# Sean MacBride

# Program: smacbrideP6.py

# Description: Simulates the card game of 'Mickslam!' using a couple of different classes.

# Input: If you would like to continue with the game, an input of c (computer) or p (player) will determine who you believe to win. Will need repeated inputs until either you run out of cards, or you enter q into the prompt

# Output: The hand dealt for the player and points associated with that hand, the hand dealt for the computer (if necessary) and points associated with that hand, and repetition of the hands dealt until you run out of cards or input q to end the game. After the game concludes, will display the win values of the computer and player.

import random #Importing Random

class Game: #Creating the class Game, which acts as the game manager for your hand and the computer's hand

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

self.wincount=0 #Setting initial wincount to 0

self.points=0 #Setting initial points to 0

def reset(self): #A reset method used to reset the point values after a round of the game

self.points=0

def Calc(self, face): #A calc method to calculate the point values of a card based on the face of the card

if face=="King" or face=="Queen" or face=="Jack":

return 10

elif face=="Ace":

return 11

else:

return face

def pointShow(self): #A method to display the amount of points a hand has

print("\nPoints:",self.points)

def Won(self): #A method to update the wincount if the computer or player won the game

self.wincount+=1

class Deck: #The deck class, which keeps track of all of the cards played

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

self.deck=list(range(52)) #Using the method used in cards.py, I set all the entries in a list of 52 items to False

for card in self.deck:

self.deck[card]=False

self.cardnum=0 #The amount of cards that have been played. Is used later as a condition on a for loop

#dealCard uses the ideas put forth in cards.py of a 52 elements representing 52 different cards in a french deck

def dealCard(self): #A method to deal a card

card = random.randint(0,51) #Determines a random number to put in the list

while self.deck[card] == True: #Checks to see if the number was already in the list

card = random.randint(0, 51) #Re runs the random number integer to see if it was in the list

self.deck[card]=True #Puts the number in the list, effectively making the card in that place "dealt"

face,suit=self.cardconvert(card) #Calling cardconvert(card) which returns the face and suit of the card

self.showCard(face,suit) #calls the showCard function that prints the card

self.cardnum+=1 #Adds one to the cardnum count

return face #Returns face

def cardconvert(self,card): #The cardconvert method which returns the face and suit of the card given the card integer value

if card < 13:

value = card+1

suit = "Hearts"

elif card < 26:

value = card-12

suit = "Diamonds"

elif card < 39:

value = card-25

suit = "Clubs"

else:

value = card-38

suit = "Spades"

face=self.stringconvert(value)

return face,suit #returns face and suit

def stringconvert(self,value): #The stringconvert method that returns the face of the card (ex. 11 corresponds to jack)

if value==1:

face="Ace"

elif value==11:

face="Jack"

elif value==12:

face="Queen"

elif value==13:

face="King"

else:

face=value

return face #returns face

def showCard(self,face,suit): #The showCard method that prints out the correct format of the card

print(face,"of",suit) #Printing the card

def main(): #The main function, which runs the game

cards=Deck() #Creating the deck of cards

player=Game() #Creating the player's game

computer=Game() #Creating the computer's game

prompt=0 #placeholder for prompt, get's updated later

while cards.cardnum < 50 and prompt != "q": #While loop to test if all the possible cards have been played (if 50 cards can be played, you can't play another full round of the game) and if the prompt asks for you to quit

print("\nPlayer cards:\n") #Prints a nice header

for i in range(5): #for loop to deal 5 cards in the player's first hand

card=cards.dealCard() #dealing a card to player

player.points+=player.Calc(card) #adding the point value of the card to the player's points

player.pointShow() #displaying the points

prompt=input("\nWho do you believe will win? Input p for player, c for computer, and q to quit. ") #Asking who you think will win

if prompt !="q": #As long as you don't want to quit, will run this loop

print("\nComputer cards:\n") #Prints a nice header

for i in range(5): #a for loop to deal the computers hand

card=cards.dealCard() #dealing a hand to the computer

computer.points+=computer.Calc(card) #adding to the computer's point total

computer.pointShow() #displaying the computer's points

if computer.points==player.points: #Checking if there's a tie

print("\nTie! Nobody wins!\n")

elif computer.points>player.points and prompt=="c" or player.points>computer.points and prompt=="p": #Checking if the player has won

print("\nPlayer wins!\n")

player.Won() #updating player win count

else: #Else, the computer has won

print("\nComputer Wins!\n")

computer.Won() #updating the computers win count

player.reset() #Resetting the player's point values

computer.reset() #resetting the computer's point values

input("Press enter to continue") #Giving a prompt to break between games

print("\nGame Over!") #nice header to display a game over

print("\nPlayer wins:",player.wincount) #player win totals

print("\nComputer wins:",computer.wincount) #computer win totals

main()

