

django

Disfruta programando.

Jorge Bastida [neo]

neo2001@gmail.com
@jorgebastida

Jaime Irurzun [etox]

jaime.irurzun@gmail.com
@jaimeirurzun



EWI

Impartial, Independent Expertise
for Justice

- Acts as a voice for experts
- Provides Training
- Makes representations to government
- Works actively with other bodies
- Helpline





Disfruta programando.



Índice

1. Python

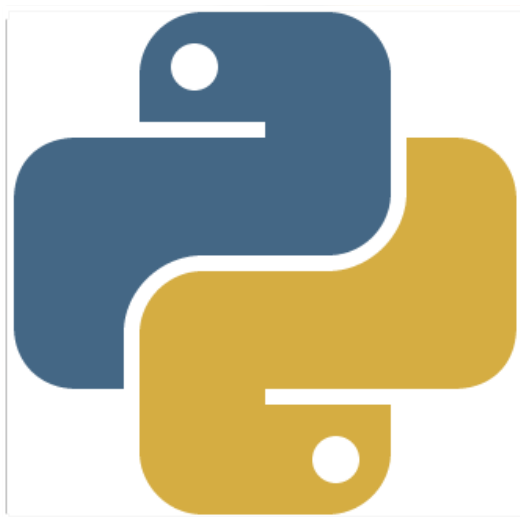
- a. Introducción
- b. Tipos de datos
- c. Operadores
- d. Usos frecuentes
- e. Estructuras
- f. Sentencias
- g. Ejercicio

2. Django

- a. Introducción
- b. URLs y Vistas
- c. Templates
- d. Modelo
- e. Administración
- f. Formularios
- g. Magia avanzada



Proyecto



python

Índice

1. Python

- a. Introducción
- b. Tipos de datos
- c. Operadores
- d. Usos frecuentes
- e. Estructuras
- f. Sentencias
- g. Ejercicio

2. Django

- a. Introducción
- b. URLs y Vistas
- c. Templates
- d. Modelo
- e. Administración
- f. Formularios
- g. Magia avanzada



Proyecto



- **Legible** Sintaxis intuitiva y estricta
- **Productivo** Entre 1/3 y 1/5 más conciso que Java o C++
- **Portable** GNU/Linux, Windows, Mac OS X, ...
- **Extenso** Standard Library, Third parties
- **Integrable** C, C++, Java, .NET, COM, WS, CORBA, ...
- **... y Divertido**

Instalación



```
$ sudo apt-get install python
```



<http://www.python.org/download/>



Snow Leopard incluye las versiones 2.5.4 y 2.6.1

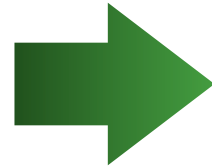
El Intérprete

El Intérprete

Modo Batch

holamundo.py

```
#!/usr/bin/env python  
print "hola mundo!"
```



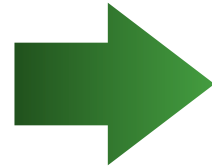
```
$ python holamundo.py  
hola mundo!  
$
```

El Intérprete

Modo Batch

holamundo.py

```
#!/usr/bin/env python  
print "hola mundo!"
```



```
$ python holamundo.py  
hola mundo!  
$
```

Modo Interactivo

```
$ python  
Python 2.6.5 (r265:79063, Apr 16 2010, 13:09:56)  
[GCC 4.4.3] on linux2  
Type "help", "copyright", "credits" or "license" for more information.  
>>> print "hola mundo!"  
hola mundo!
```


Índice

1. Python

- a. Introducción
- b. Tipos de datos**
- c. Operadores
- d. Usos frecuentes
- e. Estructuras
- f. Sentencias
- g. Ejercicio

2. Django

- a. Introducción
- b. URLs y Vistas
- c. Templates
- d. Modelo
- e. Administración
- f. Formularios
- g. Magia avanzada



Proyecto

Tipos de datos

object

+ info: <http://docs.python.org/library/stdtypes.html>

Tipos de datos

int

1234

long

35L

float

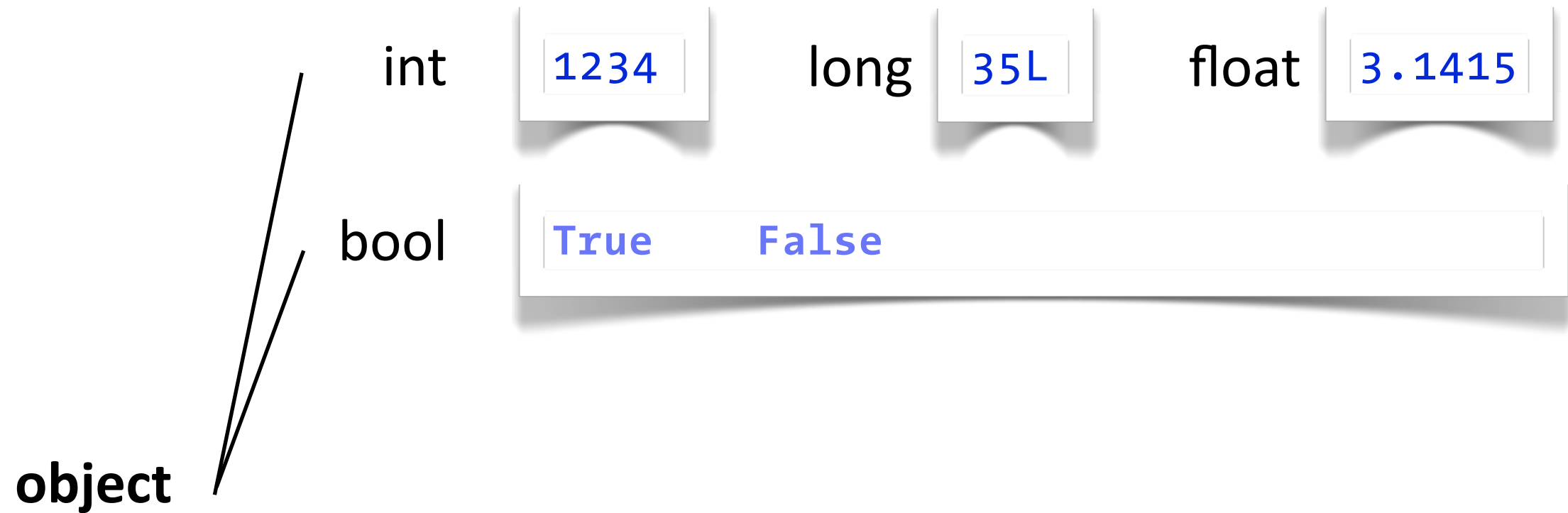
3.1415

object



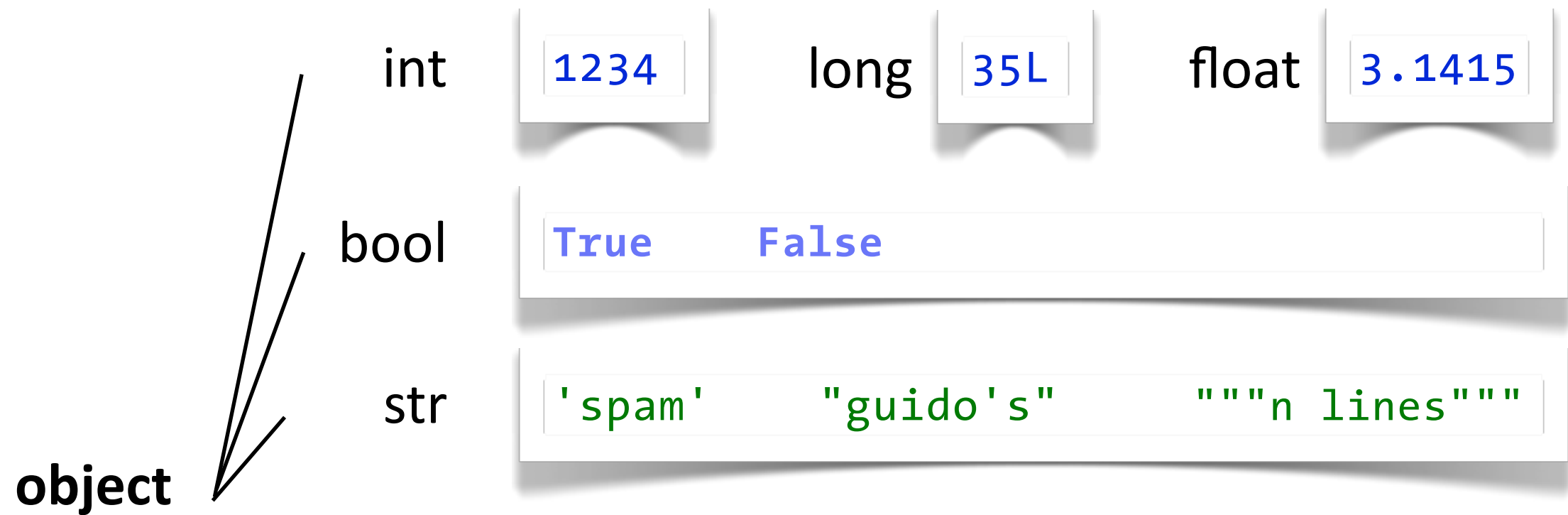
+ info: <http://docs.python.org/library/stdtypes.html>

Tipos de datos



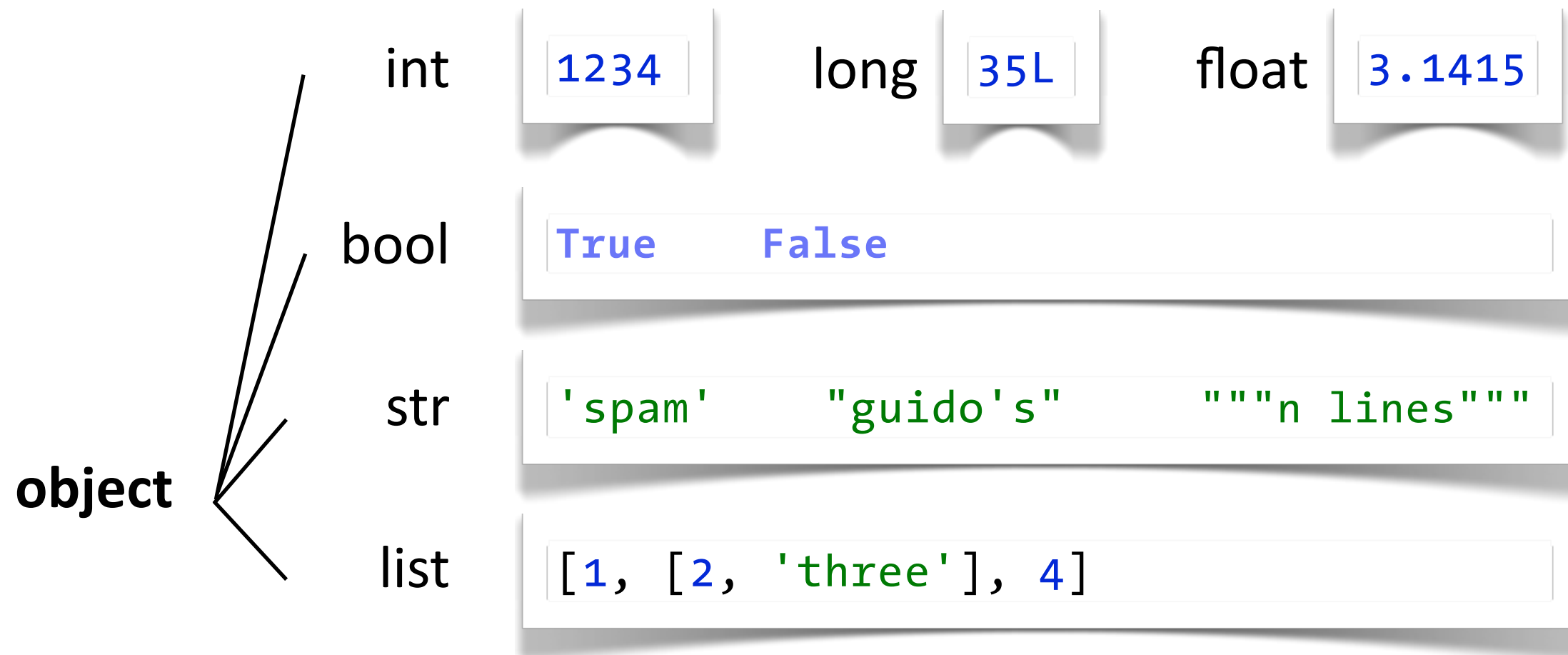
+ info: <http://docs.python.org/library/stdtypes.html>

Tipos de datos



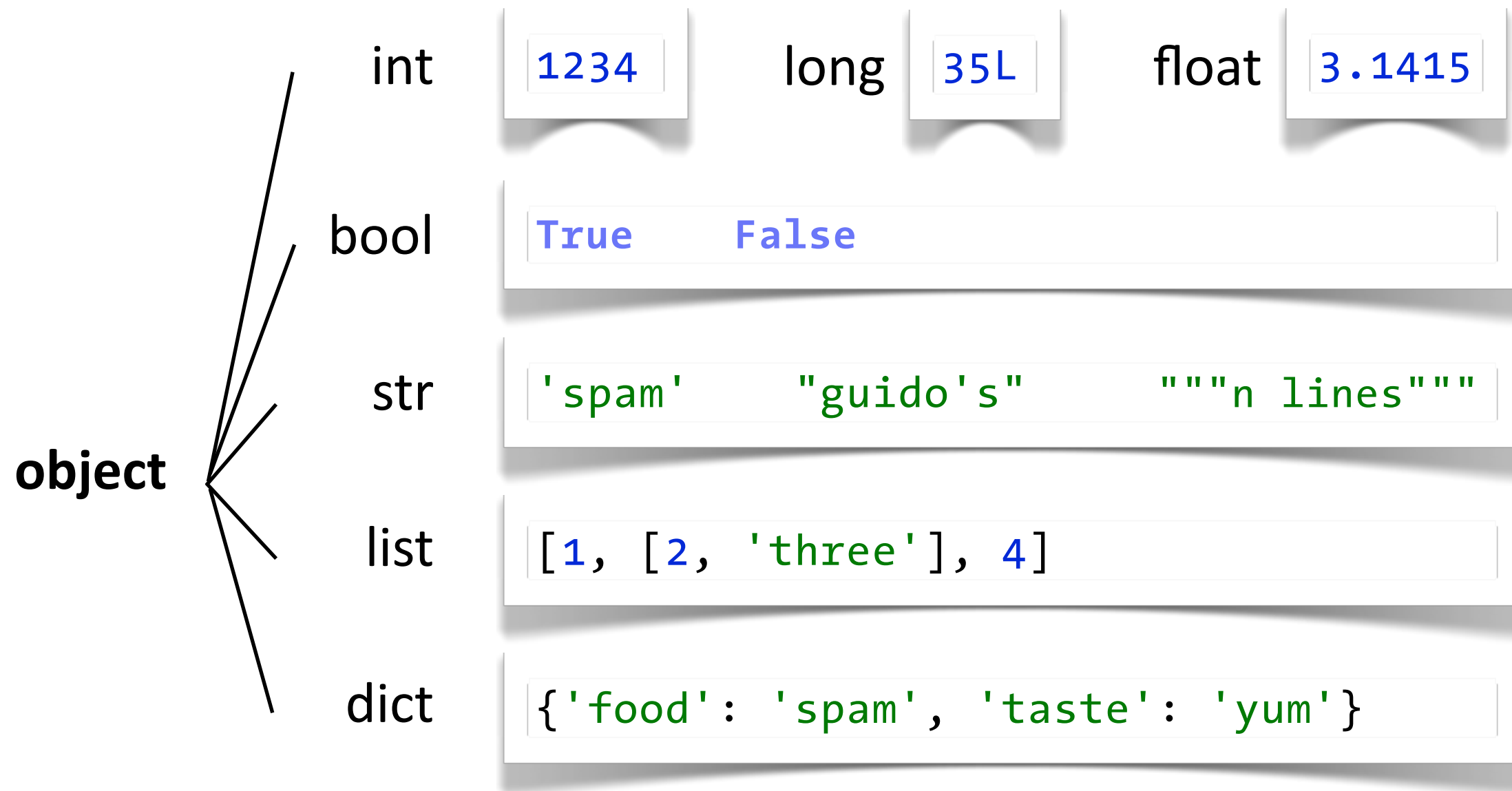
+ info: <http://docs.python.org/library/stdtypes.html>

Tipos de datos



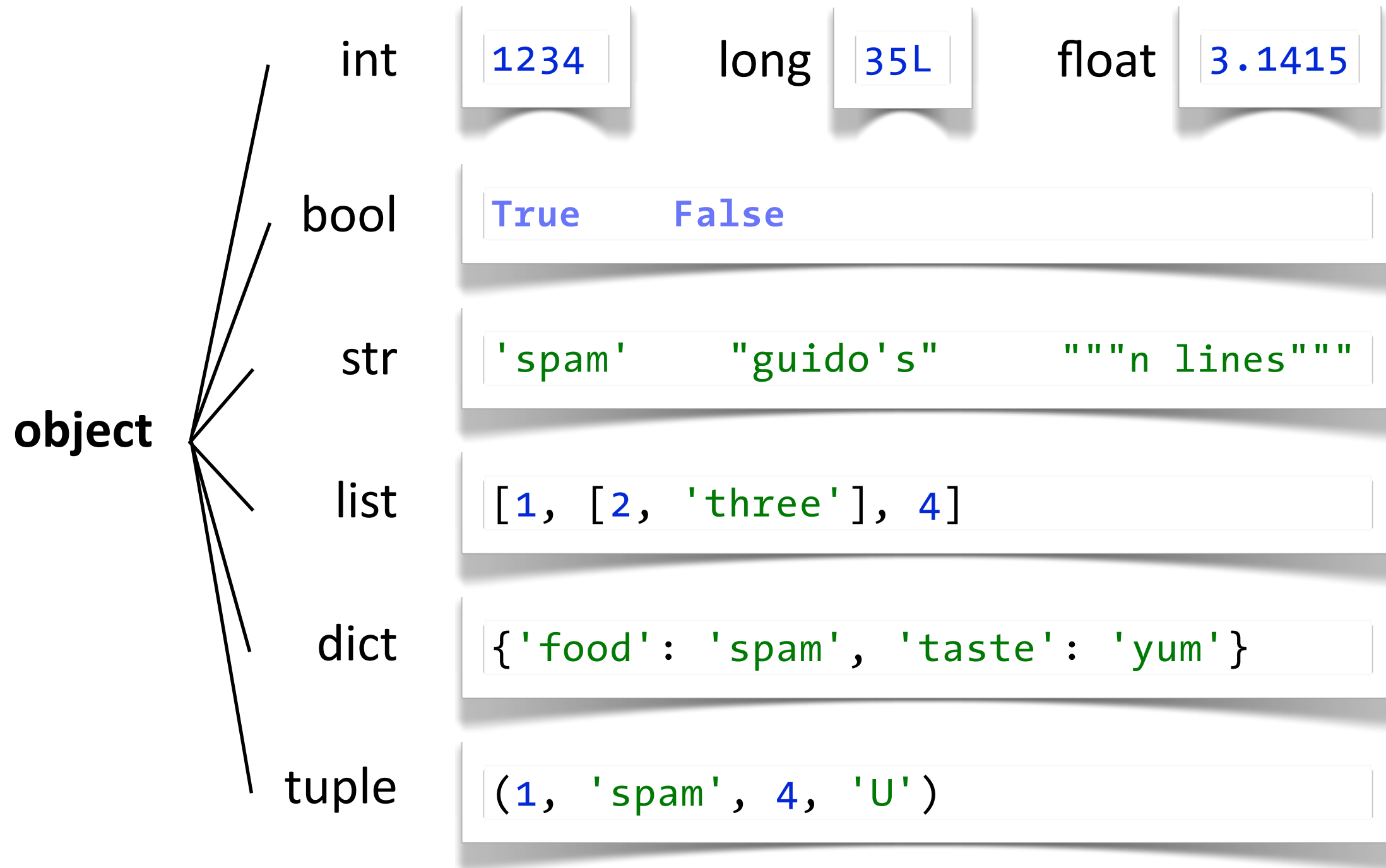
+ info: <http://docs.python.org/library/stdtypes.html>

Tipos de datos



+ info: <http://docs.python.org/library/stdtypes.html>

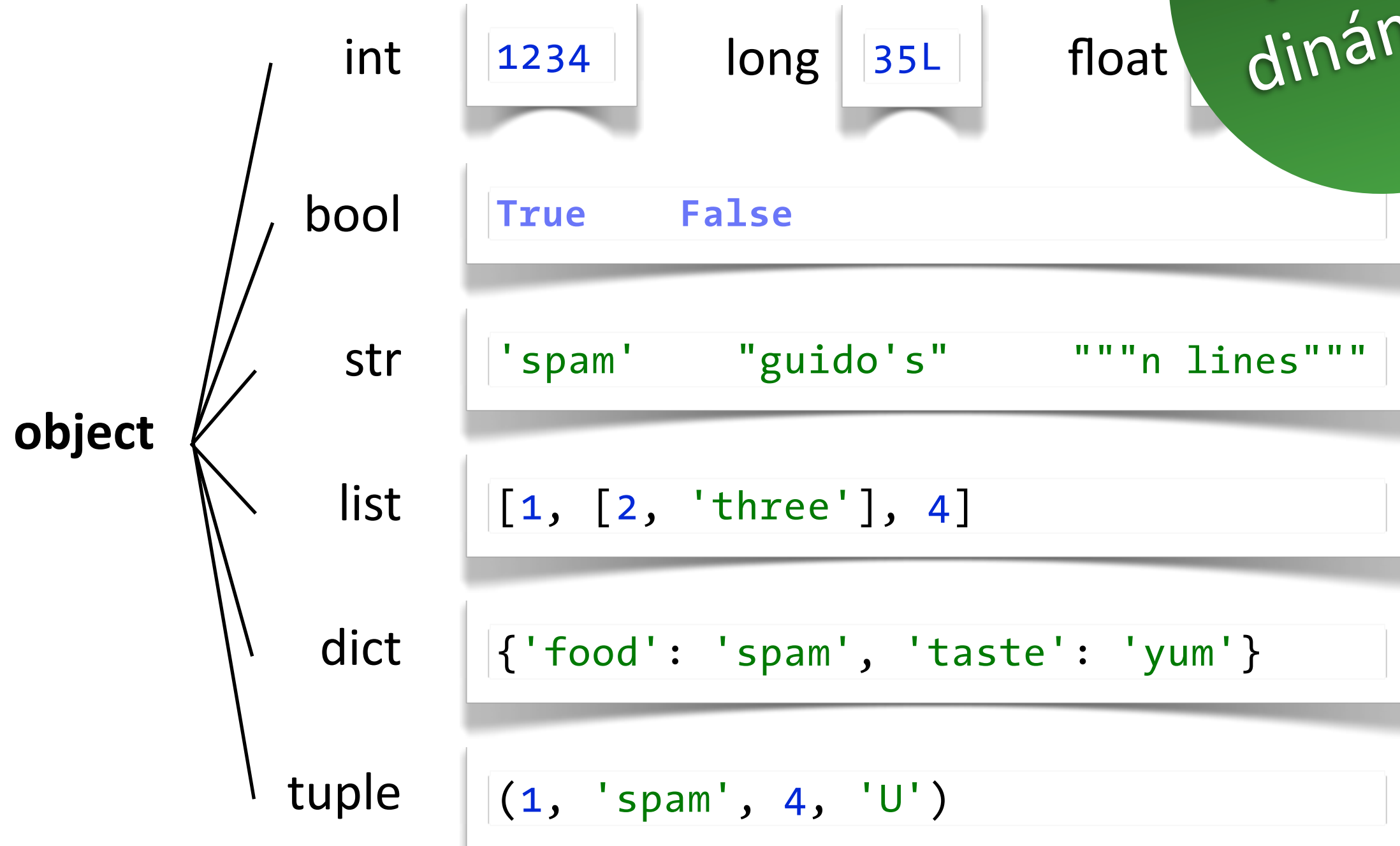
Tipos de datos



+ info: <http://docs.python.org/library/stdtypes.html>

Tipos de datos

¡Tipado
dinámico!



