Classes

Python for Ecologists

Tom Purucker, Tao Hong, Chance Pascale

Ecological Society of America Workshop
Portland, OR

purucker.tom@gmail.com

August 4, 2012

Notes

Python objects

- Everything in Python is an object with these properties
  1. an identity (id)
  2. a type (type)
  3. a value (mutable or immutable)

Notes

Each Python object has an id

```
>>> n_predators = 12
>>> id(n_predators)
4298191056
```

Notes

Each Python object has a type

```
>>> n_predators = 12
>>> type(n_predators)
<type 'int'>
```

Notes

Each Python object has a value

- String, integer, and tuple object values are *immutable*

```
>>> n_prey = 88
>>> id(n_prey)
4298193184
>>> n_prey = 96
>>> id(n_prey)
4298192992 # id for n_prey has changed
```

- Dictionary and list items are *mutable*

```
>>> birds = ["cardinal", "oriole"]
>>> id(birds)
4332756000
>>> birds.append("gnatcatcher")
>>> id(birds)
4332756000 # id is still the same
```

Notes

Classes

- Classes consist of
  - collections of data structures
  - collections of methods (functions)
- Class methods typically operate on the data structures of the class
- Class users then call methods and do not have to manipulate the data

Notes

self variable

- A class instance refers to itself as 'self'
- All methods require self as the first argument/parameter inside the class
- But users of the class do not include it in calls to the methods
- All data and methods calls are preceded by self within the class (e.g., self.age() or self.find_integral(some arguments...)

Notes

Creating a class

- object is the base class
- dunder init is a constructor
- all methods take self as the first argument/parameter

Notes

Code for creating a class

```
#create the Rabbit class, starts with 10 hit points
class Rabbit(object):
  def __init__(self, name):
    self.name = name
    self.hit_points = 10

  def hop(self):
    self.hit_points = self.hit_points − 1
    print "%s␣hops␣one␣node,␣now␣has␣%i␣hit␣points."
   % (self.name, self.hit_points)

  def eat_carrot(self):
    self.hit_points = self.hit_points + 3
    print "%s␣munches␣a␣carrot,␣now␣has␣%i␣hit␣points."
   % (self.name, self.hit_points)
```

Notes

Code to create some rabbits

- We can now create objects of Rabbit class and give them names

```
#create some Rabbits
were = Rabbit("Were−Rabbit")
harvey = Rabbit("Harvey␣Rabbit")
jessica = Rabbit("Jessica␣Rabbit")
dir(jessica)
```

Notes

Code to create some rabbits

- We can now create objects of Rabbit class and give them names

```
#create some Rabbits
were = Rabbit("Were−Rabbit")
harvey = Rabbit("Harvey␣Rabbit")
jessica = Rabbit("Jessica␣Rabbit")
dir(jessica)
```

Notes

Call the methods of the created rabbits

- We can now create objects of Rabbit class and give them names

```
#Rabbits hop around and eat carrots
were.hop()
jessica.eat_carrot()
harvey.hop()
jessica.hop()
were.eat_carrot()
```

Notes

Create a frog subclass

- Subclasses can inherit the data and methods of the original class and extend them

```
#Create a Frog class that extends the rabbit class
class Frog(Rabbit):
    # create a new croak method
    def croak(self):
        self.hit_points = self.hit_points − 1
        print "%s␣croaks,␣now␣has␣%i␣hit_points."
        % (self.name, self.hit_points)
   # override the eat_carrot method
    def eat_carrot(self):
        print "%s␣cannot␣eat␣a␣carrot,␣it␣is␣too␣big!."
        % (self.name)
   # create an eat_fly method
    def eat_fly(self):
        self.hit_points = self.hit_points + 2
        print "%s␣eats␣a␣fly,␣now␣has␣%i␣hit_points."
        % (self.name, self.hit_points)
```

Notes

Create Frog objects and call its methods

```
## Create a frog
frogger = Frog("Frogger")
# Do frog stuff
frogger.croak()
frogger.eat_carrot()
frogger.eat_fly()
frogger.hop()
```

Notes

Notes

Notes