Week 12 - Wednesday

COMP 1800

Last time

- What did we talk about last time?
- Objects
- Classes
  - Constructors
  - Accessors
  - Mutators

Questions?

Assignment 8

Class Examples

`Student` class

- Let's write a **`Student`** class
- Instance variables:
  - First Name
  - Last Name
  - GPA
  - ID
- We need accessors for all of the instance variables
- And mutators for GPA

More special methods

- Python has other special methods
- Some are useful if your class is designed to hold a collection of things
  - The **`__getitem__`** method retrieves an item based on the index specified
  - The **`__len__`** method returns the number of items in the collection
  - The **`__contains__`** method says whether or not an element is in your collection

Sentence class

- Let's make a **`Sentence`** class
- Its constructor
  - Takes a string
  - Splits that string on spaces to make a list of strings
  - Stores that list as its instance variable
- The **`__getitem__`** method should return the specified words in the list
- The **`__len__`** method returns the number of words in the sentence
- The **`__contains__`** method should say whether the list contains the string the user is looking for

Hiding Data

Encapsulation

- In many programming languages, there's a way to keep member variables hidden from the outside world
- Java, C++, and C# use the **`private`** keyword to mark member variables (and methods) as inaccessible from outside of the class
- Such variables can only be affected from the outside by methods

Hiding data in Python

- Python doesn't have a **`private`** keyword
- Instead, it uses a naming convention to hide variables
- All member variables that you want to be hidden should have names that start with double underscore (__)
- Such variables cannot be accessed directly
- I didn't talks about data hiding before because:
  - Hiding variables in Python this way is not as universal as in languages like Java
  - It makes stuff ugly to read
  - It adds another layer of confusion
- If you're serious about writing object-oriented Python, you should still do it

Hiding example

- Here's part of the **`Planet`** class from before, with appropriate hiding

```
class Planet:
    def __init__(self, name, radius, mass, distance):
        self.__name = name
        self.__radius = radius
        self.__mass = mass
        self.__distance = distance

    def getName(self):
        return self.__name

    def setName(self, name):
        self.__name = name
```

Solar System

Let's animate a solar system

- We already have a **`Planet`** class, but we need to add:
  - *x* location
  - *y* location
  - *x* velocity
  - *y* velocity
  - Color
  - A **`turtle`** object to draw the planet

Updated `Planet` constructor

```
class Planet:
    def __init__(self, name, radius, mass, distance, xVelocity, yVelocity, color):
        self.name = name
        self.radius = radius
        self.mass = mass
        self.distance = distance
        self.x = distance
        self.y = 0
        self.xVelocity = xVelocity
        self.yVelocity = yVelocity
        self.color = color
        # turtle stuff
        self.turtle = turtle.Turtle()
        self.turtle.color(self.color)
        self.turtle.shape('circle')
        self.turtle.up()
        self.turtle.goto(self.x, self.y)
        self.turtle.down()
```

Other `Planet` methods

```
def getName(self):
    return self.name

def getX(self):
    return self.x

def getY(self):
    return self.y

def getXVelocity(self):
    return self.xVelocity
```

```
def getYVelocity(self):
    return self.yVelocity

def setXVelocity(self, xVelocity):
    self.xVelocity = xVelocity

def setYVelocity(self, yVelocity):
    self.yVelocity = yVelocity

def moveTo(self, x, y):
    self.x = x;
    self.y = y
    self.turtle.goto(x, y)
```

We need a Sun class as well

```
class Sun:
    def __init__(self, radius, mass):
        self.radius = radius
        self.mass = mass
        self.x = 0
        self.y = 0
        # turtle stuff
        self.turtle = turtle.Turtle()
        self.turtle.color('yellow')
        self.turtle.shape('circle')
```

```
    def getMass(self):
        return self.mass

    def getX(self):
        return self.x

    def getY(self):
        return self.y
```

SolarSystem class

- Finally, a **SolarSystem** class will hold a **Sun** object and a list of **Planet** objects

```
class SolarSystem:
    def __init__(self, width, height):
        self.sun = None
        self.planets = []
        screen = turtle.Screen()
        screen.setworldcoordinates(-width/2.0, -height/2.0, width/2.0, height/2.0)

    def setSun(self, sun):
        self.sun = sun

    def addPlanet(self, planet):
        self.planets.append(planet)
```

Moving the planets

- The most important method in **`SolarSystem`** uses simplified (but still confusing) physics to move the planets around

```
def movePlanets(self):
        G = .1 # fake gravitational constant (real one is smaller)
        time = .001 # time in seconds
        for planet in self.planets:
            planet.moveTo(planet.getX() + time * planet.getXVelocity(),
                          planet.getY() + time * planet.getYVelocity())
            deltaX = self.sun.getX() - planet.getX()
            deltaY = self.sun.getY() - planet.getY()
            distance = math.sqrt(deltaX**2 + deltaY**2)
            accelerationX = G * self.sun.getMass() * deltaX/distance**3
            accelerationY = G * self.sun.getMass() * deltaY/distance**3
            planet.setXVelocity(planet.getXVelocity() + time * accelerationX)
            planet.setYVelocity(planet.getYVelocity() + time * accelerationY)
```

Code that uses these classes

- We can create a **`SolarSystem`**, a **`Sun`**, and four **`Planet`** objects, and make them work
- Note that these values for radius, mass, distance, and velocities are chosen to look okay and have nothing to do with reality

```
solarSystem = SolarSystem(2, 2)
sun = Sun(5000, 10)
solarSystem.setSun(sun)
solarSystem.addPlanet(Planet('Mercury', 19.5, 1000, .25, 0, 2, 'blue'))
solarSystem.addPlanet(Planet('Earth', 47.5, 5000, 0.3, 0, 2, 'green'))
solarSystem.addPlanet(Planet('Mars', 50, 9000, 0.5, 0, 1.63, 'red'))
solarSystem.addPlanet(Planet('Jupiter', 100, 49000, 0.7, 0, 1, 'black'))
steps = 2000
for step in range(steps):
    solarSystem.movePlanets()
```

Quiz

Upcoming

Next time…

- Work time for Assignment 8

Reminders

- Finish Assignment 8