
THE UNIVERSITY OF WESTERN AUSTRALIA

# Lecture 13
# Objects

# Objectives of this Lecture

- To get familiar with Objects
- To understand the concept of objects and how they can be used to simplify programs
- Understand that in Python, everything is actually an object
- To get familiar with the various objects available in the graphics library
- To be able to create objects in programs
  - *call appropriate methods to perform graphical computations*

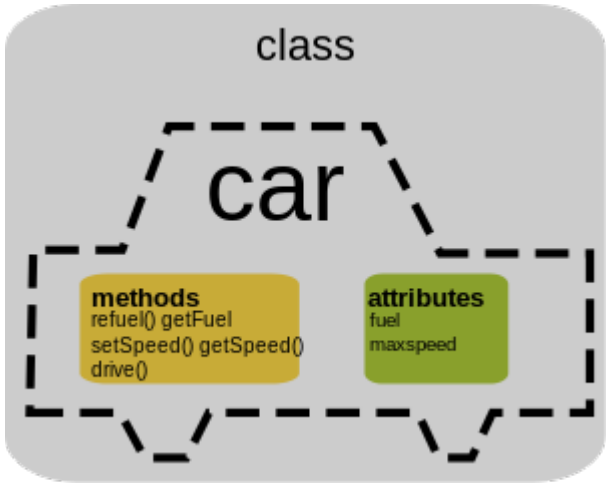
# Overview

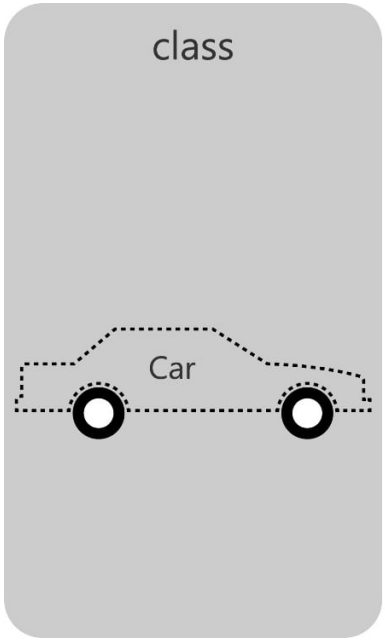
- So far, we saw that each data type can represent a certain set of values, and each had a set of associated operations.
- The traditional programming view is that data is passive – it is manipulated and combined using active operations.
- Modern computer programs are built using an object–oriented approach.

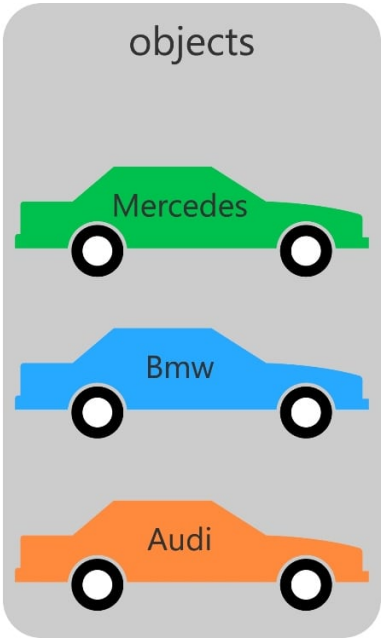
# Objects and Object Oriented Programming

- Basic idea – view a complex system as the interaction of simpler objects.
- An object is a kind of active data type that combines data and operations.
  - *Objects know stuff (contain data) and they can do stuff (have operations).*
- Objects interact by sending each other messages *(requests do to stuff)*.

