# Information Infrastructure II

**INFO I211 – Spring 2014 – Sections 18530 & 22519**

***Lecture 6 – 2014.02.03 & 2014.02.04***

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

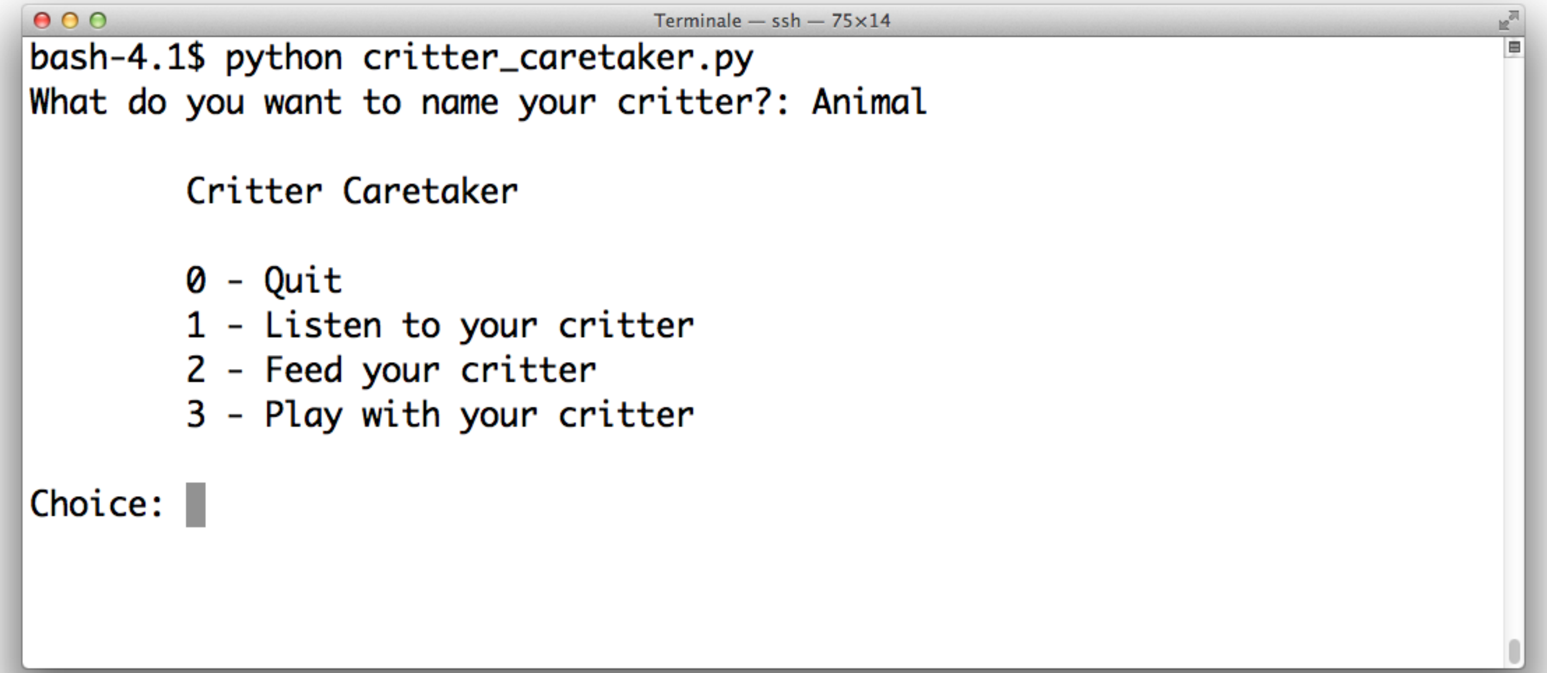
# Lecture 6 – Object Oriented Programming

Objectives:

- *Create classes* to define objects
- *Write methods* and create attributes for objects
- *Instantiate objects* from classes