#I have abided by the Wheaton Honor code in this work

#Prints a power of a set of two numbers

class number:

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

self.base=num

def pow(self,val):

return self.base\*\*val

def print(self):

print(self.base)

def main():

tmp = eval (input ("Enter number: "))

p = eval (input ("Enter a power: "))

x = number (tmp)

y = x.pow (p)

x.print()

print (y)

main()

#Prints the following:

one

2

two

flam

three

1

four

Bozo

class Zork:

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

print ("Creating Zork with:", num)

self.value = 2 \* num

def Zing (self):

num = self.value \* 2

self.value = self.value + num

num = num - 2

print (num, self.value)

def main ():

z = Zork(2) # line X

z.Zing ()

z.Zing ()

#main()

flam = 1

flim = {"two":"flam", "one":2, "three":flam}

for x in ["one", "two", "three", "four"]:

print (x)

print (flim.get (x, "Bozo"))

# Sean MacBride

# Program: smacbrideP5.py

# Description: A program that simulates a european roulette table at a casino, where you can bet in 5$ increments.

# Input: Your starting bankroll, the amount you are willing to bet for bet 1, where you would like to bet for bet 1 (Must be a number 0-36 for numbers, R or B for Colors, or X Y or Z for rows), the amount you would like to bet for bet 2 and where you would like to bet it, if applicable. The amount you are betting will repeat until you input 0$ for your first bet.

# Output: The result of the roulette spin, with the appropriate effect on your bankroll, before prompting you to bet again. Will repeat until the input of bet 1 is 0. At that point, will print out the final bankroll value

#Importing random

import random as rand

#A function that calculates the winnings of a particular bet.

#Takes the bankroll, the location of the bet, and the amount bet at that location as parameters.

#Returns the updated bankroll

#This function is only called if a bet is a winner

def winnings(bankroll, betspot,betval):

if betspot<=36 or betspot==42: #Winnings calculation for a number bet

winnings=36\*betval

elif betspot<=38: #Winnings calculation for a color bet

winnings=2\*betval

elif betspot<=41: #Winnings calculation for a row bet

winnings=3\*betval

bankroll+=winnings

return bankroll #Returning the bankroll

#A function to check if a spin is a win based on the bet

#Takes the spin number, the spin color, the spin row, and location of the bet as parameters

#Returns True or False if it is a win or not a win

def checkWin(spinnumber,spincolor,spinrow,betspot):

if betspot==spinnumber or betspot==spincolor or betspot==spinrow:

return True

else:

return False

#A function to Spin the wheel

#Takes no parameters

#Returns a numerical value for the row, color, and number

def getSpin():

numberval=rand.randrange(0,37,1) #The random number generator

redlist=[1,3,5,7,9,12,14,16,18,19,21,23,25,27,30,32,34,36] #List of all red numbers

blacklist=[2,4,6,8,10,11,13,15,17,20,22,24,26,28,29,31,33,35] #List of all black numbers

if numberval in redlist: #If statements to determine the color of the number

color=38 #using spec sheet numberic representation

elif numberval in blacklist:

color=37 #using spec sheet numeric representation

else:

color=42 #letting the number 42 as a color value equal green

if color!=42: #Making sure that the color is not green. If it isn't green, The row will return 42. Just a way to save not going through the loop

if numberval%3==0: #If statements to determine the row of the number

row=41

elif numberval%3==1:

row=39

elif numberval%3==2:

row=40

else:

row=42 #Assigning the row value=42 for a green slot

#I used the number values you gave in the spec sheet in my code, and added row and color values of 42 to be attributed to green

#Returning the number, color, and row values

return numberval, color, row

#The controller function that asks for the bankroll

#Takes no parameters

#Calls the wager function, which does most of the work

def controller():

#Asking for the first bankroll

print()

bankroll=int(input("Enter Your Starting Bankroll! $"))

wager(bankroll)

