

Artificial Intelligence Programming Objects in Python

Chris Brooks

Department of Computer Science
University of San Francisco

3-2: List Comprehensions

- Python has a great deal of support for *functional programming*
 - Can do it by hand using map, filter, and lambda.
 - Also a built-in construct called a *list comprehension*
 - List comprehensions are a convenient way to *map* a function onto a sequence.

```
[x**2 for x in [1,2,3,4]]
phoneBook = {'brooks':'422-5221',
             'wolber':'422-1234','parr':'422-3435'}
['name: %s phone: %' % (nm, ph) for nm, ph in
 phoneBook.items()]
[s.strip() for s in [' hello ', ' in ', ' there ']]
```

3-3: List comprehensions

- Some cool list comprehension tricks:

```
params = {"server":"mpilgrim", "database":"master", "uid":"sa",  
         "pwd":"secret"}  
[k for k, v in params.items()]  
[(v,k) for k,v in params.items()]  
["%s=%s" % (k, v) for k, v in params.items()]  
[x + y / 2 for (x,y) in zip([1,2,3,4], [5,6,7,8])]  
thenewlist = [x for x in theoldlist if x > 5]
```

3-4: Function Arguments

- Python functions can have optional and named arguments

```
def example_fun(arg1, arg2='cats', arg3='dogs') :  
    print arg1, arg2, ' and ', arg3  
example_fun('I love')  
I love cats and dogs  
example_fun("I love", arg2='fish')  
I love fish and dogs  
example_fun('I love', 'fish', 'chips')  
I love fish and chips  
example_fun('I love', arg3='birds')  
I love cats and birds
```

- This makes it easy to implement multiple versions of a function.

3-5: Functions as Objects

- In Python, functions are first-class objects
 - Can be assigned to variables, passed as arguments, evaluated.
 - This allows us to create higher-order functions

```
def cube(x) : return x * x * x
def my_map(list, fn) :
    list2 = []
    for item in list :
        list2.append(fn(item))
    return list2
```

3-6: Introspection

- All Python objects are capable of introspection
 - This may look familiar to you if you've used Java's reflection API
- `type(obj)`
- `dir(obj)`
- `obj.__doc__`

3-7: Python Classes

- Classes are defined with the 'class' keyword:

```
class Mammal :  
    ...
```

- This defines a class with no parent class.

3-8: __init__

- `__init__` is the first method called when an object is created.
 - Technically, it's not a constructor, but close enough
- Takes at least one argument: `self`
- This is a pointer to the object

3-9: `__init__`

```
class Mammal :  
    "Documentation for the mammal class"  
    def __init__(self, type='') :  
        self.type = type
```

- Notice def is indented one level
- Notice init can use default arguments. (this is how to implement multiple 'constructors')

3-10: Instance Variables

- Instance variables *must* be referenced with `self`
- Created when a value is assigned to them
 - No separate class definition/header file
 - This can lead to confusion
 - Convention: set up all instance variables inside `__init__`
- Data members are public

3-11: Methods

- Defined with def
- Inside the scope of the class definition.
- self is first argument in declaration
 - Not provided when calling the method - the Python interpreter fills this in.
- Can use default and keyword arguments

3-12: Methods

```
class Mammal :
    "Documentation for the mammal class"
    def __init__(self, type='') :
        self.type = type

    def sayName(self) :
        print 'I am a ', self.type
    ### this is a builtin method that controls how an object is printed
    ### return the string indicating how the object should be displayed
    def __repr__(self) :
        return self.type
```

3-13: Methods

- built-in operators can be *overloaded* to provide polymorphism.
 - repr - controls how an object is printed.
 - lt, gt, le, ge, cmp : comparison operators
 - add, sub, mul, div : arithmetic operators

3-14: Class variables

- Class variables are declared outside the scope of a method
- Referenced with the class name
- Useful for constant values, counters, mutex/semaphores, etc.
- Places where you don't want to create an object
 - `string.letters`, `string.ascii_lowercase`, etc

3-15: Inheritance

- Like all OO languages, Python supports inheritance.
- Include the name of the parent class in parentheses.
- To call a parent class' method, use the name of the parent class.

3-16: Inheritance

```
class Flyer :
    "Class for things that can fly"
    def __init__(self, type='') :
        self.type=type
    def canFly(self) :
        return True

class Bird(Flyer) :
    def init(self, type='') :
        Flyer.__init__()
        self.type = type
    def sayName(self) :
        print "I am a " + self.type
```

3-17: Multiple Inheritance

- Python also supports multiple inheritance
- List multiple classes in declaration
- Names are resolved from left to right

3-18: Multiple Inheritance

```
class Bat (Mammal, Flyer) :
    def __init__(self) :
        self.type = 'bat'
    def sayName(self) :
        Mammal.sayName(self)
        print "I am not a vampire, though"
```

3-19: Comments

- Always need to use `self` to refer to member variables
- All variables are public
 - can use `__spam` to indicate private variables
- Classes can tell you what methods they implement with `dir()`
- this is called *introspection* (reflection in Java)