+ info: <http://docs.python.org/library/stdtypes.html>

Índice

1. Python

- a. Introducción
- b. Tipos de datos
- c. Operadores**
- d. Usos frecuentes
- e. Estructuras
- f. Sentencias
- g. Ejercicio

2. Django

- a. Introducción
- b. URLs y Vistas
- c. Templates
- d. Modelo
- e. Administración
- f. Formularios
- g. Magia avanzada



Proyecto

Operadores

+ info: <http://docs.python.org/library/operator.html>

Operadores

numéricos

$a + b$	$a - b$	$a * b$	a / b
$a \% b$	$-a$	$+a$	$a ** b$

+ info: <http://docs.python.org/library/operator.html>

Operadores

numéricos

$a + b$	$a - b$	$a * b$	a / b
$a \% b$	$-a$	$+a$	$a ** b$

comparadores

$a < b$	$a <= b$	$a > b$
$a >= b$	$a == b$	$a != b$

+ info: <http://docs.python.org/library/operator.html>

Operadores

numéricos

$a + b$ $a - b$ $a * b$ a / b
 $a \% b$ $-a$ $+a$ $a ** b$

comparadores

$a < b$ $a <= b$ $a > b$
 $a >= b$ $a == b$ $a != b$

lógicos

$a \text{ or } b$ $a \text{ and } b$ $\text{not } a$

+ info: <http://docs.python.org/library/operator.html>

Índice

1. Python

- a. Introducción
- b. Tipos de datos
- c. Operadores
- d. Usos frecuentes**
- e. Estructuras
- f. Sentencias
- g. Ejercicio

2. Django

- a. Introducción
- b. URLs y Vistas
- c. Templates
- d. Modelo
- e. Administración
- f. Formularios
- g. Magia avanzada



Proyecto

Usos frecuentes: list

Usos frecuentes: list

```
>>> nums = [1, 2, 3]  
>>> nums[0]  
1
```


Usos frecuentes: list

```
>>> nums = [1, 2, 3]  
>>> nums[0]  
1
```

```
>>> 2 in nums  
True
```

Usos frecuentes: list

```
>>> nums = [1, 2, 3]
>>> nums[0]
1
```

```
>>> 2 in nums
True
```

```
>>> nums.append(4)
>>> print nums
[1, 2, 3, 4]
```

Usos frecuentes: list

```
>>> nums = [1, 2, 3]
>>> nums[0]
1
```

```
>>> 2 in nums
True
```

```
>>> nums.append(4)
>>> print nums
[1, 2, 3, 4]
```

```
>>> nums.count()
4
```

Usos frecuentes: str

Usos frecuentes: str

```
>>> "Python mola"[1:4]  
'yth'
```


Usos frecuentes: str

```
>>> "Python mola"[1:4]  
'yth'
```

```
>>> "Python mola".find("mola")  
7
```

Usos frecuentes: str

```
>>> "Python mola"[1:4]  
'yth'
```

```
>>> "Python mola".find("mola")  
7
```

```
>>> "Python mola".replace("Python", "PHP no")  
'PHP no mola'
```

Usos frecuentes: str

```
>>> "Python mola"[1:4]  
'yth'
```

```
>>> "Python mola".find("mola")  
7
```

```
>>> "Python mola".replace("Python", "PHP no")  
'PHP no mola'
```

```
>>> "Python mola".split(" ")  
['Python', 'mola']
```

Usos frecuentes: str

```
>>> "Python mola"[1:4]
'yth'
```

```
>>> "Python mola".find("mola")
7
```

```
>>> "Python mola".replace("Python", "PHP no")
'PHP no mola'
```

```
>>> "Python mola".split(" ")
['Python', 'mola']
```

```
>>> " ".join(["Python", "mola"])
"Python mola"
```

Usos frecuentes: dict

Usos frecuentes: dict

```
>>> user = {'nick': 'neo', 'age': 24}
>>> user.keys()
['nick', 'age']
>>> user.values()
['neo', 24]
```


Usos frecuentes: dict

```
>>> user = {'nick': 'neo', 'age': 24}
>>> user.keys()
['nick', 'age']
>>> user.values()
['neo', 24]
```

```
>>> user['age']
24
```

```
>>> user.get('age', 20)
24
```

Usos frecuentes: dict

```
>>> user = {'nick': 'neo', 'age': 24}
>>> user.keys()
['nick', 'age']
>>> user.values()
['neo', 24]
```

```
>>> user['age']
24
```

```
>>> user.get('age', 20)
24
```

```
>>> user.has_key('nick')
True
```

Usos frecuentes: dict

```
>>> user = {'nick': 'neo', 'age': 24}
>>> user.keys()
['nick', 'age']
>>> user.values()
['neo', 24]
```

```
>>> user['age']
24
```

```
>>> user.get('age', 20)
24
```

```
>>> user.has_key('nick')
True
```

```
>>> user.update({'nick': 'alatar', 'age': 25})
>>> user
{'nick': 'alatar', 'age': 25}
```

Usos frecuentes: str %

Usos frecuentes: str %

```
>>> "%s es muy sabio" % "Hycker"  
'Hycker es muy sabio'
```

Usos frecuentes: str %

```
>>> "%s es muy sabio" % "Hycker"  
'Hycker es muy sabio'
```

```
>>> "%s sabe %i idiomas" % ("Hycker", 5)  
'Hycker sabe 5 idiomas'
```


Usos frecuentes: str %

```
>>> "%s es muy sabio" % "Hycker"  
'Hycker es muy sabio'
```

```
>>> "%s sabe %i idiomas" % ("Hycker", 5)  
'Hycker sabe 5 idiomas'
```

```
>>> t = "%(NOMBRE)s sabe %(IDIOMAS)i idiomas"  
>>> v = {'NOMBRE': 'Hycker', 'IDIOMAS': 5}  
>>> t % v  
'Hycker sabe 5 idiomas'
```

Módulos

```
sound/  
  __init__.py  
  formats/  
    __init__.py  
    wavread.py  
    wavwrite.py  
    aiffread.py  
    aiffwrite.py  
    auread.py  
    auwrite.py  
    ...  
  effects/  
    __init__.py  
    echo.py  
    surround.py  
    reverse.py  
    ...  
  filters/  
    __init__.py  
    equalizer.py  
    vocoder.py  
    karaoke.py  
    ...
```

1. Un **módulo** es un fichero **.py**
2. Los módulos pueden organizarse en paquetes.
3. Un **paquete** es una carpeta que contiene un fichero con nombre **__init__.py**

Módulos

sound/

2

`__init__.py`
`formats/`

1

`__init__.py`
`wavread.py`
`wavwrite.py`
`aiffread.py`
`aiffwrite.py`
`auread.py`
`auwrite.py`
`...`

`effects/`

`__init__.py`
`echo.py`
`surround.py`
`reverse.py`
`...`

`filters/`

`__init__.py`
`equalizer.py`
`vocoder.py`
`karaoke.py`
`...`

3

1. Un **módulo** es un fichero **.py**

2. Los módulos pueden organizarse en paquetes.

3. Un **paquete** es una carpeta que contiene un fichero con nombre **__init__.py**

Módulos

```
import math
```

¿Dónde los busca Python?

En la lista de directorios que contiene la lista `sys.path`:

1. El directorio **actual**:

`./`

2. El directorio **site-packages** de nuestro Python:

`/usr/local/lib/python2.6/site-packages`

3. Los directorios apuntados por **PYTHONPATH**:

```
$ env | grep PYTHONPATH
```

4. El directorio de la instalación de **Python**:

`/usr/lib/python2.6/`

Módulos

```
sound/  
  __init__.py  
  formats/  
    __init__.py  
    wavread.py  
    wavwrite.py  
    aiffread.py  
    aiffwrite.py  
    auread.py  
    auwrite.py  
    ...  
  effects/  
    __init__.py  
    echo.py  
    surround.py  
    reverse.py  
    ...  
  filters/  
    __init__.py  
    equalizer.py  
    vocoder.py  
    karaoke.py  
    ...
```

```
import sound.effects.echo  
sound.effects.echo.echofilter(...)
```

Módulos

```
sound/  
  __init__.py  
  formats/  
    __init__.py  
    wavread.py  
    wavwrite.py  
    aiffread.py  
    aiffwrite.py  
    auread.py  
    auwrite.py  
    ...  
  effects/  
    __init__.py  
    echo.py  
    surround.py  
    reverse.py  
    ...  
  filters/  
    __init__.py  
    equalizer.py  
    vocoder.py  
    karaoke.py  
    ...
```

```
import sound.effects.echo  
sound.effects.echo.echofilter(...)
```

```
from sound.effects import echo  
echo.echofilter(...)
```


Módulos

```
sound/  
  __init__.py  
  formats/  
    __init__.py  
    wavread.py  
    wavwrite.py  
    aiffread.py  
    aiffwrite.py  
    auread.py  
    auwrite.py  
    ...  
  effects/  
    __init__.py  
    echo.py  
    surround.py  
    reverse.py  
    ...  
  filters/  
    __init__.py  
    equalizer.py  
    vocoder.py  
    karaoke.py  
    ...
```

```
import sound.effects.echo  
sound.effects.echo.echofilter(...)
```

```
from sound.effects import echo  
echo.echofilter(...)
```

```
from sound.effects.echo import echofilter  
echofilter(...)
```

Módulos

```
sound/  
  __init__.py  
  formats/  
    __init__.py  
    wavread.py  
    wavwrite.py  
    aiffread.py  
    aiffwrite.py  
    auread.py  
    auwrite.py  
    ...  
  effects/  
    __init__.py  
    echo.py  
    surround.py  
    reverse.py  
    ...  
  filters/  
    __init__.py  
    equalizer.py  
    vocoder.py  
    karaoke.py  
    ...
```

```
import sound.effects.echo  
sound.effects.echo.echofilter(...)
```

```
from sound.effects import echo  
echo.echofilter(...)
```

```
from sound.effects.echo import echofilter  
echofilter(...)
```

```
from sound.effects.echo import *  
echofilter(...)
```

Módulos

```
sound/  
  __init__.py  
  formats/  
    __init__.py  
    wavread.py  
    wavwrite.py  
    aiffread.py  
    aiffwrite.py  
    auread.py  
    auwrite.py  
    ...  
  effects/  
    __init__.py  
    echo.py  
    surround.py  
    reverse.py  
    ...  
  filters/  
    __init__.py  
    equalizer.py  
    vocoder.py  
    karaoke.py  
    ...
```

```
import sound.effects.echo  
sound.effects.echo.echofilter(...)
```

```
from sound.effects import echo  
echo.echofilter(...)
```

```
from sound.effects.echo import echofilter  
echofilter(...)
```

```
from sound.effects.echo import *  
echofilter(...)
```

```
from sound.effects import echo as rev  
rev.echofilter(...)
```

Índice

1. Python

- a. Introducción
- b. Tipos de datos
- c. Operadores
- d. Usos frecuentes
- e. Estructuras**
- f. Sentencias
- g. Ejercicio

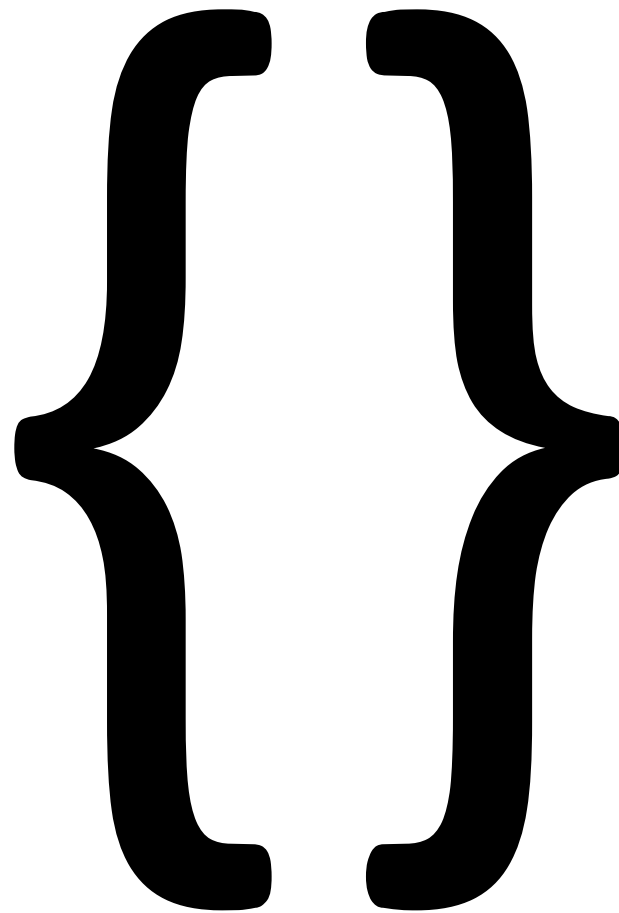
2. Django

- a. Introducción
- b. URLs y Vistas
- c. Templates
- d. Modelo
- e. Administración
- f. Formularios
- g. Magia avanzada

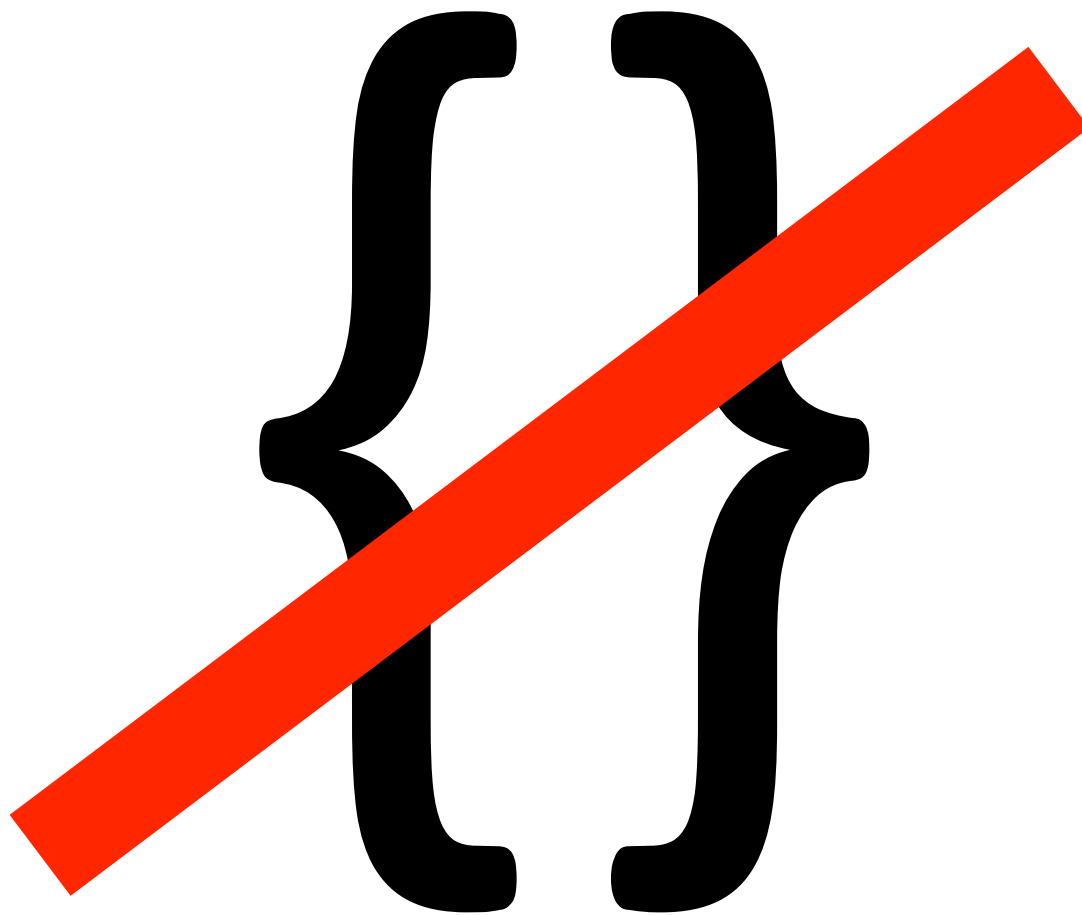


Proyecto

Estructuras



Estructuras



Identación

```
>>> from __future__ import braces  
      File "<stdin>", line 1  
      SyntaxError: not a chance
```


Identación

```
>>> from __future__ import braces
      File "<stdin>", line 1
        SyntaxError: not a chance
```

PEP 8

4 Espacios para indicar la indentación

¿PEP 8?

PEP 8

- Guido van Rossum (2001)
- Recomendaciones de estilo
- “Readability counts”
- “Code is read much more often than it is written.”
- Muy ~~recomendable~~ importante seguirlo.

<http://www.python.org/dev/peps/pep-0008/>

Funciones

```
def my_first_function(p1, p2):  
    return "Hello World!"
```

Funciones

```
1 def my_first_function(p1, p2):  
    return "Hello World!"
```

Funciones

```
1 def my_first_function(p1, 2 p2):  
    return "Hello World!"
```

Funciones

```
1 def my_first_function(p1, p2):  
    2  
    3 return "Hello World!"
```

Funciones

```
1 def my_first_function(p1, p2):  
    3 return "Hello World!"
```

Recomendaciones:

PEP 8

- Minúsculas
- Palabras separadas por _
- Evitar camelCase

Conversión de tipos

+info: <http://docs.python.org/library/functions.html>

Conversión de tipos

```
>>> int(1.3)  
1
```

+info: <http://docs.python.org/library/functions.html>

Conversión de tipos

```
>>> int(1.3)  
1
```

```
>>> str(2)  
'2'
```

+info: <http://docs.python.org/library/functions.html>

Conversión de tipos

```
>>> int(1.3)  
1
```

```
>>> str(2)  
'2'
```

```
>>> float(1)  
1.0
```

+info: <http://docs.python.org/library/functions.html>

Conversión de tipos

```
>>> int(1.3)  
1
```

```
>>> str(2)  
'2'
```

```
>>> float(1)  
1.0
```

```
>>> tuple([1,2,3])  
(1, 2, 3)
```

+info: <http://docs.python.org/library/functions.html>

Conversión de tipos

```
>>> int(1.3)  
1
```

```
>>> str(2)  
'2'
```

```
>>> float(1)  
1.0
```

```
>>> tuple([1,2,3])  
(1, 2, 3)
```

```
>>> list((1,2,3))  
[1, 2, 3]
```

+info: <http://docs.python.org/library/functions.html>

Funciones comunes

+info: <http://docs.python.org/library/functions.html>

Funciones comunes

```
>>> len("Python Mola")  
11  
>>> len([1,2,3,4])  
4
```

+info: <http://docs.python.org/library/functions.html>

Funciones comunes

```
>>> len("Python Mola")
11
>>> len([1,2,3,4])
4
```

```
>>> range(5)
[0, 1, 2, 3, 4]
>>> range(1,7)
[1, 2, 3, 4, 5, 6]
>>> range(1,7,2)
[1, 3, 5]
```

+info: <http://docs.python.org/library/functions.html>

Funciones comunes

```
>>> len("Python Mola")
11
>>> len([1,2,3,4])
4
```

```
>>> type(True)
<type 'bool'>
>>> type("Python Mola")
<type 'str'>
```

```
>>> range(5)
[0, 1, 2, 3, 4]
>>> range(1,7)
[1, 2, 3, 4, 5, 6]
>>> range(1,7,2)
[1, 3, 5]
```

+info: <http://docs.python.org/library/functions.html>

Funciones comunes

```
>>> len("Python MoIa")  
11  
>>> len([1,2,3,4])  
4
```

```
>>> range(5)  
[0, 1, 2, 3, 4]  
>>> range(1,7)  
[1, 2, 3, 4, 5, 6]  
>>> range(1,7,2)  
[1, 3, 5]
```

```
>>> type(True)  
<type 'bool'>  
>>> type("Python MoIa")  
<type 'str'>
```

```
>>> sum([0,1,2,3,4])  
10  
>>> sum(range(5))  
10
```

Y un muy largo etc...

+info: <http://docs.python.org/library/functions.html>

Funciones interesantes

Son sólo un ejemplo...

+info: <http://docs.python.org/library/functions.html>

Funciones interesantes

```
>>> a = [1,2,3]
>>> b = [4,5,6]
>>> zip(a,b)
[(1, 4), (2, 5), (3, 6)]
```

Son sólo un ejemplo...

+info: <http://docs.python.org/library/functions.html>

Funciones interesantes

```
>>> a = [1,2,3]
>>> b = [4,5,6]
>>> zip(a,b)
[(1, 4), (2, 5), (3, 6)]
```

```
>>> sorted([5,1,3,4,2])
[1, 2, 3, 4, 5]
```

Son sólo un ejemplo...

+info: <http://docs.python.org/library/functions.html>

Funciones interesantes

```
>>> a = [1,2,3]
>>> b = [4,5,6]
>>> zip(a,b)
[(1, 4), (2, 5), (3, 6)]
```

```
>>> round(1.2345,2)
1.23
```

```
>>> sorted([5,1,3,4,2])
[1, 2, 3, 4, 5]
```

Son sólo un ejemplo...

+info: <http://docs.python.org/library/functions.html>

Funciones interesantes

```
>>> a = [1,2,3]
>>> b = [4,5,6]
>>> zip(a,b)
[(1, 4), (2, 5), (3, 6)]
```

```
>>> round(1.2345,2)
1.23
```

```
>>> sorted([5,1,3,4,2])
[1, 2, 3, 4, 5]
```

```
>>> map(str,[1,2,3,4,5])
['1', '2', '3', '4', '5']
```

Son sólo un ejemplo...

+info: <http://docs.python.org/library/functions.html>

Funciones de ayuda

Funciones de ayuda

```
>>> dir([1,2,3])
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__',
 '__delslice__', '__doc__', '__eq__', '__format__', '__ge__',
 '__getattr__', '__getitem__', '__getslice__', '__gt__', '__hash__',
 '__iadd__', '__imul__', '__init__', '__iter__', '__le__', '__len__',
 '__lt__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__',
 '__repr__', '__reversed__', '__rmul__', '__setattr__', '__setitem__',
 '__setslice__', '__sizeof__', '__str__', '__subclasshook__', 'append',
 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
```

Funciones de ayuda

```
>>> dir([1,2,3])
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__',
 '__delslice__', '__doc__', '__eq__', '__format__', '__ge__',
 '__getattr__', '__getitem__', '__getslice__', '__gt__', '__hash__',
 '__iadd__', '__imul__', '__init__', '__iter__', '__le__', '__len__',
 '__lt__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__',
 '__repr__', '__reversed__', '__rmul__', '__setattr__', '__setitem__',
 '__setslice__', '__sizeof__', '__str__', '__subclasshook__', 'append',
 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
```

```
>>> help(filter)
Help on built-in function filter in module __builtin__:
```

```
filter(...)
```

```
    filter(function or None, sequence) -> list, tuple, or string
```

Return those items of sequence for which function(item) is true. If function is None, return the items that are true. If sequence is a tuple or string, return the same type, else return a list.

```
(END)
```

Classes

```
class Student(object):  
    def __init__(self, name, age):  
        self.name = name  
        self.age = age  
  
    def hello(self):  
        return 'My name is %s' % self.name
```

```
s = Student("Jorge", 24)
```

Classes

1

```
class Student(object):  
    def __init__(self, name, age):  
        self.name = name  
        self.age = age  
  
    def hello(self):  
        return 'My name is %s' % self.name
```

```
s = Student("Jorge", 24)
```

Classes

1

```
class Student(object):
```

2

```
    def __init__(self, name, age):
```

```
        self.name = name
```

```
        self.age = age
```

```
    def hello(self):
```

```
        return 'My name is %s' % self.name
```

```
s = Student("Jorge", 24)
```

Classes

```
1 class Student(object):  
2     3 def __init__(self, name, age):  
        self.name = name  
        self.age = age  
  
    def hello(self):  
        return 'My name is %s' % self.name
```

```
s = Student("Jorge", 24)
```

Classes

```
1 class Student(object):  
2     3 def __init__(self, name, age):  
4         self.name = name  
         self.age = age  
  
    def hello(self):  
        return 'My name is %s' % self.name
```

```
s = Student("Jorge", 24)
```

Classes

```
1 class Student(object):  
2     3 def __init__(self, name, age):  
4         self.name = name  
5         self.age = age  
  
6     def hello(self):  
7         return 'My name is %s' % self.name
```

```
s = Student("Jorge", 24)
```


Classes

```
1 class Student(object):  
2     3 def __init__(self, name, age):  
4         self.name = name  
5         self.age = age  
  
6 def hello(self):  
7     return 'My name is %s' % self.name
```

```
s = Student("Jorge", 24)
```

Recomendaciones:

PEP 8

- CamelCase

El operador ()

- Es importante diferenciar:

`funcion` vs `funcion()`

`Clase` vs `Clase()`

- El operador () permite:

- **Invocar** funciones:

```
resultado = funcion()
```

- **Instanciar** clases:

```
objeto = Clase("param1")
```

Índice

1. Python

- a. Introducción
- b. Tipos de datos
- c. Operadores
- d. Usos frecuentes
- e. Estructuras
- f. Sentencias**
- g. Ejercicio

2. Django

- a. Introducción
- b. URLs y Vistas
- c. Templates
- d. Modelo
- e. Administración
- f. Formularios
- g. Magia avanzada



Proyecto

Sentencias: if-else



```
$name = "Jon";

if($name == "Jon"){
    $name = "Jon Rocks!";
}
elseif($name == "Mary"){
    $name = "Hello Mary";
}
else{
    $name = "Who are you?"
}
```



```
name = "Jon"

if name == "Jon":
    name = "Jon Rocks!"
elif name == "Mary":
    name = "Hello Mary"
else:
    name = "Who are you?"
```

Sentencias: while



```
$count = 0;
while ($count < 5) {
    echo "Number ".$count;
    $count+=1;
}
```



```
count = 0
while count < 5:
    print "Number %i" % count
    count+=1
```

Sentencias: for



```
for ($i=0; $i < 5; $i++) {  
    echo "Number ".$count;  
}
```



```
for i in range(4):  
    print "Number %i" % count
```

Sentencias: try-except



```
try {  
    $result = 3 / 0;  
} catch (Exception $e) {  
    echo "Division by Zero"  
}
```



```
try:  
    result = 3 / 0  
except:  
    print "Division by Zero"
```

Conclusiones

- Python es un lenguaje fácil de aprender.
- Menos código:
 - ~~Muchos~~ Muchísimos menos errores de Sintaxis.
 - Mayor velocidad de escritura.
- ~~Prototipado...~~ Utilizado por grandes empresas.

Conclusiones

- Python es un lenguaje fácil de aprender.
- Menos código:
 - ~~Muchos~~ Muchísimos menos errores de Sintaxis.
 - Mayor velocidad de escritura.
- ~~Prototipado...~~ Utilizado por grandes empresas.