#The converter function that helps convert bet placement inputs

#Takes the location of the bet as an input

#Returns a numerical value of the betspot

#I used the numerical values given in the spec sheet, with the exception of green, which has number 0, row 42, and color 42

def converter(betspot):

if betspot=="B":

return 37

elif betspot=="R":

return 38

elif betspot=="X":

return 39

elif betspot=="Y":

return 40

elif betspot=="Z":

return 41

else:

return eval(betspot)

#A function that returns the finished string of the roulette spin

#Takes the number and color of the spin as parameters

#returns the finished string of the result of the roulette spin

def stringer(number,color):

if color==37:

return str(number)+" Black"

elif color==38:

return str(number)+" Red"

else:

return "0 Green"

#The wager function, which does most of the work with print statements and calling other functions

#Takes the bankroll for parameter

#Outputs the bet amounts, bet locations, results of the bets, and repeats until you enter 0 as you first bet amount

def wager(bankroll):

print() #A print statement for nice formatting

bet1amount=int(input("First bet amount : $")) #Asking for the first bet amount

while bet1amount!=0: #Running a loop that will repeat until you enter 0 in bet1amount (at the end of the loop)

bet1point=input("Name your bet location : ") #Asking for the location of bet1

bankroll=bankroll-bet1amount #initially updating the bankroll

bet1num=converter(bet1point) #Calling the converter function that converts the location of bet1 to a numeric value, makes it easier to deal with

bet2amount=int(input("Second bet amount : $")) #Asking for a second bet

if bet2amount!=0: #Similar to the first loop, but this time will check to see if bet2amount is not 0. If it is 0, there's no need to ask for the location, and convert it to a numeric value, or update the bankroll

bet2point=input("Name your bet location : ") #Asking for the location of bet2

bankroll=bankroll-bet2amount #Updating the bankroll from bet2

bet2num=converter(bet2point) #Calling the converter function that converts the location of bet2 to a numeric value, makes it easier to deal with

spinnumber, spincolor, spinrow = getSpin() #Spinning the wheel with the getSpin function and getting the values of the wheel

spinstring=stringer(spinnumber,spincolor) #Calling the stringer function and returning it to get the final string value

result1=checkWin(spinnumber,spincolor,spinrow,bet1num) #The result of the first bet

if bet2amount!=0: #As long as bet2 is not 0, will check to see the result of the second bet

result2=checkWin(spinnumber,spincolor,spinrow,bet2num) #The result of the second bet

else:

result2=False #Letting result2=false for a loop later in the code, as to not create any "referenced before assignment" errors

print() #a print statement for nice formatting

if result1==True or result2==True: #Printing the results of the bet if it won

print("RESULT - ", spinstring, " - WINNER", sep="") #The winning print statment

if result1==True: #Calling the winnings function to update the bankroll if result1 was a winner

bankroll=winnings(bankroll,bet1num,bet1amount)

if result2==True: #Calling the winnings function to update the bankroll if result2 was a winner

bankroll=winnings(bankroll,bet2num,bet2amount)

else: #A print statement for a spin where you did not win on either bet

print("RESULT - ", spinstring, " - NO WIN", sep="")

print() #a print statement for nice formatting

print("Bankroll: $",bankroll,sep="") #The updated bankroll from your bet

print() #a print statement for nice formatting

bet1amount=int(input("First bet amount : $")) #Asking for the first bet again

print() #a print statement for nice formatting

print("Final Bankroll: $",bankroll,". Thanks for playing!", sep="") #A print statement of the final bankroll

#Main, which calls controller

def main():

controller()

main()

#I have abided by the Wheaton Honor Code in this work

#Gives the square root of a number rounded to 2 dec places

import math

def main():

num=eval(input("Enter your number here: "))

sqroot=math.sqrt(num)

sqroot= int((sqroot + 0.005) \* 100) / 100

print("The square root of",num,"is",sqroot)

main()

#Creates a list and prints it out within a class

class myList:

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

self.list1=[]

def addValue(self, a):

self.list1.append(a)

def printList(self):

for item in self.list1:

print(item, end=" ")

def lab2():

lablist=myList()

num=int(input("Enter an integer: "))

while num!=0:

lablist.addValue(num)

num=int(input("Enter an integer: "))

lablist.printList()

print("The end!")

lab2()

#Prints out the list by going first, last, second, second last, etc.

