# gets a number and save to variable n

n = eval( input('Enter a number: ') )

s = "Hello, how, are you!?"

>>> s.replace(",","").replace("?","")

s = "Hello, how, !.;;...are you!?"

>>> for punct in ".,!?;:":

s = s.replace(punct,"")

return

# 2d search

def contains( target, lst ):

for row in lst:

for item in row:

# look for target

if item.lower()==target.lower():

#print( item )

return True

return False

####

def findTreasure( tmap ):

locations = []

lines = tmap.split() # list of strs

# visit all the locations on the map

for i in range(len(lines)):

# iterate over the str lines[i]

for j in range(len(lines[i])):

if lines[i][j]=='X': #treasure

#print( i,j, lines[i][j] )

locations.append( (i,j) )

return locations

###################

def acronym( phrase ):

acro = ""

for word in phrase.split():

#print( word[0],end='' )

acro += word[0]

return acro.upper()

# print( acro.upper() )

############

def factorial(n):

product = 1

for i in range(2,n+1):

#print(i)

product \*= i

return product

#all upper case letters are less than all lower case letters

#slicing

s[start:stop] - substring of s

>>> 'apple' < 'pear'

True

>>> 'Pear' < 'apple'

True

>>> 'Z' > 'a'

False

t = 'railroad'

>>> t[0:4]

'rail'

#lists:

append(item) - add item to the end

pop() - removes and returns last item

count(item)

index(item) - index of first occurence

remove(item) - removes (one copy)

reverse()

sort()

#########

print( arguments, sep=' ', end ='\n' )

by default sep = ' '

end = '\n'

#########

open(filename,mode='r')

modes

'r'ead - the default

'w'rite

'a'ppend

file methods

read() - reads contents as a single str

readlines() - read contents as a list

of strings, one per line

write(s) - OVERWRITE writes a single string s

to a file, less friendly than print

write() argument must be str, not int

>>> outfile.write( str(5) )

#####

def numChars(filename):

return len( open(filename).read() )

######

eval( open('numbers.csv').readlines()[1].split(',')[3])

21

#######

def getCell(filename,row,col):

# the -1's are to translate from excel row,col numbering

# to python indices

return eval( open(filename).readlines()[row-1].split(',')[col-1])

###############

def doubleVowel(w):

v = 'aeiouAEIOU'

for i in range( len(w)-1):

if w[i]in v and w[i+1] in v:

return True

return False

#######

import math

def collision(x1,y1,r1,x2,y2,r2):

distance= math.sqrt((x2-x1)\*\*2+(y2-y1)\*\*2)

if distance > r1+r2:

return False

else:

return True

#############

def vowelCount(phrase):

#countLst indexed as a,e,i,o,u

countLst=[0,0,0,0,0]

#vowels=""

for char in phrase:

if char.lower() == "a":

countLst[0]+=1

elif char.lower()=="e":

countLst[1]+=1

elif char.lower()=="i":

countLst[2]+=1

elif char.lower()=="o":

countLst[3]+=1

elif char.lower()=="u":

countLst[4]+=1

print("a, e, i, o, and u appear, respectively,

" + str(countLst).strip('[]') + " times.")

###########

def crypto(someFile):

file=open(someFile)

print(file.read().replace('secret','xxxxxx'))

file.close()

############

def exclamation(someStr):

newLst=""

for vowel in someStr:

if vowel in "aeiouAEIOU":

for i in range(4):

newLst+=vowel

else:

newLst+=vowel

return newLst+'!'

###############

def geometric(intLst):

ratios=[]

for i in range(len(intLst)-1):

ratios.append(intLst[i+1]/intLst[i])

for i in range(len(ratios)-1):

if ratios[i] != ratios[i+1]:

return False

return True

###############

def prime(num):

count=0

for i in range(num+1):

if i>=1:

if num%i==0:

count+=1

if count>2:

return False

return True

##1

class Pizza:

def \_\_init\_\_(self, size='M', toppings=None):

self.s=size

if toppings is None:

toppings = set() #done bc sets are immutable

self.t=toppings

def \_\_repr\_\_(self):

return 'Pizza(\'{}\',{})'.format(self.s,self.t)

def \_\_eq\_\_(self,other):

return (self.s,self.t)==(other.s,other.t)

def setSize(self,size):

if size in 'SML':

self.s=size.upper()

else:

self.s='M'

def getSize(self):

return self.s

def addTopping(self,topping):

self.t.add(topping)

def removeTopping(self,topping):

self.t.remove(topping)

def price(self,price=0.0):

if self.s=='S':

price=6.25

for topping in self.t:

price+=0.7

elif self.s=='M':

price=9.95

for topping in self.t:

price+=1.45

else:

price=12.95

for topping in self.t:

price+=1.85

return price

##2

def orderPizza():

myPizza = Pizza()

print('Welcome to Python Pizza!')

