

Threads

- Si conocen los thread de java o c# la firma de estos son parecidos.
- Podes ejecutar una función en un thread distinto o hacerlo *claseado*.
- Python no es thread safe... el GIL complica las cosas.
- Como buenos thread comparten memoria entre ellos.
- así que:

Como usar Threads 1

Como usar Threads 2

No user threads No user threads No user threads No user threads No user threads No
user threads No user threads No user threads No user threads No user threads No
threads No user threads No user threads No user threads No user threads No user
threads No user threads No user threads No user threads No user threads No user
threads No user threads No user threads No user threads No user threads No user
threads No user threads No user threads No user threads No user threads No user
threads No user threads

Como usar Threads 4

Como usar Threads con funciones

```
import threading

def coso():
    for i in range(10):
        print i

def cosa():
    for i in range(10):
        print i

t0 = threading.Thread(target=coso)
t1 = threading.Thread(target=cosa)
t0.start(); t1.start()
```

Threads (con classes)

```
import threading

class MyThread(threading.Thread):

    def run(self):
        for i in range(10):
            print i

t0 = MyThread()
t1 = MyThread()

t0.start(); t1.start()
```

Más Info

- http://users-cs.au.dk/chili/CSS/SlidesPowerPoint_windows/python_19.ppt
- `threading.RLock` <<< locks
- Los procesos son mejor:

```
import multiprocessing

class MyProcess(multiprocessing.Process):

    def run(self):
        print "otro proceso"

p = MyProcess()
p.start()
```


Threads (con Django)

- No es buena idea hacer servicios con threads en django. (aun así hay ejercicios de eso)
- Mejor usen cron. (web2py trae un cron multiplataforma)
- Si lo hacen de toda manera se suele arrancar los threads en el `urls.py` de cada aplicación.

Threads (ejercicio)

Escribir un servicio con un thread en un modulo interno a la aplicación polls implementado con una clase que se llame `Limpiador` que sea un thread que deshabilite todos los `Polls` que superen la cantidad de tiempo que se configure en la variable del `settings.py` `POLLS_TTL`.

Entendiendo Decoradores en Python

Todo en Python es un objeto

- Identidad
- Tipo
- Valor

Objetos

```
>>> a = 1
>>> id(a)
145217376
>>> a.__add__(2)
3
```

Otros objetos:

```
[1, 2, 3]    # listas
5.2         # flotantes
"hola"      # strings
```

Funciones

Las funciones también son objetos.

```
def saludo():  
    print "hola"
```

```
>>> id(saludo)  
3068236156L  
>>> saludo.__name__  
'saludo'  
>>> dice_hola = saludo  
>>> dice_hola()  
hola
```

Decorador (definición no estricta)

Un decorador es una *función* **d** que recibe como parámetro otra *función* **a** y retorna una nueva *función* **r**.

- d: función decoradora
- a: función a decorar
- r: función decorada

$$a = d(a)$$

Código

```
def d(a):  
    def r(*args, **kwargs):  
        # comportamiento previo a la ejecución de a  
        a(*args, **kwargs)  
        # comportamiento posterior a la ejecución de a  
    return r
```

Código

```
def d(a):  
    def r(*args, **kwargs):  
        print "Inicio ejecucion de", a.__name__  
        a(*args, **kwargs)  
        print "Fin ejecucion de", a.__name__  
    return r
```


Manipulando funciones

```
def suma(a, b):  
    print a + b
```

```
>>> suma(1,2)  
3  
>>> suma2 = d(suma)  
>>> suma2(1,2)  
Inicio ejecucion de suma  
3  
Fin ejecucion de suma  
>>> suma = d(suma)  
>>> suma(1, 2)  
Inicio ejecucion de suma  
3  
Fin ejecucion de suma
```

Azúcar sintáctica

A partir de Python 2.4 se incorporó la notación con @ para los decoradores de funciones.

```
def suma(a, b):  
    return a + b
```

```
suma = d(suma)
```

```
@d  
def suma(a, b):  
    return a + b
```

Atención

Antiejemlo: el decorador malvado.

```
def malvado(f):  
    return False
```

```
>>> @malvado  
... def algo():  
...     return 42  
...  
>>> algo  
False  
>>> algo()  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
TypeError: 'bool' object is not callable
```

Decoradores en cadenas

Similar al concepto matemático de componer funciones.

```
@registrar_uso
@medir_tiempo_ejecucion
def mi_funcion(algunos, argumentos):
    # cuerpo de la funcion
```

```
def mi_funcion(algunos, argumentos):
    # cuerpo de la funcion
```

```
mi_funcion = registrar_uso(medir_tiempo_ejecucion(mi_funcion))
```

