Lecture 16

OOP is a very powerful and common way of structuring systems.

A class: A template for an Abstract Data Types.

We can treat an Abstract Data type like a primitive data type.

Notion of modularity or abstraction.

We can cluster data and methods.

In the ideal case we would have data hiding – only access the parts using a predefined method – python does not have data hiding.

A class is a template and when we call that class then an instance is called. And inside instances we will have methods and data together we will have attributes.

Methods we had:

* \_\_init\_\_() – creates the instance and makes self point to it
* \_\_cmp\_\_() – compare to instances. if first and second are equal it will return 0. and if first is smaller than second then it will return -1 and if first is more than second then it will return 1. it is not perfectly Boolean function as it can return 3 objects.
* \_\_str\_\_() – to print out instances

Simple simulation of people:

`class person(object)` : says that `person` class will inherit from the inbuilt `object` class

How we can access variables of this instance, other instances and use methods defined before in other methods of the same class.

Super class: Person

Sub class : MITPerson

Sub class can inherit the attributes of MITPerson

Inside a class we can have a local variable that can be incremented inside any method of the class.

Inheritance can help us go up the chain. and if we want to use the methods in the base class then we can use them from classes that have been inherited from that class.

Shadowing/Overriding: Inherited class has a method of the same name and it calls the base class function then it is shadowing the base class method.

* here UG is calling say but before that it is changing somethings. so it overriding/shadowing the method in `person` class

ug > per is like ug.\_\_cmp\_\_(per)

what it does is that it will see the first \_\_cmp\_\_ that it can find. Which is the one that it will find in MITPerson. But this will compare on the basis of idNumber but `per` does not have a idNumber. So it will generate an error. But if the comparison is done other way round:

per>ug is per.\_\_cmp\_\_(ug)

then it will work because the first \_\_cmp\_\_ is in `person`.and is comparing on the basis of familyname and firstname and these two attributes are there in `ug` as well as `per`.

suppose we have an instance that is actually a collection of many other things that are as lists or anything else:

then what we may want to do is:

for p in course6:

print p

but for this we need two special forms:

* \_\_iter\_\_
* next

Whenever we use a `for … in …` structure then what python does is that it returns an iterator. it is keeping track of where we are in the list and how to get to the next thing.

the exception `StopIteration` will tell the for loop that it is done now and break out of the loop.

\_\_iter\_\_ will start the iteration whereas next will help us walk us through the instances

We can create a fairly complex architecture with just basic classes and inheritance.

For classes, we now have:

* encapsulation
  + data hide: but we can’t do it here
* inheritance
* shadow/over ride methods
* hierarchy of classes