Information Infrastructure II

INFO I211 – Spring 2014 – Sections 18530 & 22519

Lecture 7 – 2014.02.05 & 2014.02.06

Instructor:
Mitja Hmeljak,
http://mypage.iu.edu/~mitja
mitja@indiana.edu

Lecture 7 – Object Oriented Programming (continues)

Objectives:

- Working with attributes: *object* vs. *class* attributes
- Restrict access to an object's attributes
- (learn to work with both "new-style" and "old-style" Python classes)

Using Attributes

You can have object's attributes automatically created and initialized through constructor

It's a big convenience; done often.

The Attribute Critter Program

```
bash-4.1$ python attribute_critter.py
A new critter has been born!
Hi.  I'm Poochie

A new critter has been born!
Hi.  I'm Randolph

Printing crit1:
Critter object
name: Poochie

Directly accessing crit1.name:
Poochie


Press the enter key to exit.
```

Figure 8.6: Sample run of the Attribute Critter program

Each Critter object has attribute name it uses when it says hi.

Initializing Attributes

```
class Critter(object):
    def __init__(self, name):
        self.name = name
```

self

- First parameter in every instance method
- Automatically receives reference to the object invoking method
- Allows method to get at the object itself to access object's attributes or methods (or even create new attributes, as we are doing here in __init__)

Initializing Attributes (continued)

```
class Critter(object):
    def __init__(self, name):
        self.name = name
...
crit1 = Critter("Poochie")
```

self receives reference to new Critter object

name receives "Poochie"

self.name = name creates the attribute name for this object and sets it to "Poochie"

crit1 gets new Critter object named "Poochie"

Accessing Attributes

```
class Critter(object):
...
    def talk(self):
        print "Hi. I'm", self.name, "\n"
...
crit1.talk()
```

talk() method

- Uses a Critter object's name attribute
- Receives reference to the object itself as self
- Prints Hi. I'm Poochie by accessing attribute name of particular object through self.name

Accessing Attributes (continued)

```
class Critter(object):
    def __init__(self, name):
        self.name = name
...
crit1 = Critter("Poochie")
print crit1.name
```

`print crit1.name` prints string "Poochie"

We can access object attribute outside class with dot notation – but that should be avoided ...

Printing an Object

```
class Critter(object):
...
    def __str__(self):
        rep = "Critter object\n"
        rep += "name: " + self.name + "\n"
        return rep
...
print crit1
```

attribute_critter.py

__str__

- Another special method
- Returns string representation of object

book.py

Using Class Attributes and Static Methods

Class attribute: A single attribute that's associated with a class itself

Static method: A method that's associated with a class itself

Class attribute could be used for number of objects instantiated, for example

How many spaceships are there?

Static methods often work with class attributes

The Classy Critter Program

```
bash-4.1$ python classy_critter.py
Accessing the class attribute Critter.total: 0

Creating critters.
A critter has been born!
A critter has been born!
A critter has been born!

The total number of critters is 3

Accessing the class attribute through an object: 3


Press the enter key to exit.
```

Figure 8.7: Sample run of the Classy Critter program

Total number of objects in class attribute, displayed by static method

Creating a Class Attribute

```
class Critter(object):
    total = 0
```

total = 0 creates class attribute total set to 0

Assignment statement inside the class --but outside any method-- creates a *class attribute*

This assignment statement is executed *only once*, when Python first sees the class definition

Class attributes exist ***even before a single object is created*** ← ←

We can thus use class attributes without any objects of that class in existence

Accessing a Class Attribute

```
class Critter(object):
    total = 0

    def status():
        print "Total critters", Critter.total

    status = staticmethod(status)

    def __init__(self, name):
        Critter.total += 1

print Critter.total
...
print crit1.total
```

Accessing a Class Attribute (continued)

Access class attribute with dot notation - both inside class or out

```
Critter.total += 1    #inside class
print Critter.total  #outside class
```

Can access class attribute through class instance (object of that class)

```
print crit1.total
```

But we can't assign a new value through instance

```
crit1.total += 1  # won't work as might expect
```

Creating a Static Method

```
class Critter(object):
...
    def status():
        print "Total critters", Critter.total

    status = staticmethod(status)   # old-style static method declaration

status()
```

Is a *static* method

Doesn't have self in parameter list because method will be invoked through class not object

Creating a Static Method (continued)

staticmethod()

- Built-in Python function
- Takes method and returns static method

status = staticmethod(status)

- In our example:
- it takes method status() and returns static method
- Assigns static method to status and that name is then used to call the static method

Invoking a Static Method

```
...
crit1 = Critter("critter 1")
crit2 = Critter("critter 2")
crit3 = Critter("critter 3")

Critter.status()

Critter.status()
```

- Invokes static method status() defined in Critter
- Prints a message stating that 3 critters exist
- Works because constructor increments class attribute total, which status() displays

classy_critter.py

Group Task: student.py

Design a *Python class* to retain information about students:

- Each *object* should contain these attributes:

  name

  GPA

  bursar balance

  credit hours (in current semester)

- The *Python class* should contain this attribute:

  total number of students

Write a *method* so that printing a student object will display all of its attributes

In your main Python program, *instantiate* two students and print them