Decoradores con parámetros

- Permiten tener decoradores más flexibles.
- Ejemplo: un decorador que fuerce el tipo de retorno de una función.

```
@to_string  
def count():  
    return 42
```

```
>>> count()  
'42'
```

Decoradores con parámetros

Primera aproximación.

```
def to_string(f):  
    def inner(*args, **kwargs):  
        return str(f(*args, **kwargs))  
    return inner
```

Decoradores con parámetros

Algo más genérico?

```
@typer(str)
def c():
    return 42

@typer(int)
def edad():
    return 25.5
```

```
>>> edad()
25
```

Decoradores con parámetros

typer es una fábrica de decoradores.

```
def typer(t):  
    def _typer(f):  
        def inner(*args, **kwargs):  
            r = f(*args, **kwargs)  
            return t(r)  
        return inner  
    return _typer
```


Clases decoradoras

- Decoradores con estado.
- Código mejor organizado.

```
class Decorador(object):  
  
    def __init__(self, a):  
        self.variable = None  
        self.a = a  
  
    def __call__(self, *args, **kwargs):  
        # comportamiento previo a la ejecución de a  
        self.a(*args, **kwargs)  
        # comportamiento posterior a la ejecución de a
```

Clases decoradoras

```
@Decorator
def nueva_funcion(algunos, parametros):
    # cuerpo de la funcion
```

- Se instancia un objeto del tipo Decorador con nueva_función como argumento.
- Cuando llamamos a nueva_funcion se ejecuta el método `__call__` del objeto instanciado.

```
def nueva_funcion(algunos, parametros):
    # cuerpo de la funcion
nueva_funcion = Decorador(nueva_funcion)
```

Decorador (definición más estricta)

Un decorador es una *callable* **d** que recibe como parámetro un *objeto* **a** y retorna un nuevo objeto **r** (por lo general del mismo tipo que el original o con su misma interfaz).

- **d**: objeto de un tipo que defina el método `__call__`
- **a**: cualquier objeto
- **r**: objeto decorado

```
a = d(a)
```

Decorar clases (Python >= 2.6)

Identidad:

```
def identidad(C):  
    return C
```

```
>>> @identidad  
... class A(object):  
...     pass  
...  
>>> A()  
<__main__.A object at 0xb7d0db2c>
```

Decorar clases (Python >= 2.6)

Cambiar totalmente una clase:

```
def abuse(C):  
    return "hola"
```

```
>>> @abuse  
... class A(object):  
...     pass  
...  
>>> A()  
Traceback (most recent call last):  
  File "", line 1, in  
TypeError: 'str' object is not callable  
>>> A  
'hola'
```

Decorar clases (Python >= 2.6)

Reemplazar con una nueva clase:

```
def reemplazar_con_X(C):  
    class X():  
        pass  
    return X
```

```
>>> @reemplazar_con_X  
... class MiClase():  
...     pass  
...  
>>> MiClase  
<class __main__.X at 0xb78d7cbc>
```

Decorar clases (Python >= 2.6)

Instancia:

```
def instanciar(C):  
    return C()
```

```
>>> @instanciar  
... class MiClase():  
...     pass  
...  
>>> MiClase  
<__main__.MiClase instance at 0xb7d0db2c>
```

Dónde encontramos decoradores?

Permisos en Django

```
@login_required
def my_view(request):
    ...
```

URL routing en Bottle

```
@route('/')
def index():
    return 'Hello World!'
```

Standard library

```
classmethod, staticmethod, property
```


Muchas gracias!



Datos y contacto

- Original <http://www.juanjoconti.com.ar>
- Twitter: @jjconti

- <http://www.juanjoconti.com.ar/categoria/aprendiendo-python/>
- <http://www.juanjoconti.com.ar/2008/07/11/decoradores-en-python-i/>
- <http://www.juanjoconti.com.ar/2009/07/16/decoradores-en-python-ii/>
- <http://www.juanjoconti.com.ar/2009/12/30/decoradores-en-python-iii/>
- http://www.juanjoconti.com.ar/2010/08/07/functools-update_wrapper/

Ejercicio Thread

Crear un modulo en la app django de ejemplo llamado `decorators.py` que contenga un decorador para las vistas llamado `only_allowed_ips` que solo permita recibir peticiones de una lista de ips definidas en la variable `ALLOWED_IPS` en el `settings.py`. Además puede recibir un parámetro opcional llamado *allow* que reemplaza a las lista de ip autorizadas. En todo caso cualquier petición no permitida responderá con un http code 403.