The Google logo, featuring the word "Google" in its characteristic multi-colored font.The Yahoo! logo, featuring the word "YAHOO!" in a purple, stylized font.The IBM logo, featuring the word "IBM" in a blue, striped font.

etc...



Ejemplo & Ejercicio

django

The Web framework for perfectionists with deadlines.

Índice

1. Python

- a. Introducción
- b. Tipos de datos
- c. Operadores
- d. Usos frecuentes
- e. Estructuras
- f. Sentencias
- g. Ejercicio

2. Django

- a. Introducción
- b. URLs y Vistas
- c. Templates
- d. Modelo
- e. Administración
- f. Formularios
- g. Magia avanzada



Proyecto

Evolución de la Web



Desarrollo Web

1ª Generación

HTML
CGI

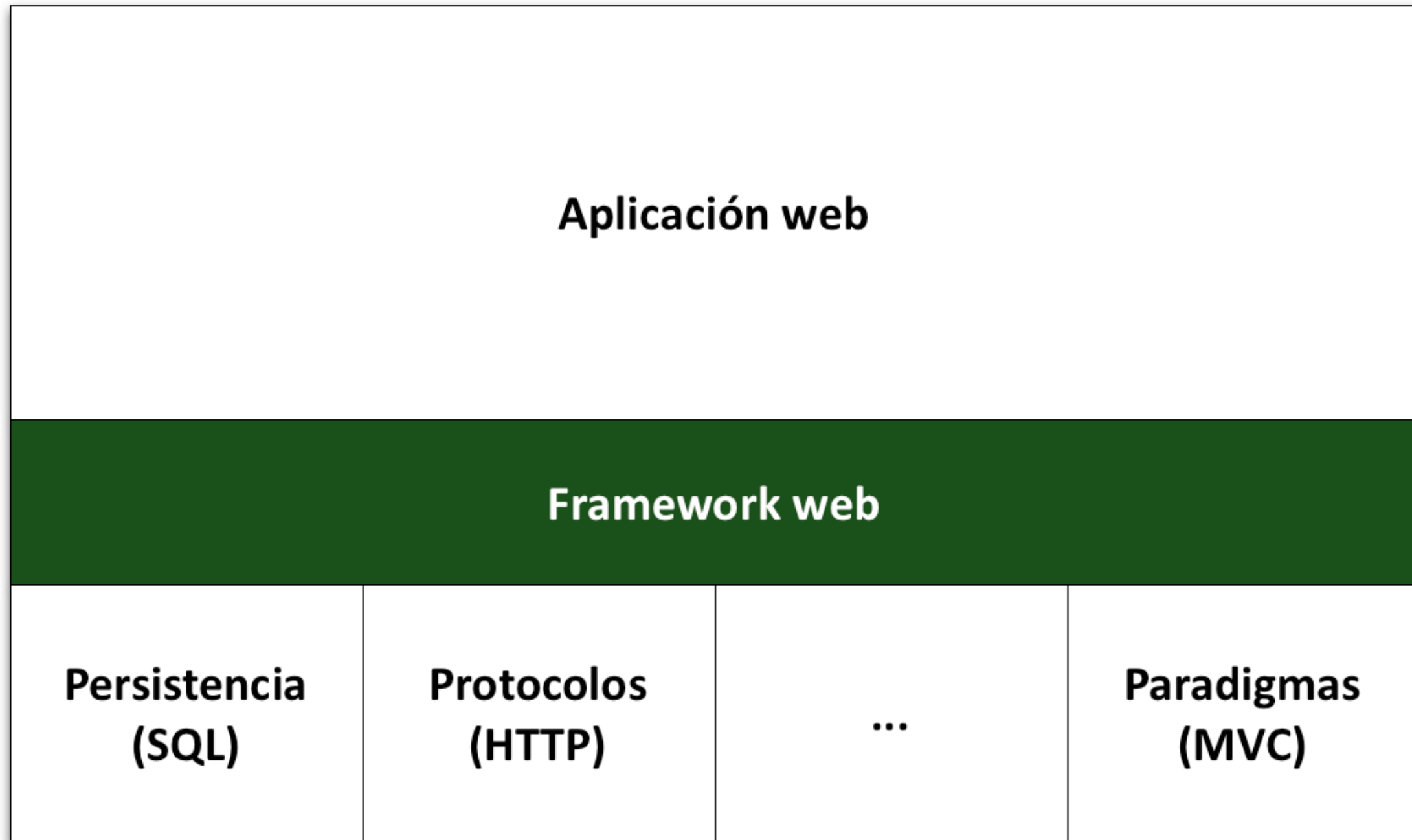
2ª Generación

PHP
ASP
JSP
...

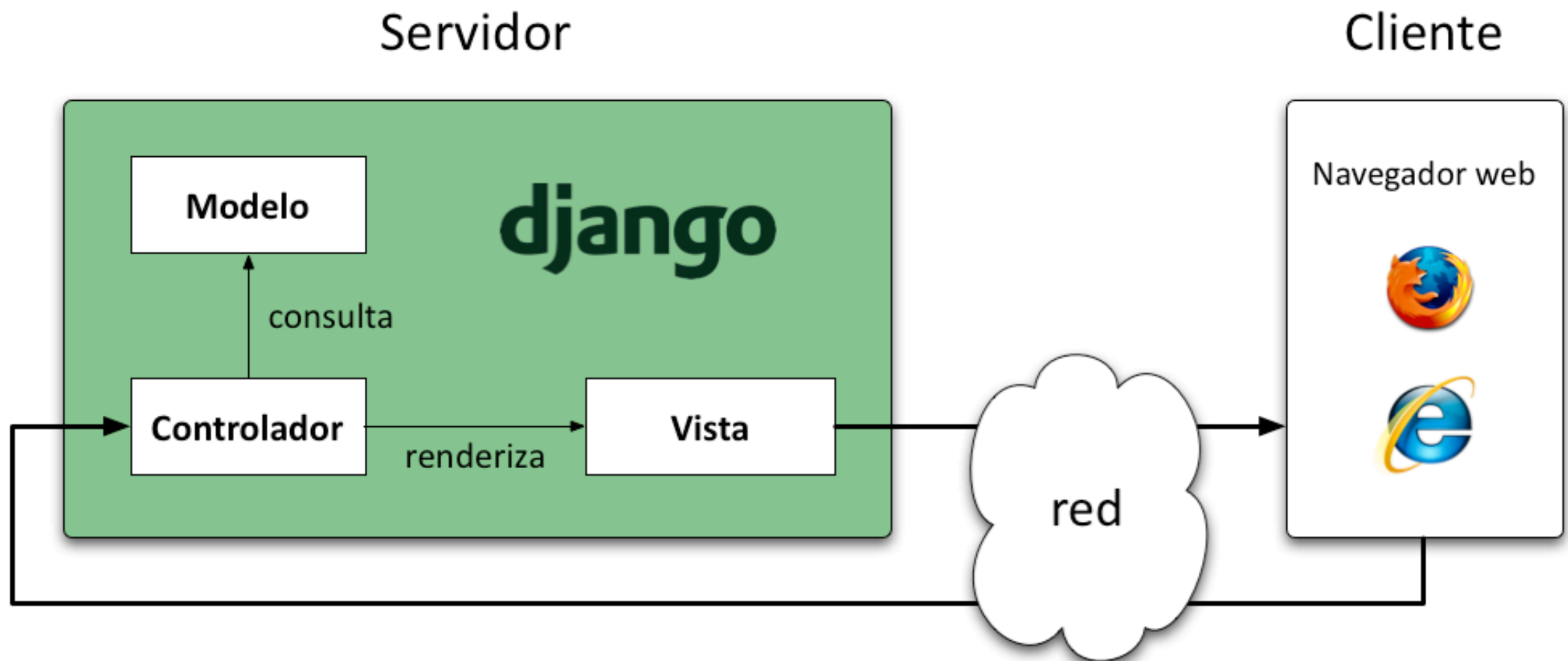
3ª Generación

Django
Rails
Symfony
...

Frameworks web



Django: Qué y dónde



Filosofía

- Loose coupling, Acoplamiento débil.
 - Cada capa es independiente y desconoce completamente a las demás.
- Menos código.
- Rápido desarrollo.
 - Esto es el siglo 21, todo el trabajo tedioso hay que evitarlo.
- Don't Repeat Yourself (DRY)

Filosofía

- Loose coupling, Acoplamiento débil.
 - Cada capa es independiente y desconoce completamente a las demás.
- Menos código.
- Rápido desarrollo.
 - Esto es el siglo 21, todo el trabajo tedioso hay que evitarlo.
- Don't Repeat Yourself (DRY)

“Every distinct concept and/or piece of data should live in one, and only one, place. Redundancy is bad. Normalization is good.”

Filosofía

- Explicit is better than implicit.
 - Este es un principio de Python.
 - Django no debe hacer demasiada “Magia”.
 - Si algo es “Mágico” ha de haber una buena razón.
- Consistencia
 - Ha de ser consistente a todos los niveles.
- Eficiencia, Seguridad, Flexibilidad y Simplicidad.

<http://docs.djangoproject.com/en/dev/misc/design-philosophies/>

La comunidad



La comunidad

- **django-users**

18.000 miembros



La comunidad

- **django-users**

18.000 miembros

- **django-developers**

5.900 miembros



La comunidad

- **django-users**

18.000 miembros

- **django-developers**

5.900 miembros

- **djangoproject.com**

500.000 visitas únicas mensuales



¿Quién usa Django?

¿Quién usa Django?

Google™

YAHOO!®

Discovery
COMMUNICATIONS™

 **the ONION**®
America's Finest News Source

LUCASFILM
Ltd

theguardian

The Washington Post


WALT DISNEY


NATIONAL
GEOGRAPHIC™

The New York Times


six apart

etc...

1. Instalación Python



```
$ sudo apt-get install python
```



<http://www.python.org/download/>



Snow Leopard incluye las versiones 2.5.4 y 2.6.1

2. Instalación SGBD

Configurar un motor de Base de Datos:

- Oficiales:

- PostgreSQL
- MySQL
- Oracle
- **SQLite**

- Comunidad:

- Sybase SQL Anywhere
- IBM DB2
- Microsoft SQL Server 2005
- Firebird
- ODBC

2. Instalación SGBD

Configurar un motor de Base de Datos:

- Oficiales:
 - PostgreSQL
 - MySQL
 - Oracle
 - **SQLite**
- Comunidad:
 - Sybase SQL Anywhere
 - IBM DB2
 - Microsoft SQL Server 2005
 - Firebird
 - ODBC



2. Instalación SGBD

- Vamos a utilizar **SQLite** por comodidad
- Desde Python 2.5 podemos utilizar sqlite sin instalar nada.

2. Instalación SGBD

- Vamos a utilizar **SQLite** por comodidad
- Desde Python 2.5 podemos utilizar sqlite sin instalar nada.



3. Instalación Django

A: Paquetes de cada distro

- `apt-get install python-django`

B: Official Release

- <http://www.djangoproject.com/download/>
- `python setup.py install`

C: Trunk

- <http://code.djangoproject.com/svn/django/trunk/>

All Right?

All Right?

```
>>> import django
>>> django.VERSION
(1, 1, 0, 'final', 0)
```



```
>>> import django
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: No module named django
```





django



django

Bienvenidos al mundo de Oz

Ficheros y Carpetas

¿Es Django tan simple y fácil de usar?



* Incluida la carpeta del proyecto

Ficheros y Carpetas

¿Es Django tan simple y fácil de usar?

Ficheros

Carpetas

* Incluida la carpeta del proyecto



Ficheros y Carpetas

¿Es Django tan simple y fácil de usar?

	Rails
Ficheros	149
Carpetas	35

* Incluida la carpeta del proyecto



Ficheros y Carpetas

¿Es Django tan simple y fácil de usar?

	Rails	Symfony
Ficheros	149	117
Carpetas	35	29

* Incluida la carpeta del proyecto



Ficheros y Carpetas

¿Es Django tan simple y fácil de usar?

	Rails	Symfony	Django
Ficheros	149	117	4
Carpetas	35	29	1

* Incluida la carpeta del proyecto



Let's enjoy



Crear nuestro proyecto

de 4 ficheros ;-)

Crear nuestro proyecto

de 4 ficheros ;-)

```
$ django-admin.py startproject dwitter
```

Crear nuestro proyecto

de 4 ficheros ;-)

```
$ django-admin.py startproject dwitter
```

```
__init__.py  
manage.py  
settings.py  
urls.py
```

Crear nuestro proyecto

de 4 ficheros ;-)

```
$ django-admin.py startproject dwitter
```

```
__init__.py  
manage.py  
settings.py  
urls.py
```

- ¿Esto es un proyecto... ? **Si os ve mi jefe...**

Crear nuestro proyecto

de 4 ficheros ;-)

```
$ django-admin.py startproject dwitter
```

```
__init__.py  
manage.py  
settings.py  
urls.py
```

- ¿Esto es un proyecto... ? **Si os ve mi jefe...**
- **Sí**, disponemos de un proyecto '*funcional*' en django.



settings.py

Arrancar nuestro proyecto

de 4 ficheros ;-)

Arrancar nuestro proyecto

de 4 ficheros ;-)

```
$ cd dwitter
$ python manage.py runserver

Validating models...
0 errors found

Django version 1.1, using settings 'dwitter.settings'
Development server is running at http://
127.0.0.1:8000/
Quit the server with CONTROL-C.
```

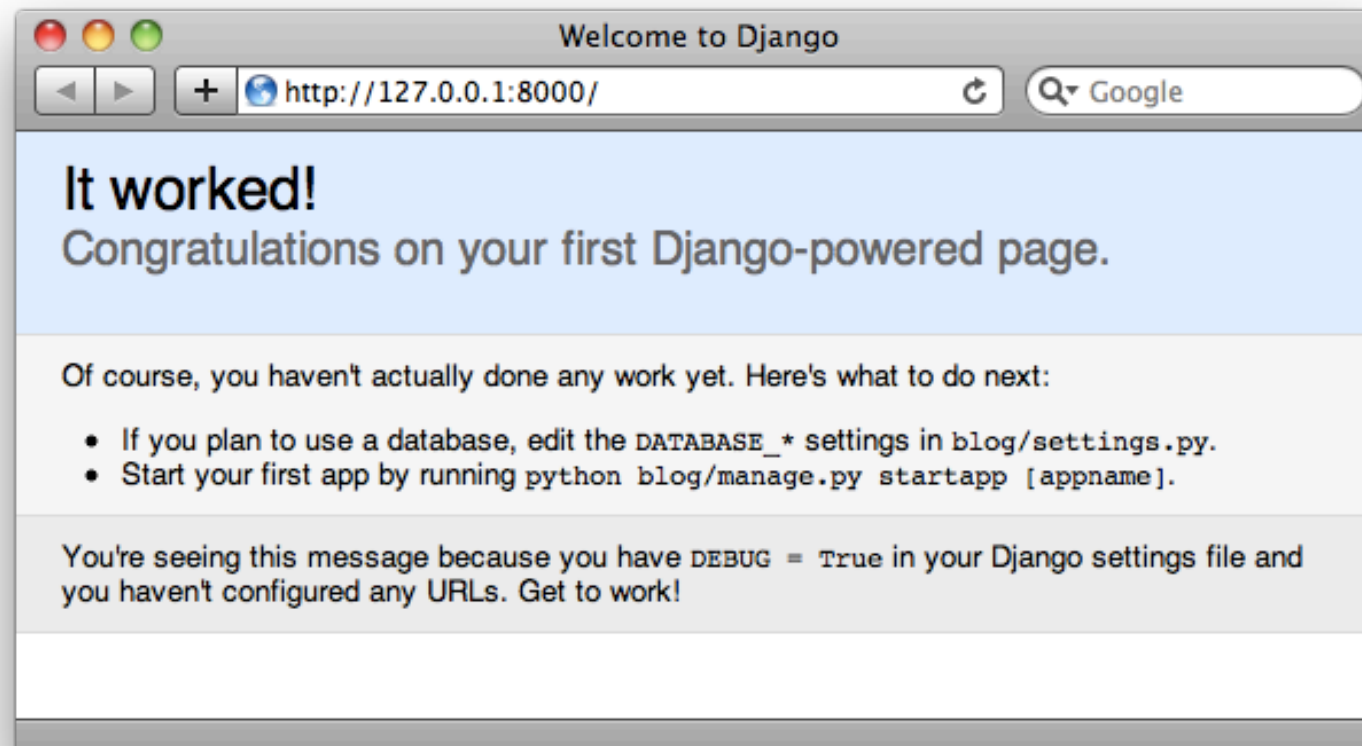

Arrancar nuestro proyecto

de 4 ficheros ;-)

```
$ cd dwitter
$ python manage.py runserver

Validating models...
0 errors found

Django version 1.1, using settings 'dwitter.settings'
Development server is running at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```



Índice

1. Python

- a. Introducción
- b. Tipos de datos
- c. Operadores
- d. Usos frecuentes
- e. Estructuras
- f. Sentencias
- g. Ejercicio

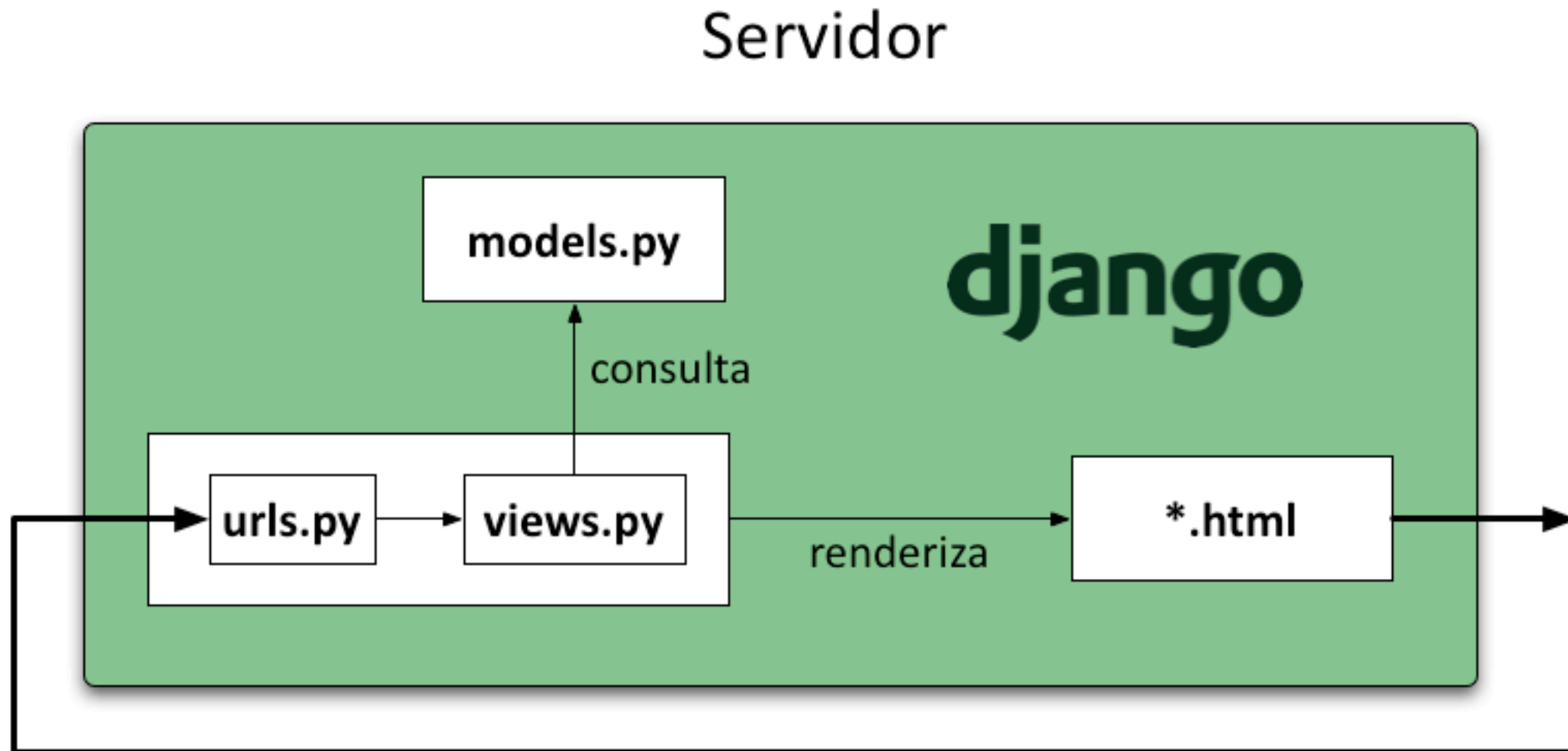
2. Django

- a. Introducción
- b. URLs y Vistas**
- c. Templates
- d. Modelo
- e. Administración
- f. Formularios
- g. Magia avanzada



Proyecto

MVC en Django



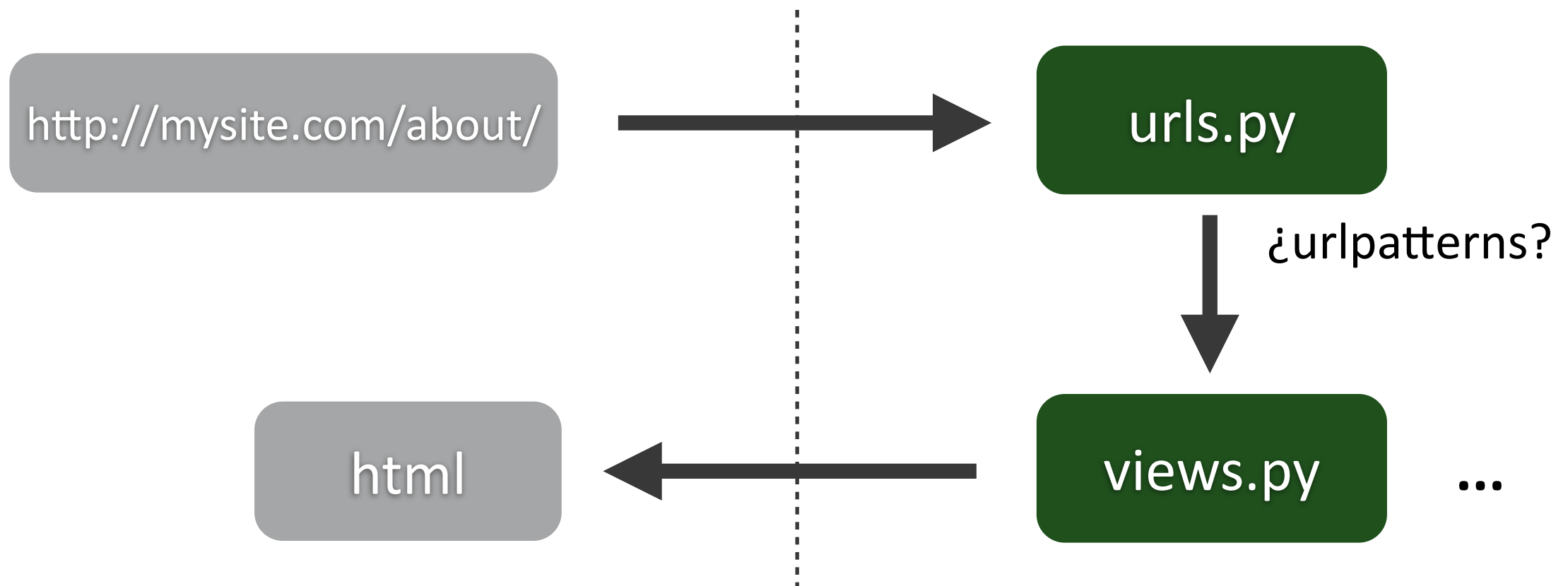
Modelo = Model

Vista = Template

Controlador = URL+View

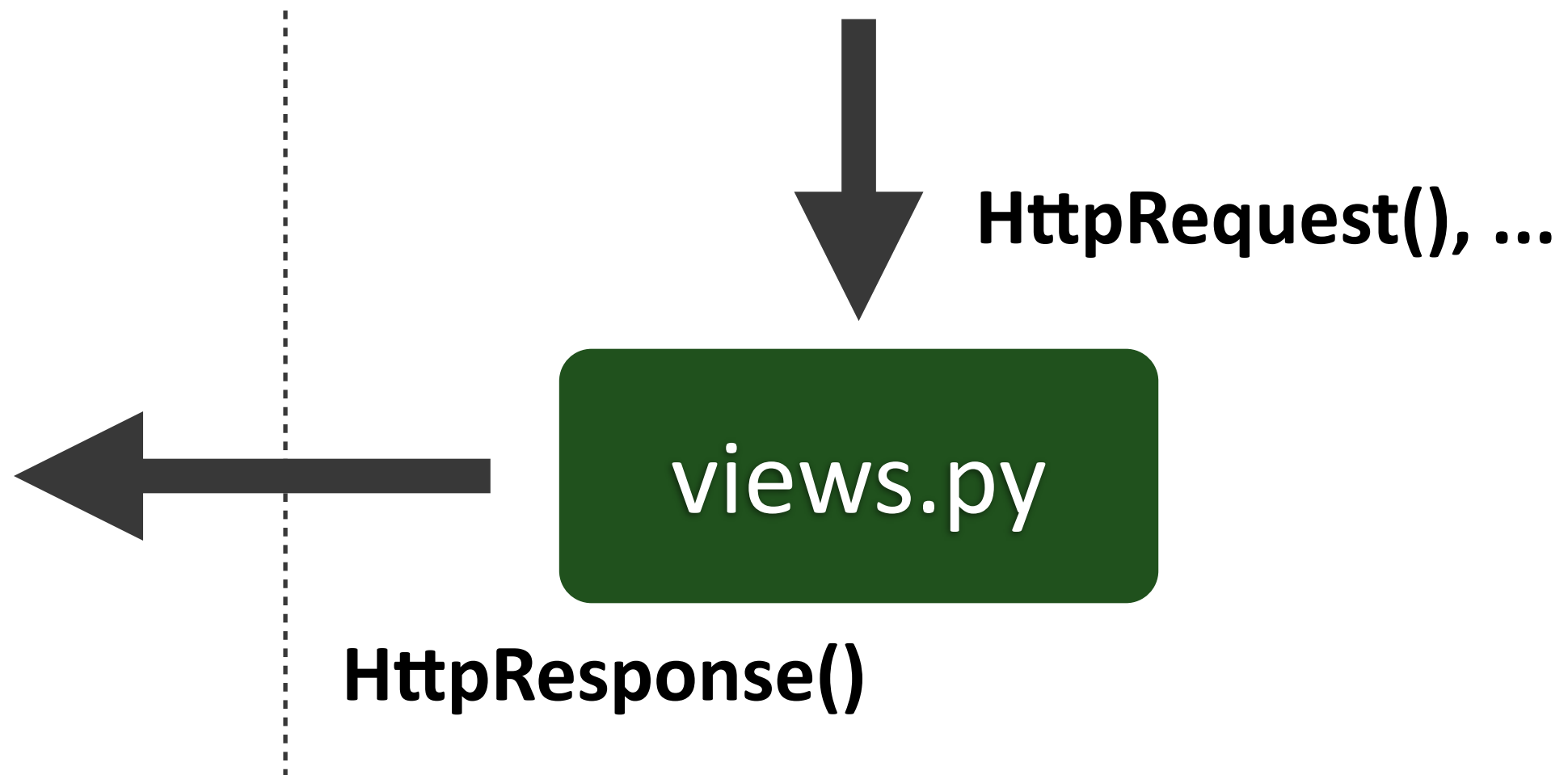
URLs y Vistas

- El fichero **urls.py** actúa como puerta de entrada para las peticiones HTTP
- Se definen URLs elegantes mediante expresiones regulares que redirigen a funciones de **views.py**



URLs y Vistas

- La función de `views.py` recibe como parámetros un objeto **HttpRequest** y todos los parámetros de la URL capturados, teniendo que devolver siempre un objeto **HttpResponse**



URLs y Vistas

Ejemplo 1: `http://mysite.com/time`

urls.py

