Classes and Objects

Basics

Copyright © Software Carpentry 2010
This work is licensed under the Creative Commons Attribution License
See http://software-carpentry.org/license.html for more information.
Two basic concepts in OOP are \textit{class} and \textit{object}
Two basic concepts in OOP are **class** and **object**.

A class defines the behavior of a new kind of thing.
Two basic concepts in OOP are class and object
A class defines the behavior of a new kind of thing
An object is a thing with particular properties
Two basic concepts in OOP are class and object
A class defines the behavior of a new kind of thing
An object is a thing with particular properties

<table>
<thead>
<tr>
<th></th>
<th>Biology</th>
<th>Programming</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specific</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Two basic concepts in OOP are *class* and *object*

A class defines the behavior of a new kind of thing

An object is a thing with particular properties

<table>
<thead>
<tr>
<th>General</th>
<th>Biology</th>
<th>Programming</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><em>canis lupus</em></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Specific</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Organism Waya</td>
<td></td>
</tr>
</tbody>
</table>
Two basic concepts in OOP are **class** and **object**

A class defines the behavior of a new kind of thing

An object is a thing with particular properties

<table>
<thead>
<tr>
<th>General</th>
<th>Biology</th>
<th>Programming</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specific</td>
<td><strong>Species</strong> <em>canis lupus</em></td>
<td><strong>Class</strong> Vector</td>
</tr>
<tr>
<td></td>
<td><strong>Organism</strong> Waya</td>
<td><strong>Object</strong> velocity</td>
</tr>
</tbody>
</table>
Define a new class with no behavior
Define a new class with no behavior

```python
>>> class Empty(object):
...     pass
```
Define a new class with no behavior

```python
>>> class Empty(object):
...     pass
```

Create two objects of that class
Define a new class with no behavior

```python
>>> class Empty(object):
...     pass
```

Create two objects of that class

```python
>>> first = Empty()
>>> second = Empty()
```
Define a new class with no behavior

```python
>>> class Empty(object):
    ...
    pass
```

Create two objects of that class

```python
>>> first = Empty()
>>> second = Empty()
>>> print 'first is', id(first)
35855140
>>> print 'second is', id(second)
35855152
```
Contents of memory

Empty

first

second
Contents of memory

- object
- Empty
- first
- second
Define the class's behavior with *methods*
Define the class's behavior with *methods*

A function defined inside a classé
Define the class's behavior with *methods*

A function defined inside a class that is called for an object of that class
Define the class's behavior with *methods*

A function defined inside a class that is called for an object of that class

class Greeter(object):
    def greet(self, name):
        print 'hello', name, '!'
Define the class's behavior with *methods*

A function defined inside a classé that is called for an object of that class

class Greeter(object):
    def greet(self, name):
        print 'hello', name, '!'
Define the class's behavior with *methods*

A function defined inside a class that is called for an object of that class

```python
class Greeter(object):
    def greet(self, name):
        print 'hello', name, '!
```
Define the class's behavior with *methods*

A function defined inside a classé that is called for an object of that class

```python
class Greeter(object):
    def greet(self, name):
        print 'hello', name, '!' 

g = Greeter()
g.greet('Waya')

hello Waya !
```
Define the class's behavior with *methods*
A function defined inside a class is called for an object of that class

```python
class Greeter(object):
    def greet(self, name):
        print 'hello', name, '!

g = Greeter()
g.greet('Waya')
Hello Waya!
```
Define the class's behavior with *methods*

A function defined inside a classé that is called for an object of that class

class Greeter(object):
    def greet(self, name):
        print 'hello', name, '!

g = Greeter()
g.greet('Waya')

*hello Waya!*
Define the class's behavior with *methods*

A function defined inside a class that is called for an object of that class

```python
class Greeter(object):
    def greet(self, name):
        print 'hello', name, '!

g = Greeter()
g.greet('Waya')
hello Waya !
```
Contents of memory

stack  heap

```
greet
```

```
Greeter
```

g
Contents of memory

```
stack
  self
  name

heap
  g
  Greeter
  greet
  'Waya'
```