- learn about *self* in Python

## Book Project: The Critter Caretaker Program

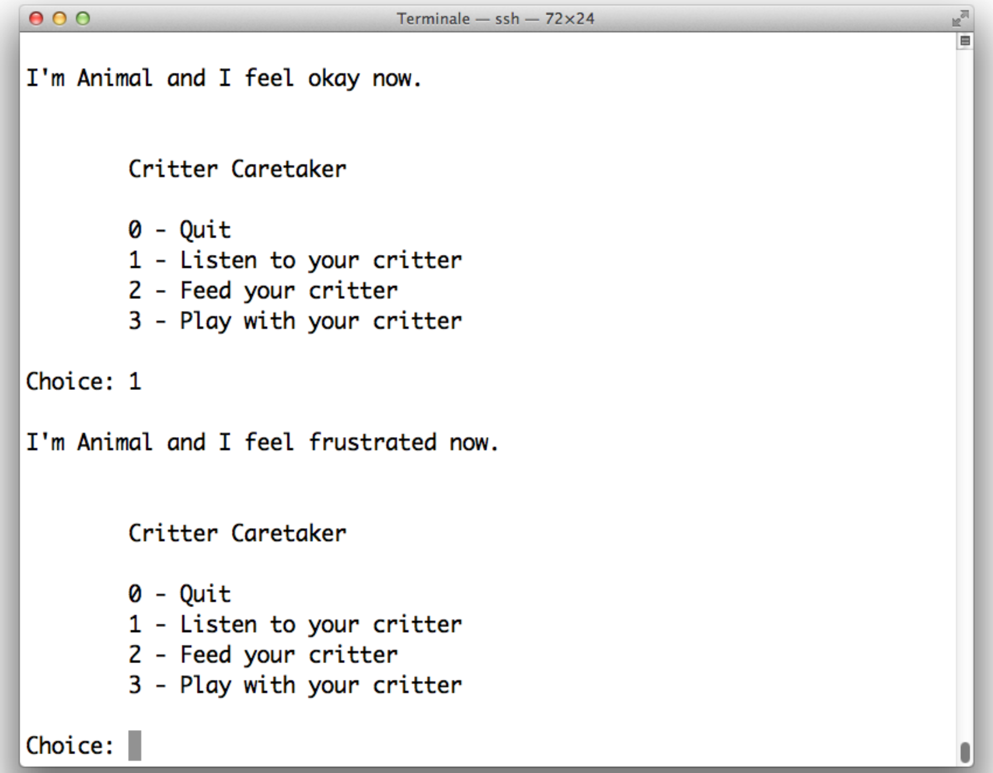


```
bash-4.1$ python critter_caretaker.py
What do you want to name your critter?: Animal

        Critter Caretaker

        0 - Quit
        1 - Listen to your critter
        2 - Feed your critter
        3 - Play with your critter

Choice: 
```

Figure 8.1: Sample run of the Critter Caretaker program

You get to name your very own critter.

## Chapter Project: The Critter Caretaker Program

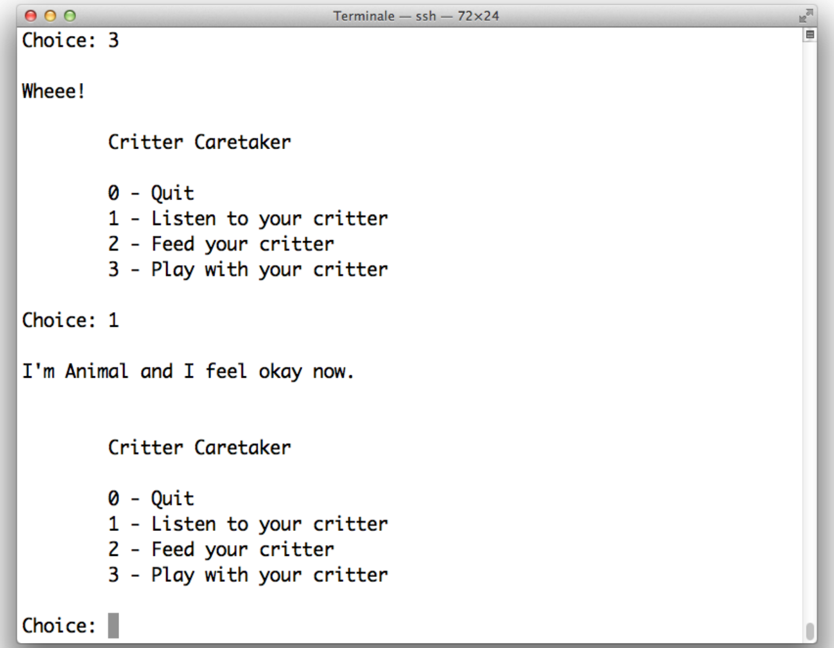


```
I'm Animal and I feel okay now.


        Critter Caretaker

        0 - Quit
        1 - Listen to your critter
        2 - Feed your critter
        3 - Play with your critter

Choice: 1

I'm Animal and I feel frustrated now.


        Critter Caretaker

        0 - Quit
        1 - Listen to your critter
        2 - Feed your critter
        3 - Play with your critter

Choice: 
```

Figure 8.2: Sample run of the Critter Caretaker program

If you neglect your critter, it will have a mood change for the worse.