size=input('What size pizza would you like (S,M,L): ')

myPizza.setSize(size)

topping=input('Type topping to add (or Enter to quit): ')

while topping !='':

myPizza.addTopping(topping)

topping=input('Type topping to add (or Enter to quit): ')

print ('Thanks for ordering!')

print ('Your pizza costs ${}'.format(myPizza.price()))

return myPizza

####3

class Stack(list):

#LIFO

#sub/child class of the super/parent class

#pop and len already inherited

#must define push: single item

#must write repr

#be careful for infinite recursion

#inheritance needs diff repr:

def \_\_repr\_\_(self):

return 'Stack({})'.format(list.\_\_repr\_\_(self))

#called extending a method. could also do list(self) but not as good

def push(self,item):

self.append(item)

def isEmpty(self):

return self==[]

####4

def parenthesesMatch(string):

#dont need to index...compare the popped item

#to closing par w/ iteration

#check beginning first or

#else dont do

opp = {'{':'}','[':']','(':')'}

for bracket in string:

if bracket not in '[]{}()':

return False

else:

continue

stack=Stack()

for bracket in string:

if bracket in '[{(':

stack.push(bracket)

elif bracket in ']})' and stack.isEmpty():

return False

elif bracket in ']})' and not stack.isEmpty():

openBracket = stack.pop()

if opp[openBracket] == bracket:

continue

else:

return False

return stack.isEmpty()

#####1

class Volume:

#stereo volume b/w 0 and 11

def \_\_init\_\_(self, initialLvl=0):

self.set(initialLvl)

#gaurantees that every volume gets a new value

def \_\_repr\_\_(self):

return 'Volume({})'.format(self.level)

def \_\_eq\_\_(self,other):

return self.level==other.level

#returns True if Volumes have the same value

def get(self):

return self.level

def set(self, vol):

if vol<0:

self.level=0

elif vol>11:

self.level=11

else:

self.level=vol

def up(self, moveUpBy):

self.set(self.level+moveUpBy)

def down(self, moveDownBy):

self.set(self.level-moveDownBy)

############2

def partyVolume(fileStr):

myFile=open(fileStr)

initValue=eval(myFile.readline())

myVol=Volume(initValue)

restV=myFile.readlines()

myFile.close()

for line in restV:

line=line.strip('\n').split(' ')

if line[0]=='U':

myVol.up(eval(line[1]))

elif line[0]=='D':

myVol.down(eval(line[1]))

return myVol

#5.48 done

def sublist(list1, list2):

startAt=0

sublist1=[]

for i in range(len(list1)):

if startAt==len(list2):

break

for j in range(startAt,len(list2)):

if list1[i]==list2[j]:

startAt=j+1

sublist1.append(list1[i])

break

else:

continue

return sublist1==list1

#'cross out' irrelevant items in second list-->hint

#i and j counters

#while you still have numbers in both lists

#i=j mean increment both

#no match means increment just j

#if first list exhausted, then True

#do not use sets

#similiar to previous problem stategy

#try printing or printing vars()

#6.22 done

def mirror(word):

letterDict={'q':'p', 'p':'q','d':'b','b':'d'}

newWord=''

for letter in word:

if letter not in 'qwtuiopdlxvbnmWTYUIOAHXVM':

return 'INVALID'

for i in range(len(word)):

if word[i] in letterDict:

newWord+=letterDict[word[i]]

else:

newWord+=word[i]

return newWord[::-1]

#6.30 done

def rps():

poss=['R','P','S']

player1=random.choice(poss)

player2=random.choice(poss)

ans=str(player1+player2)

if ans=='RS' or ans=='SP' or ans=='PR':

return -1

elif ans=='SR' or ans=='PS' or ans=='RP':

return 1

else:

return 0

def simul(intN):

rounds=intN

player1Win=0

player2Win=0

ties=0

numGames=0

while numGames<=rounds:

numGames+=1

winner=rps()

if winner==-1:

player1Win+=1

elif winner==1:

player2Win+=1

else:

ties+=1

if player1Win>player2Win:

print('Player 1')

elif player2Win>player1Win:

print ('Player 2')

else:

print ('Tie')

