Write a python program to generate Calendar for the given month and year?

```
import calendar

yy = 2023  # year
mm = 12    # month

# To take month and year input from the user
# yy = int(input("Enter year: "))
# mm = int(input("Enter month: "))

# display the calendar
print(calendar.month(yy, mm))
```

Write a Python program to implement Depth First Search algorithm. Refer the following graph as an Input for the program.[Initial node=1,Goal node=7].

```
graph = {
  'A' : ['B','C','G'],
  'B' : ['D', 'E'],
  'C' : ['F'],
  'G' : [],
  'D' : [],
  'E' : ['F'],
  'F' : []
}
visited=set()
def dfs(graph,start,visited):
  if visited is None:
    visited.add(start)
  print(start)
  for i in graph[start]:
    dfs(graph,i,visited)


  return visited
dfs(graph,'A',visited)
```

Write a python program to remove punctuations from the given string?

```
punctuations = '''!()-[]{};:'"\,<>./?@#$%^&*_~''' # define punctuation
                                                  # To take input from the user
                                                  # my_str = input("Enter a string: ")# remove punctuation from the string
my_str = "Hello!!!, How  --- r @ U ."
no_punct = ""
for char in my_str:
   if char not in punctuations:
       no_punct = no_punct + char
print(no_punct)
```

Write a Python program to implement Depth First Search algorithm. Refer the following graph as an Input for the program.[Initial node=2,Goal node=7]

Q.1)Write a program to implement Hangman game using python. [10 Marks] Description: Hangman is a classic word-guessing game. The user should guess the word correctly by entering alphabets of the user choice. The Program will get input as single alphabet from the user and it will matchmaking with the alphabets in the original

```
import time       #importing the time module
name = input("What is your name? ") #welcoming the user
print ("Hello, " + name, "Time to play hangman!")
print ("Start guessing...")
time.sleep(0.5)
word = ("secret")       #here we set the secret. You can select any word to play with.
guesses = ''            #creates an variable with an empty value
turns = 10 #determine the number of turns
while turns > 0:     # Create a while loop  #check if the turns are more than zero
    failed = 0
    for char in word:    # for every character in secret_word
```

```
        if char in guesses:   # see if the character is in the players guess
            print (char,end=""),  # print then out the character
        else:                             # if not found, print a dash
            print ("_",end=""),     # and increase the failed counter with one
            failed += 1    # if failed is equal to zero
    if failed == 0:   # print You Won
        print ("You won")
        break         # exit the script
    guess = input("guess a character:")  # ask the user go guess a character
    guesses += guess     # set the players guess to guesses
    if guess not in word:  # if the guess is not found in the secret word
        turns -= 1        # turns counter decreases with 1 (now 9
        print ("Wrong")  # print wrong
        print ("You have", + turns, 'more guesses' ) # how many turns are left
        if turns == 0:      # if the turns are equal to zero
            print ("You Lose"  )# print "You Lose"
```

Write a Python program to implement Breadth First Search algorithm. Refer the following graph as an Input for the program.[Initial node=1,Goal node=8]

```
graph = {
  'A' : ['B','C','G'],
  'B' : ['D', 'E'],
  'C' : ['F'],
  'G' : [],
  'D' : [],
  'E' : ['F'],
  'F' : []
}
visited = [] # List to keep track of visited nodes.
queue = []     #Initialize a queue
def bfs(visited,graph,node):
  visited.append(node)
  queue.append(node)
  while queue:
    s = queue.pop(0)
    print (s, end = "   ")
    for i in graph[s]:#checking adjacent
      if i not in visited:
        visited.append(i)
        queue.append(i)
bfs(visited, graph, 'A')
```

Write a python program to implement Lemmatization using NLTK

```
import nltk
nltk.download('wordnet')
```

```
# import these modules
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
print("frogs :", lemmatizer.lemmatize("frogs"))
# a denotes adjective in "pos"
print("better :", lemmatizer.lemmatize("better", pos ="a"))
```

Q.2) Write a Python program to implement Breadth First Search algorithm. Refer the following graph as an Input for the program.[Initial node=1,Goal node=8]

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

Write a Python program to implement Breadth First Search algorithm. Refer the following graph as an Input for the program.[Initial node=1,Goal node=8].