## Chapter Project: The Critter Caretaker Program



```
Choice: 3

Wheee!

        Critter Caretaker

        0 - Quit
        1 - Listen to your critter
        2 - Feed your critter
        3 - Play with your critter

Choice: 1

I'm Animal and I feel okay now.


        Critter Caretaker

        0 - Quit
        1 - Listen to your critter
        2 - Feed your critter
        3 - Play with your critter

Choice: 
```

Figure 8.3: Sample run of the Critter Caretaker program

With the proper care, your critter will return to its sunny mood.

## Understanding Object-Oriented Basics

Object-oriented Programming (OOP): A methodology of programming where software objects are used to represent data and actions

where new types of objects are defined

**Object:** A single software unit that combines attributes (data) and methods (actions)

**Attribute:** A "characteristic" of an object; like a variable associated with a kind of object

## Understanding Object-Oriented Basics (continued)

**Method:** A "behavior" of an object; like a function associated with a kind of object

**Instance:** A single object

**Instantiate:** To create an object

**Class:** Code that defines the attributes and methods of a kind of object

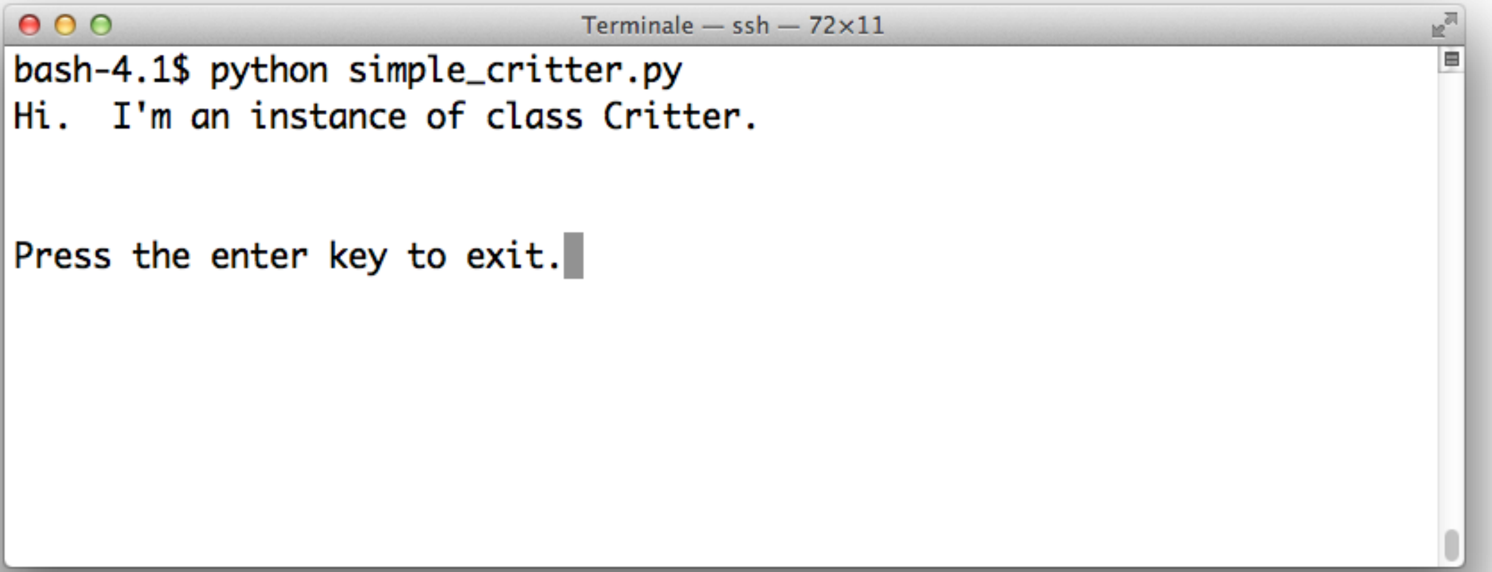
## Creating Classes, Methods, and Objects

OOP allows representation of real-life objects as software objects

e.g. Spacecraft *objects*

- *Attribute*: Energy level
- *Method*: Fire weapons
- Each object has similar structure (energy level and fire weapons) but each has unique values (one might have energy level of 3, another energy level of 10)

## The Simple Critter Program



```
bash-4.1$ python simple_critter.py
Hi.  I'm an instance of class Critter.


Press the enter key to exit.
```

Figure 8.4: Sample run of the Simple Critter program

The Critter object's talk() method makes the critter greet the world.

