9.#pgm\_to\_check\_the\_season

#function\_that\_return\_the\_season...

def season(month, date):

if((month >= 11 and date >=21) or (month <=3 and date <=19)):

return 'winter'

if((month >= 3 and date >=20) or (month <=6 and date <=20)):

return 'summer'

if((month >= 6 and date >=21) or (month <=9 and date <=21)):

return 'spring'

if((month >= 9 and date >=22) or (month <=11 and date <=20)):

return 'fall'

#------main\_function-------

month = input('Which month is this :')

date = int(input('Who told the date :'))

if(month == 'january'):

m=1

elif(month == 'february'):

m=2

elif(month == 'march'):

m= 3

elif(month == 'april'):

m = 4

elif(month == 'may'):

m=5

elif(month == 'june'):

m=6

elif(month == 'july'):

m=7

elif(month == 'august'):

m=8

elif(month == 'september'):

m=9

elif(month == 'october'):

m=10

elif(month == 'november'):

m=11

elif(month == 'december'):

m=12

2. leap year

def isLeap(year):

if(year % 4 == 0):

if(year % 100 == 0):

if(year % 400 == 0):

return True

else:

return False

else:

return True

else:

return False

year = int(input("Enter a valid year :"))

if(isLeap(year)):

print("Leap Year....")

else:

print("Not a Leap year....")

4. merge two sorted list

#function\_to\_take\_input..

def insert\_list(l,s):

for i in range(1,s+1):

data = int(input())

l.append(data)

#function\_to-merge\_lists..

def merged(l1, l2):

l1.sort()

l2.sort()

l1 += l2

l1.sort()

return l1

s1 = int(input('size-1 :'))

s2 = int(input('size-2 :'))

firstList = []

secondList = []

print("Enter ",s1," elements :")

insert\_list(firstList,s1) #insert\_data into list1

print("Enter ",s2," elements :")

insert\_list(secondList,s2) #insert\_data into list2

print(merged(firstList, secondList))

10.remove words

from collections import Counter

def removeCommonWords(sent1, sent2):

sentence1 = list(sent1.split())

sentence2 = list(sent2.split())

frequency1 = Counter(sentence1)

frequency2 = Counter(sentence2)

word = 0

for i in range(len(sentence1)):

if sentence1[word] in frequency2.keys():

sentence1.pop(word)

word = word-1

word += 1

word = 0

for i in range(len(sentence2)):

if sentence2[word] in frequency1.keys():

sentence2.pop(word)

word = word-1

word += 1

print(\*sentence1)

print(\*sentence2)

sentence1 = input("s1=")

sentence2 = input("s2=")

removeCommonWords(sentence1, sentence2)

3.max words

def max\_words(my\_list):

max\_ = 0

empty = []

for i in my\_list:

current = len(i.split(' '))

#print(current)

if(current > max\_):

max\_ = current

return max\_

listOfSentences = []

sizeOfList = int(input("May I know the size of list :"))

print("Start entering sentences limit is :", sizeOfList)

for i in range(1,sizeOfList+1):

newSentence = input()

listOfSentences.append(newSentence)

print("The sentence with max words that are :",max\_words(listOfSentences))

5. basic calculater

print("Enter the expression")

class Solution:

def calculate(self, s: str) -> int:

stack = []

def append(operator, operand):

if operator == "+":

stack.append(operand)

elif operator == "-":

stack.append(-operand)

elif operator == "\*":

stack.append(stack.pop() \* operand)

else:

stack.append(int(stack.pop() / operand))

operator, operand = "+", 0

for c in s:

if c in "0123456789":

operand = operand \* 10 + int(c)

elif c in "+-\*/":

append(operator, operand)

operator, operand = c, 0

append(operator, operand)

return sum(stack)

6.Keyword

class Solution(object):

def letterCombinations(self, digits):

if len(digits) == 0:

return []

characters = {2:"abc",3:"def",4:"ghi",5:"jkl",6:"mno",7:"pqrs",8:"tuv",9:"wxyz"}

result = []

self.solve(digits,characters,result)

return result

def solve(self, digits, characters, result, current\_string="",current\_level = 0):

if current\_level == len(digits):

result.append(current\_string)

return

for i in characters[int(digits[current\_level])]:

self.solve(digits,characters,result,current\_string+i,current\_level+1)

print("Enter the number")

s=input()

ob1 = Solution()

print(ob1.letterCombinations(s))

7.GENERATE PARENTHASES

class Solution(object):

def generateParenthesis(self, n):

"""

:type n: int

:rtype: List[str]

"""

result = []

self.generateParenthesisUtil(n,n,"",result)

return result

def generateParenthesisUtil(self, left,right,temp,result):

if left == 0 and right == 0:

result.append(temp)

return

if left>0:

self.generateParenthesisUtil(left-1,right,temp+'(',result)

if right > left:

self.generateParenthesisUtil(left, right-1, temp + ')', result)

ob = Solution()

n = int(input("enter the number:"))

print(ob.generateParenthesis (n))

8.regular expression

import re

s = input("enter the first string:")

p = input("enter the second string:")

p = r"{}".format(p)

p = re.compile(p)

if p.fullmatch(s):

print("true")

else:

print("false")

1. STAIRCASE

def stair(n):

if n <= 1:

return n

else:

return stair(n-1) + stair(n-2)

s=int(input("n="))

print("Number of possible ways for climbing the stair cases",stair(s+1))