```
from django.conf.urls.defaults import *
from mysite.views import hora_actual

urlpatterns = patterns('',
    (r'^time/$', hora_actual),
)
```

views.py

```
from django.http import HttpResponse
from datetime import datetime

def hora_actual(request):
    now = datetime.now()
    html = "Son las %s." % now
    return HttpResponse(html)
```

URLs y Vistas

Ejemplo 2: `http://mysite.com/time/plus/2`

urls.py

```
from django.conf.urls.defaults import *
from mysite.views import dentro_de

urlpatterns = patterns('',
    (r'^time/plus/(\d{1,2})/$', dentro_de),
)
```

views.py

```
from django.http import HttpResponse
from datetime import datetime, timedelta

def dentro_de(request, offset):
    offset = int(offset)
    dt = datetime.now() + timedelta(hours=offset)
    html = "En %i hora(s), serán las %s." % (offset, dt)
    return HttpResponse(html)
```




2129

083526

DNK

URLs y Vistas

Ejemplo 2: `http://mysite.com/time/plus/2`

urls.py

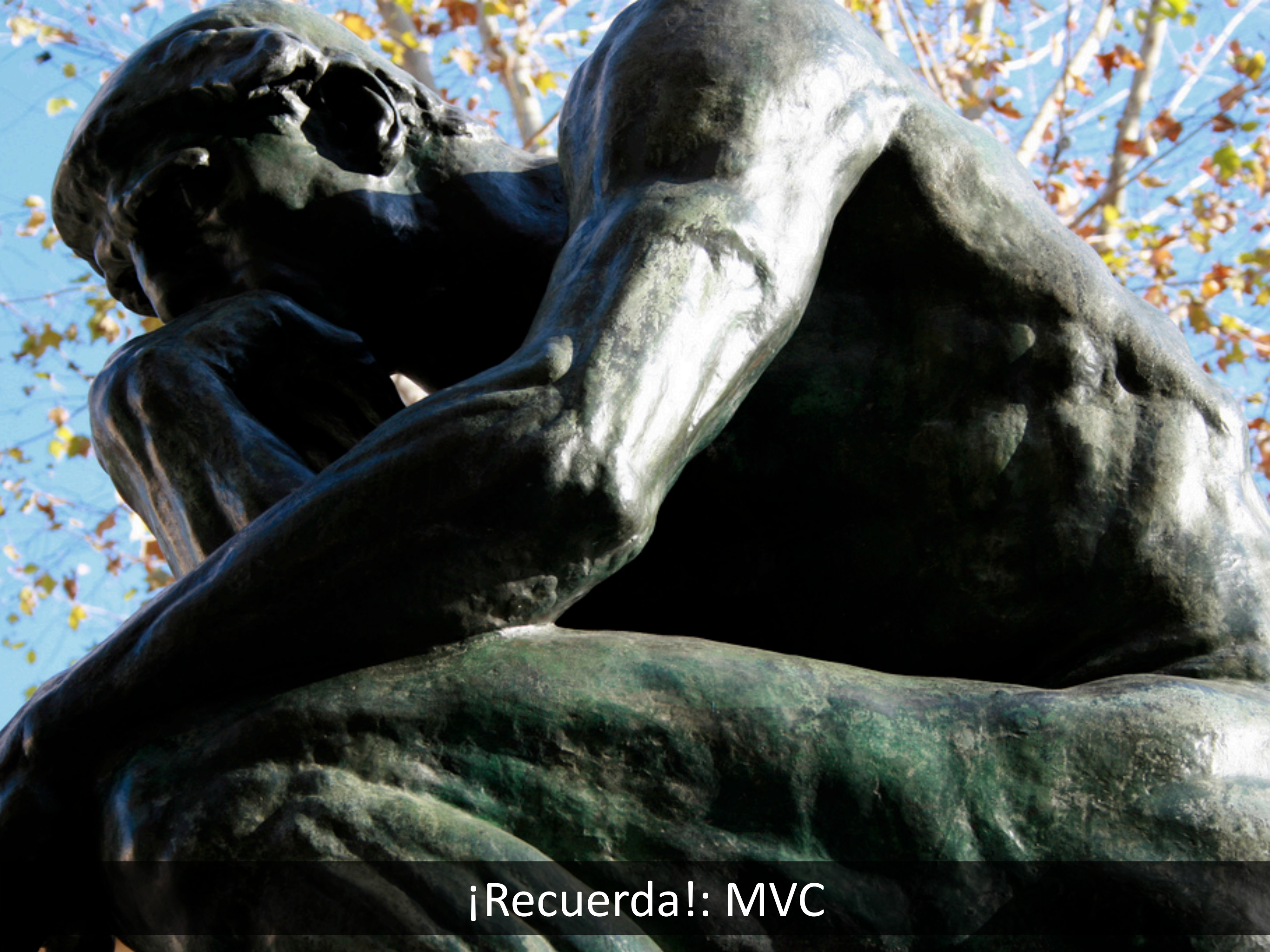
```
from django.conf.urls.defaults import *
from mysite.views import dentro_de

urlpatterns = patterns('',
    (r'^time/plus/(\d{1,2})/$', dentro_de),
)
```

views.py

```
from django.http import HttpResponse
from datetime import datetime, timedelta

def dentro_de(request, offset):
    offset = int(offset)
    dt = datetime.now() + timedelta(hours=offset)
    html = "En %i hora(s), serán las %s." % (offset, dt)
    return HttpResponse(html)
```

¡Recuerda!: MVC

Índice

1. Python

- a. Introducción
- b. Tipos de datos
- c. Operadores
- d. Usos frecuentes
- e. Estructuras
- f. Sentencias
- g. Ejercicio

2. Django

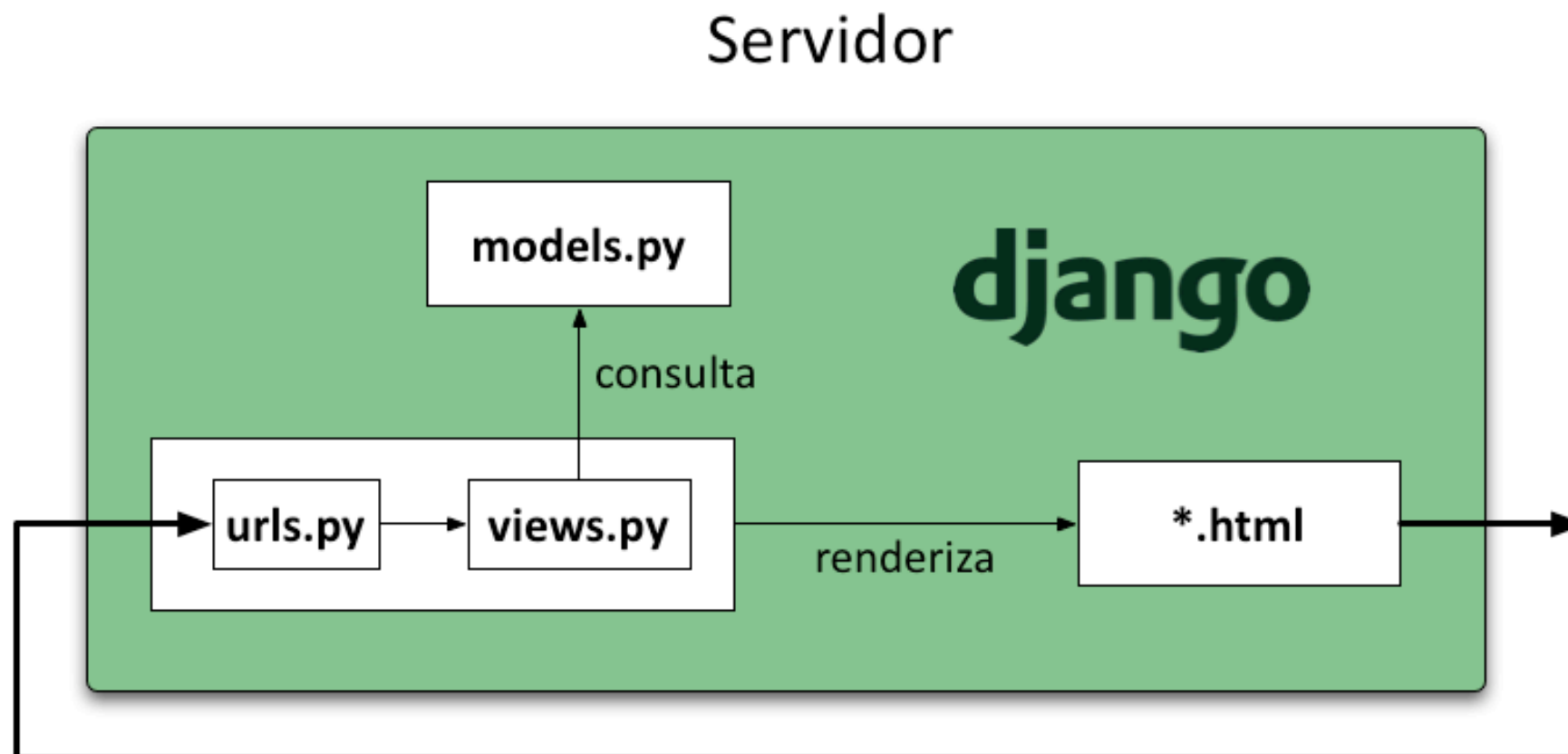
- a. Introducción
- b. URLs y Vistas
- c. Templates**
- d. Modelo
- e. Administración
- f. Formularios
- g. Magia avanzada



Proyecto

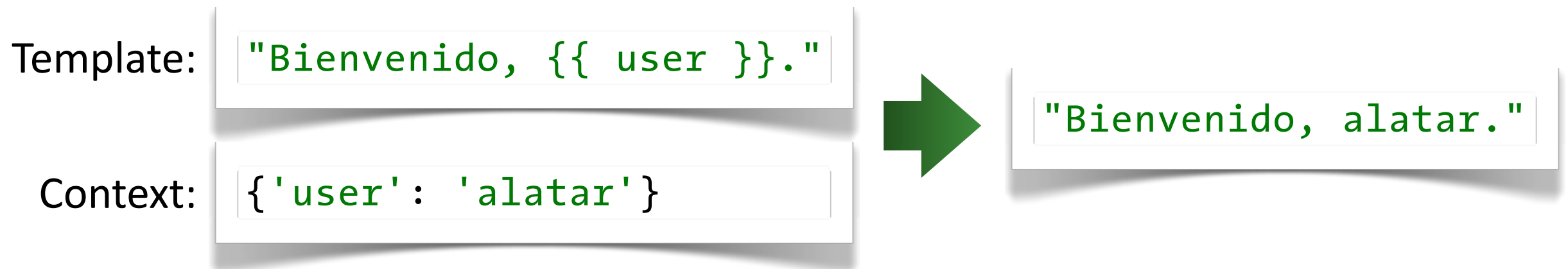
Templates

- Separan la lógica de **presentación** a una capa independiente.
 - Ficheros independientes (.html)
 - Lenguaje independiente (¡para diseñadores!)



Templates

- Se basan en dos tipos de objetos: `Template()` y `Context()`.
 - Un objeto **`Template()`** contiene el **string** de salida que queremos devolver en el `HttpResponse` (normalmente HTML), pero incluyendo etiquetas especiales de Django.
 - Un objeto **`Context()`** contiene un **diccionario** con los valores que dan contexto a una plantilla, los que deben usarse para renderizar un objeto `Template()`.



Templates

- **Primera aproximación al objetivo: Template + Context**

```
from django.http import HttpResponse
from django.template import Template, Context
from datetime import datetime
```

```
PLANTILLA = """<html><body>
Son las {{ hora }}.
</body></html>"""
```

```
def hora_actual(request):
    now = datetime.now()
    t = Template(PLANTILLA)
    c = Context({'hora': now})
    html = t.render(c)
    return HttpResponse(html)
```

Templates

- Segunda aproximación al objetivo: `open()`, `read()`, `close()`

```
from django.http import HttpResponse
from django.template import Template, Context
from datetime import datetime

def hora_actual(request):
    now = datetime.now()
    fp = open('/home/django/templates/hora.html')
    t = Template(fp.read())
    fp.close()
    c = Context({'hora': now})
    html = t.render(c)
    return HttpResponse(html)
```

Templates

- Segunda aproximación al objetivo: `open()`, `read()`, `close()`

```
from django.http import HttpResponse
from django.template import Template, Context
from datetime import datetime

def hora_actual(request):
    now = datetime.now()
    fp = open('/home/django/templates/hora.html')
    t = Template(fp.read())
    fp.close()
    c = Context({'hora': now})
    html = t.render(c)
    return HttpResponse(html)
```


Templates

- Segunda aproximación al objetivo: `open()`, `read()`, `close()`

```
from django.http import HttpResponse
from django.template import Template, Context
from datetime import datetime

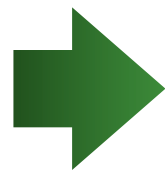
def hora_actual(request):
    now = datetime.now()
    fp = open('/home/django/templ1.html')
    t = Template(fp.read())
    fp.close()
    c = Context({'hora': now})
    html = t.render(c)
    return HttpResponse(html)
```

Boring
boilerplate
code!

Templates

- Tercera aproximación al objetivo: `get_template()`

settings.py



```
TEMPLATE_DIRS = (  
    '/home/django/templates',  
)
```

```
from django.http import HttpResponseRedirect  
from django.template.loader import get_template  
from django.template import Context  
from datetime import datetime  
  
def hora_actual(request):  
    now = datetime.now()  
    t = get_template('hora.html')  
    c = Context({'hora': now})  
    html = t.render(c)  
    return HttpResponseRedirect(html)
```

Templates

- Tercera aproximación al objetivo: `get_template()`

settings.py



```
TEMPLATE_DIRS = (  
    '/home/django/templates',  
)
```

```
from django.http import HttpResponseRedirect  
from django.template.loader import get_template  
from django.template import Context  
from datetime import datetime  
  
def hora_actual(request):  
    now = datetime.now()  
    t = get_template('hora.html')  
    c = Context({'hora': now})  
    html = t.render(c)  
    return HttpResponseRedirect(html)
```

Templates

- Tercera aproximación al objetivo: `get_template()`

settings.py



```
TEMPLATE_DIRS = (  
    '/home/django/templates',  
)
```

```
from django.http import HttpResponseRedirect  
from django.template.loader import get_template  
from django.template import Context  
from datetime import datetime  
  
def hora_actual(request):  
    now = datetime.now()  
    t = get_template('hora.html')  
    c = Context({'hora': now})  
    html = t.render(c)  
    return HttpResponseRedirect(html)
```



Still
boring...

Templates

- Objetivo alcanzado: **render_to_response()**

```
from django.shortcuts import render_to_response
from datetime import datetime

def hora_actual(request):
    now = datetime.now()
    return render_to_response('hora.html', {'hora': now})
```

Templates

- Objetivo alcanzado: **render_to_response()**

```
from django.shortcuts import render_to_response
from datetime import datetime

def hora_actual(request):
    now = datetime.now()
    return render_to_response('hora.html', {'hora': now})
```



Boring...?

Templates

- Objetivo alcanzado: `render_to_response()`

```
from django.shortcuts import render_to_response
from datetime import datetime

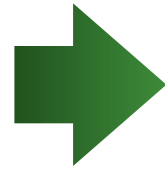
def hora_actual(request):
    now = datetime.now()
    return render_to_response('hora.html', {'hora': now})
```



AWESOME!

Templates: Tip

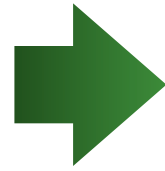
a) settings.py



```
TEMPLATE_DIRS = (  
    '/home/django/templates',  
)
```


Templates: Tip

a) settings.py

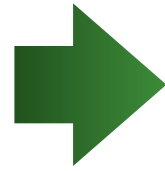


```
TEMPLATE_DIRS = (  
    '/home/django/templates',  
)
```

Reusable
apps?

Templates: Tip

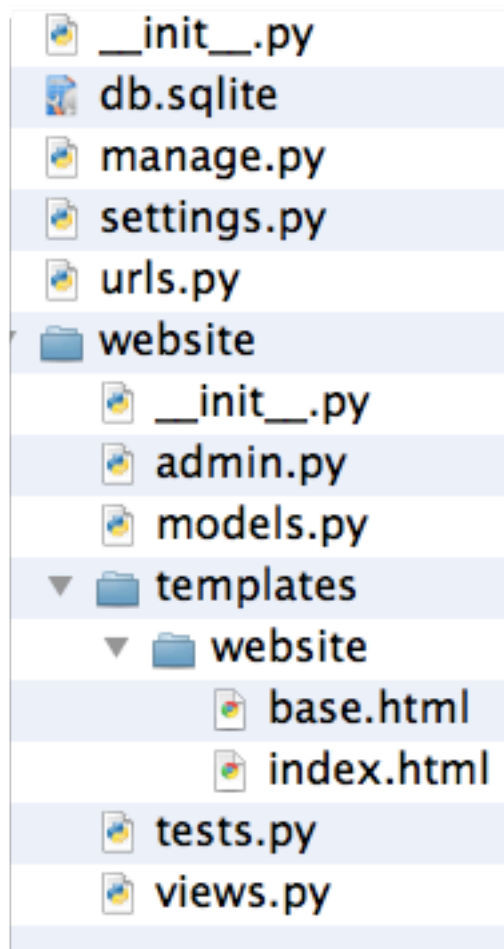
a) settings.py



```
TEMPLATE_DIRS = (  
    '/home/django/templates',  
)
```

Reusable
apps?

b)



Alternativa reutilizable:

```
TEMPLATE_LOADERS = (  
    'django.template.loaders.filesystem.load_template_source',  
    'django.template.loaders.app_directories.load_template_source',  
)
```

- La carpeta **/templates** es buscada dentro de cada app.
- Conviene incluir una carpeta con el nombre de la app por claridad.

```
return render_to_response('website/index.html')
```


- a) Crear la app dwitter.**website**.
- b) Incluir la app en settings.py.
- c) Definir URLs en urls.py:
 - ▶ `user/<username>/status/<tweet_id>/`
 - ▶ `user/<username>/follow/`
 - ▶ `user/<username>/unfollow/`
 - ▶ `user/<username>/`
 - ▶ `search/`
 - ▶ `users/`
- d) Crear una vista para “/” que renderice un timeline de twitter en una template `index.html` a partir de datos hardcodedos en un diccionario.

¡Empieza nuestro primer proyecto!

Templates en detalle

- Si, otro sistema de templates
 - Smarty, Tiles, ClearSilver ...
- Describen cuál va a ser el resultado que ven los usuarios.
 - Desacoplado de Python (Diseñadores muy lejos de Python)
- HTML (o no)... con **esteroides**.
- Muy sencillo de aprender
 - KISS: Keep It Simple, Stupid
- Muy sencillo de extender

Filosofía y Limitaciones

- La sintaxis debe estar desacoplada del HTML/XML.
 - Los diseñadores saben HTML.
 - Los diseñadores no saben Python.
 - No consiste en inventarse un lenguaje.
-
- Una variable no puede cambiar el valor de una variable.
 - Una template no puede ejecutar código Python.

Templates: `{{}}`

```
<html>
  <head>Ejemplo templates</head>
  <body>
    Hola, {{ username }}.
  </body>
</html>
```

```
{'username': 'juan'}
```

```
<html>
  <head>Ejemplo templates</head>
  <body>
    Hola, juan.
  </body>
</html>
```



Tea for One

1981-1990

...nes recalled. "Jimmy
...g about something
...n what happen
...post-bender
...ll—only a ye
...mes the same
...a lot to drink
...d then he doe
...s that [was] lo
...s, with a wife, Pa
...old daughter, Zoë,
...lity at Mill House was the
...on by pulmonary edema
...and having knocked back
...one drink or vodka in the
...and Jimmy was

I was not sufficiently enlightened
the man had nothing to do with
his because lay in the invisibil
generations of men.

Filters y Tags

Filters y Tags

filter

```
{{ variable|filter }}
```

Filters y Tags

filter

```
{{ variable|filter }}
```

inline tag

```
{% tag var1 var2 %}
```

Filters y Tags

filter

```
{{ variable|filter }}
```

inline tag

```
{% tag var1 var2 %}
```

block tag

```
{% tag var1 %}  
...  
{% endtag %}
```

Templates: tags {% %}

Templates: tags {% %}

comment

```
{% comment %} Bu! {% endcomment %}
```

Templates: tags {% %}

comment

```
{% comment %} Bu! {% endcomment %}
```

for

```
{% for elemento in lista %}  
    <li>{{ elemento }}</li>  
{% endfor %}
```

Templates: tags {% %}

comment

```
{% comment %} Bu! {% endcomment %}
```

for

```
{% for elemento in lista %}  
    <li>{{ elemento }}</li>  
{% endfor %}
```

if

```
{% if username == "Juan" %}  
    Hola Juan, me gustas!  
{% else %}  
    Hola {{ username }},  
{% endif %}
```

```
== != > < >= <=  
in and or not
```


Tarea 1: Tags

a) Utilizar tags para que el timeline se muestre con el siguiente formato:

- <username>: <tweet>
- <username>: <tweet>
- ...

Templates: Filters

title

```
<html>
  <head>Ejemplo templates</head>
  <body>
    Hola, {{ username|title }}.
  </body>
</html>
```

```
{'username': 'juan'}
```

```
<html>
  <head>Ejemplo templates</head>
  <body>
    Hola, Juan.
  </body>
</html>
```

Templates: Filters

Templates: Filters

```
{'username': 'Juan es majo'}
```

length

```
{{ username|length }}
```

12

Templates: Filters

```
{'username': 'Juan es majo'}
```

length

```
{{ username|length }}
```

12

cut

```
{{ username|cut }}
```

Juanesmaj

Templates: Filters

```
{'username': 'Juan es majo'}
```

length

```
{{ username|length }}
```

12

cut

```
{{ username|cut }}
```

Juanesmajo

slugify

```
{{ username|slugify }}
```

juan-es-majo

Templates: Filters

```
{'username': 'Juan es majo'}
```

length

```
{{ username|length }}
```

12

cut

```
{{ username|cut }}
```

Juanesmaj

slugify

```
{{ username|slugify }}
```

juan-es-majo

wordcount

```
{{ username|wordcount }}
```

3

Templates: Filters

```
{'username': 'Juan es majo'}
```

length

```
{{ username|length }}
```

12

cut

```
{{ username|cut }}
```

Juanesmaj

slugify

```
{{ username|slugify }}
```

juan-es-majo

wordcount

```
{{ username|wordcount }}
```

3

upper

```
{{ username|upper }}
```

JUAN ES MAJO

Templates: Filters

```
{'username': 'Juan es majo'}
```

length

```
{{ username|length }}
```

12

cut

```
{{ username|cut }}
```

Juanesmaj

slugify

```
{{ username|slugify }}
```

juan-es-majo

wordcount

```
{{ username|wordcount }}
```

3

upper

```
{{ username|upper }}
```

JUAN ES MAJO

```
{'username': None}
```

default

```
{{ username|default:"Desconocido" }}
```

Desconocido

Templates: Filters

Templates: Filters