## The Simple Critter Program (continued)

```
class Critter(object):
    """A virtual pet"""
    def talk(self):
        print "Hi. I'm an instance of class Critter."

# main
crit = Critter()
crit.talk()
```

## Defining a Class

```
class Critter(object):
   """A virtual pet"""
```

class - a Python keyword

the, the Class name should begin with a capital letter

Critter

Parentheses contain the class's parent. In this case, the class is based on object, a Python fundamental built-in type

Docstring, describes kind of objects

"""A virtual pet"""

## Defining a Method

```
def talk(self):
    print "Hi. I'm an instance of class Critter."
```

- you *define* a method like a function
  - When you define it "inside" a Class, it is a method
- Every *instance method* must have a special first parameter, called *self* by convention
  - It's provided by Python, not to be written when calling the method
- self = Special first parameter provides way for a method to refer to object itself
  - *Q: Am I Ship A, with an energy level of 10, or am I Ship B, with an energy level of 3?*
  - A: Consult self!

## Instantiating an Object

crit = Critter()

Create new object of the specified class by using the class name followed by set of parentheses

- Critter() creates new object of class Critter

Can assign a newly instantiated object to a variable of any name

- crit = Critter() assigns new Critter object to crit

Avoid using variable that's same name as the class name in lowercase letters.

## Invoking a Method

crit.talk()

Every Critter object has a talk() method

crit.talk() invokes the talk() method of the Critter object crit

Prints string "Hi. I'm an instance of class Critter."

simple_critter.py

## Using Constructors

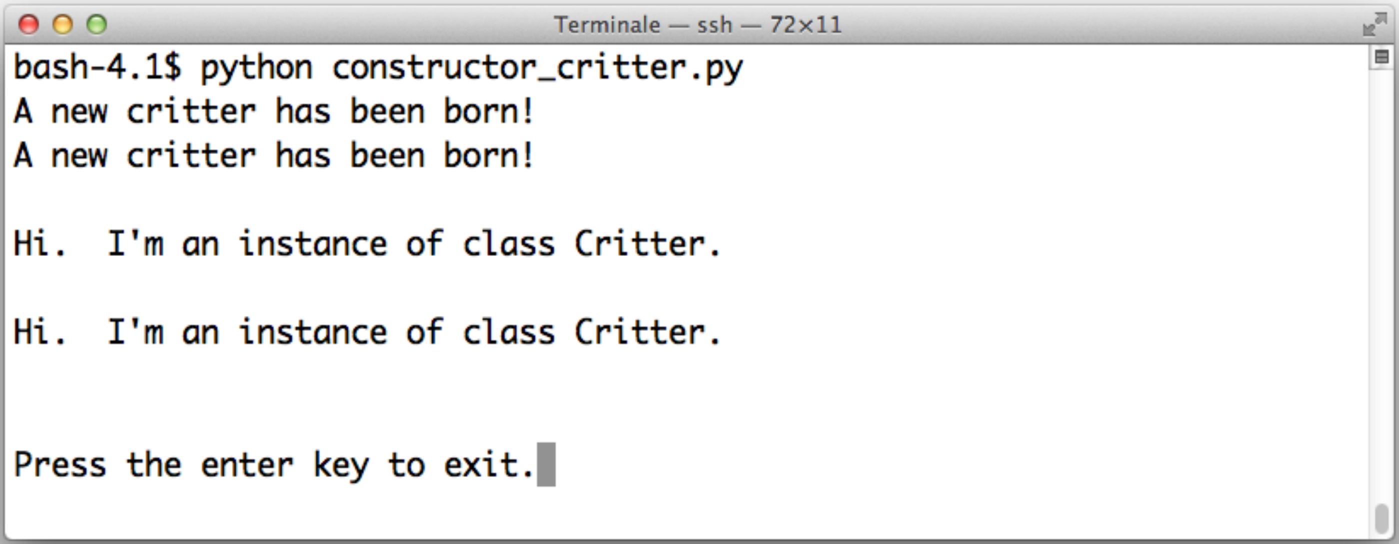
**Constructor:** A special method that is automatically invoked every time a new object is instantiated

Usually write one in each class

Usually sets up the *initial attribute values* of new object

- You might give a spaceship 10 units of energy to start with, that it then uses up by flying around and getting shot at.