def lab11\_1c():

list1=[]

inp=int(input("Enter an integer "))

while inp!=0:

list1.append(inp)

inp=int(input("Enter an integer "))

front=0

back=len(list1)-1

while front<back:

print(list1[front], list1[back],end=" ")

front+=1

back-=1

lab11\_1c()

#Prints out a list of numbers

def lab11\_1a():

list1=[]

inp=int(input("Enter an integer "))

while inp!=0:

list1.append(inp)

inp=int(input("Enter an integer "))

for item in list1:

print(item,end=" ")

print()

s=int(input("Enter an s "))

for item in range(0,len(list1),s+1):

print(list1[item], end=" ")

lab11\_1a()

#A class with the output

Hello

This string has the letter l 2 times

This string has the letter e 1 times

The strings are not the same

The strings are the same

The strings are not the same

The strings are the same

class String:

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

self.string=string

def display(self):

print(self.string)

def countLetter(self, letter):

print("This string has the letter", letter, self.string.count(letter)+self.string.count(letter.capitalize()), "times")

def compare(self, string):

if self.string==string.string:

return True

else:

return False

def sameWord(self, string):

if self.string.upper()==string.string.upper():

return True

else:

return False

def main():

str1=String("Hello")

str1.display()

str1.countLetter("l")

str1.countLetter("e")

str2=String("Hello")

if str1==str2:

print("The strings are the same")

else:

print("The strings are not the same")

if str1.compare(str2)==True:

print("The strings are the same")

else:

print("The strings are not the same")

str3=String("hello")

if str1.compare(str3)==True:

print("The strings are the same")

else:

print("The strings are not the same")

if str1.sameWord(str3)==True:

print("The strings are the same")

else:

print("The strings are not the same")

main()

#a GREAT example of using dictionaries for a state capitals list

class database:

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

self.states = {"Massachusetts":"MA"}

self.capitals = {"MA":"Boston"}

def displayStates(self):

for i in self.states:

print("\n",i)

def addState(self):

fullname=input("\nWhat is the state's full name? ")

abbreviation=input("\nWhat is the state's abbreviation? ")

capital=input("\nWhat is the state's capital? ")

self.states[fullname]=abbreviation

self.capitals[abbreviation]=capital

def displayCapital(self):

inp=input("\nWhat is the two letter abbreviation of the state that you are looking for? ")

local=self.capitals.get(inp,"bad")

while local=="bad":

print("\nThat input was invalid")

inp=input("\nWhat is the two letter abbreviation of the state that you are looking for? ")

local=self.capitals.get(inp,"bad")

print("\n",self.capitals[inp])

def showMenu(self):

print("\n1 -- Display the states in the database")

print("2 -- Add a new state")

print("3 -- Display state capital")

print("4 -- Display state abbreviation")

print("5 -- Quit\n")

choice=eval(input("What is your choice? "))

return choice

def displayAbbrev(self):

inp=input("\nEnter the state's full name to get its abbreviation: ")

local=self.states.get(inp,"bad")

while local=="bad":

print("\nThat input was invalid")

inp=input("\nWhat is the two letter abbreviation of the state that you are looking for? ")

local=self.states.get(inp,"bad")

print("\n",self.states[inp])

def main():

myDB = database()

choice = myDB.showMenu()

while choice != 5:

if choice == 1:

myDB.displayStates()

elif choice == 2:

myDB.addState()

elif choice == 3:

myDB.displayCapital()

elif choice == 4:

myDB.displayAbbrev()

else:

print ("Choice is invalid")

choice = myDB.showMenu()

print ("\nAdios!")

main()

#Intro to classes

#A fraction class

class Fraction:

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

self.numerator=0

self.denominator=1

self.result=0

self.whole=0

def input(self):

fract=input()

slash=fract.find("/")

self.numerator=int(fract[0:slash])

self.denominator=int(fract[slash+1:])

def mult(self, val1,val2):

self.numerator=val1.numerator \* val2.numerator

self.denominator=val1.denominator \* val2.denominator

def display (self) :

if self.numerator==0:

print(self.numerator)

elif self.whole!=0:

print(self.whole, " ",self.numerator, "/",self.denominator, sep="",end="")

else:

print(self.numerator, "/",self.denominator, sep="",end="")

def add (self, val1, val2):

answer=Fraction()