Ejemplo

```
# settings.py
```

```
ALLOWED_IPS = ["localhost", "127.0.0.1"]
```

```
# polls/views.py
```

```
from polls import decorators
```

```
@decorator.only_allowed_ips()
```

```
def my_view():
```

```
    pass
```

```
@decorator.only_allowed_ips(allowed=["192.168.1.1"])
```

```
def my_view_2():
```

```
    pass
```

OS y otros servicios de File System

Que tiene os adentro

Implementación portable de servicios del sistema operativo

- all functions from posix, nt, os2, or ce, e.g. unlink, stat, etc.
- os.path is one of the modules posixpath, or ntpath
- os.name is 'posix', 'nt', 'os2', 'ce' or 'riscos'
- os.curdir is a string representing the current directory ('.' or ':')
- os.pardir is a string representing the parent directory ('..' or '::')
- os.sep is the (or a most common) pathname separator ('/' or ':' or '\')
- os.extsep is the extension separator ('.' or '/')
- os.altsep is the alternate pathname separator (None or '/')
- os.pathsep is the component separator used in \$PATH etc
- os.linesep is the line separator in text files ('r' or 'n' or 'rn')
- os.defpath is the default search path for executables
- os.devnull is the file path of the null device ('/dev/null', etc.)

Jugando con os.path

```
import os
```

```
os.listdir(".") # una lista con el contenido del directorio
```

```
for dp, dnames, fnames in os.walk("."):
    for fname in fnames:
        print os.path.join(dp, fname) # join sabe cual es el os.path.sep
```

```
>> os.path.split("path/a/un.archivo")
```

```
Out[38]: ('path/a', 'un.archivo')
```

```
>> os.path.splitdrive("path/a/un.archivo")
```

```
(' ', 'path/a/un.archivo')
```

```
>> os.path.splitext("path/a/un.archivo")
```

```
('path/a/un', '.archivo')
```

```
>> os.path.split("path/a/un.archivo")
```

```
('path/a', 'un.archivo')
```

Jugando con shutil

`shutil` tiene implementaciones de alto nivel para el tratamiento de archivos.

funciones interesantes:

- `shutil.copytree`
- `shutil.rmtree`
- `shutil.copy`
- `shutil.copy2`

JSON

- Viene en la librería estándar.
- Emite diccionarios, listas y demás tipos nativos en python.
- Api Simétrica:

```
json.load  
json.loads  
json.dump  
json.dumps
```

JSON

```
d = {"nombre": "Armando E. Banquito",  
     "edad": 25,  
     "altura": 1.7,  
     "conyugue": None,  
     "direcciones": {"casa": "Fake 123",  
                     "trabajo": "Real 456"},  
     "vehiculos": ["Fiat Uno", "VW Gol"],  
     "vivienda_propia": False}
```

```
json.dumps(d)
```

Ejercicios

Implementar un demonio con un thread o un proceso (como ejercicio de thread) que recorra los diferentes polls y los persista como json en una determinada carpeta definida por variable del settings `JSON_BACKUPS`.

Dividir las carpetas por fechas y hora de backup.

Mongo

HomePage: <http://www.mongodb.org/> Mogo vs the world: <http://goo.gl/xxwsN>

- **Written in:** C++
- **Main point:** Retains some friendly properties of SQL. (Query, index)
- **License:** AGPL (Drivers: Apache)
- **Protocol:** Custom, binary (BSON)
- Master/slave replication (auto failover with replica sets)
- Queries are javascript expressions
- Run arbitrary javascript functions server-side
- Better update-in-place than CouchDB
- Uses memory mapped files for data storage
- Performance over features
- Journaling (with --journal) is best turned on
- On 32bit systems, limited to ~2.5Gb
- An empty database takes up 192Mb
- GridFS to store big data + metadata (not actually an FS)

PyMongo

- Es medio el estandar de facto de manejo de mongo con python

```
import pymongo, bson

conn = pymongo.MongoClient('localhost', 27017)
db = conn['tudb']

d = {
    "nombre": "Armando E. Banquito",
    "edad": 25,
    "altura": 1.7,
    "conyugue": None,
    "direcciones": {
        "casa": "Fake 123",
        "trabajo": "Real 456"
    },
    "vehiculos": ["Fiat Uno", "VW Gol"],
    "vivienda_propia": False
}

db.persona.insert(d) # retorna un bson.ObjectId

d = {
    "nombre": "Armando Barro",
    "edad": 35,
    "altura": 1.8,
    "conyugue": None,
    "direcciones": {
        "casa": "Fake 123"
    },
}
```

```
"vehiculos": [],  
"vivienda_propia": False}  
db.persona.save(d) # update si tiene objectid insert si no
```

PyMongo Map Reduce

```
db.person.find() # todos
```

```
db.persona.find_one({"nombre": "Armando E. Banquito"})
```

```
db.persona.find({"edad":  
                 {"$gt": 15}  
                })
```

