

Lecture VI

Objects

- The OOP Concept
- Defining Classes
- Methods
- Instance, Class and Static Members
- Instance Variable
- Privacy
- Inheritance
- Calling Parent Methods

O O P

- Although Python is a multi-paradigm language, object oriented programming is a key component of its model.
- Everything is an object. Everything.
- Objects are simply data structures that map attribute names to values (also objects).
- Unlike many other languages, Python does not distinguish between data members and methods much. A method is simply a data member whose type is a bound function.

Defining Classes

- Classes are defined with the `class` keyword, which opens a new scope and creates a new type. Base class(es) can be specified in the definition after the name between brackets.

```
- class Vehicle:  
    ...  
- class Car(Vehicle):  
    ...  
- class Boat(Vehicle):  
    ...  
- class AmphibianCar(Car, Boat):  
    ...
```

Class Scope

- Note that everything inside of a class block is ordinary Python code.
- You can have variable assignments, if statements, etc.
- The code in this case is evaluated during the definition, not during the usage of a class.

Class Scope

- Example:

- `is_cat = True`
`my_age = 10`

```
class Pet:
    if is_cat:
        def speak(self):
            print "Meow!"
    else:
        def speak(self):
            print "Roar!"

    talk = speak
    age = min(5, my_age)
my_pet = Pet()
print my_pet.age
```

Methods

- Methods are defined in classes the same way they are defined in the global scope, using the `def` keyword.

```
- class Vehicle:
    def __init__(self):
        self.fuel = 0

    def move(self):
        if self.fuel > 0:
            self.fuel -= 1
            print "I'm moving! I'm moooviiiiing!"

    def refuel(self, fuel_amount):
        self.fuel += fuel_amount
```

Methods

- By default, when a method defined inside a class is called on an instance of that class, the instance is passed in the first parameter. The two variants below are identical:
 - `v = Vehicle()`
`v.move()`
 - `v = Vehicle()`
`Vehicle.move(v)`
- The convention is to name the first variable of a class method `self`, though this is not necessary and means nothing to Python itself.

Special Methods

- Python allows you to make your class better integrated into the rest of the code by giving you control of methods called on it by Python itself in special cases.
- Special methods all start and end with two underscores.
- The best known special method is the constructor, called `__init__`.
- Other special methods exist to overload operators, facilitate iteration, etc.

Instance Methods

- By default, methods defined in classes are bound to the class's instances. They are called from instances and receive these instances in their first parameters.
- When you assign an instance method to a variable, its parent instance is "bound" to it.

Example:

```
- v = Vehicle()
  x = v.move
  y = Vehicle.move
  x()           → Calls Vehicle.move(v)
  y()           → Error! Not enough parameters!
```

Class Methods

- In some cases you may want to create method that take the class object itself rather than the instance object as their first parameter. This is done using the `@classmethod` decorator:

```
- class Animal:
    def whoAmI(self):
        print self

    @classmethod
    def whatAmI(self):
        print self
a = Animal()
a.whoAmI()
a.whatAmI()
```

Static Methods

- Sometimes you want to define a method in a class that is treated like a normal function and is not passed any special parameters. This is done using the `@staticmethod` decorator:

```
- class Animal:  
    def speak(self):  
        print '%s says: Hello!' % self
```

```
    @staticmethod  
    def say(self):  
        print self
```

```
a = Animal()
```

```
a.speak()
```

```
a.say()
```

```
parameters!
```

→ Calls `Animal.speak(a)`

→ Error! Not enough

Instance Variables

- A method called from an instance can access the instance's variables only through its first parameter (e.g. `self`). This includes both data attributes and other methods.

```
- class Animal:  
    def __init__(self, my_name):  
        self.name = my_name  
  
    def speak(self):  
        self.say(self.name)  
  
    def say(self):  
        print self
```

Privacy

- As we have mentioned previously, Python does not have real privacy.
- There are conventions that specify that variable starting with a single underscore are private and two underscores as name-mangled.
- This applies to class methods and data attributes the same way it applies to modules.

Inheritance

- Python's inheritance model is very similar to that of other OOP languages, but like C++ it allows inheriting from multiple base classes.
- A child class inherits its parents' attributes, be they data members or methods.
- Classes can have no parents, but in such case it is suggested that they inherit from `object`.
- Parent classes do not need to be defined in the same scope or file - any expression that returns a class object will do.

Inheritance

- Example:

```
- class Animal:  
    def __init__(self, my_name):  
        self.name = my_name
```

```
class Cat(Animal):  
    def speak(self):  
        print '%s says: Meow!' % self.name
```

```
class Dog(Animal):  
    def speak(self):  
        print '%s says: Rruff!' % self.name
```

Inheritance

- Example:

```
- class A:  
    x = 'Hello'  
  
class B:  
    z = 'Goodbye'  
  
class Combined(A, B):  
    y = 'Wow!'  
  
c = Combined()  
print c.x  
print c.y  
print c.z
```


Method Overriding

- Python methods are always overridden by child classes. For those coming from C++, we can say that all Python methods are "virtual".
- When a base class calls a method that has been overridden in by a child class *on an instance of the child class*, the overridden method is called.

Method Overriding

- Example:

```
- class Parent:  
    def greet(self):  
        self.say('Hello')  
  
    def say(self, what):  
        print 'Parent says:', what
```

```
class Child(Parent):  
    def say(self, what):  
        print 'Child says:', what
```

```
p = Parent()
```

```
c = Child()
```

```
p.greet()
```

→ prints "Parent says: Hello"

```
c.greet()
```

→ prints "Child says: Hello"

Calling Base Methods

- Child classes can call methods of their parents even if they override these methods.
- This is done by calling the "unbound" version of the method and manually passing the current instance (`self`) as the first parameter:

```
- class Parent:  
    def greet(self):  
        print 'Hello'  
  
class Child(Parent):  
    def greet(self):  
        Parent.greet(self)  
        print 'How are you?'
```