Implementing Classes – Key Concepts

Why are Classes useful?

- A class provides a way to ***bundle/organize* data and actions** under a single name. Data is stored in *instance variables* and actions are specified by *methods*.
- A class specifies a ***type*** of thing, from which we can produce multiple ***instances***. All instances have the same form (i.e., same names for instance variables and methods), but each has its own data.

  As such, a class is like *int* or *float*, but the *programmer* gets to specify the type of data and what can be done with it.

  Functions are important in part because we can call them with different arguments to get different results. Classes extend that idea to include ***objects with different data***, as well as methods that can be called with different arguments.
- Designing solutions to problems by thinking about the *types* of things needed for the solution, and the *relationships* between those types of things, is fundamental to **object-oriented design**.

What is the *syntax* (notation) for implementing a class?

Indicates a ***class*** definition (as opposed to **`def`** for a ***function*** definition)

Name of class – always starts with upper-case letter

What the class "inherits" from – for now, always **`object`** (the most basic class)

```
class Blah(object):

    def __init__(self, whatever, ...):
        self.whatever = whatever
        self.xxx = ...

    def foo(self, ...):
        ... self.whatever ...

    ...
```

The ***constructor*** method. Always named **`__init__`**. Called when an instance of the class is constructed, e.g.: **`b = Blah(...)`**

Instance variables. They always begin with **`self`**.

Other ***methods***. Just like functions, except the first parameter is always **`self`**.
("Class" methods are an exception, but we won't discuss them in this course.)

Why is the `__init__` method special?

```
class Point(object):
    def __init__(self, x, y):
        self.x = x
        self.y = y
        self.number_of_moves = 0
```

Causes the Point class' **`__init__`** method to be called, with:

`x = 100`
`y = 50`
`self =` *the Point being constructed*

Then sets **p1** to that Point.

Causes the Point class' **`__init__`** method to be called, with:

`x = 77`
`y = 33`
`self =` *the Point being constructed*

Then sets **p2** to that Point.

```
def main():
    p1 = Point(100, 50)
    p2 = Point(77, 33)
```

```
class Point(object):
    def __init__(self, x, y):
        self.x = x
        self.y = y
        self.number_of_moves = 0

    def move_by(self, dx, dy):
        self.x = self.x + dx
        self.y = self.y + dy
```

What is *self*?

Answer: It is set to the ***thing before the dot*** when a method of the class is called.

```
def main():
    p1 = Point(100, 50)
    p2 = Point(77, 33)
    p1.move_by(20, 30)
    p2.move_by(11, 22)
    p1.move_by(1, 1)
```

self = p1 in the call to ***__init__***
self = p1 in the call to ***move_by***
self = p1 in the call to ***move_by***

It is the ***same p1*** all 3 times.
So the one and only **p1** is set to (100, 50), then moves to (120, 80), then moves to (121, 81) in this code snippet.

self = p2 in the other two statements.
So the one and only **p2** is set to (77, 33) and moves to (88, 55).

What is *self*.blah? Answer: It is the ***blah* instance variable** for the ***thing before the dot***.

```
class Point(object):
    def __init__(self, x, y):
        self.x = x
        self.y = y
```

```
def main():
    p1 = Point(100, 50)
    p2 = Point(77, 33)
    x1 = p1.x
    x2 = p2.x
```

self = p1
in this call to ***__init__.***

So **self.x = 100**
in ***__init__***
means that
p1.x in ***main***
is set to **100**.

self = p2 in this call to ***__init__.***
So **self.x = 77** in ***__init__***
means that
p2.x in ***main*** is set to **77**.

What is *self*.blah()?

[Note the parentheses]

```
class Point(object):
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def get_distance_from(self, p2):
        ...

    def get_distance_from_start(self):
        start_point = ...
        return self.distance_from(start_point)
```

```
def main():
    p1 = Point(100, 50)
    ...
    d = p1.get_distance_from_start()
```

Answer: It is the ***blah* method** applied to the ***thing before the dot***, and is used to call one method from within another method.