PyMongo MapReduce

```
fmap = bson.Code(
    """function () {
        emit(this.direcciones.casa, 1);
    }""")
freduce = bson.Code(
    """function (key, values) {
        return values.length;
    }""")

results = core.DB.responses.map_reduce(fmap, freduce, "myresults")
for r in results:
    print r # {u'_id': 0.016666666666666666, u'value': 3.0}
```


Disclaimer

De aca en mas robe las filiminas de esta dirección:

<http://www.slideshare.net/namlook/mongokit-presentation-mongofr2010>



A python ODM for MongoDB

```
class MyDocument(Document) :
```

Structure

Descriptors

Options

```
class MyDocument(Document) :
```

```
    structure = {  
        'foo': int,  
        'bar': float,  
        'spam': {  
            'eggs': [unicode],  
            'blah': None,  
        }  
    }
```

Descriptors

Options

```
class MyVeryNestedDoc(Document):
    structure = {
        '1':{
            '2':{
                '3':{
                    '4':{
                        '5':{
                            '6':{
                                '7':int,
                                '8':{
                                    '9':float,
                                }
                            }
                        }
                    }
                }
            }
        }
    }
```



```
class MyDocument(Document) :
```

```
    structure = {  
        'foo': unicode,  
        'bar': int,  
        'spam': {  
            'eggs': [unicode],  
            'blah': float,  
        }  
    }
```

Descriptors

Options

```
class MyDocument(Document) :
```

```
    structure = {  
        'foo': unicode,  
        'bar': int,  
        'spam': {  
            'eggs': [unicode],  
            'blah': float,  
        }  
    }
```

```
    required = ['foo', 'spam.eggs']  
    default_values = {'spam.blah' : 1.0}  
    validators = {'bar': lambda x > 0}
```

Options

```
class MyDocument(Document) :
```

```
    structure = {  
        'foo': unicode,  
        'bar': int,  
        'spam': {  
            'eggs': [unicode],  
            'blah': float,  
        }  
    }
```

```
    required = ['foo', 'spam.eggs']  
    default_values = {'spam.blah' : 1.0}  
    validators = {'bar': lambda x > 0}
```

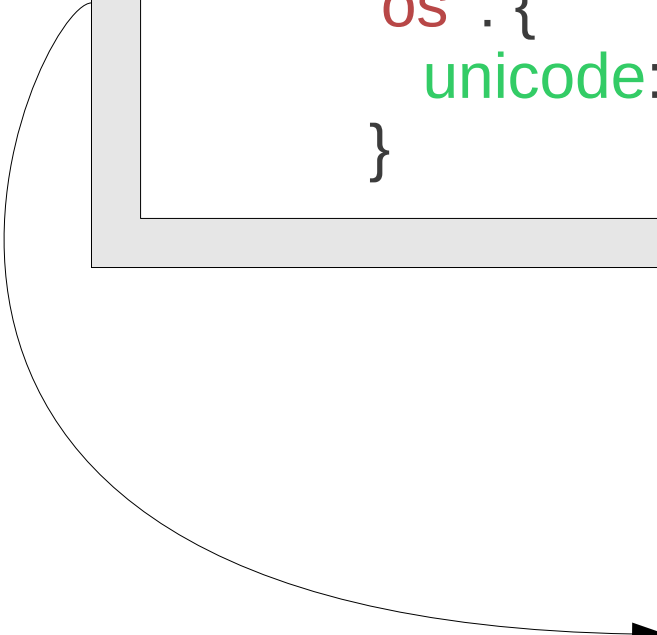
```
    use_dot_notation = True  
    skip_validation = True
```


Advantages

- ✓ **Great** readability
- ✓ **Simple** python dict
- ✓ **Pure** python types
- ✓ **Nested** and complex schema declaration
- ✓ **Fast** : don't instanciate objects
- ✓ **Live** update via instrospection
- ✓ **Dynamic** keys

Dynamic keys

```
class MobilePhones(Document) :  
    structure = {  
        'os' : {  
            unicode:[{'version': float, 'name':unicode}],  
        }  
    }
```