## The Constructor Critter Program



```
bash-4.1$ python constructor_critter.py
A new critter has been born!
A new critter has been born!

Hi.  I'm an instance of class Critter.

Hi.  I'm an instance of class Critter.


Press the enter key to exit.
```

Figure 8.5: Sample run of the Constructor Critter program
Two separate critters are created. Each says hi.

## Creating a Constructor

```
def __init__(self):
    print "A new critter has been born!"
```

New Critter object automatically announces itself to world

`__init__`

- Is special method name
- Automatically called by each new Critter object

## Creating Multiple Objects

```
crit1 = Critter()
crit2 = Critter()
```

Creating multiple objects is easy

Two objects created here

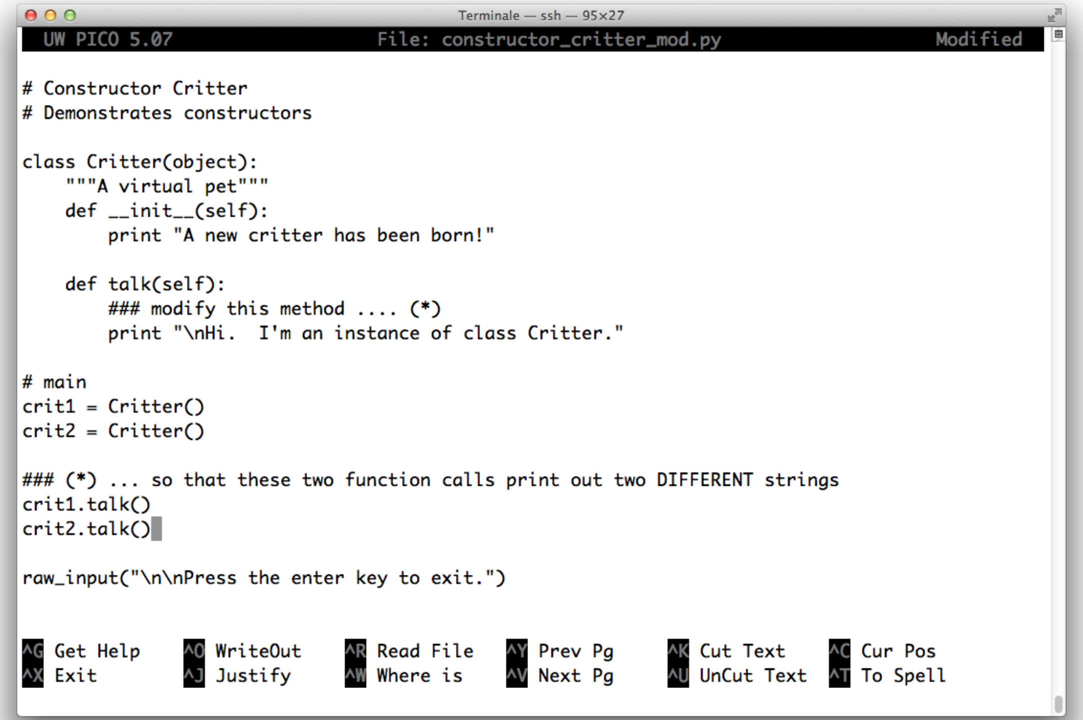
Each object is independent, full-fledged critter

constructor_critter.py

## Group Task

Instructions:

follow the comments marked with (*) to complete the task.



```
  UW PICO 5.07              File: constructor_critter_mod.py                  Modified

# Constructor Critter
# Demonstrates constructors

class Critter(object):
    """A virtual pet"""
    def __init__(self):
        print "A new critter has been born!"

    def talk(self):
        ### modify this method .... (*)
        print "\nHi.  I'm an instance of class Critter."

# main
crit1 = Critter()
crit2 = Critter()

### (*) ... so that these two function calls print out two DIFFERENT strings
crit1.talk()
crit2.talk()

raw_input("\n\nPress the enter key to exit.")


^G Get Help  ^O WriteOut  ^R Read File  ^Y Prev Pg   ^K Cut Text   ^C Cur Pos
^X Exit      ^J Justify   ^W Where is   ^V Next Pg   ^U UnCut Text ^T To Spell
```

**Hint 1:**

print in Python can print *anything*

**Hint 1:**

self is *unique* to each Python object