Inheritance and Polymorphism

Basic Inheritance

We have mentioned before that software re-use is considered one of the golden rules of object-orientated programming, and that generally speaking we will prefer to eliminate duplicated code whenever possible. One mechanism to achieve this is *inheritance*, which means that the attributes and methods of one class are available to another one in such a way the it appears those attributes and methods were declared within the second class. We call the first class, the *parent class* or the *superclass*, and the second class is called the *child class*, or *subclass*. To declare that one class is a subclass of another, when you are declaring that class, just put the name of the superclass as a parameter in the class declaration of the subclass:

```
class Subclass(Superclass):
      <Class Declaration>
# END Class.
```

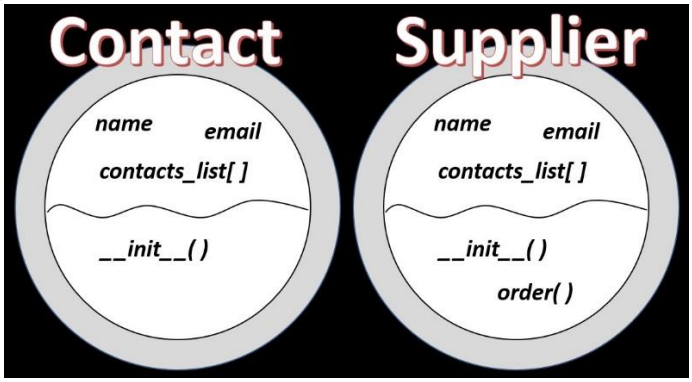
Let's look at an example in practice. Let's say we have a simple address book that keeps track of names and e-mail addresses. We'll call the class Contact, and this class keeps the list of all contacts, and initialises the names and addresses for new contacts:

```
class Contact:
    contacts_list = []
    def __init__(self, name, email):
        self.name = name
        self.email = email
        Contact.contacts_list.append(self)
    # END Init.
# END Class.
```

Now let's say that the contact list is for our company, and some of our contacts are suppliers and some others are other staff members in the company. If we wanted to add an order method, so that we can order supplies off our suppliers, we need to do it in such a way as we cannot try to accidently order supplies off other staff members. So we do:

```
class Supplier(Contact):
   def order(self, order):
       print("The order is for"
          "'{}' to '{}'".format(order,self.name))
   # END order.
# END Class.
```

So as a diagram, this looks like:



Inheritance and Polymorphism

So we can declare objects of the classes:

```
c1 = Contact("Tom StaffMember", "TomStaff@MyCompany.com")
s1 = Supplier("Joe Supplier", "JoeSupplier@Supplies.com")
```

And if order something from our supplier by saying `s1.order("Bag of sweets")` we get the following message back:

```
The order is for 'Bag of sweets' to 'Joe Supplier'
```

But if we tried to order from our contacts by saying `c1.order("Bag of sweets")` we get an error message back:

```
Traceback (most recent call last):
  File "C:/Inheritance.py", line 64, in <module>
    c1.order("Bag of sweets")
AttributeError: 'Contact' object has no attribute 'order'
```

Because the method `order` exists only in the class `Supplier`, not in `Contact`.

Overriding and Super

Overriding means that Python allows a superclass and a subclass to have methods of the same name, and objects of each particular class can use the method associated with that class, by calling it in the normal way.

Super is a function, typically called as `super()`, that allows a subclass to call a method in a superclass, by saying `super().SuperclassMethod`.

Multiple Inheritance

A subclass can inherit for more than one superclass in a very simple way:

```
class Subclass(Superclass1, Superclass2):
      <Class Declaration>
# END Class.
```

In this scenario, when a specific attribute or method is mentioned, the Python interpreter first looks for it in the current class, and if it isn't there the interpreter will check the first superclass for the attribute or method, and if it isn't found there the interpreter will check if that superclass itself has a superclass, if so it will check that one, if not it will move onto the second superclass (this is called a depth-first search).

Polymorphism

Polymorphism means that we can call the same method name with different parameters, and depending on the parameters, it will do different things. For example:

```
>>> print(6 * 5)                    [30]
>>> print("Hello" * 5)               [HelloHelloHelloHelloHello]
```