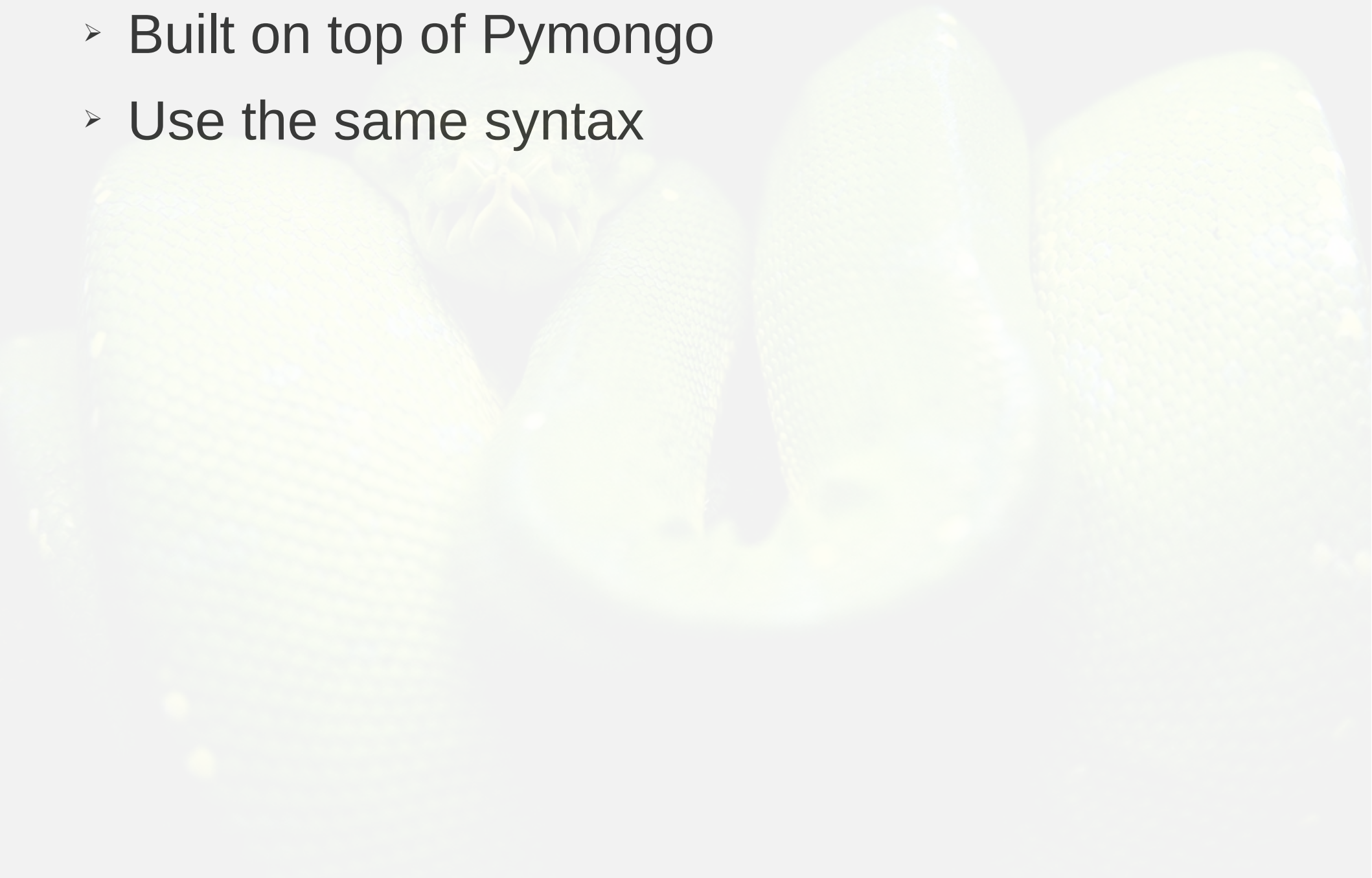


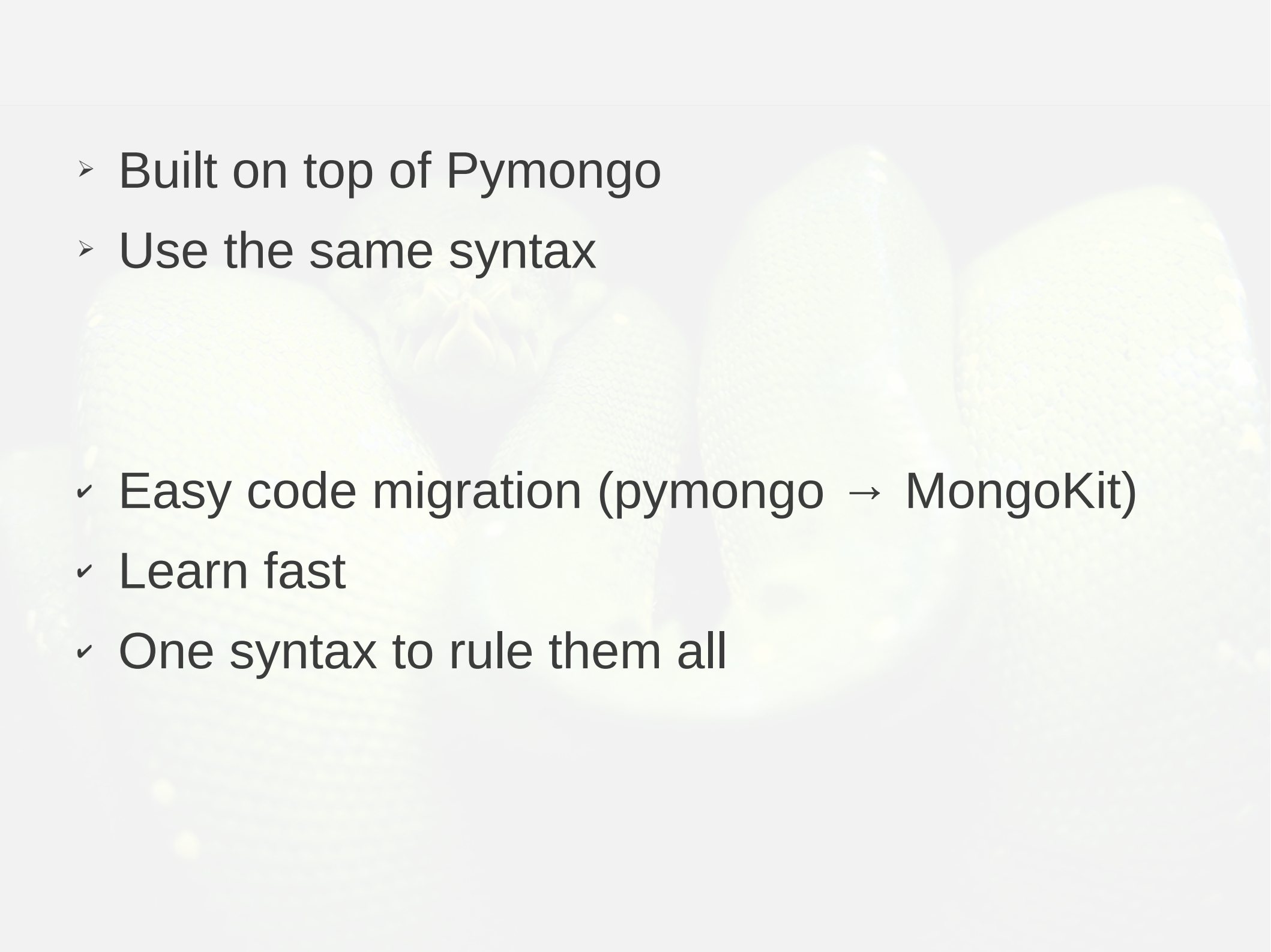
```
{'os' : {  
    'android':[  
        {'version': 2.2, 'name' : 'froyo'},  
        {'version': 2.1, 'name' : 'eclair'}  
    ],  
    'iphone': [{'version': 4, 'name' : 'iOS'}],  
}
```