```
{'username': 'Juan es <b>majo, guapo y <em>listo</em></b>'}
```

striptags

```
{{ username|striptags }}
```

```
Juan es majo guapo y listo
```

Templates: Filters

```
{'username': 'Juan es <b>majo, guapo y <em>listo</em></b>'}
```

striptags

```
{{ username|striptags }}
```

Juan es majo guapo y listo

truncatewords_html

```
{{ username|truncatewords_html:4 }}
```

Juan es majo guapo ...

Templates: Filters

```
{'username': 'Juan es <b>majo, guapo y <em>listo</em></b>'}
```

striptags

```
{{ username|striptags }}
```

Juan es majo guapo y listo

truncatewords_html

```
{{ username|truncatewords_html:4 }}
```

Juan es majo guapo ...

removetags

```
{{ username|removetags:"em a br" }}
```

Juan es majo guapo y listo

Templates: Filters

Templates: Filters

```
{'url': 'Visitad http://www.djangoproject.com'}
```

urlize

```
{{ url|urlize }}
```

```
Visitad <a href="http://www.djangoproject.com">  
http://www.djangoproject.com </a>
```

Templates: Filters

```
{'url': 'Visitad http://www.djangoproject.com'}
```

urlize

```
{{ url|urlize }}
```

```
Visitad <a href="http://www.djangoproject.com">  
http://www.djangoproject.com </a>
```

urlizetrunc

```
{{ url|urlizetrunc:16 }}
```

```
Visitad <a href="http://www.djangoproject.com">  
http://www.djang...</a>
```

Templates: Filters

```
{'lista': ['States', ['Kansas', ['Lawrence', 'Topeka'], 'Illinois']]}
```


Templates: Filters

```
{'lista': ['States', ['Kansas', ['Lawrence', 'Topeka'], 'Illinois']]}
```

unordered_list

```
{{ lista|unordered_list }}
```

```
<li>States
  <ul>
    <li>Kansas
      <ul>
        <li>Lawrence</li>
        <li>Topeka</li>
      </ul>
    </li>
    <li>Illinois</li>
  </ul>
</li>
```

Templates: Filters

Templates: Filters

```
{'value': 123456789}
```

add

```
{{ value|add:"1" }}
```

```
123456790
```

Templates: Filters

```
{'value': 123456789}
```

add

```
{{ value|add:"1" }}
```

123456790

filesizeformat

```
{{ value|filesizeformat }}
```

117.7MB

Templates: Filters

```
{'value': 123456789}
```

add

```
{{ value|add:"1" }}
```

123456790

filesizeformat

```
{{ value|filesizeformat }}
```

117.7MB

```
{'date': datetime.datetime(2010, 9, 11, 17, 1, 59, 385323) }
```

date

```
{{ date|date:"d M Y" }}
```

11 Sep 2010

Templates: Filters

```
{'value': 123456789}
```

add

```
{{ value|add:"1" }}
```

```
123456790
```

filesizeformat

```
{{ value|filesizeformat }}
```

```
117.7MB
```

```
{'date': datetime.datetime(2010, 9, 11, 17, 1, 59, 385323) }
```

date

```
{{ date|date:"d M Y" }}
```

```
11 Sep 2010
```

timesince

```
{{ date|timesince }}
```

```
4 days, 6 hours
```

Templates: Filters

```
{'value': 123456789}
```

add

```
{{ value|add:"1" }}
```

```
123456790
```

filesizeformat

```
{{ value|filesizeformat }}
```

```
117.7MB
```

```
{'date': datetime.datetime(2010, 9, 11, 17, 1, 59, 385323) }
```

date

```
{{ date|date:"d M Y" }}
```

```
11 Sep 2010
```

timesince

```
{{ date|timesince }}
```

```
4 days, 6 hours
```

timeuntil

```
{{ date|timeuntil }}
```

```
1 days, 6 hours
```


Tarea 2: Filters

a) Añadir a cada tweet del timeline el tiempo transcurrido:

- <username>: <tweet>
 <n> seconds ago
- <username>: <tweet>
 <n> minutes ago
- ...

Templates: tags {% %}

Templates: tags {% %}

cycle

```
{% for elemento in lista %}  
    <li class="{% cycle 'rojo' 'azul' %}">{{ elemento }}</li>  
{% endfor %}
```

Templates: tags {% %}

cycle

```
{% for elemento in lista %}  
    <li class="{% cycle 'rojo' 'azul' %}">{{ elemento }}</li>  
{% endfor %}
```

include

```
{% include "foo/bar.html" %}
```

Templates: tags {% %}

cycle

```
{% for elemento in lista %}  
    <li class="{% cycle 'rojo' 'azul' %}">{{ elemento }}</li>  
{% endfor %}
```

include

```
{% include "foo/bar.html" %}
```

forloop

```
{% for elemento in lista %}  
    <li>{{ forloop.counter }}.{{ elemento }}</li>  
{% endfor %}
```

Templates: tags {% %}

cycle

```
{% for elemento in lista %}  
    <li class="{% cycle 'rojo' 'azul' %}">{{ elemento }}<li>  
{% endfor %}
```

include

```
{% include "foo/bar.html" %}
```

forloop

```
{% for elemento in lista %}  
    <li>{{ forloop.counter }}.{{ elemento }}<li>  
{% endfor %}
```

empty

```
{% for elemento in lista %}  
    <li class="{% cycle 'rojo' 'azul' %}">{{ elemento }}<li>  
{% empty %}  
    Sin elementos.  
{% endfor %}
```


Tarea 3: Tags avanzados

- a) Hacer que el timeline muestre el mensaje “No hay tweets.” si el diccionario recibido está vacío.

extend y block

base.html

```
<html>
  <head>
    <title>Mi página personal</title>
  </head>
  <body>
    {% block content %}
      Contenido por defecto.
    {% endblock %}
  </body>
</html>
```

base.html



hija.html

hija.html

```
{% extends "base.html" %}
{% block content %}
  Hola desde la portada.
{% endblock %}
```


Tarea 4: Herencia de templates

- a) Descargar nuestra template base.html.
- b) Incluirla en vuestra app.
- c) Hacer que vuestro index.html herede de base.html y extienda el bloque 'content' con su contenido.

Índice

1. Python

- a. Introducción
- b. Tipos de datos
- c. Operadores
- d. Usos frecuentes
- e. Estructuras
- f. Sentencias
- g. Ejercicio

2. Django

- a. Introducción
- b. URLs y Vistas
- c. Templates
- d. Modelo**
- e. Administración
- f. Formularios
- g. Magia avanzada



Proyecto



Modelos

acos_2110
UPDATE

SET des

WHERE

¿SQL?

Ejemplo SQL

```
def book_list(request):
    try:
        db = MySQLdb.connect(user='me', db='mydb',
                               passwd='secret', host='localhost')

        cursor = db.cursor()
        cursor.execute('SELECT nama FROM books ORDER BY name')
        names = []
        for row in cursor.fetchall():
            names.append(row[0])
        db.close()
    except:
        return render_to_response('500.html')
    return render_to_response('book_list.html', {'names':names})
```

Ejemplo SQL

```
def book_list(request):  
    try: 1  
        db = MySQLdb.connect(user='me', db='mydb',  
                               passwd='secret', host='localhost')  
  
        cursor = db.cursor()  
        cursor.execute('SELECT nama FROM books ORDER BY name')  
        names = []  
        for row in cursor.fetchall():  
            names.append(row[0])  
        db.close()  
    except:  
        return render_to_response('500.html')  
    return render_to_response('book_list.html', {'names':names})
```

Ejemplo SQL

```
def book_list(request):  
    try: 1  
        db = MySQLdb.connect(user='me', db='mydb', 2  
                               passwd='secret', host='localhost')  
        cursor = db.cursor()  
        cursor.execute('SELECT nama FROM books ORDER BY name')  
        names = []  
        for row in cursor.fetchall():  
            names.append(row[0])  
        db.close()  
    except:  
        return render_to_response('500.html')  
    return render_to_response('book_list.html', {'names':names})
```

Ejemplo SQL

```
def book_list(request):  
    try:  
        db = MySQLdb.connect(user='me', db='mydb',  
                               passwd='secret', host='localhost')  
        cursor = db.cursor()  
        cursor.execute('SELECT nama FROM books ORDER BY name')  
        names = []  
        for row in cursor.fetchall():  
            names.append(row[0])  
        db.close()  
    except:  
        return render_to_response('500.html')  
    return render_to_response('book_list.html', {'names':names})
```

Ejemplo SQL

```
def book_list(request):  
    try:  
        db = MySQLdb.connect(user='me', db='mydb',  
                               passwd='secret', host='localhost')  
        cursor = db.cursor()  
        cursor.execute('SELECT nama FROM books ORDER BY name')  
        names = []  
        for row in cursor.fetchall():  
            names.append(row[0])  
        db.close()  
    except:  
        return render_to_response('500.html')  
    return render_to_response('book_list.html', {'names':names})
```


Ejemplo SQL

```
def book_list(request):  
    try:  
        db = MySQLdb.connect(user='me', db='mydb',  
                               passwd='secret', host='localhost')  
        cursor = db.cursor()  
        cursor.execute('SELECT nama FROM books ORDER BY name')  
        names = []  
        for row in cursor.fetchall():  
            names.append(row[0])  
        db.close()  
    except:  
        return render_to_response('500.html')  
    return render_to_response('book_list.html', {'names':names})
```

Ejemplo SQL

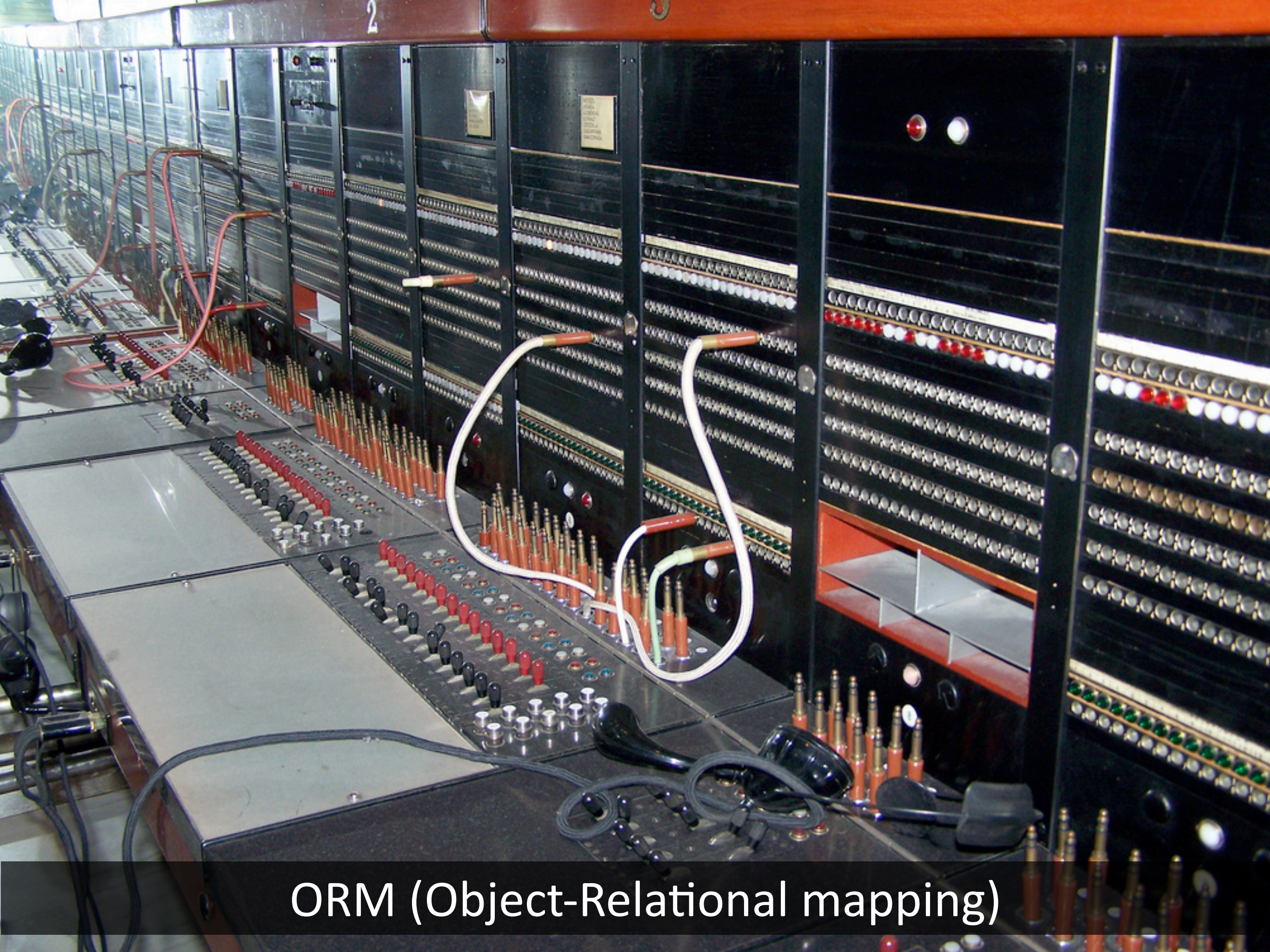
```
def book_list(request):  
    try:  
        db = MySQLdb.connect(user='me', db='mydb',  
                               passwd='secret', host='localhost')  
        cursor = db.cursor()  
        cursor.execute('SELECT nama FROM books ORDER BY name')  
        names = []  
        for row in cursor.fetchall():  
            names.append(row[0])  
        db.close()  
    except:  
        return render_to_response('500.html')  
    return render_to_response('book_list.html', {'names':names})
```




12 lineas de Python... ٧_٧



ORM (Object-Relational mapping)



ORM (Object-Relational mapping)

Ejemplo ORM

```
def book_list(request):  
    names = Books.objects.all().order_by('name')  
    return render_to_response('book_list.html', {'names':names})
```

Ejemplo ORM

```
def book_list(request):  
    1 names = Books.objects.all().order_by('name')  
    return render_to_response('book_list.html', {'names':names})
```

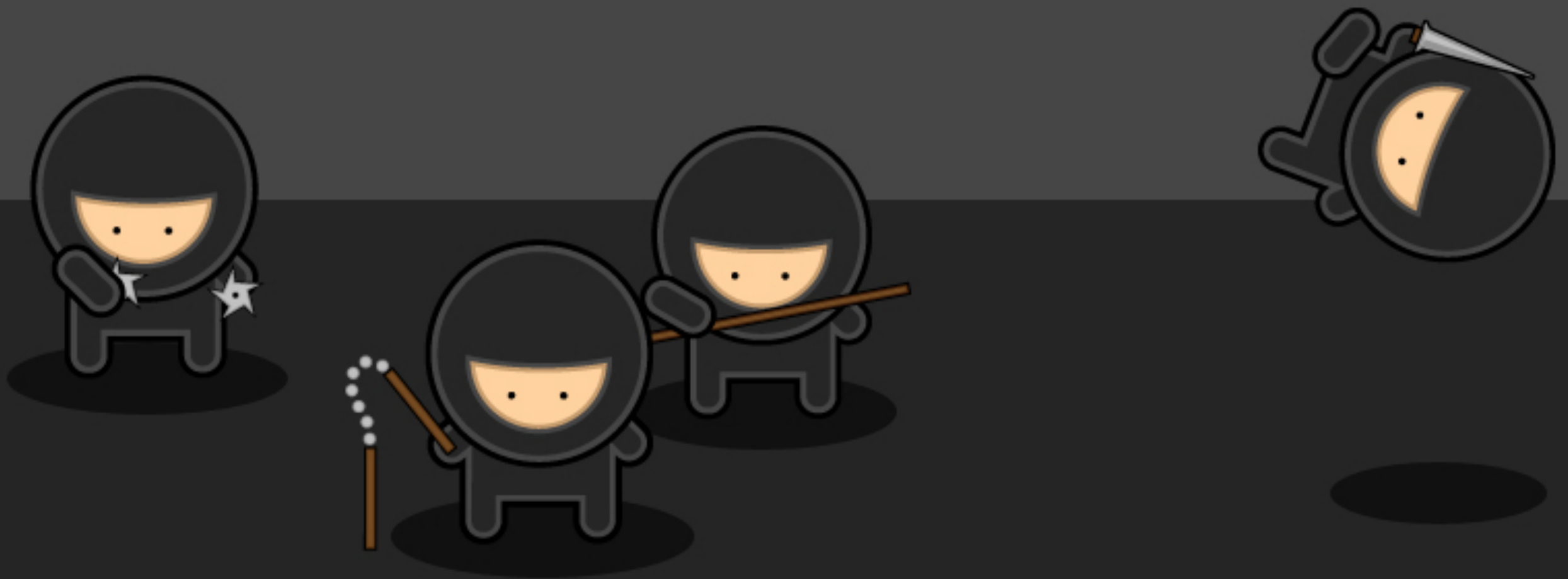
Ejemplo ORM

```
def book_list(request):
```

```
1 names = Books.objects.all().order_by('name')
```

2

```
    return render_to_response('book_list.html', {'names':names})
```

Fucking Ninjas!

Modelo

```
from django.db import models

class Books(models.Model):
    name = models.CharField(blank=True, max_length=100)
    created = models.DateTimeField(blank=False)
    available = models.BooleanField(default=True)
```

- **Independencia SGBD!**
 - Definimos estructuras de información **genéricas**.
 - Definimos **restricciones** (*notnull, blank, max_lenght...*)
- **Única** definición del modelo (*configuración, mapeo a db*)

Modelo

```
from django.db import models
```

1

```
class Books(models.Model):
```

```
    name = models.CharField(blank=True, max_length=100)
```

```
    created = models.DateTimeField(blank=False)
```

```
    available = models.BooleanField(default=True)
```

- **Independencia SGBD!**
 - Definimos estructuras de información **genéricas**.
 - Definimos **restricciones** (*notnull, blank, max_lenght...*)
- **Única** definición del modelo (*configuración, mapeo a db*)

Modelo

```
from django.db import models
```

1

2

```
class Books(models.Model):  
    name = models.CharField(blank=True, max_length=100)  
    created = models.DateTimeField(blank=False)  
    available = models.BooleanField(default=True)
```

- Independencia SGBD!
 - Definimos estructuras de información **genéricas**.
 - Definimos **restricciones** (*notnull, blank, max_lenght...*)
- Única definición del modelo (*configuración, mapeo a db*)

Modelo

```
from django.db import models
```

1

3

2

```
class Books(models.Model):
```

```
    name = models.CharField(blank=True, max_length=100)
```

```
    created = models.DateTimeField(blank=False)
```

```
    available = models.BooleanField(default=True)
```

- Independencia SGBD!
 - Definimos estructuras de información **genéricas**.
 - Definimos **restricciones** (*notnull, blank, max_lenght...*)
- Única definición del modelo (*configuración, mapeo a db*)

Modelo

```
from django.db import models 1
class Books(models.Model): 2
    name = models.CharField(blank=True, max_length=100) 3 4
    created = models.DateTimeField(blank=False)
    available = models.BooleanField(default=True)
```

- Independencia SGBD!
 - Definimos estructuras de información **genéricas**.
 - Definimos **restricciones** (*notnull, blank, max_lenght...*)
- Única definición del modelo (*configuración, mapeo a db*)

Modelo

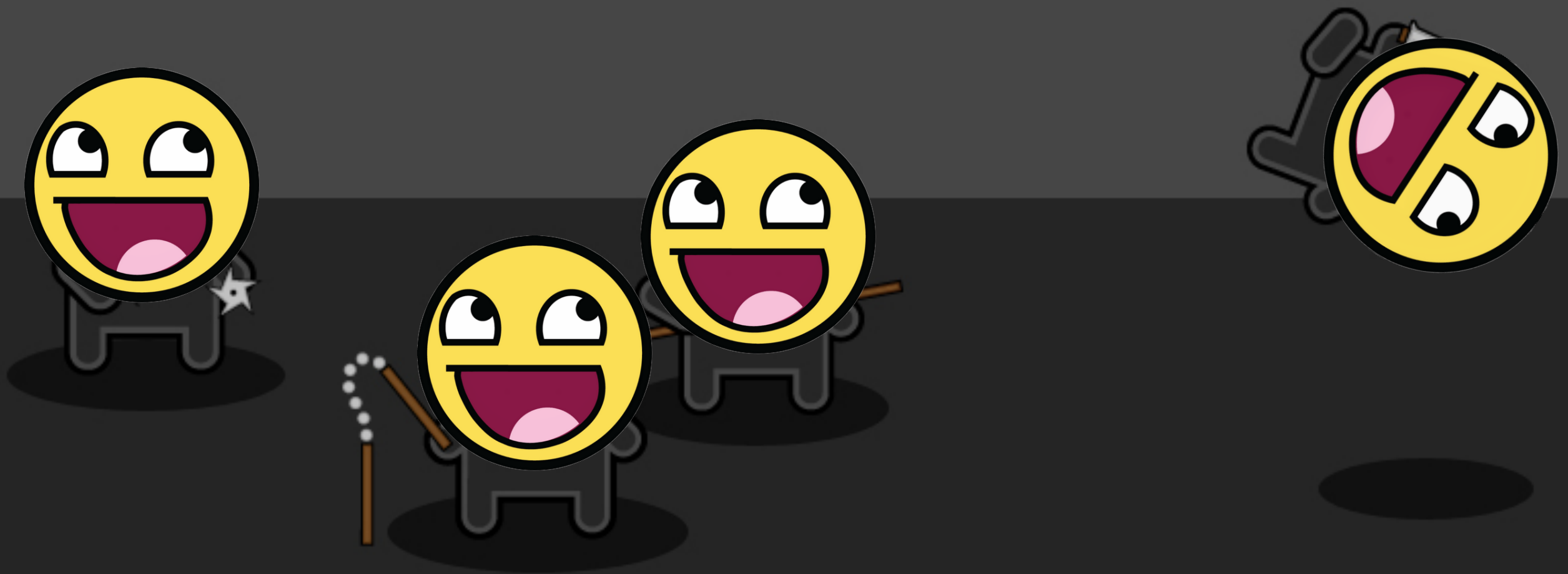
```
from django.db import models 1
class Books(models.Model): 2
    name = models.CharField(blank=True, max_length=100) 3 4
    created = models.DateTimeField(blank=False) 5
    available = models.BooleanField(default=True)
```

- Independencia SGBD!
 - Definimos estructuras de información **genéricas**.
 - Definimos **restricciones** (*notnull, blank, max_lenght...*)
- Única definición del modelo (*configuración, mapeo a db*)

Modelo

```
from django.db import models 1
class Books(models.Model): 2
    name = models.CharField(blank=True, max_length=100) 3 4
    created = models.DateTimeField(blank=False) 5
    available = models.BooleanField(default=True) 6
```

- Independencia SGBD!
 - Definimos estructuras de información **genéricas**.
 - Definimos **restricciones** (*notnull, blank, max_lenght...*)
- Única definición del modelo (*configuración, mapeo a db*)



Fucking Awesome Ninjas!

Tipos de Datos

- AutoField
- BigIntegerField
- BooleanField
- CharField
- CommaSeparatedIntegerField
- DateField
- DateTimeField
- DecimalField
- EmailField
- FileField
- FilePathField
- FloatField
- ImageField
- IntegerField
- IPAddressField
- NullBooleanField
- PositiveIntegerField
- PositiveSmallIntegerField
- SlugField
- SmallIntegerField
- TextField
- TimeField
- URLField
- XMLField
- ForeignKey
- ManyToManyField
- OneToOneField

Tipos de Datos

- AutoField
- BigIntegerField
- BooleanField
- CharField
- CommaSeparatedIntegerField
- DateField
- DateTimeField
- DecimalField
- EmailField
- FileField
- FilePathField
- FloatField
- ImageField
- IntegerField
- IPAddressField
- NullBooleanField
- PositiveIntegerField
- PositiveSmallIntegerField
- SlugField
- SmallIntegerField
- TextField
- TimeField
- URLField
- XMLField
- ForeignKey
- ManyToManyField
- OneToOneField

Propiedades de las Field

- `null (True|False)`
- `blank (True|False)`
- `choices (lista)`
- `default (valor)`
- `editable (True|False)`
- `help_text (String)`
- `unique (True|False)`
- `primary_key`
- `unique_for_date`
- `unique_for_month`
- `unique_for_year`

Propiedades de las Field

- `null (True|False)`
- `blank (True|False)`
- `choices (lista)`
- `default (valor)`
- `editable (True|False)`
- `help_text (String)`
- `unique (True|False)`
- `primary_key`
- `unique_for_date`
- `unique_for_month`
- `unique_for_year`

¿Es magia? No.