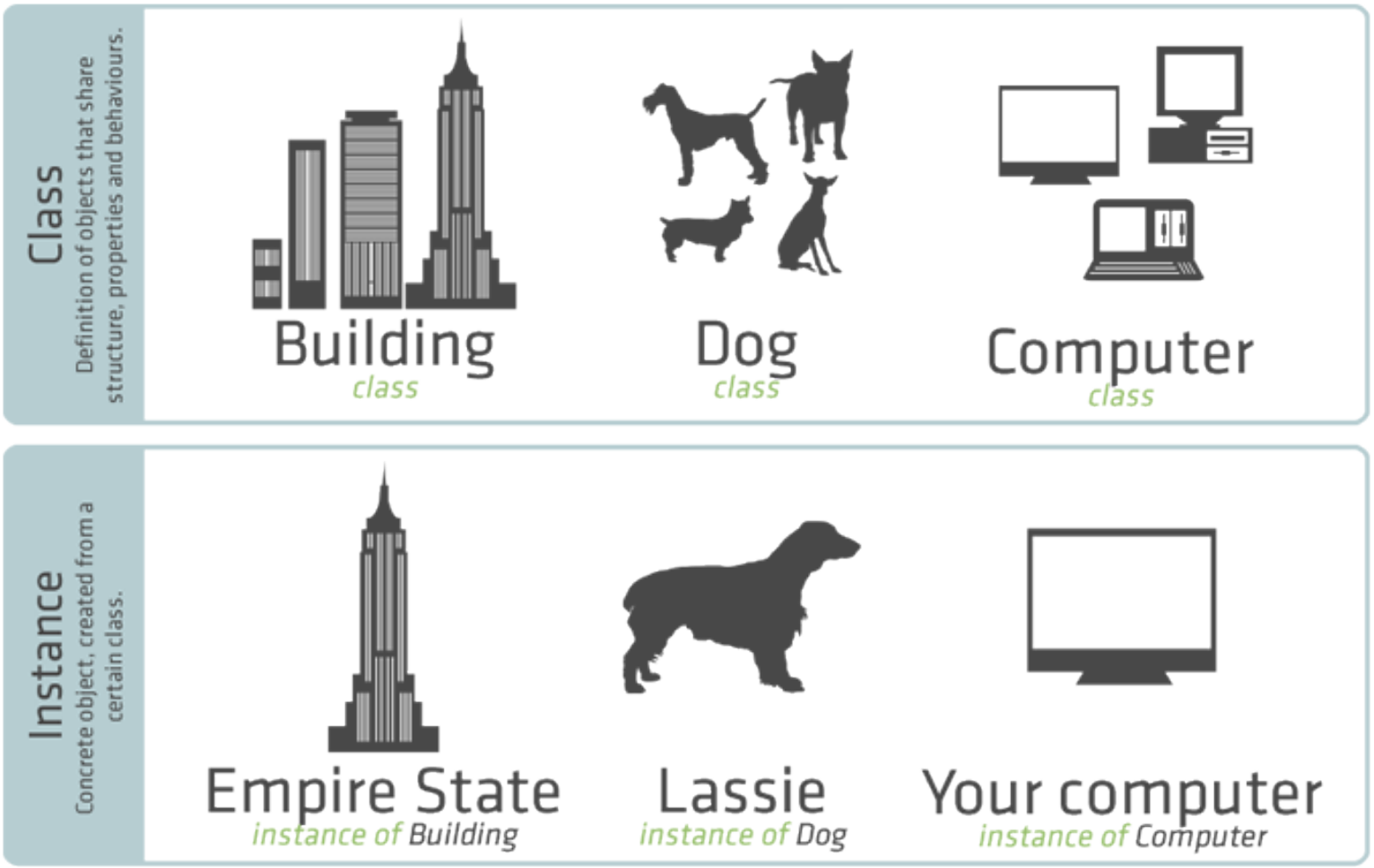
# OOP concept


class
car
methods
refuel() getFuel
setSpeed() getSpeed()
drive()
attributes
fuel
maxspeed