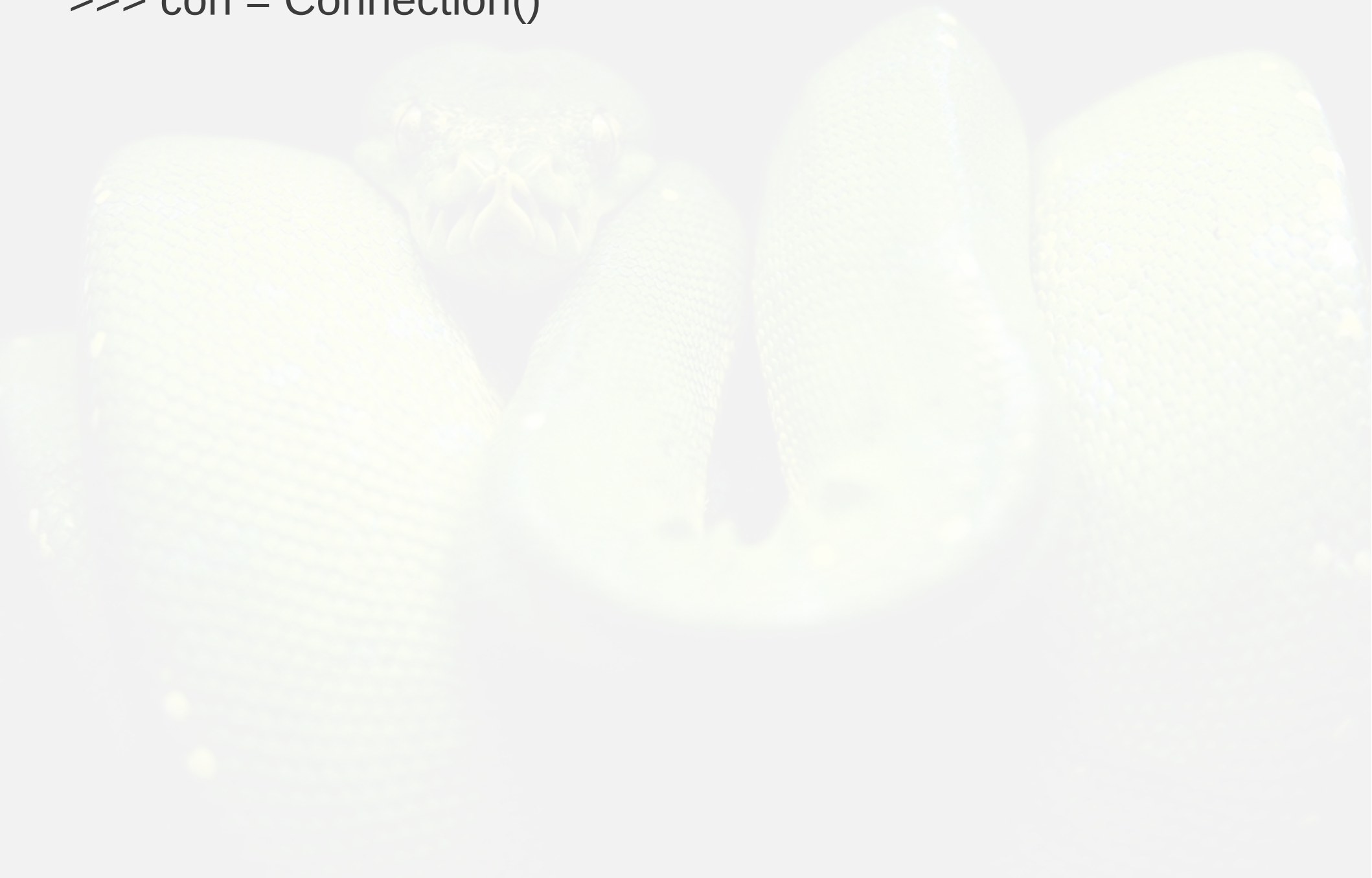

It's all about Pymongo

- Built on top of Pymongo
- Use the same syntax



- 
- A green snake with a yellow underbelly is coiled around a large, smooth, green rock. The snake's head is raised, and its body is wrapped around the rock. The background is a light, hazy green.
- Built on top of Pymongo
 - Use the same syntax
 - ✓ Easy code migration (pymongo → MongoKit)
 - ✓ Learn fast
 - ✓ One syntax to rule them all

```
>>> from mongokit import *  
>>> con = Connection()
```




```
>>> from mongokit import *
```

```
>>> con = Connection()
```

Pymongo's way

```
>>> doc = con.mydb.mycol.find_one() # very fast !
```

doc is a dict instance

```
>>> from mongokit import *
```

```
>>> con = Connection()
```

Pymongo's way

```
>>> doc = con.mydb.mycol.find_one() # very fast !
```

doc is a dict instance

Mongokit's way

```
>>> doc = con.mydb.mycol.MyDocument.find_one()
```

doc is a MyDocument instance

```
>>> doc.spam.eggs.append(u'foo')
```

```
>>> doc.save()
```




Features

Inheritance / Polymorphism

```
class A(Document) :  
    structure = {  
        'a': {  
            'foo' : unicode,  
        }  
    }
```


```
class B(Document) :  
    structure = {  
        'b': {  
            'bar' : [float],  
        }  
    }
```

Inheritance / Polymorphism

```
class A(Document) :  
    structure = {  
        'a': {  
            'foo' : unicode,  
        }  
    }
```

```
class B(Document) :  
    structure = {  
        'b': {  
            'bar' : [float],  
        }  
    }
```

```
class C(A,B) :  
    structure = {  
        'c': {  
            'spam' : int,  
        }  
    }
```



Inheritance / Polymorphism

```
class A(Document) :  
    structure = {  
        'a': {  
            'foo' : unicode,  
        }  
    }
```

```
class B(Document) :  
    structure = {  
        'b': {  
            'bar' : [float],  
        }  
    }
```

```
class C(A,B) :  
    structure = {  
        'c': {  
            'spam' : int,  
        }  
    }
```

>>> con.mydb.mycol.C()

```
{  
    'a' : {'foo' : None},  
    'b' : {'bar' : []},  
    'c' : {'spam' : None}  
}
```

Dot notation

```
class MyDocument(Document) :  
    structure = {  
        'foo': unicode,  
        'bar': int,  
        'spam': {  
            'eggs': [unicode],  
            'blah': float,  
        }  
    }
```