```
BEGIN;  
CREATE TABLE "website_books" (  
    "id" integer NOT NULL PRIMARY KEY,  
    "name" varchar(100) NOT NULL,  
    "created" datetime NOT NULL,  
    "available" bool NOT NULL  
);  
COMMIT;
```

```
BEGIN;  
CREATE TABLE "website_books" (  
    "id" serial NOT NULL PRIMARY KEY,  
    "name" varchar(100) NOT NULL,  
    "created" timestamp with time zone NOT NULL,  
    "available" boolean NOT NULL  
);  
COMMIT;
```


¿Es magia? No.

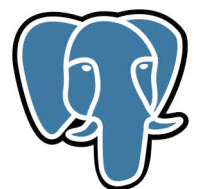
```
$ python manage.py sqlall website
```

```
BEGIN;  
CREATE TABLE "website_books" (  
    "id" integer NOT NULL PRIMARY KEY,  
    "name" varchar(100) NOT NULL,  
    "created" datetime NOT NULL,  
    "available" bool NOT NULL  
);  
COMMIT;
```

SQLite

```
BEGIN;  
CREATE TABLE "website_books" (  
    "id" serial NOT NULL PRIMARY KEY,  
    "name" varchar(100) NOT NULL,  
    "created" timestamp with time zone NOT NULL,  
    "available" boolean NOT NULL  
);  
COMMIT;
```

PostgreSQL



¿Es magia? No.

- Nombres de tablas generados automáticamente.
 - `<app_name>_lower(<model_name>)`
- **id** como Primary Key (*Personalizable*)
- Las Foreign Key terminan en **_id** (*Personalizable*)
- Los tipos de datos se ajustan en función del SGBD

Configurar settings.py

```
DATABASE_ENGINE = 'sqlite3'  
DATABASE_NAME = 'db.sqlite'  
DATABASE_USER = ''  
DATABASE_PASSWORD = ''  
DATABASE_HOST = ''  
DATABASE_PORT = ''
```

Configurar settings.py

```
DATABASE_ENGINE = 'sqlite3'  
DATABASE_NAME = 'db.sqlite'  
DATABASE_USER = ''  
DATABASE_PASSWORD = ''  
DATABASE_HOST = ''  
DATABASE_PORT = ''
```

1

Configurar settings.py

```
DATABASE_ENGINE = 'sqlite3'  
DATABASE_NAME = 'db.sqlite'  
DATABASE_USER = ''  
DATABASE_PASSWORD = ''  
DATABASE_HOST = ''  
DATABASE_PORT = ''
```

1

2

Configurar settings.py

```
DATABASE_ENGINE = 'sqlite3'  
DATABASE_NAME = 'db.sqlite'  
DATABASE_USER = ''  
DATABASE_PASSWORD = ''  
DATABASE_HOST = ''  
DATABASE_PORT = ''
```

1

2

3

Creando Tablas

- Crea las tablas para **todos** los modelos de las **apps instaladas** en el fichero settings.py
- **No actualiza esquemas** si la tabla existe.
 - Eliminar tabla y volver a ejecutar **syncdb**

Creando Tablas

```
$ python manage.py syncdb
```

- Crea las tablas para **todos** los modelos de las **apps instaladas** en el fichero settings.py
- **No actualiza esquemas** si la tabla existe.
 - Eliminar tabla y volver a ejecutar **syncdb**

- ```
Manager.py
// 4 objects to
// lets load some
var i = 0; i < 4; i++
pick a random object
number = Math.floor(Math
object = libraryArrange(number)
screen
attachMove(object)
```
- a) Crear modelo Tweet
  - b) Configurar settings.py
  - c) Ver el SQL generado con sqlalchemy

```
var
// chuck
clip = screen
clip = screen
dragging mech
onpress = function
// to do
start move
number
```

Nuestro primer modelo



# syncdb

```
$ python manage.py syncdb
Creating table auth_permission
Creating table auth_group
Creating table auth_user
Creating table auth_message
Creating table django_content_type
Creating table django_session
Creating table django_site
Creating table website_tweet
```

You just installed Django's auth system, which means you don't have any superusers defined.

Would you like to create one now? (yes/no): yes

Username (Leave blank to use 'neo'): admin

E-mail address: user@example.com

Password:

Password (again):

Superuser created successfully.

Installing index for auth.Permission model

Installing index for auth.Message model

Installing index for website.Tweet model

# syncdb

```
$ python manage.py syncdb
Creating table auth_permission
Creating table auth_group
Creating table auth_user
Creating table auth_message
Creating table django_content_type
Creating table django_session
Creating table django_site
Creating table website_tweet
```

django.contrib.auth

You just installed Django's auth system, which means you don't have any superusers defined.

Would you like to create one now? (yes/no): yes

Username (Leave blank to use 'neo'): admin

E-mail address: user@example.com

Password:

Password (again):

Superuser created successfully.

Installing index for auth.Permission model

Installing index for auth.Message model

Installing index for website.Tweet model





¿Cuántos sistemas de Autenticación habéis programado?





¿Alguno era mejor que el anterior? ;-)





contrib.auth puede ser el último :D



# syncdb

```
$ python manage.py syncdb
Creating table auth_permission
Creating table auth_group
Creating table auth_user
Creating table auth_message
Creating table django_content_type
Creating table django_session
Creating table django_site
Creating table website_tweet
```

website.tweet

You just installed Django's auth system, which means you don't have any superusers defined.

Would you like to create one now? (yes/no): yes

Username (Leave blank to use 'neo'): admin

E-mail address: user@example.com

Password:

Password (again):

Superuser created successfully.

Installing index for auth.Permission model

Installing index for auth.Message model

Installing index for website.Tweet model

# syncdb

```
$ python manage.py syncdb
Creating table auth_permission
Creating table auth_group
Creating table auth_user
Creating table auth_message
Creating table django_content_type
Creating table django_session
Creating table django_site
Creating table website_tweet
```

You just installed Django's auth system, which means you don't have any superusers defined.

Would you like to create one now? (yes/no): yes

Username (Leave blank to use 'neo'): admin

E-mail address: user@example.com

Password:

Password (again):

Superuser created successfully.

Installing index for auth.Permission model

Installing index for auth.Message model

Installing index for website.Tweet model

django.contrib.auth

**BD Lista para usarse**



**BD Lista para usarse**



```
Manager.prototype.loadSomeObjects = function() {
 // 4 objects to load
 for(var i = 0; i < 4; i++) {
 // pick a random object
 var number = Math.floor(Math.random() * 1000);
 var object = libraryArrange[number];
 // add object to screen
 puzzle.attachMove(object, random location);
 }
}
```

a) Hacer syncdb

Nuestro primer modelo







# Previo: Clases del ORM

```
ts = Publisher.objects.all()
```

---

**Model**

---

**Manager**

---

**QuerySet**



**Previo: `__unicode__()`**

# Previo: `__unicode__()`

```
>>> Publisher.objects.all()
[<Publisher: Publisher object>]
```

# Previo: `__unicode__()`

```
>>> Publisher.objects.all()
[<Publisher: Publisher object>]
```

# Previo: `__unicode__()`

```
>>> Publisher.objects.all()
[<Publisher: Publisher object>]
```



# Previo: `__unicode__()`

```
>>> Publisher.objects.all()
[<Publisher: Publisher object>]
```



```
class Publisher(models.Model):
 ...

 def __unicode__(self):
 return self.name
```

# Previo: `__unicode__()`

```
>>> Publisher.objects.all()
[<Publisher: Publisher object>]
```



```
class Publisher(models.Model):
 ...

 def __unicode__(self):
 return self.name
```



```
>>> Publisher.objects.all()
[<Publisher: Apress>]
```



# INSERT

# INSERT

```
o = Model(...) o.save()
```

a)

```
>>> p = Publisher(
... name='Apress',
... address='2855 Telegraph Avenue',
... city='Berkeley',
... state_province='CA',
... country='U.S.A.',
... website='http://www.apress.com/')
>>> p.save()
```

# INSERT

```
o = Model(...) o.save()
```

a)

```
>>> p = Publisher(
... name='Apress',
... address='2855 Telegraph Avenue',
... city='Berkeley',
... state_province='CA',
... country='U.S.A.',
... website='http://www.apress.com/')
>>> p.save()
```

```
manager.create(...)
```

b)

```
>>> p = Publisher.objects.create(
... name='O'Reilly',
... address='10 Fawcett St.',
... city='Cambridge',
... state_province='MA',
... country='U.S.A.',
... website='http://www.oreilly.com/')
>>>
```

# UPDATE

# UPDATE

```
o.save()
```

1

```
>>> ...
>>> p.id
52
>>> p.name = 'Apress Publishing'
>>> p.save()
```

# UPDATE

```
o.save()
```

1

```
>>> ...
>>> p.id
52
>>> p.name = 'Apress Publishing'
>>> p.save()
```

```
queryset.update(...)
```

n

```
>>> Publisher.objects.all().update(country='USA')
2
```



**DELETE**

# DELETE

```
o.delete()
```

1

```
>>> ...
>>> p.id
52
>>> p.delete()
```

# DELETE

```
o.delete()
```

1

```
>>> ...
>>> p.id
52
>>> p.delete()
```

```
queryset.delete()
```

n

```
>>> ps = Publisher.objects.all()
>>> ps.delete()
```

# **SELECT de 1 resultado**

# SELECT de 1 resultado

```
.get(...)
```

```
>>> Publisher.objects.get(name="Apress")
<Publisher: Apress>
```

# SELECT de 1 resultado

```
.get(...)
```

```
>>> Publisher.objects.get(name="Apress")
<Publisher: Apress>
```

```
>>> Publisher.objects.get(name="Anaya")
Traceback (most recent call last):
...
DoesNotExist: Publisher matching query does not exist.
```



# SELECT de 1 resultado

```
.get(...)
```

```
>>> Publisher.objects.get(name="Apress")
<Publisher: Apress>
```

```
>>> Publisher.objects.get(name="Anaya")
Traceback (most recent call last):
...
DoesNotExist: Publisher matching query does not exist.
```

```
>>> Publisher.objects.get(country="U.S.A.")
Traceback (most recent call last):
...
MultipleObjectsReturned: get() returned more than one Publisher --
it returned 2! Lookup parameters were {'country': 'U.S.A.'}
```

# **SELECT de N resultados**

# SELECT de N resultados

`.all()`

```
>>> Publisher.objects.all()
[<Publisher: Apress>, <Publisher: O'Reilly>]
```

# SELECT de N resultados

`.all()`

```
>>> Publisher.objects.all()
[<Publisher: Apress>, <Publisher: O'Reilly>]
```

`.filter(...)`

```
>>> Publisher.objects.filter(
 country="U.S.A.", state_province="CA")
[<Publisher: Apress>]
```

# SELECT de N resultados

`.all()`

```
>>> Publisher.objects.all()
[<Publisher: Apress>, <Publisher: O'Reilly>]
```

`.filter(...)`

```
>>> Publisher.objects.filter(
 country="U.S.A.", state_province="CA")
[<Publisher: Apress>]
```

`.exclude(...)`

```
>>> Publisher.objects.exclude(
 country="U.S.A.", state_province="CA")
[<Publisher: O'Reilly>]
```

# SELECT de N results

¡Devuelven  
QuerySets, no  
listas!

`.all()`

```
>>> Publisher.objects.all()
[<Publisher: Apress>, <Publisher: O'Reilly>]
```

`.filter(...)`

```
>>> Publisher.objects.filter(
 country="U.S.A.", state_province="CA")
[<Publisher: Apress>]
```

`.exclude(...)`

```
>>> Publisher.objects.exclude(
 country="U.S.A.", state_province="CA")
[<Publisher: O'Reilly>]
```



# **get(), filter() y exclude()**

# get(), filter() y exclude()

```
Modelo.objects.filter(campo1="valor1", campo2="valor2")
```

Los parámetros pueden indicar mucho más que igualdad (=)

# get(), filter() y exclude()

```
Modelo.objects.filter(campo1="valor1", campo2="valor2")
```

Los parámetros pueden indicar mucho más que igualdad (=)



campo\_\_exact=' '  
campo\_\_iexact=' '  
campo\_\_contains=' '  
campo\_\_icontains=' '  
campo\_\_isnull=T|F  
campo\_\_day=31

campo\_\_gt=0  
campo\_\_gte=0  
campo\_\_lt=0  
campo\_\_lte=0  
campo\_\_in=[ , ]  
campo\_\_month=12

campo\_\_startswith=' '  
campo\_\_istartswith=' '  
campo\_\_endswith=' '  
campo\_\_iendswith=' '  
campo\_\_range=( , )  
campo\_\_year=2010

# ORDER BY

# ORDER BY

```
.order_by(...)
```



```
>>> Publisher.objects.order_by("name")
[<Publisher: Apress>, <Publisher: O'Reilly>]
```

# ORDER BY

```
.order_by(...)
```



```
>>> Publisher.objects.order_by("name")
[<Publisher: Apress>, <Publisher: O'Reilly>]
```



```
>>> Publisher.objects.order_by("-name")
[<Publisher: O'Reilly>, <Publisher: Apress>]
```



# ORDER BY

```
.order_by(...)
```



```
>>> Publisher.objects.order_by("name")
[<Publisher: Apress>, <Publisher: O'Reilly>]
```



```
>>> Publisher.objects.order_by("-name")
[<Publisher: O'Reilly>, <Publisher: Apress>]
```

Múltiples campos:

```
>>> Publisher.objects.order_by("-name", "country")
[<Publisher: O'Reilly>, <Publisher: Apress>]
```

# ORDER BY

¡También  
devuelve  
QuerySets!

```
.order_by(...)
```



```
>>> Publisher.objects.order_by("name")
[<Publisher: Apress>, <Publisher: O'Reilly>]
```



```
>>> Publisher.objects.order_by("-name")
[<Publisher: O'Reilly>, <Publisher: Apress>]
```

Múltiples campos:

```
>>> Publisher.objects.order_by("-name", "country")
[<Publisher: O'Reilly>, <Publisher: Apress>]
```

# Slicing

# Slicing

[n:m]

```
>>> Publisher.objects.order_by("name")[0]
<Publisher: Apress>
```

```
>>> Publisher.objects.order_by("name")[:2]
[<Publisher: Apress>, <Publisher: O'Reilly>]
```

```
>>> Publisher.objects.order_by("name")[1:2]
[<Publisher: O'Reilly>]
```

# Slicing

[n:m]

```
>>> Publisher.objects.order_by("name")[0]
<Publisher: Apress>
```

```
>>> Publisher.objects.order_by("name")[:2]
[<Publisher: Apress>, <Publisher: O'Reilly>]
```

LIMIT

```
>>> Publisher.objects.order_by("name")[1:2]
[<Publisher: O'Reilly>]
```

# Slicing

[n:m]

```
>>> Publisher.objects.order_by("name")[0]
<Publisher: Apress>
```

```
>>> Publisher.objects.order_by("name")[:2]
[<Publisher: Apress>, <Publisher: O'Reilly>]
```

LIMIT

```
>>> Publisher.objects.order_by("name")[1:2]
[<Publisher: O'Reilly>]
```

OFFSET



# Related Objects

# Related Objects

## OneToOneField

1

```
class Coche(models.Model):
 motor = OneToOneField(Motor)
```

1

```
class Motor(models.Model):
 ...
```

# Related Objects

## OneToOneField

```
1 class Coche(models.Model):
 motor = OneToOneField(Motor)
```

```
1 class Motor(models.Model):
 ...
```

¿Cómo usamos la relación desde las instancias?

Gracias a nosotros

```
>>> c.motor
<Motor: Motor object>
```

# Related Objects

## OneToOneField

1

```
class Coche(models.Model):
 motor = OneToOneField(Motor)
```

1

```
class Motor(models.Model):
 ...
```

¿Cómo usamos la relación desde las instancias?

Gracias a nosotros

```
>>> c.motor
<Motor: Motor object>
```

Gracias a **Django**

```
>>> m.coche
<Coche: Coche object>
```

# Related Objects

# Related Objects

## ForeignKeyField

1

```
class Blog(models.Model):
 ...
```

n

```
class Post(models.Model):
 blog = ForeignKeyField(Blog)
```



# Related Objects

## ForeignKeyField

1 `class Blog(models.Model):`  
    `...`

n `class Post(models.Model):`  
    `blog = ForeignKeyField(Blog)`

¿Cómo usamos la relación desde las instancias?

Gracias a nosotros

```
>>> p.blog
<Blog: Blog object>
```

Gracias a **Django**

```
>>> b.post_set.all()
[<Post: Post object>, ...]
```

# Related Objects

# Related Objects

## ManyToManyField

n

```
class Post(models.Model):
 tags = ManyToManyField(Tag)
```

m

```
class Tag(models.Model):
 ...
```

# Related Objects

## ManyToManyField

n

```
class Post(models.Model):
 tags = ManyToManyField(Tag)
```

m

```
class Tag(models.Model):
 ...
```

¿Cómo usamos la relación desde las instancias?

Gracias a nosotros

```
>>> p.tags.add(t1, t2)
>>> p.tags.all()
[<Tag: Tag object>, ...]
```

Gracias a **Django**

```
>>> t.post_set.add(p1, p2)
>>> t.post_set.all()
[<Post: Post object>, ...]
```

# Related Objects

# Related Objects

En todas ellas podemos **renombrar** el puntero inverso

1

```
class Blog(models.Model):
 ...
```

n

```
class Post(models.Model):
 blog = ForeignKeyField(Blog, related_name='posts')
```



# Related Objects

En todas ellas podemos **renombrar** el puntero inverso

1

```
class Blog(models.Model):
 ...
```

n

```
class Post(models.Model):
 blog = ForeignKeyField(Blog, related_name='posts')
```

Gracias a nosotros

```
>>> p.blog
<Blog: Blog object>
```

Gracias a **Django**

```
>>> b.posts.all()
[<Post: Post object>, ...]
```

# Related Objects

En todas ellas podemos **renombrar** el puntero inverso

1

```
class Blog(models.Model):
 ...
```

n

```
class Post(models.Model):
 blog = ForeignKeyField(Blog, related_name='posts')
```

Gracias a nosotros

```
>>> p.blog
<Blog: Blog object>
```

Gracias a **Django**

```
>>> b.posts.all()
[<Post: Post object>, ...]
```

Cuando haya **2 relaciones entre 2 modelos**, será **obligatorio**



Y la guinda final: ¡Todo es LAZY!

# Laziness

# Laziness

- Las consultas **sólo** se ejecutarán cuando realmente se necesite obtener los objetos. En las siguientes situaciones:

- Iteraciones

```
for p in Publisher.objects.all():
```

- Slicing

```
Publisher.objects.filter(country='USA')[0]
```

- Serialización

[Caché]

- repr()

```
[<Publisher: Publisher object>]
```

- len() !!!

```
len(Publisher.objects.all())
```

- list()

```
list(Publisher.objects.all())
```

- bool()

```
if Publisher.objects.filter(country='USA'):
```



- a) Insertar varios Tweets desde la shell
- b) Modificar `views.index` para que utilice los Tweets reales de la BD
- c) Implementar URL y View para ofrecer permalinks a los tweets:
  - ▶ URL: `user/<username>/status/<tweet_id>`
  - ▶ View: `tweet_page`
  - ▶ Template: `tweet_page.html` (Bajar)
- d) shortcut!: `get_object_or_404()`

Usando el ORM





Profiles

# Profile

- **Problema:** El modelo *User* de *django.contrib.auth* no puede contener toda la información que necesitamos.
  - Username, Password, Name.... y poco más.
- **Solución:** Definir un **Profile** (Un Modelo Agregado) para guardar esa información.

# Profile

**models.py**

# Profile

**models.py**

```
class Profile(models.Model):
 user = models.OneToOneField(User, unique=True)
```

# Profile

## models.py

```
class Profile(models.Model):

 user = models.OneToOneField(User, unique=True)
 bio = models.CharField(blank=True, max_length=200)
```

# Profile

## models.py

```
class Profile(models.Model):
 user = models.OneToOneField(User, unique=True)
 bio = models.CharField(blank=True, max_length=200)
 ...
```



# Profile

## models.py

```
class Profile(models.Model):

 user = models.OneToOneField(User, unique=True)
 bio = models.CharField(blank=True, max_length=200)
 ...
```

## settings.py

```
AUTH_PROFILE_MODULE = "website.Profile"
```

# Profile

## models.py

```
class Profile(models.Model):

 user = models.OneToOneField(User, unique=True)
 bio = models.CharField(blank=True, max_length=200)
 ...
```

## settings.py

```
AUTH_PROFILE_MODULE = "website.Profile"
```

- Cada objeto *User* de *django.contrib.auth* dispone de un método `get_profile()` que nos permitirá acceder al Profile.

# Profile

- Donde tengamos el User tendremos el Profile.
- Donde tengamos el Profile, tendremos el User.

# Profile

```
>>> from django.contrib.auth.models import User
>>> u = User.objects.get(id=1)
>>> type(u)
<class 'django.contrib.auth.models.User'>

>>> profile = u.get_profile()
>>> type(profile)
<class 'website.models.Profile'>
```

- Donde tengamos el User tendremos el Profile.
- Donde tengamos el Profile, tendremos el User.





1 User = 1 Profile ¿Cuándo se crea un User?



# Signals

- Django incorpora un dispatcher de señales para ayudar a crear aplicaciones reutilizables.
- Permite subscribirnos a “eventos” a lo largo de todo el framework y actuar frente a ellos.
- Señales interesantes:
  - `pre_save`, `post_save`
  - `pre_delete`, `post_delete`
  - `m2m_changed`
  - `request_started`, `request_finished`

<http://docs.djangoproject.com/en/dev/ref/signals/>



# Signals

# Signals

```
def profile_signal_callback(sender, **kwargs): 1
 if kwargs.get('created'):
 profile = Profile()
 profile.user = User.objects.get(id=kwargs.get('instance').id)
 profile.save()

from django.db.models.signals import post_save

post_save.connect(profile_signal_callback, sender=User,
 dispatch_uid="users-profilecreation-signal")
```

# Signals

```
def profile_signal_callback(sender, **kwargs): 1
 if kwargs.get('created'):
 profile = Profile()
 profile.user = User.objects.get(id=kwargs.get('instance').id)
 profile.save()

from django.db.models.signals import post_save

post_save.connect(profile_signal_callback, sender=User,
 dispatch_uid="users-profilecreation-signal") 2
```

# Signals

```
def profile_signal_callback(sender, **kwargs): 1
 if kwargs.get('created'): 2
 profile = Profile()
 profile.user = User.objects.get(id=kwargs.get('instance').id)
 profile.save()

from django.db.models.signals import post_save 3

post_save.connect(profile_signal_callback, sender=User,
 dispatch_uid="users-profilecreation-signal")
```

# Signals

```
def profile_signal_callback(sender, **kwargs): 1
 if kwargs.get('created'): 2
 profile = Profile()
 profile.user = User.objects.get(id=kwargs.get('instance').id)
 profile.save()

from django.db.models.signals import post_save 3

post_save.connect(profile_signal_callback, sender=User, 4
 dispatch_uid="users-profilecreation-signal")
```

# Signals

```
def profile_signal_callback(sender, **kwargs): 1
 if kwargs.get('created'): 2
 profile = Profile()
 profile.user = User.objects.get(id=kwargs.get('instance').id)
 profile.save()

from django.db.models.signals import post_save 3

post_save.connect(profile_signal_callback, sender=User, 4
 dispatch_uid="users-profilecreation-signal")
```

**Puede estar en cualquier app de nuestro proyecto.**

Se recomienda models.py



- a) Crear el modelo Profile
- b) Crear trigger de agregación con User mediante signals
- c) Eliminar BD
- d) Hacer syncdb

Users en dwitter





Followers



# Profile

**models.py**

# Profile

**models.py**

```
class Profile(models.Model):

 user = models.OneToOneField(User, unique=True)
 bio = models.CharField(blank=True, max_length=200)
```

# Profile

## models.py

```
class Profile(models.Model):

 user = models.OneToOneField(User, unique=True)
 bio = models.CharField(blank=True, max_length=200)
 followers = models.ManyToManyField("self", blank=True,
```





# Profile

# models.py

[illegible]

# Profile

## models.py

