Object Oriented Programming (part 2)

Inheritance

In [64]:
```
class Animal(object):
    def __init__(self, name):
        self._name = name
    def say(self, message):
        print '%s the animal says %s' % (self._name, message)
    def get_number_of_legs(self):
        raise NotImplementedError, 'get_number_of_legs'

class Cat(Animal):
    def __init__(self, name='Felix'):
        Animal.__init__(self, name)
    def say(self, message):
        print '%s the cat meows %s' % (self._name, message)
    def get_number_of_legs(self):
        return 4

class Dog(Animal):
    def __init__(self, name='Fido'):
        super(Dog, self).__init__(name)
    def say(self, message):
        print '%s the dog barks %s' % (self._name, message)
    def get_number_of_legs(self):
        return 4

class Monkey(Animal):
    def __init__(self, name='George'):
        Animal.__init__(self, name)
    def say(self, message):
        print '%s the monkey says %s' % (self._name, message)
    def get_number_of_legs(self):
        return 2
```

In [65]:
```
animal = Animal('Generic')
animal.say('hello')
```

```
Generic the animal says hello
```

In [66]:
```
print animal.get_number_of_legs()
```

```
---------------------------------------------------------------------------
NotImplementedError                       Traceback (most recent call last)
/vagrant/<ipython-input-66-3623e2c96566> in <module>()
----> 1 print animal.get_number_of_legs()

/vagrant/<ipython-input-64-798984796887> in get_number_of_legs(self)
      5         print '%s the animal says %s' % (self._name, message)
      6     def get_number_of_legs(self):
----> 7         raise NotImplementedError, 'get_number_of_legs'
      8
      9 class Cat(Animal):

NotImplementedError: get_number_of_legs
```

In [67]:
```
animal = Cat()
animal.say('hello')
```

```
Felix the cat meows hello
```

In [68]:
```
animal = Dog()
animal.say('hello')
print animal.get_number_of_legs()
```

```
Fido the dog barks hello
4
```

In [69]:
```
animal = Monkey()
animal.say('I have %s legs' % animal.get_number_of_legs())
```

```
George the monkey says I have 2 legs
```

In [70]:
```
isinstance(animal, Monkey)
```

Out[70]: True

In [71]:
```
isinstance(animal, Cat)
```

Out[71]: False

In [72]:
```
isinstance(animal, Animal)
```

Out[72]: True

In [73]:
```
issubclass(Cat, Animal)
```

Out[73]: True

In [74]:
```
class MonkeyDog(Monkey, Dog):
    pass

x = MonkeyDog('What is this thing?!')
print x.say('hello?')
```

```
What is this thing?! the monkey says hello?
None
```

In [75]:
```
print MonkeyDog.mro()
```

```
[<class '__main__.MonkeyDog'>, <class '__main__.Monkey'>, <class '__main__.Dog'>, <class '__main
```

Magic methods

In [76]:
```
print animal
```

```
<__main__.Monkey object at 0x2847590>
```

In [77]:
```
print str(animal)
```

```
<__main__.Monkey object at 0x2847590>
```

In [78]:
```
class Animal(object):
    def __init__(self, name):
        self._name = name
    def __str__(self):
        return '<Animal %s>' % self._name
```

In [79]:
```
animal = Animal('generic')
print animal
```

```
<Animal generic>
```

In [80]:
```
class Animal(object):
    def __init__(self, name):
        self._name = name
    def __str__(self):
        return '<Animal %s>' % self._name
    def __repr__(self):
        return 'Animal(%r)' % self._name
```

In [81]:
```
Animal('with repr')
```

Out[81]:
```
Animal('with repr')
```

In [82]:
```
print Animal('with repr')
```

```
<Animal with repr>
```

Override attribute access

In [83]:
```
class MyClass(object):
    def __init__(self):
        self.a = 'avalue'
    def __getattr__(self, name):
        print 'Trying to get %s' % name
        return None

x = MyClass()
print x.a
print x.unknown_attribute
```

```
avalue
Trying to get unknown_attribute
None
```

In [84]:
```
class MyClass(object):
    a = 0
    def __setattr__(self, name, value):
        print 'Set %s <= %s' % (name, value)

x = MyClass()
x.a = 'avalue'
print 'x.a is still %s' % x.a
```

```
Set a <= avalue
x.a is still 0
```

In [85]:
```
class MyClass(object):
    def __init__(self):
        self.a = 'avalue'
    def __getattribute__(self, name):
        print 'Trying to get %s' % name
        return None

x = MyClass()
print x.a
print x.unkown_attribute
```

```
Trying to get a
None
Trying to get unkown_attribute
None
```

Override Container Methods

In [86]:
```
class DefaultDict(object):
    def __init__(self, default):
        self._data = {}
        self._default = default
    def __getitem__(self, key):
        try:
            return self._data[key]
        except KeyError:
            return self._default()
    def __setitem__(self, key, value):
        self._data[key] = value
    def __delitem__(self, key):
        del self._data[key]
    def __contains__(self, key):
        return key in self._data
    def __repr__(self):
        return '<DefaultDict %r>' % self._data

mydict = DefaultDict(lambda:5)
mydict[1] = 1
mydict[2] = 2
print mydict
print 2 in mydict
```

```
<DefaultDict {1: 1, 2: 2}>
True
```

In [87]:
```
print mydict[5]
print mydict
```

```
5
<DefaultDict {1: 1, 2: 2}>
```

Other Magic Methods

- Comparison override (`__lt__`, `__gt__`, `__le__`, `__ge__`, `__eq__`, `__ne__`)
- Emulating numeric types (`__add__`, `__sub__`, etc.)
- ... more ... (full list at http://docs.python.org/reference/datamodel.html#special-method-names)

Exercises

- Update your phone directory to support looking up a number using the `[]` operator
- Create two phone directories, one which throws exceptions when looking up phone numbers, and a subclass that always returns the same number for unknown phone numbers.