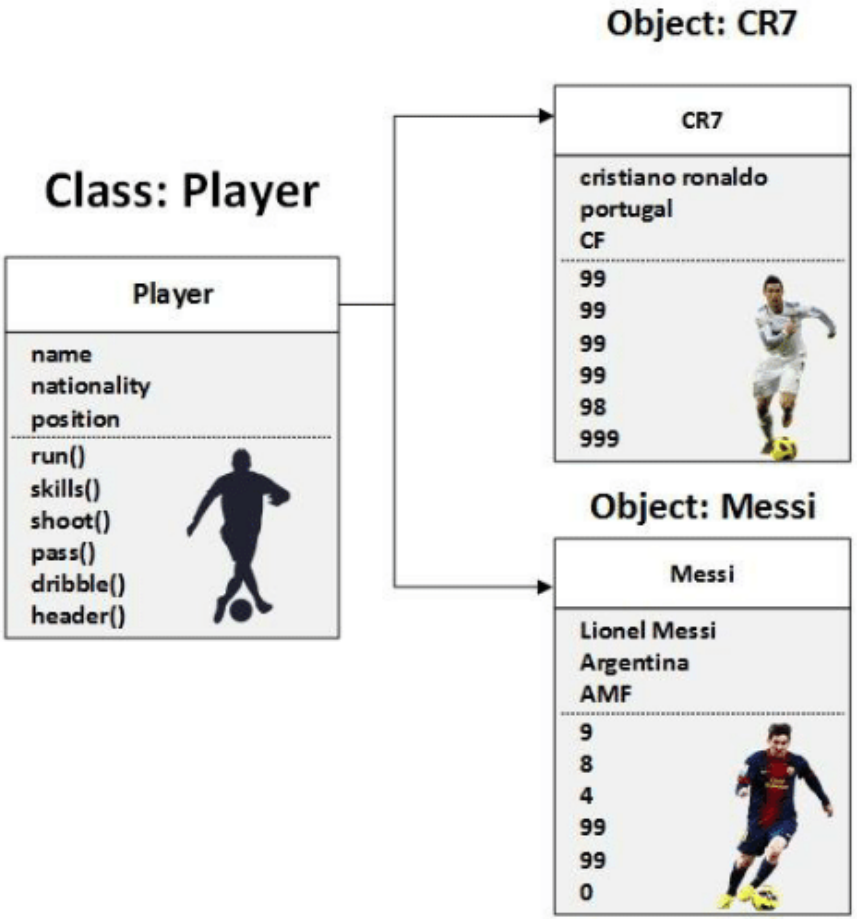

class
Car


objects
Mercedes
Bmw
Audi

# Other Examples



# Example (2)




How to learn OOP using football

# Objects: Explained with an Example

- Suppose we want to develop a data processing system for a university.
- We must keep records on students who attend the university.
- Each student, each unit, etc., will be represented as different sorts of objects.

# Univesity Student Object

- What information would be in a student object?
  - *Name*
  - *Home address*
  - *Residential address (if away from home)*
  - *Units*

# What would the student object do?

- The student object should respond to requests.
- We may want to send out a campus-wide mailing, so we need a campus address for each student.
- We could send the getHomeAddress() to each student object. When the student object receives the message, it responds with the home address.

# Course Object

- Each course might also be represented by an object:
- The Course-object:
  - *Instructor*
  - *Students enrolled*
  - *Pre-requisite courses*
  - *When and where the class meets*

# Objects within Objects

- An object can have one or more objects inside it
- For example, the course-object will have student-objects inside
- Similarly, the course-object may have an instructor-object.

# Sample operations of the Course-object

- `addStudent()`
  - *`Student-object` added to course-object*
- `delStudent()`
- `changeRoom()`
- The point is that different operations are appropriate for objects (like different data-types)

# Objects for Graphics Programming

- Most applications you're familiar with have Graphical User Interfaces (GUI)
- GUI provides windows, icons, buttons and menus (these are also known as objects).
- There's a simple graphics library written specifically to go with your text book.
- Operations using this library will be used to illustrate object-oriented programming in Python

# Aside: Importing Library Functions

- Many Python programmers believe it is tedious to prepend library names in front of library functions, objects, etc,
  - `math.sqrt()`
- Python allows you to import all functions from a module
  - `from math import *`

    *All the functions from this library will be imported and can be used without further qualification.*
  - `sqrt(5)  # rather than math.sqrt(5)`

# Importing Library Functions

- We can also import one function from a library

```
>>> from math import sqrt
>>> sqrt(5)
```

- Problem is that after the import, further down the program, when you see the name of a function you have no idea where it came from.
  - *Can make debugging harder later*
- Better to leave original module name, or create short-hand:

```
>>> import math as
>>> win = m.sqrt(5)
```

# Simple Graphics Programming

- Python provides graphics capabilities through Tkinter.
- Your text book comes with graphics.py library
  - *http://mcsp.wartburg.edu/zelle/python/graphics.py*
  - *Copy on LMS*
- Where to put the library
  - *In the same folder as your other Python programs for this unit*