Classes and Objects
Basics
Every object has its own variables
Every object has its own variables — members
Every object has its own variables
Create new ones by assigning them values
Every object has its own variables
Create new ones by assigning them values

class Empty(object):
    pass

e = Empty()
e.value = 123
print e.value
123
Every object has its own variables
Create new ones by assigning them values

class Empty(object):
    pass

e = Empty()
e.value = 123
print e.value

e2 = Empty()
print e2.value

AttributeError: 'Empty'

'Empty'

123

object has no attribute

'value'
The values of member variables customize objects
The values of member variables customize objects

Use them in methods
The values of member variables customize objects

Use them in methods

class Greeter(object):
    def greet(self, name):
        print self.hello, name, '!'
The values of member variables customize objects

Use them in methods

class Greeter(object):
    def greet(self, name):
        print self.hello, name, '!'
Every object has its own variables
Create new ones by assigning them values

class Greeter(object):
    def greet(self, name):
        print self.hello, name, '!

g = Greeter()
Every object has its own variables
Create new ones by assigning them values

class Greeter(object):
    def greet(self, name):
        print self.hello, name, '!

g = Greeter()
g.hello = 'Bonjour'
Every object has its own variables
Create new ones by assigning them values

class Greeter(object):
    def greet(self, name):
        print self.hello, name, '!

g = Greeter()
g.hello = 'Bonjour'
g.greet('Waya')

Bonjour Waya !
Every object has its own variables
Create new ones by assigning them values

```python
class Greeter(object):
    def greet(self, name):
        print self.hello, name, '!

g = Greeter()
g2 = Greeter()
g.hello = 'Bonjour'
g2.hello = 'Salut'
g.greet('Waya')
g2.greet('Waya')

Bonjour Waya !
Salut Waya !
```
Contents of memory

stack

heap

Greeter

hello

'Bonjour'

greet
Contents of memory

```
self
name
'Waya'

stack

Greeter

hello

heap

self
name
'Bonjour'
g

Greeter

greet
```
Every object's names are separate
Every object's names are separate

class Greeter(object):
    def greet(self, name):
        print self.hello, name, '!

hello = 'Hola'
g = Greeter()
g.hello = 'Bonjour'
g.greet('Waya')

Bonjour Waya !
Creating objects and *then* giving them members is error-prone
Creating objects and *then* giving them members is error-prone.

Might forget some (especially when making changes)
Creating objects and then giving them members is error-prone. Might forget some (especially when making changes). Any code repeated in two or more places is...
Creating objects and *then* giving them members is error-prone

Might forget some (especially when making changes)

Any code repeated in two or more places

Define a *constructor* for the class
Creating objects and *then* giving them members is error-prone.

Might forget some (especially when making changes).

Any code repeated in two or more places.

Define a *constructor* for the class.

Automatically called as new object is being created.
Creating objects and *then* giving them members is error-prone

Might forget some (especially when making changes)

Any code repeated in two or more places

Define a *constructor* for the class

Automatically called as new object is being created

A natural place to customize individual objects
Creating objects and *then* giving them members is error-prone. Might forget some (especially when making changes). Any code repeated in two or more places is error-prone. Define a *constructor* for the class. Automatically called as new object is being created. A natural place to customize individual objects. Python uses the special name `__init__(self, ...)`.
A better Greeter

class Greeter(object):

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

    def greet(self, name):
        print self.hello, name, '!'
Why it's better

```python
first = Greeter('Hello')
first.greet('Waya')
Hello Waya !
```
Why it's better

```python
first = Greeter('Hello')
first.greet('Waya')
Hello Waya !
second = Greeter('Bonjour')
second.greet('Waya')
Bonjour Waya !
```
Contents of memory

Classes and Objects

Basics
A common mistake

class Greeter(object):

    def __init__(self, what_to_say):
        hello = what_to_say

    def greet(self, name):
        print self.hello, name, '!'
What goes wrong

first = Greeter('Hello')
What goes wrong

first = Greeter('Hello')
first.greet('Wayya')

Attribute Error: 'Greeter' object has no attribute 'hello'
What goes wrong

```python
first = Greeter('Hello')
first.greet('Wayu')
```

