:** Write a python program to remove stop words for a given passage from a text file using NLTK?

**Solution:**

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

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)

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

filtered\_sentence = []

for w in word\_tokens:

    if w not in stop\_words:

        filtered\_sentence.append(w)

print(word\_tokens)

print(filtered\_sentence)

**Experiment No. 8**

**Task:** Write a python program to implement stemming for a given sentence using NLTK?

**Solution:**

# importing modules

from nltk.stem import PorterStemmer

from nltk.tokenize import word\_tokenize

ps = PorterStemmer()

sentence = "Programmers program with programming languages"

words = word\_tokenize(sentence)

for w in words:

print(w, " : ", ps.stem(w))

**Experiment No. 9**

**Task:** Write a python program to POS (Parts of Speech) tagging for the give sentence using NLTK?

**Solution:**

import nltk

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize, sent\_tokenize

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

 # Dummy text

txt = "Sukanya, Rajib and Naba are my good friends. "

"Sukanya is getting married next year. "

"Marriage is a big step in one’s life."

"It is both exciting and frightening. "

"But friendship is a sacred bond between people."

"It is a special kind of love between us. "

"Many of you must have tried searching for a friend "

    "but never found the right one."

# sent\_tokenize is one of instances of

# PunktSentenceTokenizer from the nltk.tokenize.punkt module

tokenized = sent\_tokenize(txt)

for i in tokenized:

    # Word tokenizers is used to find the words

    # and punctuation in a string

    wordsList = nltk.word\_tokenize(i)

    # removing stop words from wordList

    wordsList = [w for w in wordsList if not w in stop\_words]

    #  Using a Tagger. Which is part-of-speech

    # tagger or POS-tagger.

    tagged = nltk.pos\_tag(wordsList)

    print(tagged)

**Experiment No. 10**

**Task:** Write a python program to implement Lemmatization using NLTK?

**Solution:**

# import these modules

from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

print("rocks :", lemmatizer.lemmatize("rocks"))

print("corpora :", lemmatizer.lemmatize("corpora"))

# a denotes adjective in "pos"

print("better :", lemmatizer.lemmatize("better", pos ="a"))

**Experiment No. 11**

**Task:** Write a python program to for Text Classification for the give sentence using NLTK

**Solution:**

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"])

**Some extra programs:**

1. Write a python program to implement Depth First Search Traversal?

# Using a Python dictionary to act as an adjacency list

graph = {

'5' : ['3','7'],

'3' : ['2', '4'],

'7' : ['8'],

'2' : [],

'4' : ['8'],

'8' : []

}

visited = set() # Set to keep track of visited nodes of graph.

def dfs(visited, graph, node): #function for dfs

if node not in visited:

print (node)

visited.add(node)

for neighbour in graph[node]:

dfs(visited, graph, neighbour)

# Driver Code

print("Following is the Depth-First Search")

dfs(visited, graph, '5')