)Write a python program implement tic-tac-toe using alpha beeta pruning

Write a Python program to implement Simple Chatbot.

```
pip install chatterbot
```

```
from chatterbot import ChatBot

chatbot = ChatBot("Chatpot")

exit_conditions = (":q", "quit", "exit")
while True:
    query = input("> ")
    if query in exit_conditions:
        break
    else:
        print(f"🪴 {chatbot.get_response(query)}")
```

Double-click (or enter) to edit

Write a Python program to accept a string. Find and print the number of upper case alphabets and lower case alphabets.

```
Str="GeeksForGeeks"
lower=0
upper=0
for i in Str:
  if(i.islower()):
      lower+=1
  else:
      upper+=1
print("The number of lowercase characters is:",lower)
print("The number of uppercase characters is:",upper)
```

Write a Python program to solve tic-tac-toe problem

```
# Tic-Tac-Toe game in Python
board = [" " for x in range(9)]
def print_board():
    row1 = "| {} | {} | {} |".format(board[0], board[1], board[2])
    row2 = "| {} | {} | {} |".format(board[3], board[4], board[5])
    row3 = "| {} | {} | {} |".format(board[6], board[7], board[8])
    print()
    print(row1)
    print(row2)
    print(row3)
    print()
def player_move(icon):
    if icon == "X":
        number = 1
    elif icon == "O":
        number = 2
    print("Your turn player {}".format(number))
    choice = int(input("Enter your move (1-9): ").strip())
    if board[choice - 1] == " ":
        board[choice - 1] = icon
    else:
        print()
        print("That space is taken!")
def is_victory(icon):
    if (board[0] == icon and board[1] == icon and board[2] == icon) or \
       (board[3] == icon and board[4] == icon and board[5] == icon) or \
       (board[6] == icon and board[7] == icon and board[8] == icon) or \
       (board[0] == icon and board[3] == icon and board[6] == icon) or \
       (board[1] == icon and board[4] == icon and board[7] == icon) or \
       (board[2] == icon and board[5] == icon and board[8] == icon) or \
       (board[0] == icon and board[4] == icon and board[8] == icon) or \
       (board[2] == icon and board[4] == icon and board[6] == icon):
```

```
        (board[2] == icon and board[4] == icon and board[6] == icon):
            return True
        else:
            return False
    def is_draw():
        if " " not in board:
            return True
        else:
            return False
    while True:
        print_board()
        player_move("X")
        print_board()
        if is_victory("X"):
            print("X wins! Congratulations!")
            break
        elif is_draw():
            print("It's a draw!")
            break
        player_move("O")
        if is_victory("O"):
            print_board()
            print("O wins! Congratulations!")
            break
        elif is_draw():
            print("It's a draw!")
            break2
```

Write Python program to implement crypt arithmetic problem TWO+TWO=FOUR

```
from itertools import permutations

def is_valid_assignment(mapping):
    two1 = mapping['T']
    w = mapping['W']
    o1 = mapping['O']
    f = mapping['F']
    u = mapping['U']
    r = mapping['R']

    two2 = two1 * 10 + w
    four = f * 1000 + o1 * 100 + u * 10 + r

    return two2 + two2 == four

def solve_cryptarithmetic():
    unique_digits = set(range(10))
    letters = ['T', 'W', 'O', 'F', 'U', 'R']

    for perm in permutations(unique_digits, len(letters)):
        mapping = dict(zip(letters, perm))
        if is_valid_assignment(mapping):
            print("Valid assignment found:")
            print(f"TWO: {mapping['T']}{mapping['W']}{mapping['O']}")
            print(f"+ TWO: {mapping['T']}{mapping['W']}{mapping['O']}")
            print(f"--------")
            print(f"FOUR: {mapping['F']}{mapping['O']}{mapping['U']}{mapping['R']}")
            return

    print("No valid assignment found.")

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

Write a Python program to implement Simple Chatbot.