answer.denominator=val1.denominator\*val2.denominator

firstvalnum=val1.numerator\* (answer.denominator//val1.denominator)

secondvalnum=val2.numerator\* (answer.denominator//val2.denominator)

answer.numerator=firstvalnum+secondvalnum

return answer

def mixed(self):

self.whole=self.numerator//self.denominator

self.numerator=self.numerator%self.denominator

def main():

f1=Fraction()

f2=Fraction()

print("The value of f1 is: ", end="")

f1.display()

print("Enter a fraction: ",end="")

f1.input()

print("Enter another fraction: ", end="")

f2.input()

print("The fractions are: ",end="")

f1.display()

print(" and ", end="")

f2.display()

print()

f3=Fraction()

f3.mult(f1,f2)

print("The product of f1 and f2 is: ", end="")

f3.display()

print()

f4=Fraction()

f5=Fraction()

f5=f4.add(f1,f2)

print("The value of f4 is: ", end="")

f4.display()

print("The value of f5 is: ", end="")

f5.display()

f5.mixed()

print()

print("The mixed value of f5 is: ", end="")

f5.display()

main()

#Making a nice litte x o o o x o o o x tic tac toe board

def printer(board):

for i in range(3):

for j in range (3):

print(board[i][j], end="")

if j<2:

print(" | ", end="")

print()

if i<2:

print("--+----+--")

def main():

tictactoe=[[" " for i in range (3)] for j in range (3)]

for i in range (3):

for j in range(3):

if i==j:

tictactoe[i][j]="X"

else:

tictactoe[i][j]="O"

printer(tictactoe)

main()

#Function practice, prints out

56 81 14, all on separate lines

x=2

def getval(one,two):

two=two\*x

return one+two

def main():

x=5

for i in range(5,1,-2):

i=getval(i,x)+i

x=x+i

print(i)

print(x)

print(getval(getval(2,1),5))

main()

#Create a myMath library

class myMath:

def abs(number):

if number>0:

return number

else:

return number\*-1

def round(number):

if number>0:

number=int(number+0.5)

else:

number=int(number-0.5)

return number

def round1(number):

if number>0:

number=int((number+0.05)\*10)/10

else:

number=int((number-0.05)\*10)/10

return number

def main():

m=myMath()

num=eval(input("Enter a number "))

print("Rounded, that number is",myMath.round(num))

print("Rounded to one decimal place, that number is",myMath.round1(num))

print("The absolute value of that number is", myMath.abs(num))

main()

#Deals a random poker hand

# cards.py

#

# List and class example simulating playing cards.

#

# Cards are assigned values as follows:

# 1=A, 2-10, 11=J, 12=Q, 13=K

# There are 52 cards, 13 each in Hearts, Diamonds, Spades, and Clubs

import random

class Cards:

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

# constructor: creates a deck of cards

self.deck = list(range(52)) # makes a deck of 52 cards

# set all cards to false

for card in self.deck:

self.deck [card] = False;

self.count = 0 # counts number of cards used

def getCount (self):

# access function to return current count

return self.count

def dealCard (self):

# method to randomly select a card from the deck;

# if a card has already been selected, it may not be chosen again

if self.count < 52:

card = random.randint(0,51) # INCLUSIVE

# check if card has been used before and get another

while self.deck[card] == True:

card = random.randint(0, 51)

# card has been "taken"

self.deck[card] = True

self.count = self.count + 1

# print ("card number", self.count, "is", card)

else:

print ("All the cards have been dealt")

def showCard (self, value, suit):

# method to display one card in nice way

if value == 1:

face = "Ace"

elif value == 11:

face = "Jack"

elif value == 12:

face = "Queen"

elif value == 13:

face = "King"

else:

face = value

print (face, "of", suit)

def displayCards (self):

# method to display all cards dealt so far

for card in range (52):

if self.deck[card] == True:

# print (card)

if card < 13:

value = card + 1

suit = "Hearts"

elif card < 26:

value = card - 12

suit = "Diamonds"

elif card < 39:

value = card - 25

suit = "Clubs"

else:

value = card - 38

suit = "Spades"

self.showCard (value, suit)

def main ():

deck = Cards()

while deck.getCount () < 52:

deck.dealCard()

# can now play game or whatever

newdeck = Cards()

for i in range (5):

newdeck.dealCard()

print()

print ("Your poker hand is:")

print()

newdeck.displayCards()

main()