[Code partially from Zelle/Lambert]

V.Rego,

Tu Mar 22, 2016

Week 11, Examples 1

```
#__________________Cannonball with simple
#graphics______________________________
#1.py

#Here is the code from 4.py of Week11-Lec1. We modified it VERY slightly by
#adding a graphics
#window and a circle to represent the CB.

''' Look for lines flagged with "##".
These are the only lines that were changed or added.

In particular, look at how we passed the graphics window to the
constructor so that
the projectile class has access to a window to draw in. Look at how we
defined instance
variables and got a circle object for this class.

NOTE: In this example we gave the Projectile class the ability to do the
graphing.

Sometimes you may not want to do that. You may want to keep all the
graphics outside
such a class. If we wanted to take this route in this example, we can
simple define the
circle object in the main program and then use "move" inside the while
loop to draw
cb once we have its position.

DIMENSIONING: In this example, its possible that the cb flies out of the
window. It means
velocity and/or angle are very high and our window was not big enough.
To solve this
problem, you can

(a) run the loop once without graphics and get the max height and max
distance, or
(b) use calculus to get max height and max distance.

Once you have the proper dimensions (max height, max distance) for any
given input parameters,
you can create a graphics window to handle that size and then run the
simulation again, but
 *this time graph the cb.*

...

# HW: How would you make the graphics work for any projectile input
parameters?

```python
from graphics import *  #

from math import sin, cos, radians

def wait():             #
    x = input()        #

class Projectile:

    def __init__(self, angle, velocity, height, w):  #
        self.xp = 0
        self.yp = 0
        theta = radians(angle)
        self.xv = velocity * cos(theta)
        self.yv = velocity * sin(theta)
        self.win = w

        self.c = Circle(Point(self.xp, self.yp), 30)  #
        self.c.draw(self.win)

        self.c.setFill("red")

    def position_update(self, t):
        oldx = self.xp  #
        oldy = self.yp  #

        self.xp = self.xp + t * self.xv
        yv_new = self.yv - 9.8 * t
        average_velocity = (self.yv + yv_new) / 2
        self.yp = self.yp + t * average_velocity
        self.yv = yv_new

        self.c.move(self.xp - oldx, self.yp - oldy)  #
```

```python
def height(self):
    return(self.yp)

def distance(self):
    return(self.xp)

#____________ code below is not in the class

def get_input_values():
    a = eval(input("Enter launch angle (degrees): "))
    v = eval(input("Enter initial velocity (meters/sec): "))
    h = eval(input("Enter initial height (meters): "))
    t = eval(input("Enter small time-interval between position updates of cb: "))
    return(a,v,h,t)

# Here is a main program that uses the above class definition for projectiles

# Note that you can add as many methods as you need, and you can make them as complex as you need

def main():
    win = GraphWin("Cannonball trajectory",800,800)  
    win.setCoords(-50,-50,4050,8050)  

    angle, vel, h0, t = get_input_values()

    cb = Projectile(angle, vel, h0,win)

    while (cb.height() >= 0):
        cb.position_update(t)

        print ("CB is now {0:0.2f} meters high and is {1:0.2f} meters away.").format(cb.height(),cb.distance())

    wait()

main()
```
#2.py

# You have now seen how we defined
# a Projectile class (so we could get a cannonball object), and
# a MSD (many-sided random die) class, so we could toss very general die in games/gambling
# REVIEW, to make sure you understand class. Let's use the MSD

#msd.py an n-sided die, using Python class definition

from random import randrange

# notice below how we will defin a class, it's constructor (a function that initializes it), and
# its any methods.

# 1. How to define the class and constructor function?

class MSD:

    def __init__(self, n):
        # self is a "special parameter" used to make
        # whatever object you are
        # defining refer (point) to itself.
        # You don't have to use the word "self"
        # because other languages
        # also use it, though they do not
        # have to use it as a parameter

        self.n = n
        self.value = 1  # any valid value is okay for an initial value

    # notice the word self. It helps the class refer to ITS OWN variables.
    # These are called "instance
    # variables". The first parameter in each call, while defining the class,
    # will always be "self".

    # __init__() will give the caller an MSD object. But what can he do with
    # it? To use this object
    # he needs functions, but such functions in a class are called "methods".

    ''' Let's write some functions that do the kinds of things we do when we
    play with dice

http://courses.cs.purdue.edu/
```python
def roll(self):
    # note that roll() throws but does not return a value
    self.value = randrange(1, self.n + 1)  # adding 1 because of how randrange works
    # calling roll is like crossing the MSD object