```
class Profile(models.Model):

 user = models.OneToOneField(User, unique=True)
 bio = models.CharField(blank=True, max_length=200)
 followers = models.ManyToManyField("self", blank=True,
 symmetrical=False,
 related_name="following")
```

- m2m symmetrical (*por defecto*)
  - Si yo te sigo a ti, tú me sigues a mí.
- related\_name (*en este caso*)
  - Limpieza y evitar profile\_set



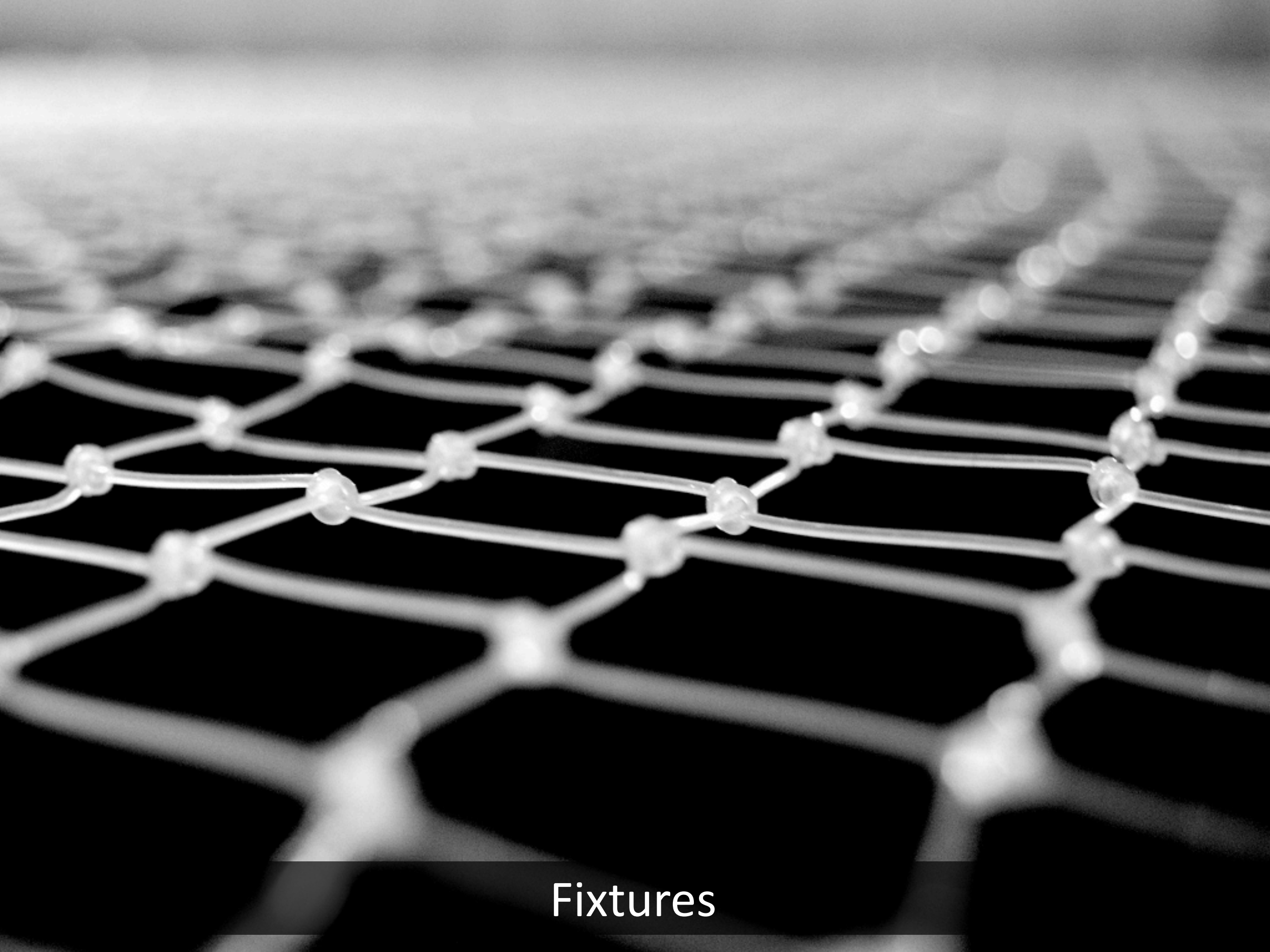
- ```
Manager.php
// 4 objects to
// lets load some
var i = 0; i < 4; i++) {
    // pick a random object
    number = Math.floor(Math
    object = libraryArrange(number)
}

// chuck
clip = screen
clip = screen
// dragging mech
onpress = function() {
    // to do
    // start
    // end
}
```
- a) Añadir ManyToMany para los followers
 - b) Eliminar BD
 - c) Hacer syncdb

Users en dwitter



1,2,3 Borrar la BD... 1,2,3... Borrar la BD



Fixtures

Fixtures

- Es muy aburrido crear juegos de datos cada vez que se elimina una tabla / BD.
- **No** nos gustan las cosas aburridas.
- Ficheros (json/xml/yaml) para inicializar una Tabla.
- Pueden crearse una vez dispongamos de la información:

Fixtures

- Es muy aburrido crear juegos de datos cada vez que se elimina una tabla / BD.
- **No** nos gustan las cosas aburridas.
- Ficheros (json/xml/yaml) para inicializar una Tabla.
- Pueden crearse una vez dispongamos de la información:

```
python manage.py dumpdata <appName appName appName.model ...>
```

Fixtures

- Es muy aburrido crear juegos de datos cada vez que se elimina una tabla / BD.
- **No** nos gustan las cosas aburridas.
- Ficheros (json/xml/yaml) para inicializar una Tabla.
- Pueden crearse una vez dispongamos de la información:

```
python manage.py dumpdata <appName appName appName.model ...>
```

- Se cargan utilizando:

```
python manage.py loaddata <fixture fixture ...>
```

- Si se llaman *initial_data* y están dentro de la carpeta fixtures de la app, se cargan al hacer el syncdb.

- a) Crear la carpeta website/fixtures
- b) Descargar initial_data.json y meterlo en la carpeta fixtures
- c) Hacer syncdb
- d) Implementar URL y View para ofrecer páginas de usuario:
 - ▶ URL: `user/<username>/`
 - ▶ View: `user_page`
 - ▶ Template: `user_page.html` (Bajar)

El poder de los fixtures

Índice

1. Python

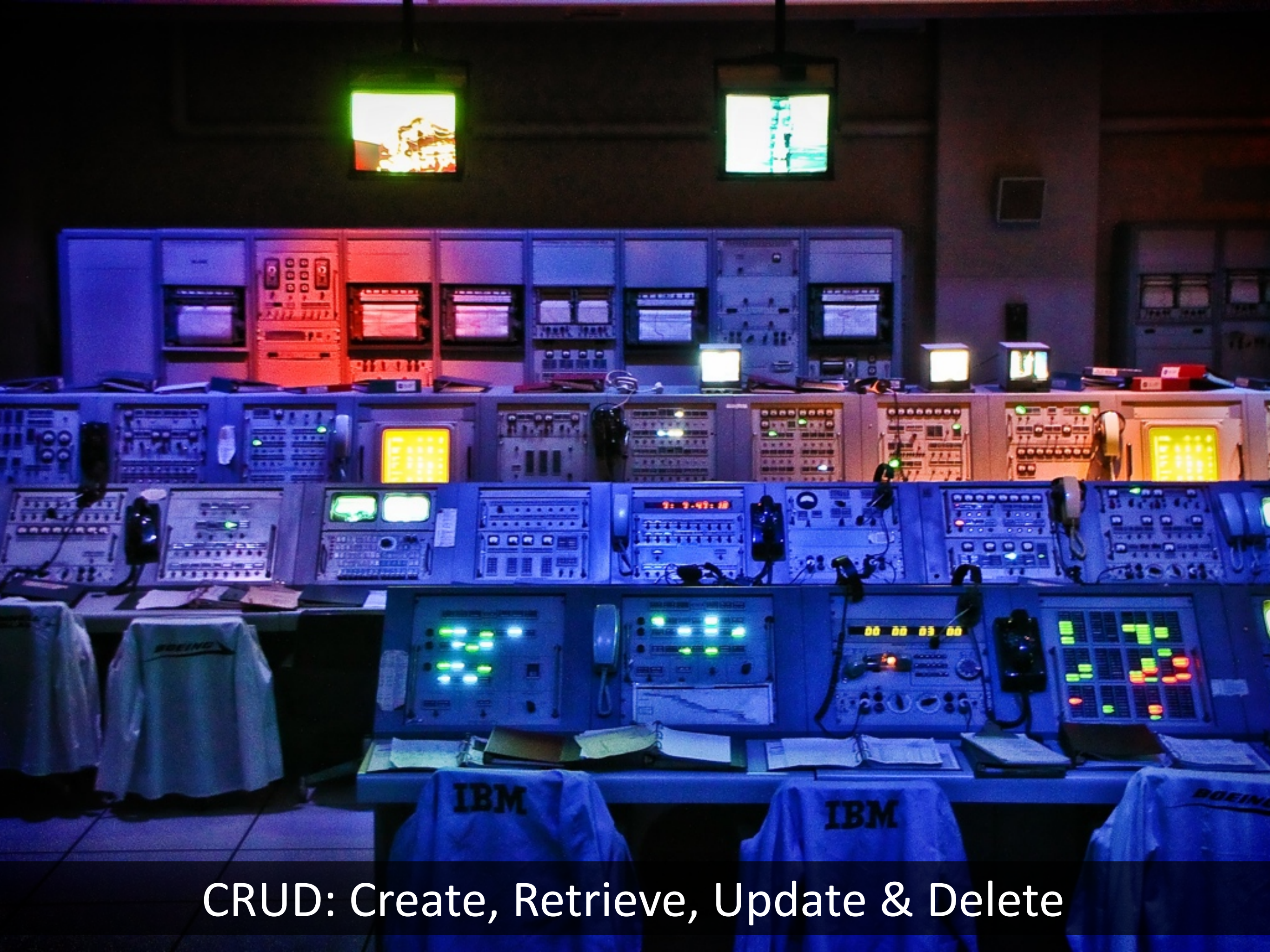
- a. Introducción
- b. Tipos de datos
- c. Operadores
- d. Usos frecuentes
- e. Estructuras
- f. Sentencias
- g. Ejercicio

2. Django

- a. Introducción
- b. URLs y Vistas
- c. Templates
- d. Modelo
- e. Administración**
- f. Formularios
- g. Magia avanzada

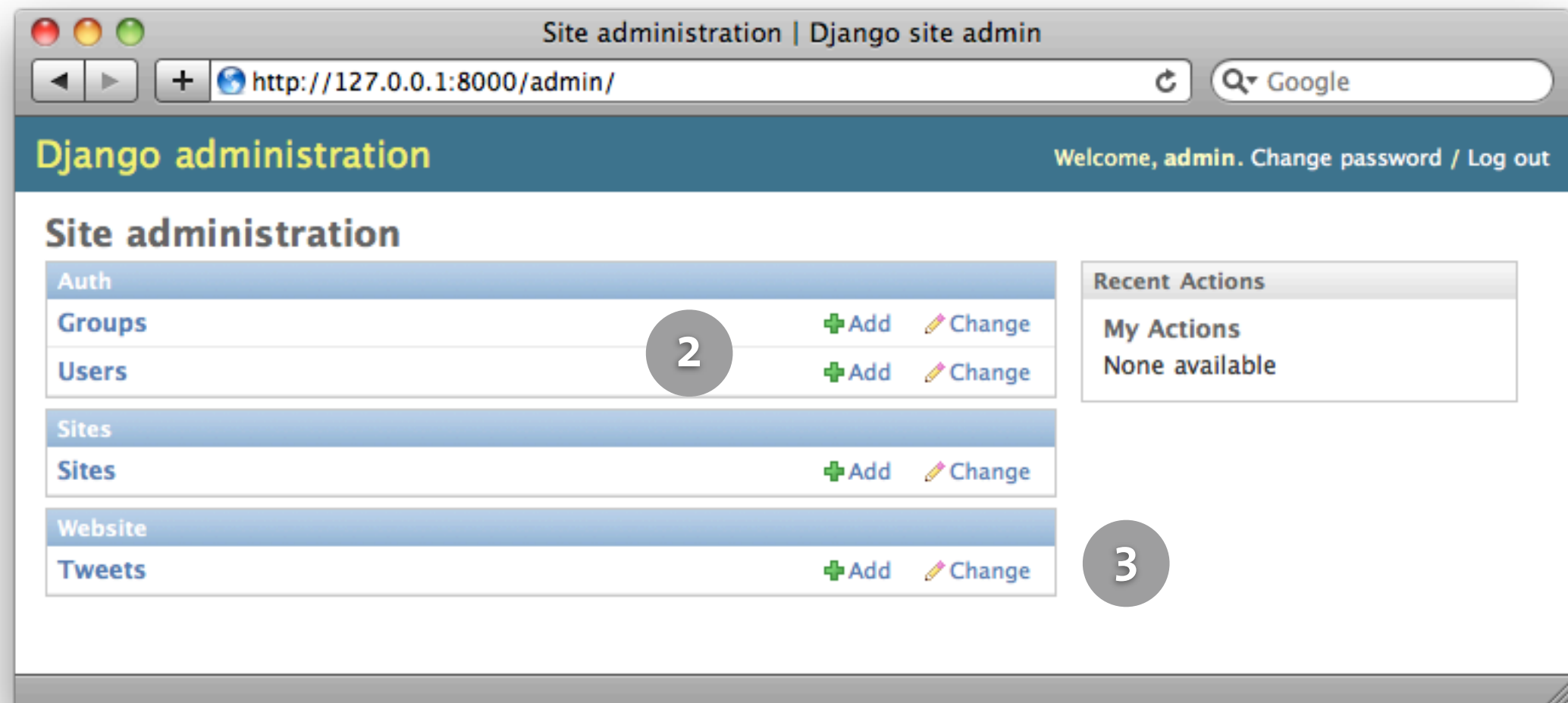


Proyecto



CRUD: Create, Retrieve, Update & Delete

django.contrib.admin



django.contrib.admin

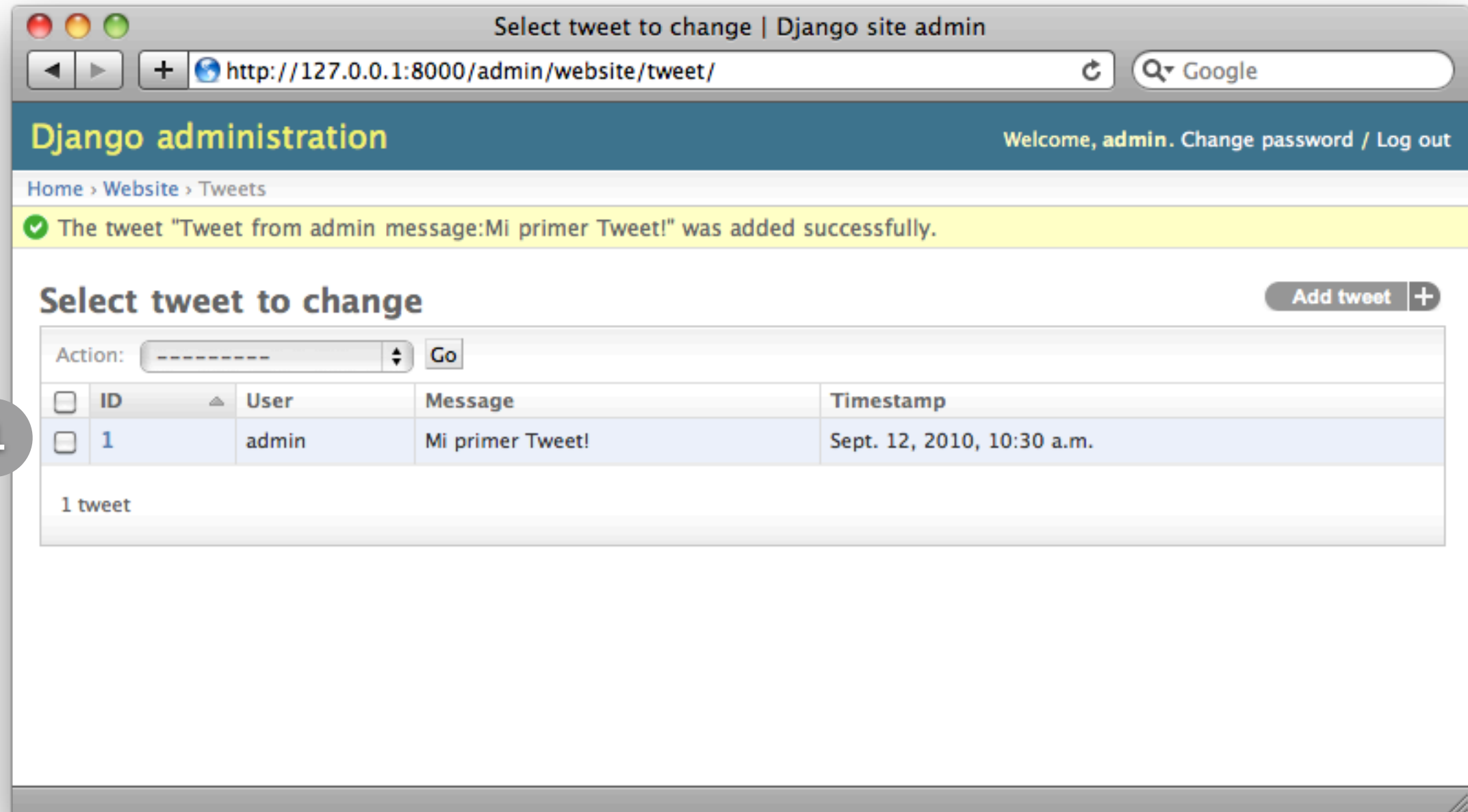
The screenshot shows a web browser window with the title "Add tweet | Django site admin". The address bar displays the URL `http://127.0.0.1:8000/admin/website/tweet/add/`. The page header includes the "Django administration" logo and a welcome message for the user "admin", with links for "Change password" and "Log out". The breadcrumb trail indicates the path: "Home > Website > Tweets > Add tweet".

The main content area is titled "Add tweet" and contains two primary form fields:

- Message:** A large text area containing the text "Mi primer Tweet!". A circular callout with the number "1" points to this field.
- User:** A dropdown menu currently showing "admin" with a green plus icon to its right. A circular callout with the number "2" points to this dropdown.

At the bottom of the form, there are three buttons: "Save and add another", "Save and continue editing", and a blue "Save" button. A circular callout with the number "3" points to the "Save" button.

django.contrib.admin



Django admin: Instalación

urls.py

```
from django.conf.urls.defaults import *

# Uncomment the next two lines to enable the admin:
# from django.contrib import admin
# admin.autodiscover()

urlpatterns = patterns('',

    ...

    2 # (r'^admin/', include(admin.site.urls)),

    ...

)
```

Django admin: Instalación

urls.py

```
from django.conf.urls.defaults import *

# Uncomment the next two lines to enable the admin:
1 from django.contrib import admin
  admin.autodiscover()

urlpatterns = patterns('',

    ...

2 (r'^admin/', include(admin.site.urls)),

    ...

)
```

Django admin: Instalación

settings.py

```
INSTALLED_APPS = (  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.sites',  
  
    ...  
)
```

Django admin: Instalación

settings.py

```
INSTALLED_APPS = (  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.sites',  
    'django.contrib.admin',  
    ...  
)
```

1

Actualizando la BD

Actualizando la BD

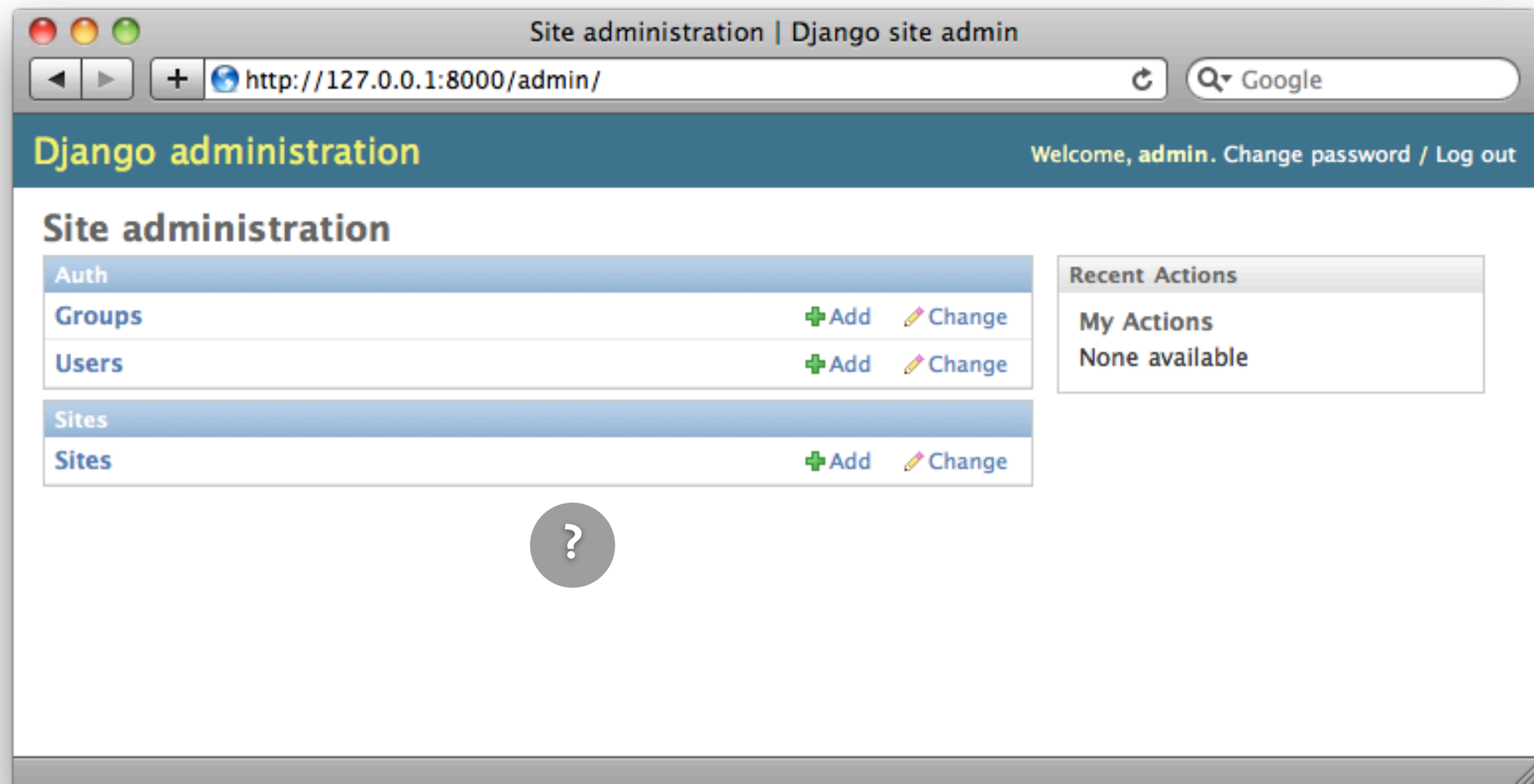
```
$ python manage.py syncdb
```

```
Creating table django_admin_log  
Installing index for admin.LogEntry model
```

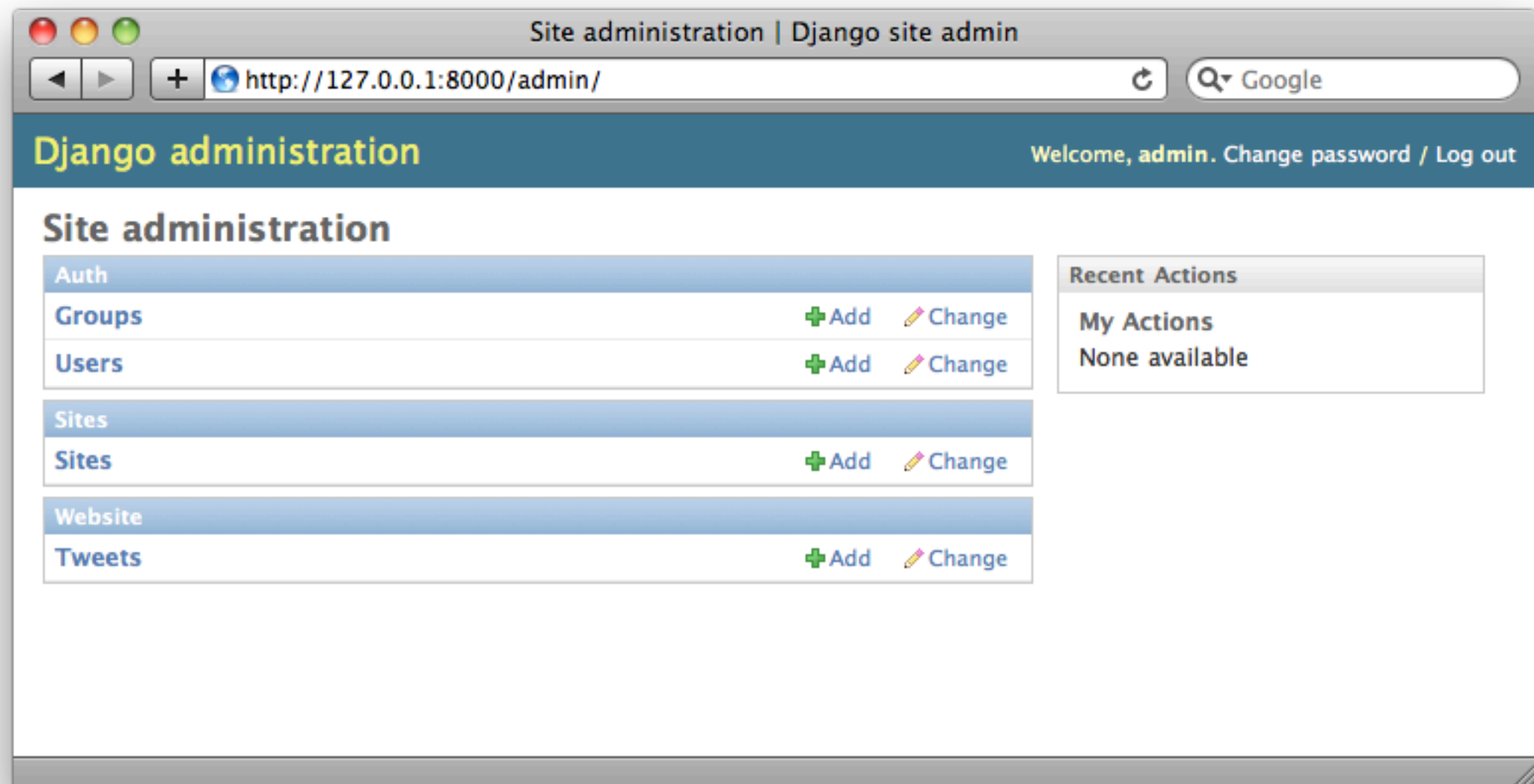


Admin lista para usar

¿Y nuestra app?



¿Y nuestra app?



admin.py

- Cada **app** debe de tener el suyo.
- Define qué modelos serán visibles desde el admin y permite personalizar su aspecto.