# Using the graphics.py Library

- We need to import the library first

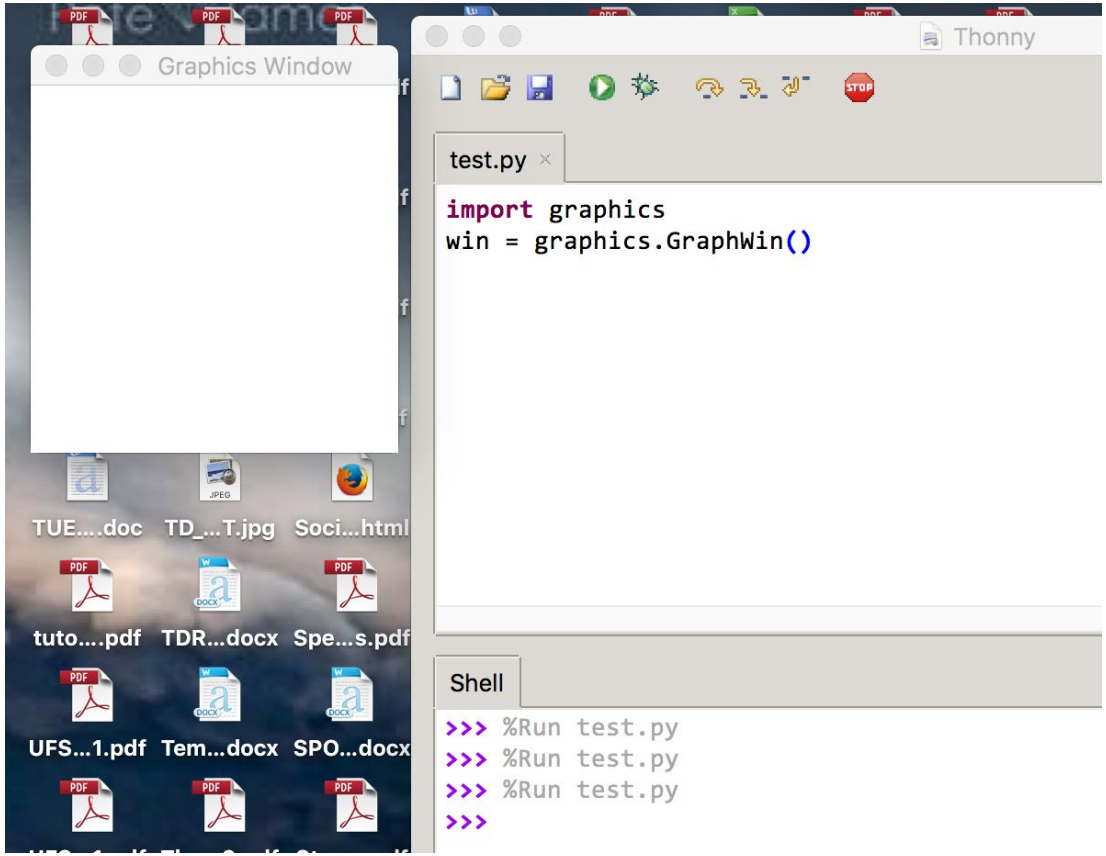
```
>>> import graphics
```

- A graphics window is a place on the screen where the graphics will appear.

```
>>> win = graphics.GraphWin()
```

- This command creates a new window object titled "Graphics Window"

# Using the graphics.py Library



# Graphics and Objects

- `GraphWin()` creates an object which is assigned to the variable *win*.
- We can manipulate the window object through this variable.
  - *Like having* `x = 6` *and then performing integer operations, e.g.* `x *= 7`
- For example, windows can be closed/destroyed by issuing the command

```
>>> win.close()
```