    # Again, look at how we had to use "self" as the first (and in this case only) parameter. It allows
    # the die to get a random integer and store it in its own variable "self.value"

    The functions below are similar.

    def getValue(self):
        # to get the face value after throwing we must call getValue()
        return(self.value)  # notice how we are working with the instance variables defined
        __init__() is the CONSTRUCTOR

    def setValue(self, value):
        self.value = value  # since self always has to be there, we'll say setValue() has only
        # one parameter which is "value"

    # instance variables + methods = attributes [that's class terminology]
```

# Now you know everything you need to know to define a class. To know how to write a class
# you need to think of your application in terms of objects (i.e., projectiles, dice, airplanes, shapes).

# If you look at 3.py in the last week lecture's example set you will see a main() that create
# 3 MSD objects for 3 players in a simple game. This is how you will create and use objects
# in general.

#
# Datafile for next program (in 3.py)

# save this tab-separated data "as is" in file "list.txt" and then run the program

# the format is: student name, total credit-hours, grade-points based on total credit-hours  (e.g. A at 3 credits gives 4X3 = 12 pts, if A=4)
# make sure there is no blank line at the end of the file, or the program will try to read it

Abercrombie, Fitch  129  230
Baggins, Bilbo  100  370
Baggins, Frodo  93  320
Cromwell, Oliver  150  160
Duck, Daffy  121  75
Fudd, Elmer  120  75
Fisher, Kin  130  139
Gearloose, Gyro  50  160
Defrog, Kermit  60  100
Scissorhands, Edward  100  200
Dagreek, Zorba  175  321

#______ Example: Using Python "class" for data processing___________________
#3.py

class Student:
    def __init__(self, name, hours, points):  # constructor for class "Student"
        self.name = name
        self.hours = float(hours)  # cast it as a real number
        self.points = float(points)  # "      (will help when using division later)

    def Name(self):
        return (self.name)  # a method, and so caller accesses instance variable via
does not touch it directly and

    def Hours(self):
        return(self.hours)  # possibly mess it up (change it unknowingly).
def Points(self):
    return(self.points)

def gpa(self):
    return(self.points/self.hours)

#________________________ Student Class ends here _________________________

def makeStudent(istring):
    #str is "tab separated": name, hours, points [see datafile list.txt]
    #makeStudent returns a Student Object
    name, hours, points = istring.split("	"

    return (Student(name,hours,points)) # IMPORTANT! Observe how it returns a
    # Student object via the constructor

    #Let's write a program to find max gpa, min gpa and average gpa

    def main():

        infile = open("list.txt","r")

        high = makeStudent(infile.readline()) #assume temporarily that first student is both
        low  = high
        
        # highest and lowest gpa student

        total_points = 0 # for average gpa
        total_hours = 0

        for line in infile:

            s = makeStudent(line)

            if (s.gpa() > high.gpa() ): high = s #update high gpa object if necc.
            if (s.gpa() < low.gpa() ): low = s # likewise

            total_points = total_points + s.Points()
            total_hours = total_hours + s.Hours()
infile.close()  # loop is done, change indentation to exit

print("\n")
print("Highest:")
print(high.Name(), "\t", high.Hours(), "(hours)", "\t", high.gpa(), "(gpa)")
print("\n")

print("Lowest:")
print(low.Name(), "\t", low.Hours(), "(hours)", "\t", low.gpa(), "(gpa)")
print("\n")

print("Average over all data:", "\t", total_hours, "(hours)", "\t", total_points/total_hours, "(gpa)")

main()

# ____________________________ How to play the game of
# Craps
#4.py

# In the next file we will have two classes and then a main program (with
two functions)
# that plays the game of craps using 2 dice.

# How does this game work?
...

__________________________ CRAPS (game rules)__________________________

There is a single player.

The player rolls a pair of dice (i.e., two dice, having 6 sides)
Record the "initial roll" (i.e., first roll of both dice)

Player LOSES: if die1 + die2 is equal to 2, 3, or 12

Player WINS: if die1 + die2 is equal to 7 or 11

If player neither wins nor loses on initial roll, player continues to roll.

On any roll:

player WINS: if die1 + die2 = sum of die1 and die2 on the
"initial roll"
player LOSES: if die1 + die2 equals 7

---

We'll have a Die class (so we can have two 6-sided dice)

We'll have a Player class (so the player object can get 2 die objects and play the game using methods in the Player class and methods in the Die class)

# What does the Die method do?

