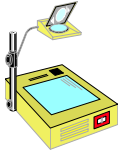




## Python: Classes



1

## Classes: unit of encapsulation

- s/b an abstraction of an easily understood concept or entity
- Reflect real objects in your app domain
- Customization through inheritance: general/special
- Composition: teamwork
- Operator overloading

2

## basics

- “class” creates a class object and assigns a name
- Assignments inside class statements make data attributes
- Def statements inside class generate methods
- Instances are generated from classes
- Assignments to attributes of self change data in the instance, not the class

3

## Example of class attributes

```
class First:
    def setData(self, val):
        self.data = val
    def display(self):
        print self.data

x = First()
y = First()
x.setData("Hello World")
y.setData(3.14)

x.display()
y.display()
```

4

## Global attributes

```
data = "cat"
class First:
    def setData(self, val):
        data = val
    def display(self):
        print data

x = First()
y = First()
x.setData("Hello World")
y.setData(3.14)

x.display()
y.display()
```

5

## constructors

```
class First:
    def __init__(self, d): self.data = d
    def setData(self, val): self.data = val
    def display(self):
        print self.data

class Second(First):
    def display(self):
        print "current value: %s" % self.data

x = Second(77)
x.display()
x.setData(99)
x.display()
```

6

## Operator overloading

- Can overload operators for built-in types: add, slice, print, etc
- Allows programmers to write natural code
- Permits tighter integration with Python's object model
- Interfaces s/b easier to learn
- Can overload most built-in operators
- Method names such as `__X__` are hooks

7

## Example of op overloading

```
class Imaginary:
    def __init__(self, r, i):
        self.re = r
        self.im = i
    def __add__(self, rhs):
        return Imaginary(self.re+rhs.re, self.im+rhs.im)
    def display(self):
        print self.re, ",", self.im

x = Imaginary(3, 5)
y = Imaginary(1, 2)
z = x + y
z.display()
```

8

Method	Overloads	example
<code>__init__</code>	Constructor	Object create
<code>__repr__</code>	Printing, conv	Print x
<code>__del__</code>	Destructor	
<code>__add__</code>	'+'	X + Y
<code>__or__</code>	' '	X   Y
<code>__call__</code>	Function calls	X()
<code>__getitem__</code>	Indexing	X[key]
<code>__setitem__</code>	Index assign	X[key] = val
<code>__len__</code>	Length	
<code>__cmp__</code>	Comparison	x==y, x<y

9

## Overloading print

```
class Imaginary:
    def __init__(self, r, i):
        self.re = r
        self.im = i
    def __add__(self, rhs):
        return Imaginary(self.re+rhs.re, self.im+rhs.im)
    def __repr__(self):
        return str(self.re)+" , "+str(self.im)

x = Imaginary(3, 5)
y = Imaginary(1, 2)
z = x + y
print z
```

10

## The "class" statement

- In C++, the word "class" is a declaration
- In Python, it's an object builder and an implicit assignment
- When executed, it generates a class object and stores a ref to it
- "class" is a compound statement
- Super classes are listed in parens
- Classes are namespaces, i.e., local scope

11

## Inheritance

```
class First:
    def setData(self, val):
        self.data = val
    def display(self):
        print self.data

class Second(First):
    def display(self):
        print "current value: '%s'" % self.data

x = Second()
x.setData(99)
x.display()
```

12