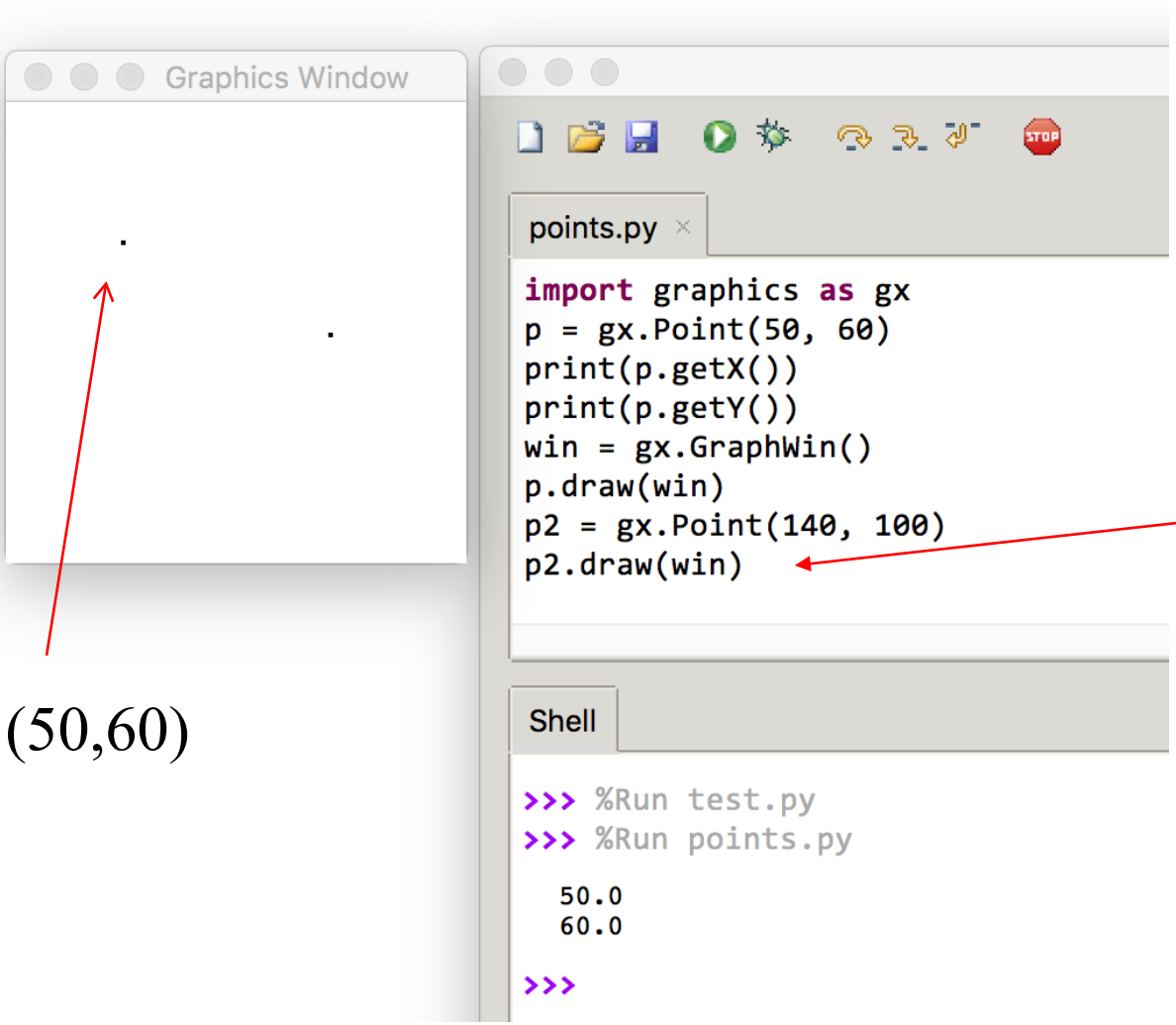
# Graphics Window

- A graphics window is a collection of points called pixels (picture elements).
- The default GraphWin is 200 pixels tall by 200 pixels wide (40,000 pixels total).
- One way to get pictures into the window is one pixel at a time, which would be tedious.
- The graphics library has a number of predefined routines to draw geometric shapes.

# A Point in Graphics

- The simplest object is the `Point`.
- Point locations are represented with a coordinate system (*x*, *y*), where *x* is the horizontal location of the point and *y* is the vertical location.
- The origin (0,0) in a graphics window is the upper left corner.
- X values increase from left to right, y values **from top to bottom**.
- Lower right corner is (199, 199)

