##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 craps2():

import random

di1=random.randrange(1,7)

di2=random.randrange(1,7)

dice=di1+di2

if dice==7 or dice==11:

return 1

elif dice==2 or dice==3 or dice==12:

return 0

else:

di3=random.randrange(1,7)

di4=random.randrange(1,7)

dice2=di3+di4

while dice2!=dice and dice2!=7:

if dice2==7:

return 1

elif dice2==dice:

return 0

else:

break'''

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