*Attribute Error: 'Greeter' object has no attribute 'hello'*

```python
self.name stores the value in the object
```
What goes wrong

```python
first = Greeter('Hello')
first.greet('Waya')
```

*Attribute Error: 'Greeter' object has no attribute 'hello'*

`self.name` stores the value in the object `name` on its own is a local variable on the stack
What goes wrong

first = Greeter('Hello')
first.greet('Waya')

*Attribute Error: 'Greeter' object has no attribute 'hello'*

self.name stores the value in the object name on its own is a local variable on the stack

class Greeter(object):
    def __init__(self, what_to_say):
        hello = what_to_say
Object data is not protected or hidden in Python
Object data is not protected or hidden in Python

```python
first = Greeter('Hello')
first.greet('Waya')
Hello Waya !
first.hello = 'Kaixo'
Kaixo Waya !
```
Object data is not protected or hidden in Python

```python
first = Greeter('Hello')
first.greet('Waya')
Hello Waya!
first.hello = 'Kaixo'
Kaixo Waya!
```

Some languages prevent this
Object data is not protected or hidden in Python

```python
first = Greeter('Hello')
first.greet('Waya')
Hello Waya !
first.hello = 'Kaixo'
Kaixo Waya !

Some languages prevent this
All discourage it
```
A more practical example

class Rectangle(object):

    def __init__(self, x0, y0, x1, y1):
        assert x0 < x1, 'Non-positive X extent'
        assert y0 < y1, 'Non-positive Y extent'
        self.x0 = x0
        self.y0 = y0
        self.x1 = x1
        self.y1 = y1
A more practical example

```python
class Rectangle(object):
    def __init__(self, x0, y0, x1, y1):
        assert x0 < x1, 'Non-positive X extent'
        assert y0 < y1, 'Non-positive Y extent'
        self.x0 = x0
        self.y0 = y0
        self.x1 = x1
        self.y1 = y1
```
Benefit #1: fail early, fail often
Benefit #1: fail early, fail often

# Programmer thinks rectangles are written
# [[x0, x1], [y0, y1]]
>>> field = [[50, 100], [0, 200]]
Benefit #1: fail early, fail often

# Programmer thinks rectangles are written
# [[x0, x1], [y0, y1]]

```python
>>> field = [[[50, 100], [0, 200]]
```

# Class knows rectangles are (x0, y0, x1, y1)

```python
>>> field = Rectangle(50, 100, 0, 200)
```

AssertionError: non-positive X extent
Benefit #2: readability

class Rectangle(object):
    ...

    def area(self):
        return (self.x1-self.x0)*(self.y1-self.y0)

    def contains(self, x, y):
        return (self.x0 <= x <= self.x1) and (self.y0 <= y <= self.y1)
## Compare

<table>
<thead>
<tr>
<th>List of Lists</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>field = <code>[[0, 0], [100, 100]]</code></td>
<td>field = <code>Rectangle(0, 0, 100, 100)</code></td>
</tr>
<tr>
<td><code>rect_area(field)</code></td>
<td><code>field.area()</code></td>
</tr>
<tr>
<td><code>rect_contains(field, 20, 25)</code></td>
<td><code>field.contains(20, 25)</code></td>
</tr>
</tbody>
</table>
## Compare

<table>
<thead>
<tr>
<th>List of Lists</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>field = [[0, 0], [100, 100]]</td>
<td>field = Rectangle(0, 0, 100, 100)</td>
</tr>
<tr>
<td>rect_area(field)</td>
<td>field.area()</td>
</tr>
<tr>
<td>rect_contains(field, 20, 25)</td>
<td>field.contains(20, 25)</td>
</tr>
</tbody>
</table>

Make it even clearer by creating a `Point2D` class
## Compare

<table>
<thead>
<tr>
<th>List of Lists</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>field = [[0, 0], [100, 100]]</td>
<td>field = Rectangle(0, 0, 100, 100)</td>
</tr>
<tr>
<td>rect_area(field)</td>
<td>field.area()</td>
</tr>
<tr>
<td>rect_contains(field, 20, 25)</td>
<td>field.contains(20, 25)</td>
</tr>
</tbody>
</table>

Make it even clearer by creating a **Point2D class**

Then re-defining **Rectangle in terms of it**