#6.31 done

def craps():

roll = random.randrange(1,7) + random.randrange(1,7)

if roll==7 or roll==11:

return 1

elif roll==2 or roll==3 or roll==12:

return 0

else:

newRoll=random.randrange(1,7) + random.randrange(1,7)

while newRoll!=roll and newRoll!=7:

newRoll = random.randrange(1,7) + random.randrange(1,7)

return int(newRoll == roll)

def testCraps(someInt):

n=abs(someInt)

wins=0

total=0

while total<=n:

wins+=craps()

total+=1

return wins/total

class Queue:

def \_\_init\_\_(self, lst = []):

# very important

# make sure you call list()

# constructor in the body

# of the method

# guarantees that every

# Queue gets a new list

self.q = list( lst )

def \_\_repr\_\_(self):

return "Queue({})".format( self.q )

def enqueue(self, item):

self.q.append( item )

def dequeue(self):

return self.q.pop(0)

# q[0] -> Queue.\_\_getitem\_\_(q,0)

# note this also makes iteration with

# in possible

def \_\_getitem\_\_(self,index):

return self.q[index]

# len(q) -> Queue.\_\_len\_\_(q)

def \_\_len\_\_(self):

return len(self.q)

def \_\_eq\_\_(self,other):

# can also write as if,else

return self.q==other.q

# + -> \_\_add\_\_

def \_\_add\_\_(self,other):

return Queue(self.q + other.q)

class Queue(list):

# inherit \_\_init\_\_, i.e.,

# dont write it, use list version

# extend the method

def \_\_repr\_\_(self):

return "Queue({})".format( list.\_\_repr\_\_(self) )

# must write enqueue, dequeue

def enqueue(self,item):

self.append(item)

def dequeue(self):

return self.pop(0)

# in, \_\_getitem\_\_, \_\_len\_\_. \_\_eq\_\_ is inherited

# q[1]=99 -> Queue.\_\_setitem\_\_(q,1,99)

def \_\_setitem\_\_(self,index,item):

raise TypeError('no cutting, bad code!')

def \_\_add\_\_(self,other):

return Queue( list.\_\_add\_\_(self,other) )

class Temp:

def \_\_init\_\_(self,degree=0,fc='C'):

self.d=degree

self.fc=fc

def \_\_repr\_\_(self):

return "Temp({},{})".format(self.d,self.fc)

def \_\_eq\_\_(self,other):

(self.d,self.fc)==(other.d,other.fc)

def convert(self):

if self.fc=='C':

self.d=(self.d\*(9/5))+32

self.fc='F'

else:

self.d=(self.d-32)\*(5/9)

self.fc='C'

def lineNumber(multiLine,searchWrds):

multiLine2=multiLine.lower().replace('.','').split('\n')

searchWrds2=set(searchWrds.lower().split(' ')

for line in range(len(multiLine2)):

words=set()

for wrd in searchWrds2:

if wrd in multiLine2[line].split(' '):

words.add(wrd)

if words==searchWrds2:

return line

return -1

class Counter(dict):

def \_\_init\_\_(self,d={}):

self=d

def count(self,item):

if item in self:

self[item]+=1

else:

self[item]=1

def \_\_repr\_\_(self):

return 'Counter({})'.format(dict.\_\_repr\_\_(self))

def \_\_getitem\_\_(self,index):

if index not in self:

return 0

else:

return dict.\_\_getitem\_\_(self,index)

def countItems(self,items):

for item in items:

self.count(item)

def printCounts(self):

for item in sorted(self):

print (item,self[item])

def diceGame():

from random import randrange as rr

prevRolls=[]

finalRoll=()

count=0

while True:

if len(prevRolls)<5:

roll = ((rr(1,7),rr(1,7)))

if roll in prevRolls or roll[::-1] in prevRolls:

finalRoll=roll

prevRolls.append(roll)

break

prevRolls.append(roll)

continue

roll=((rr(1,7),rr(1,7)))

if roll in prevRolls[-5:] or roll[::-1] in prevRolls[-5:]:

finalRoll=roll

prevRolls.append(roll)

break

prevRolls.append(roll)

for rolls in prevRolls:

if sum(rolls)==sum(finalRoll):

count+=1

return count