Objects and classes

So, an object is a collection of data with associated behaviors. How do we differentiate between types of objects? Apples and oranges are both objects, but it is a common adage that they cannot be compared. Apples and oranges aren't modeled very often in computer programming, but let's pretend we're doing an inventory application for a fruit farm. To facilitate the example, we can assume that apples go in barrels and oranges go in baskets.

Now, we have four kinds of objects: apples, oranges, baskets, and barrels. In object-oriented programming, the term used for *kind of object* is class. So, in technical terms, we now have four classes of objects.

What's the difference between an object and a class? Classes describe objects. They are like blueprints for creating an object. You might have three oranges sitting on the table in front of you. Each orange is a distinct object, but all three have the attributes and behaviors associated with one class: the general class of oranges.

The relationship between the four classes of objects in our inventory system can be described using a Unified Modeling Language (invariably referred to as UML, because three letter acronyms never go out of style) class diagram. Here is our first class diagram:

Diagram Here Plz

This diagram shows that an Orange is somehow associated with a Basket and that an Apple is also somehow associated with a Barrel. Association is the most basic way for two classes to be related.

Specifying attributes and behaviours

We now have a grasp of some basic object-oriented terminology. Objects are instances of classes that can be associated with each other. An object instance is a specific object with its own set of data and behaviors; a specific orange on the table in front of us is said to be an instance of the general class of oranges. That's simple enough, but what are these data and behaviors that are associated with each object?

Data describes objects

In C++ and Java, things are pretty straight-forward. There are 3 magical and easy to remember access modifiers, that will do the job (public, protected and private). But there's no such a thing in Python.

Protected

Protected member is (in C++ and Java) accessible only from within the class and it's subclasses. How to accomplish this in Python? The answer is – by convention. By prefixing the name of your member with a single underscore, you're telling others "don't touch this, unless you're a subclass". See the example below:

```
class Cup:

    def __init__(self):

        self.color = None

        self._content = None # protected variable

    def fill(self, beverage):

        self._content = beverage

    def empty(self):

        self._content = None

cup = Cup()

cup._content = "tea"
```


Public

All member variables and methods are public by default in Python. So when you want to make your member public, you just do nothing. See the example below:

```
class Cup:

    def __init__(self):
```

```
        self.color = None
        self.content = None
    def fill(self, beverage):
        self.content = beverage
    def empty(self):
        self.content = None
redCup = Cup()
redCup.color = "red"
redCup.content = "tea"
redCup.empty()
redCup.fill("coffee")
```


Private

By declaring your data member private you mean, that nobody should be able to access it from outside the class, i.e. strong you can't touch this policy. Python supports a technique called name mangling. This feature turns every member name prefixed with at least two underscores and suffixed with at most one underscore into _<className><memberName> . So how to make your member private? Let's have a look at the example below:

```
class Cup:
    def __init__(self, color):
        self._color = color    # protected variable
```