```
d = Die()               returns a die object, face = 1

d.roll()                   rolls the die, so face is 1 or 2 or 3 ... or 6.
It only rolls, does not return a face

d.getFace()          returns the face value

d.__str__()           same as str(d); returns string representation of
die's face value

# What does the Player method do?

```
p = Player()                         returns a new Player object. This
object plays the game.

p.play()                                plays the game; if player wins it
returns True, else False

p.getNumberOfRolls()        returns the number of rolls

p.__str__()                          same as str(p); it returns a formatted
string representation of the
number of rolls

#___________ Python Classes to play a game of Craps____________________________

#5.py
'IMPORTANT: I have put (a) Class Die, (b) Class Player, and (2) two functions that play games, invoked by main() in this one file 5.py.

So there are there separate things (modules) in this one file. This makes it easy to upload as 5.py, and also makes it easy to run in class.

However, what YOU need to do is to

(a) put the Class Die in file die.py [this file will say "from random import randint" at the top]

(b) put the Class Player in file craps.py [this file will say "from die import Die" at the top]

(c) put the two last functions in a separate file along with main(). Call it anything you want, but this is the module you will run.

So you now see how

** different classes go into different files as separate modules **

and you can import these classes for use wherever you want 

File: die.py

This module defines the die class

from random import randint
class Die:

"""This class offers a 6-sided die""

    def __init__(self):
        """The initial face of the die""
        self.face = 1

    def roll(self):
        """Make face of die some random number between 1 and 6""
        self.face = randint(1,6)

    def getFace(self):
        return(self.face)
def __str__(self):  #will be useful in building history of tosses
    return(str(self.face))     
#--------------------------------------------------------------------------------------
#--------------------------------------------------------------------------------------
#--------------------------------------------------------------------------------------
###

File: craps.py

This module studies and plays the game of craps

#from die import Die
class Player:

    def __init__(self):
        """Player object gets a pair of dice and an empty list of rolls.""

            self.die1 = Die()
            self.die2 = Die()
            self.rolls = []

        def __str__(self):
            #always make sure the indentation is correct

                """Returns the string representation of the history of rolls""

                result = " "
                for (v1, v2) in self.rolls:

                        result = result + str((v1, v2)) + " " + str(v1 + v2) + "\n"

                return(result)

        def getNumberOfRolls(self):
            """Returns the number of rolls in one game""
            return (len(self.rolls))

        def play(self):
            """Plays one game, saves all the rolls for that game, and finally returns True
for a win and False for a loss.""

                self.rolls = [ ]
self.die1.roll()
self.die2.roll()

(v1, v2) = (self.die1.getFace(), self.die2.getFace())

self.rolls.append((v1, v2))

initialSum = v1 + v2

if (initialSum in (2, 3, 12)):
    return False  # loss
elif (initialSum in (7, 11)):
    return True  # win

while True:  # otherwise keep playing
    self.die1.roll()
    self.die2.roll()

    (v1, v2) = (self.die1.getFace(), self.die2.getFace())

    self.rolls.append((v1, v2))

    sum = v1 + v2

    if (sum == 7):
        return False
    elif (sum == initialSum):
        return True

# from craps import Player

# Now some functions that interact with the user to play the game

def playOneGame():
    """ Plays a single game and prints the results."""

    player = Player()
    youWin = player.play()

    print(player)  # print forces a call to str(player) so it can print a string

    if youWin:
        print(" You win!")
Now some functions that interact with the user to play the game

```python
def playOneGame():
    """ Plays a single game and prints the results."""

    player = Player()
    youWin = player.play()

    print(player)       # print forces a call to str(player)
    so it can print a string
    
    if youWin:
        print(" You win!")
    else:
        print(" You lose!")

def playManyGames():
    """ Plays a number of games and prints statistics."""

    number = int(input("How many games will you play? "))
    
    wins = 0
    losses = 0
    winRolls = 0
    lossRolls = 0

    player = Player()

    for count in range(number):

        hasWon = player.play()
        rolls = player.getNumberOfRolls()
        if hasWon:
            wins = wins + 1
            winRolls = winRolls + rolls
        else:
            losses = losses + 1
            lossRolls = lossRolls + rolls

    print("Total number of wins: ",wins)
    print("Total number of losses: ",losses)

    print("Average number of rolls per win: %0.2f" % (winRolls / wins))
    print("Average number of rolls per loss: %0.2f" % (lossRolls / losses))
    print("Winning percentage: %0.3f" %( wins / number))
```
def main():
    print("Play one game: \n")
    playOneGame()
    print(" ")
    print("Now play many games. \n")
    playManyGames()
main()