Classes

Now, we're going to work with **classes**. We've been creating these variables which are not very structured:

```
customer1 = "John Doe"
pickupLocation1 = "350 5th Ave"
package1 = customer1 + pickupLocation1

customer2 = "Jane Doe"
pickupLocation2 = "100 7th Ave"
package2 = customer2 + pickupLocation2

customer3 = "Joe Daniels"
pickupLocation3 = "11 1st Ave"
package4 = customer3 + pickupLocation3

customerList = [customer1, customer2, customer3]
```

Let's create some object that contains particular attributes like name and address, maybe even the pick up location. So we're going to start off by creating that kind of class:

```
class Customer:
    name = ""
    pickupLocation = ""

    def getPackage(self):
        return self.name + ", " + self.pickupLocation
```

This class has two variables set to empty strings.

'Customer' Class

Let's create an instance of this `Customer` class:

```
cus1 = Customer()
cus1.name = "Joe Blow"
print(cus1.name)
print(cus1.getPackage())
```

Note that we are able to set the name and fetch it later. `self` is implied here.

'Person' Class

Now create a bit more complex class:

```
class Person:
    firstName = ""
    lastName = ""
    streetAddress = ""
```

```
    city = ""
    state = ""
    zipcode = ""

    def __init__(self, fname, lname, streetAddress, city, state, zipcode):
        self.firstName = fname
        self.lastName = lname
        self.streetAddress = streetAddress
        self.city = city
        self.state = state
        self.zipcode = zipcode

    def printName(self):
        print("First name is %s" % (self.firstName))
```

Initialization Function

`__init__` is a special initialization function that gets called automatically upon the creating of an instance of the class.

'self'

Inside the initialization function we prepend all variables with `self` - this way we say that we work with attributes of the class, not with the local variables that won't be visible to the code outside of this function.

'self__customerId = custId'

Now create a new version of the `Customer` class:

```
class Customer (Person):
    name = ""
    pickupLocation = ""

    def __init__(self, name, pickupLocation, custId):
        self.name = name
        self.pickupLocation = pickupLocation
        self.__customerId = custId
        return

    def getPackage(self, delimeter):
        return "%s %s %s. Customer ID = %d" % (self.name, delimeter,self.pickupLocation, self.__customerId)

    #overrides
    #def printName(self):
     #   print("Name is %s" % (self.name))

    #hide data member
    __customerId = 0

cus1 = Customer("Joe Blow", "350 5th Ave", 20)
cus1.name = "Joe Blow"
cus1.pickupLocation = "350 5th Ave"
#print(cus1.getPackage())
print(cus1.getPackage(":"))

#overriding
cus1.printName()
```

Don't forget to comment outour original `Customer` class definition.

This class inherits from the `Person` class. It means that this class inherits all the methods from the parent class. However, if you define a method with the same name in the child class, you effectively override the initial method. For example you may define a new `printName` method and observe the result - you will get a different output.