Dot notation

```
class MyDocument(Document) :  
    structure = {  
        'foo': unicode,  
        'bar': int,  
        'spam': {  
            'eggs': [unicode],  
            'blah': float,  
        }  
    }  
    use_dot_notation = True
```

Dot notation

```
class MyDocument(Document) :  
    structure = {  
        'foo': unicode,  
        'bar': int,  
        'spam': {  
            'eggs': [unicode],  
            'blah': float,  
        }  
    }  
    use_dot_notation = True
```

```
>>> doc = con.mydb.mycol.MyDocument()  
>>> doc.foo = u'the foo'  
>>> doc.spam.eggs.append(u'bla', u'toto')
```

Document reference

```
class User(Document) :  
    structure = {  
        'login': unicode,  
        'name': unicode,  
    }
```

```
class Comment(Document) :  
    structure = {  
        'title': unicode,  
        'body': unicode,  
        'author' : ObjectId,  
    }
```


Document reference

```
class User(Document) :  
    structure = {  
        'login': unicode,  
        'name': unicode,  
    }
```

```
class Comment(Document) :  
    structure = {  
        'title': unicode,  
        'body': unicode,  
        'author' : User,  
    }  
    use_autorefs = True
```

Document reference

```
class User(Document) :  
    structure = {  
        'login': unicode,  
        'name': unicode,  
    }
```

```
class Comment(Document) :  
    structure = {  
        'title': unicode,  
        'body': unicode,  
        'author' : User,  
    }  
    use_autorefs = True
```

```
>>> con.mydb.mycol.find_one()
```

```
{  
    'title' : 'Hello world !',  
    'body' : 'My first blog post',  
    'author' : DBRef(...),  
}
```

Document reference

```
class User(Document) :  
    structure = {  
        'login': unicode,  
        'name': unicode,  
    }
```

```
class Comment(Document) :  
    structure = {  
        'title': unicode,  
        'body': unicode,  
        'author' : User,  
    }  
    use_autorefs = True
```

```
>>> con.mydb.mycol.find_one()
```

```
{  
    'title' : 'Hello world !',  
    'body' : 'My first blog post',  
    'author' : DBRef(...),  
}
```

```
>>> con.mydb.mycol.BlogPost.find_one()
```

```
{  
    'title' : 'Hello world !',  
    'body' : 'My first blog post',  
    'author' : {  
        'login' : 'timy',  
        'name' : 'Timy Donzy'  
    }  
}
```

GridFS support

```
class MyDocument(Document) :  
    structure = {  
        'foo': unicode,  
        'bar': int,  
    }  
    grid_fs = {  
        'files': ['source', 'template'],  
        'containers': ['images'],  
    }
```

```
>>> doc = con.mydb.mycol.MyDocument()  
>>> doc.fs.source = '...'  
>>> doc.fs.images['image1.png'] = '...'
```

i18n

```
class MyDocument(Document) :  
    structure = {  
        'foo': unicode,  
        'bar': int,  
    }  
    i18n = ['foo']  
    use_dot_notation = True
```

```
>>> doc = con.mydb.mycol.MyDocument()  
>>> doc.set_lang('fr')  
>>> doc.foo = u'Salut'  
  
>>> doc.set_lang('en')  
>>> doc.foo = u'Hello'  
>>> doc.save()
```

- ✓ Inheritance / Polymorphism
- ✓ Dot notation
- ✓ Document auto-reference support (DBRef)
- ✓ GridFS
- ✓ I18n support

- ✓ Inheritance / Polymorphism
- ✓ Dot notation
- ✓ Document auto-reference support (DBRef)
- ✓ GridFS
- ✓ I18n support
- ✓ Schema migration
- ✓ Json export/import

- ✓ Inheritance / Polymorphism
- ✓ Dot notation
- ✓ Document auto-reference support (DBRef)
- ✓ GridFS
- ✓ I18n support
- ✓ Schema migration
- ✓ Json export/import

CAN BE DISABLED

mongokit-django

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.sqlite3',  
        'NAME': 'example-sqlite3.db',  
    },  
    'mongodb': {  
        'ENGINE': 'django_mongokit.mongodb',  
        'NAME': 'example',  
    },  
}
```

mongokit-django

```
from django_mongokit.document import DjangoDocument
from django_mongokit import connection

from django_mongokit import get_database
database = get_database()

class Talk(DjangoDocument):
    structure = {
        'topic': unicode,
        'date': datetime.datetime
    }

connection.register([Talk])
```

mongokit-django

```
database = conn.mydb
collection = database.mycollection
instances = collection.Talk.find()

instance = collection.Computer()
...
instance.save()
```

Ejercicio

Implementar el poll con mongokit-django