# Simple Graphics Commands

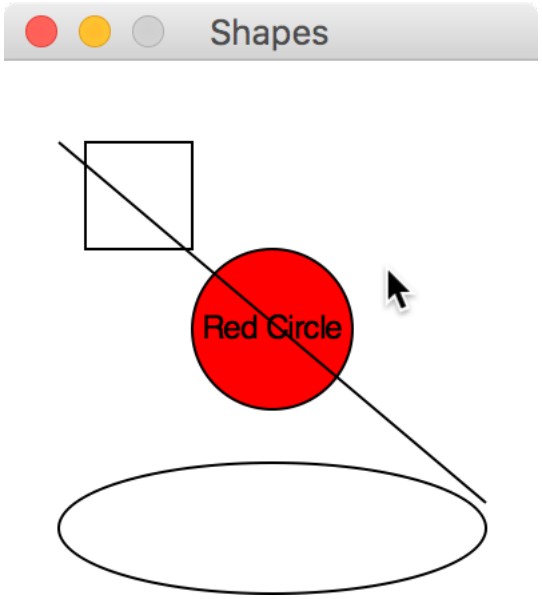



Objects only become visible when the object is drawn in the window

# Drawing Geometric Shapes

```
import graphics as gx

### Open a graphics window
win = gx.GraphWin('Shapes')
### Draw a red circle centered at point (100, 100) with
    radius 30
center = gx.Point(100, 100)
circ = gx.Circle(center, 30)
circ.setFill('red')
circ.draw(win)
### Put a textual label in the center of the circle
label = gx.Text(center, "Red Circle")
label.draw(win)
### Draw a square using a Rectangle object
rect = gx.Rectangle(gx.Point(30, 30), gx.Point(70, 70))
rect.draw(win)
### Draw a line segment using a Line object
line = gx.Line(gx.Point(20, 30), gx.Point(180, 165))
line.draw(win)
### Draw an oval using the Oval object
oval = gx.Oval(gx.Point(20, 150), gx.Point(180, 199))
oval.draw(win)
```



# Using Graphics Objects

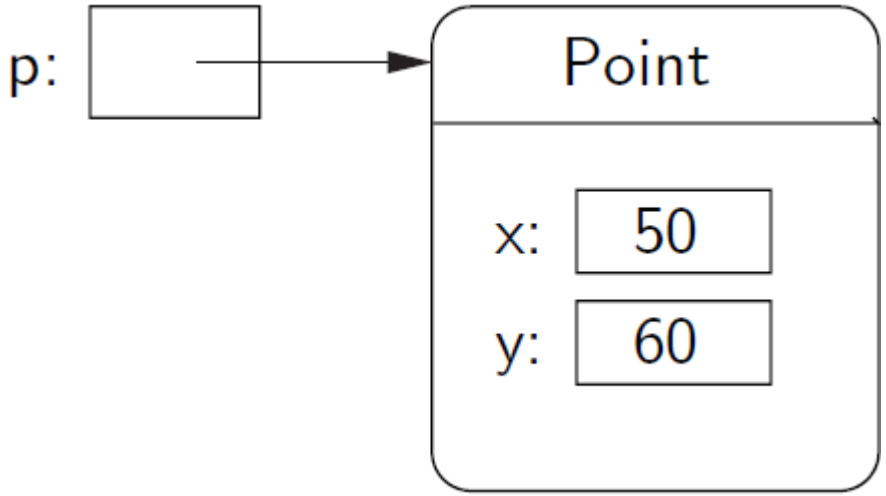
- Computation is preformed by asking an object to carry out one of its operations; “message”.
- In the previous example we manipulated GraphWin, Point, Circle, Oval, Line, Text and Rectangle. These are examples of *classes*.
- Each object is an instance of some class and the class describes the properties of the instance.
  - int, float, str, None *are classes*
- If we say Snoopy is a dog, we mean Snoopy is a specific individual of the class of dogs. Snoopy is an instance of the dog class.

# Creating a New Instance

- To create a new instance of a class, we use a special operation called a *constructor*.
  <class-name>(<param1>, <param2>, …)
- A <class-name> is the name of the class we want to create a new instance of, e.g. `Circle` or `Point`.
- *The parameters are required to initialize the object. For example,* `Point` *requires two numeric values;* `GraphWin` *can, optionally, take a name for the window.*
  - `Point(50, 60)`

# Example of Creating a New Instance

- `p = Point(50, 60)`
- The constructor for the `Point` class requires two parameters, the *x* and *y* coordinates for the point.
- These values are stored as *instance variables* inside of the object.



# Class – Instance - Object

**Class:** Think of it as a “template” or a “blueprint” used to create objects.

**Instance:** A unique copy of a Class representing an Object.

**Object:** An Object is an Instance of a Class. It knows stuff and can do stuff.

# Summary

- We learned some basics of Object Oriented programming
- We learned what are objects and how to use them in our programs
- We learned the difference between classes, instances and objects
- We learned how to write simple graphics programs
- We haven't learned how to define our own classes yet. This will be covered in a few weeks time.