Lecture 7:
Classes and Object Oriented Programming, Part 2

FAES BIOF 309 Introduction to Python

Christopher Coletta

lecture outline

- review of classes
- methods = functions with an extra first argument
- instance variables vs class variables
- review of functionality by implementing object hooks
- functionality by inheritance

Why classes again?

	A	B	C	D	E
1	Last Name	First Name	email	hw grades	final proj grade
2	Busby	Ben	ben.busby@gmail.com	89,78,90,76	98
3	Coletta	Christopher	me@chriscoletta.com	12,34,56,54	89
4	Shirley	Matthew	matt@mattshirley.com	100,97,98,99	100

Using list of tuples:

```
In [1]: my_data = [ ( "Busby", "Ben", "ben.busby@gmail.com", [89,78,90,76], 98 ), \
   ...:             ( "Coletta", "Christopher", "me@chriscoletta.com", [12,34,56,54], 89 ), \
   ...:             ( "Shirley", "Matthew", "matt@mattshirley.com", [100,97,98,99], 100 ) ]

In [2]:

In [2]: my_data[0][4]
Out[2]: 98
```

Using list of instances of class `Student`:

```
In [1]: from Student import Student

In [2]: my_data = [ Student( "Busby Ben ben.busby@gmail.com 89 78 90 76 98" ),\
                    Student( "Coletta Christopher me@chriscoletta.com 12 34 56 54 89" ), \
                    Student( "Shirley Matthew matt@shirley.com 100 97 98 99 100 " ) ]

In [3]: my_data[0].final_proj
Out[3]: 98.0
```

Why classes again?

- Help manage data complexity
  - e.g. nested information
- Help manage data navigation
- Help shield programs that use the data from changes in the data's representation

A basic class example*

```
# !!!!!!!!!!!!!!!WARNING!!!!!!!!!!!!!!!!!!!!!
# THE FOLLOWING IS PSEUDOCODE AND DOES NOT
# REPRESENT PROPER PYTHON SYNTAX!!!!!!!!!!!!!

class ExampleClass( object ):
  """This is the class docstring"""

  # Some Fundamental Methods:

  def __init__( self, ... ):
    """Initialization Method"""
    # initialize new instance's fields
    # no return
    self.valueA = ...
    self.valueB = ...
    self._private_value = ...

  def __repr__( self ):
    """The String used by the interpreter to print instances"""
    return ...

  def __str__( self ):
    """string used by print and str()"""
    return ...

  # Predicate Methods:
  # Methods that return only True or False

  def __lt__( self, other_instance ):
    """The method called by sorted() and list.sort() when
       no key arument is provided."""
    if type( self ) != type( other_instance ):
      raise TypeError( 'Incompatible argument to __lt__: ' +
          str( other_instance ) )
    return self.get_something() < other_instance.get_something()
```

* Model, M., 2010. *Bioinformatics Programming Using Python*

A basic class example, con't*

```
37  # !!!!!!!!!!!!!!!WARNING!!!!!!!!!!!!!!!!!!!!!
38  # THE FOLLOWING IS PSEUDOCODE AND DOES NOT
39  # REPRESENT PROPER PYTHON SYNTAX!!!!!!!!!!!!!
40
41  # More Predicate (returns only True or False) Methods:
42  def is_some_characteristic( self ):
43    return # True or False
44
45  def isPassing( self ):
46    if self.final_grade >= 60:
47      return True
48    return False
49
50  # Access Methods
51  def get_something( self ):
52    return # value obtained by one of:
53               # accessing a field
54               # lookup by key
55               # computation
56               # filtering a collection
57               # searching a collection
58
59  # Modification Methods
60  def set_something( self, ... ):
61    """Change the value of one or more fields passed on the
62    parameters supplied in the call. Generally no return value"""
63
64  # Action Methods
65  def do_something( self, ... ):
66    """Do something that has effects outside the class"""
67
68  # Private Support Methods
69  def _helper_method( self ):
70    """Something used by other methods of the class only"""
```

* Model, M., 2010. *Bioinformatics Programming Using Python*

Constructor problem

- One should be able to create an instance of class Student from different types of inputs
  - last time: a string of values
    - This was because we were reading in lines of input from a file
  - want to add: tuple of values
- Question: Can __init__(…) take different arguments?

`@classmethod` decorator

- Can be used on any method
- The decorated method will have the class type ("`cls`") passed as its first argument instead of the instance on which the method was called ("`self`").

```
class Cheer( object ):

  def __init__( self, what_to_say ):
    self.the_cheer = what_to_say

  @classmethod
  def NewHoorayCheer( cls ):
    return cls( "Hooray!" )

  @classmethod
  def NewYipeeCheer( cls ):
    return cls( "Yipee!" )
```

Instance attributes vs. Class attributes

- Thus far we have only used instance methods (functions) and instance attributes (data fields)
- Some fields and methods should be associated with the class itself rather than individual instances.*
  - e.g., generating unique ID number for each instance, or keeping track of all class instances
- Class attributes go in the class namespace

Inheritance

- Inheritance describes a relationship between two types, or classes, of objects in which one is said to be a *subtype* or *child* of the other. The child inherits features of the parent, allowing for shared functionality.
- One class will get most or all of its features from a parent class
- Common functionality goes in the parent class, specialized functionality goes in the base class
- Read this: http://learnpythonthehardway.org/book/ex44.html

Inheritance Considerations

- Composition vs. Inheritance
  - "Is-A" vs. "Has-A"
- Three ways that the parent and child classes can interact:*
  - Actions on the child imply an action on the parent.
  - Actions on the child override the action on the parent.
  - Actions on the child alter the action on the parent.
- The use of `super()`

*from *Learn Python the Hard Way*

Extend built-in Python types via inheritance

- Example: extend list object to contain methods returning only the odds or only the evens.

Abstract base class

- A class that describes how its children classes will behave, but leaves the implementation details up to the children.
- Design by contract:
  - Any object that inherits from an abstract base class is required to implement
- Examples
  - class mammal, method speak
  - class shape, method area
- Opposite of Abstract class = Concrete class

To review:

(from learncodethehardway.com)

- class : Tell Python to make a new kind of thing.
- object : Two meanings: the most basic kind of thing, and any instance of some thing.
- instance : What you get when you tell Python to create a class.
- def : How you define a method (function) inside a class.
- self : The first value passed to a method will be a reference to the object on which the method was called.
- inheritance : The concept that one class can inherit traits from another class, much like you and your parents.
- composition : The concept that a class can be composed of other classes as parts, much like how a car has wheels.
- attribute : A property classes have that are from composition and are usually variables.
- is-a : A phrase to say that something inherits from another, as in a Salmon is-a Fish.
- has-a : A phrase to say that something is composed of other things or has a trait, as in a Salmon has-a mouth.