```
from django.contrib import admin
from website.models import Tweet

class TweetAdmin(admin.ModelAdmin):
    list_display = ('id', 'user', 'message', 'timestamp')

admin.site.register(Tweet, TweetAdmin)
```


- a) Configurar urls.py
- b) Configurar settings.py
- c) Hacer syncdb
- d) Comprobar que no está Tweet
- e) Añadir admin.py
- f) Probar Administración

Instalar Admin

Índice

1. Python

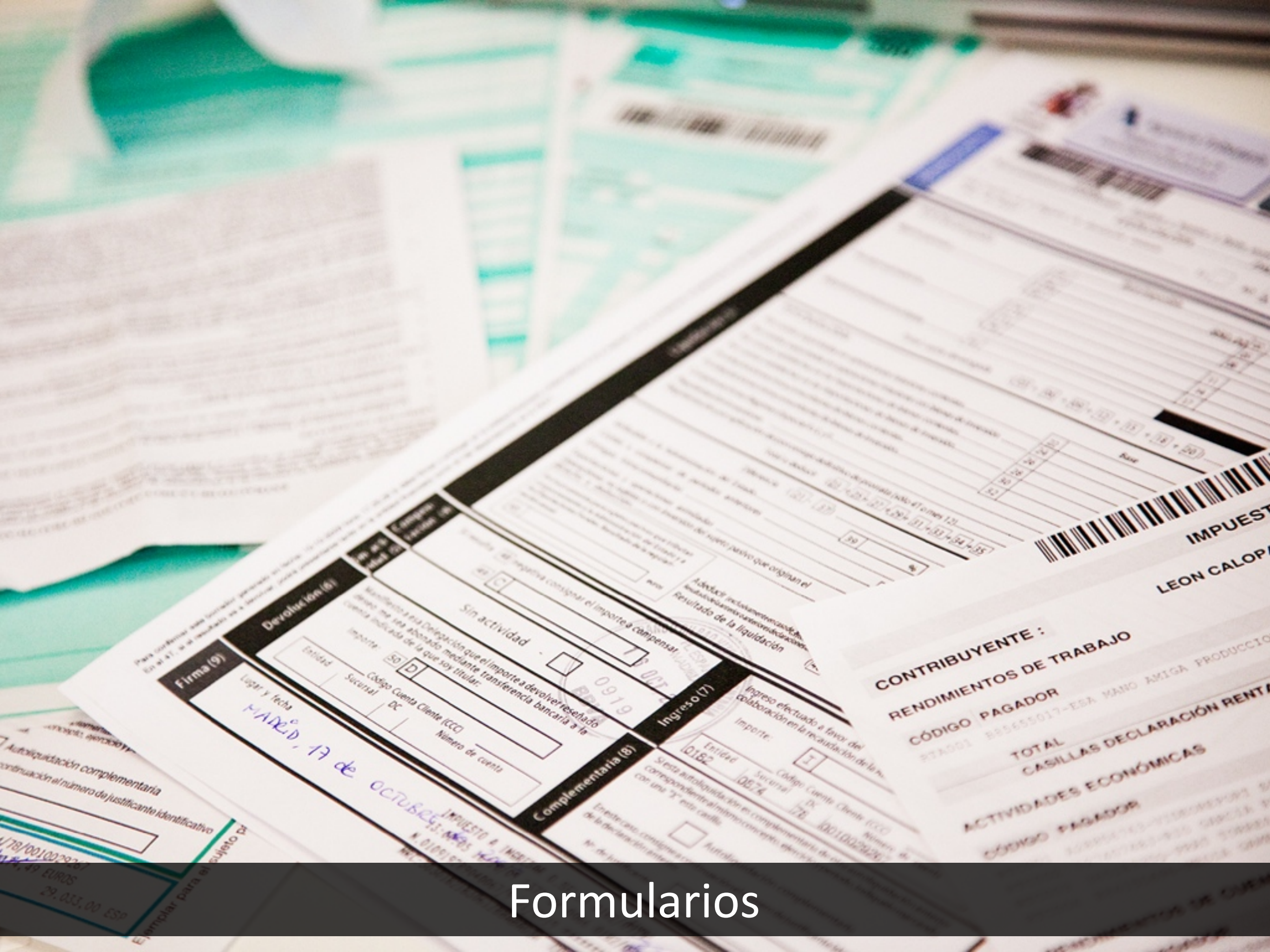
- a. Introducción
- b. Tipos de datos
- c. Operadores
- d. Usos frecuentes
- e. Estructuras
- f. Sentencias
- g. Ejercicio

2. Django

- a. Introducción
- b. URLs y Vistas
- c. Templates
- d. Modelo
- e. Administración
- f. Formularios**
- g. Magia avanzada



Proyecto



Formularios

Clases involucradas

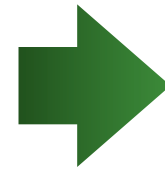
Clases involucradas

Widget

Componente **visual** equivalente a HTML

TextInput

CheckboxInput



```
<input type='text' ...>
```

```
<input type='checkbox' ...>
```

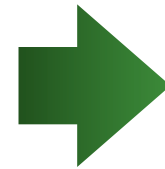
Clases involucradas

Widget

Componente **visual** equivalente a HTML

TextInput

CheckboxInput



```
<input type='text' ...>
```

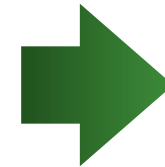
```
<input type='checkbox' ...>
```

Field

Lógica de un campo, asociado a un Widget

EmailField

IPAddressField



```
widget, initial, error, ...
```

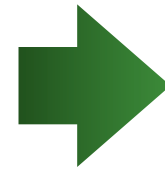
Clases involucradas

Widget

Componente **visual** equivalente a HTML

TextInput

CheckboxInput



```
<input type='text' ...>
```

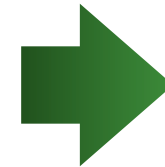
```
<input type='checkbox' ...>
```

Field

Lógica de un campo, asociado a un Widget

EmailField

IPAddressField



```
widget, initial, error, ...
```

Form

Conjunto de Fields de un formulario

```
ContactForm [nombre, email, telefono, mensaje, ...]
```


Fields

Fields

- BooleanField
- CharField
- ChoiceField
- TypedChoiceField
- DateField
- DateTimeField
- DecimalField
- EmailField
- FileField
- FilePathField
- FloatField
- ImageField
- IntegerField
- IPAddressField
- MultipleChoiceField
- NullBooleanField
- RegexField
- SlugField
- TimeField
- URLField
- ComboField
- MultiValuefield
- SplitDateTimeField
- ModelChoiceField
- ModelMultipleChoiceField

Fields

- BooleanField
- CharField
- ChoiceField
- TypedChoiceField
- DateField
- DateTimeField
- DecimalField
- EmailField
- FileField
- FilePathField
- FloatField
- ImageField
- IntegerField
- IPAddressField
- MultipleChoiceField
- NullBooleanField
- RegexField
- SlugField
- TimeField
- URLField
- ComboField
- MultiValuefield
- SplitDateTimeField
- ModelChoiceField
- ModelMultipleChoiceField

Creación de un Form

- **Paso 1/3:** Definición del formulario en **forms.py**

```
from django import forms
```

```
class ContactForm(forms.Form):
```

```
    subject = forms.CharField(max_length=100, label='Topic')
```

```
    email = forms.EmailField(required=False)
```

```
    message = forms.CharField(widget=forms.Textarea)
```

Creación de un Form

- **Paso 2/3:** Maquetación del formulario en su **template**

```
<html>
<body>
  <h1>Contact us</h1>

  {% if form.errors %}
    <p style="color: red;">
      Please correct the error{{ form.errors|pluralize }} below.
    </p>
  {% endif %}

  <form action="" method="post">
    <table>
      {{ form.as_table }}
    </table>
    <input type="submit" value="Submit">
  </form>
</body>
</html>
```

Creación de un Form

- **Paso 3/3:** Programación de la vista en **views.py**

```
from django.shortcuts import render_to_response
from mysite.contact.forms import ContactForm

def contact(request):
    if request.method == 'POST':
        form = ContactForm(request.POST)
        if form.is_valid():
            cd = form.cleaned_data
            send_mail(cd['subject'], cd['message'], ...)
            # ...
            return HttpResponseRedirect('/contact/thanks/')
    else:
        form = ContactForm()
    return render_to_response('contact_form.html', {'form': form})
```


Creación de un Form

- **Paso 3/3:** Programación de la vista en **views.py**

```
from django.shortcuts import render_to_response
from mysite.contact.forms import ContactForm

def contact(request):
    if request.method == 'POST':
        form = ContactForm(request.POST)
        if form.is_valid():
            cd = form.cleaned_data
            send_mail(cd['subject'], cd['message'], ...)
            # ...
            return HttpResponseRedirect('/contact/thanks/')
    else:
        form = ContactForm()
    return render_to_response('contact_form.html', {'form': form})
```



Pattern

Creación de un Form

- Paso 3/3: Programación de la vista en **views.py**

```
from django.shortcuts import render_to_response
from mysite.contact.forms import ContactForm

def contact(request):
    if request.method == 'POST':
        form = ContactForm(request.POST)
        if form.is_valid():
            cd = form.cleaned_data
            send_mail(cd['subject'], cd['message'], ...)
            # ...
            return HttpResponseRedirect('/contact/thanks/')
    else:
        form = ContactForm()
    return render_to_response('contact_form.html', {'form': form})
```



Pattern

Creación de un Form

- Paso 3/3: Programación de la vista en **views.py**

```
from django.shortcuts import render_to_response
from mysite.contact.forms import ContactForm

def contact(request):
    if request.method == 'POST':
        form = ContactForm(request.POST)
        if form.is_valid():
            cd = form.cleaned_data
            send_mail(cd['subject'], cd['message'], ...)
            # ...
            return HttpResponseRedirect('/contact/thanks/')
    else:
        form = ContactForm()
    return render_to_response('contact_form.html', {'form': form})
```



Pattern

Validación propia

Podemos programar validación extra asociada a cada Field del formulario escribiendo un método `clean_<fieldname>`:

```
from django import forms

class ContactForm(forms.Form):
    subject = forms.CharField(max_length=100)
    email = forms.EmailField(required=False)
    message = forms.CharField(widget=forms.Textarea)

    def clean_message(self):
        message = self.cleaned_data['message']
        num_words = len(message.split())
        if num_words < 4:
            raise forms.ValidationError("Not enough words!")
        return message
```

Validación propia

Podemos programar validación extra asociada a cada Field del formulario escribiendo un método `clean_<fieldname>`:

```
from django import forms

class ContactForm(forms.Form):
    subject = forms.CharField(max_length=100)
    email = forms.EmailField(required=False)
    message = forms.CharField(widget=forms.Textarea)

    def clean_message(self):
        message = self.cleaned_data['message']
        num_words = len(message.split())
        if num_words < 4:
            raise forms.ValidationError("Not enough words!")
        return message
```

Maquetación propia

Podemos personalizar la maquetación tanto como queramos, prescindiendo de las ayudas de `form.as_table`:

```
...
<form action="" method="post">
  <div class="field">
    {{ form.subject.errors }}
    <label for="id_subject">Subject:</label>
    {{ form.subject }}
  </div>
  <div class="field">
    {{ form.email.errors }}
    <label for="id_email">E-mail:</label>
    {{ form.email }}
  </div>
  ...
  <input type="submit" value="Submit">
</form>
...
```

Para los diseñadores:

```
<style type="text/css">
  ul.errorlist {
    ...
  }
  .errorlist li {
    ...
  }
</style>
```




Forms a partir de Models: El COLMO del DRY

Forms a partir de Models

models.py

```
from django.db import models

class Author(models.Model):
    name = models.CharField(max_length=100)
    birth_date = models.DateField(blank=True, null=True)
    country = models.ModelChoiceField(Country)
    ...
```

forms.py

```
from django import forms
from books.models import Author

class AuthorForm(forms.ModelForm):
    class Meta:
        model = Author
        exclude = ('country',)
```


a) Crear un Form que permita publicar tweets desde index.html al usuario que esté logueado desde django.contrib.admin (no tenemos login propio).

Primer Form en dwitter

Índice

1. Python

- a. Introducción
- b. Tipos de datos
- c. Operadores
- d. Usos frecuentes
- e. Estructuras
- f. Sentencias
- g. Ejercicio

2. Django

- a. Introducción
- b. URLs y Vistas
- c. Templates
- d. Modelo
- e. Administración
- f. Formularios
- g. Magia avanzada



Proyecto



Magia avanzada

Magia avanzada

1. Context Processors
2. Middleware
3. Caching
4. Deploying Django
5. Apps recomendadas

Magia avanzada

1. Context Processors

2. Middleware

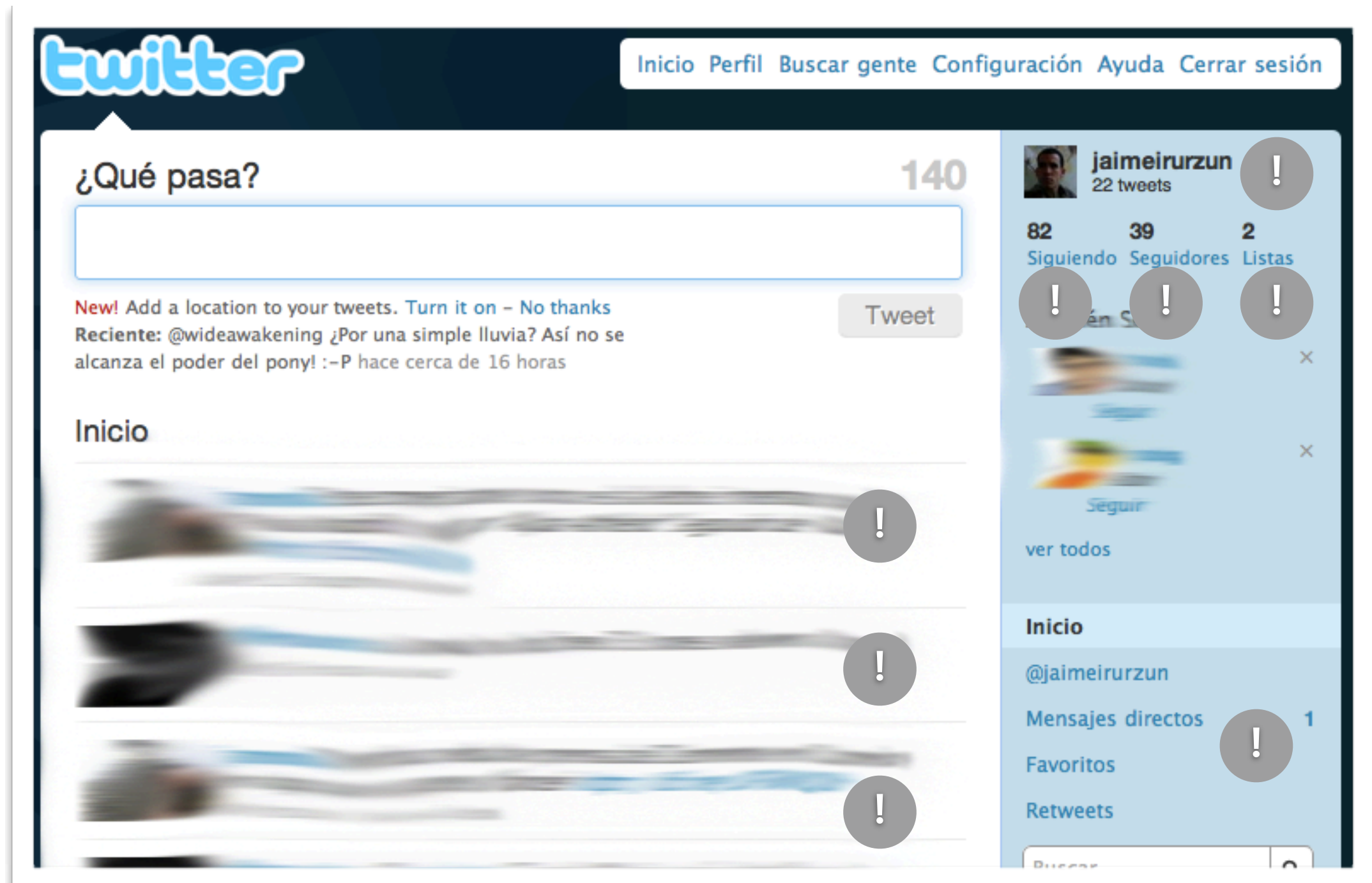
3. Caching

4. Deploying Django

5. Apps recomendadas

Context Processors

Problema



Context Processors

Solución: el objeto **RequestContext()**

```
c = RequestContext(request, {'tweets': tweets})
```

- Sustituye a Context().
- Recibe un HttpRequest() antes del diccionario.
- Automáticamente alimenta nuestro diccionario con todos los diccionarios devueltos por los **Context Processors** que tengamos habilitados.

Context Processors

¿Context Processors? Son funciones.

- Reciben un `HttpRequest()` como único parámetro.
- Devuelven un diccionario para ser usado como `Context()`.

Context Processors

¿Context Processors? Son funciones.

- Reciben un HttpRequest() como único parámetro.
- Devuelven un diccionario para ser usado como Context().

```
def ip_address_processor(request):  
    return {'ip_address': request.META['REMOTE_ADDR']}
```


Context Processors

¿Context Processors? Son funciones.

- Reciben un HttpRequest() como único parámetro.
- Devuelven un diccionario para ser usado como Context().

```
def ip_address_processor(request):  
    return {'ip_address': request.META['REMOTE_ADDR']}
```

```
TEMPLATE_CONTEXT_PROCESSORS = (  
    "django.contrib.auth.context_processors.auth",  
    "django.core.context_processors.debug",  
    "django.core.context_processors.i18n",  
    "django.core.context_processors.media",  
    "django.contrib.messages.context_processors.messages",  
    "website.context_processors.ip_address_processor",  
)
```

Context Processors

¿Qué ocurre con `render_to_response()`?

```
def some_view(request):  
    # ...  
    return render_to_response('my_template.html',  
                              my_data_dictionary,  
                              context_instance=RequestContext(request))
```

Context Processors

¿Qué ocurre con `render_to_response()`?

```
def some_view(request):  
    # ...  
    return render_to_response('my_template.html',  
                              my_data_dictionary,  
                              context_instance=RequestContext(request))
```

Context Processors

¿Qué ocurre con `render_to_response()`?

```
def some_view(request):  
    # ...  
    return render_to_response('my_template.html',  
                              my_data_dictionary,  
                              context_instance=RequestContext(request))
```



Consejo

```
from django.shortcuts import render_to_response  
from django.template import RequestContext  
  
def render_response(req, *args, **kwargs):  
    kwargs['context_instance'] = RequestContext(req)  
    return render_to_response(*args, **kwargs)
```

Magia avanzada

1. Context Processors

2. Middleware

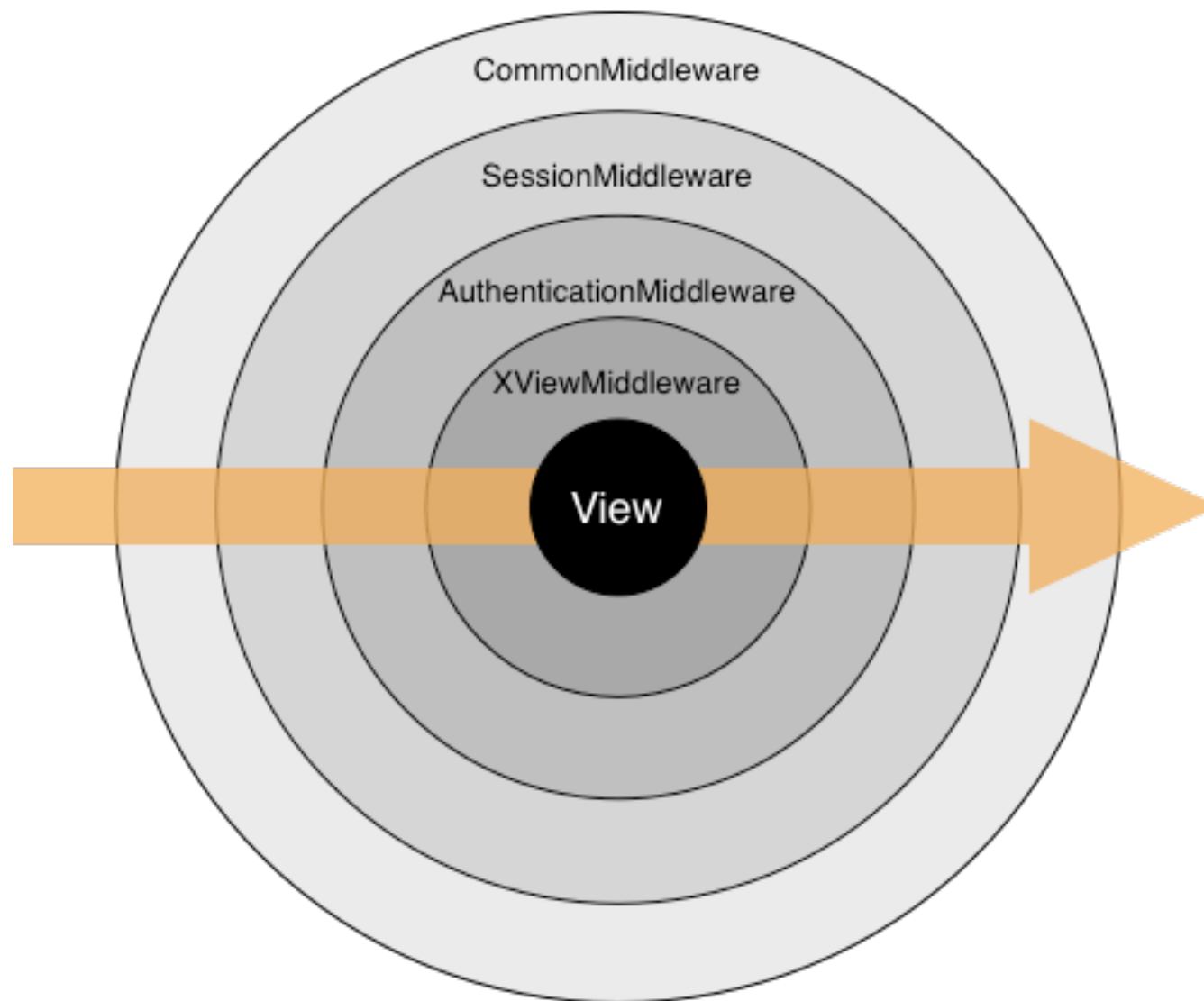
3. Caching

4. Deploying Django

5. Apps recomendadas

Middleware

Nos permite escribir funcionalidad reutilizable que se inserta en el flujo de ejecución de Django



Middleware

```
class MyMiddleware(object):

    def process_request(self, request):
        """Se ejecuta antes de que Django decida a qu     View llamar.
        -> HttpRequest(): Se corta el flujo de middleware.
        -> None: El flujo sigue con el siguiente middleware."""
        # ...

    def process_view(self, request, view_func, view_args, view_kwargs):
        """Se ejecuta antes de que se llame a la View.
        -> HttpRequest(): Se corta el flujo de middleware.
        -> None: El flujo sigue con el siguiente middleware."""
        # ...

    def process_response(self, request, response):
        """Se ejecuta despu    s de que la View devuelva una response.
        -> HttpResponse()"""
        # ...

    def process_exception(self, request, exception):
        """Se ejecuta cuando una View lanza una excepci    n.
        -> HttpResponse().
        -> None: La excepci    n se propaga hasta el cliente."""
        # ...
```

Middleware

Tenemos que incluir nuestro middleware en el lugar que nos convenga, **teniendo en cuenta el orden de ejecución en la Request y en la Response:**

```
MIDDLEWARE_CLASSES = (  
    'django.middleware.common.CommonMiddleware',  
    'django.contrib.sessions.middleware.SessionMiddleware',  
    'django.middleware.csrf.CsrfViewMiddleware',  
    'django.contrib.auth.middleware.AuthenticationMiddleware',  
    'django.contrib.messages.middleware.MessageMiddleware',  
)
```

Magia avanzada

1. Context Processors
2. Middleware
- 3. Caching**
4. Deploying Django
5. Apps recomendadas

caching

- Si queremos escalar nuestro proyecto rápidamente.... **Cache!**
 - Recomendado = *memcached*
- Acceder a la BD es caro, muy caro.
- El Hardware es “gratis” en comparación con optimizar código optimizado.
- Ejemplo (Facebook):

caching

- Si queremos escalar nuestro proyecto rápidamente.... **Cache!**
 - Recomendado = *memcached*
- Acceder a la BD es caro, muy caro.
- El Hardware es “gratis” en comparación con optimizar código optimizado.
- Ejemplo (Facebook):

300+Tb Memcached

caching

views

```
from django.views.decorators.cache import cache_page

@cache_page(60 * 15)
def my_view(request):
    ...
```

caching

views

```
from django.views.decorators.cache import cache_page

@cache_page(60 * 15)
def my_view(request):
    ...
```

templates

```
{% load cache %}
{% cache 500 sidebar %}
    .. sidebar ..
{% endcache %}
```

caching

low level

caching

low level

```
>>> from django.core.cache import cache  
>>> cache.set('my_key', 'hello, world!', 30)  
>>> cache.get('my_key')  
'hello, world!'
```

- Podemos cachear cualquier tipo de objeto que se pueda serializar.
 - QuerySets
 - Listas
 - ...



memcached DEMO

Magia avanzada

1. Context Processors
2. Middleware
3. Caching
- 4. Deploying Django**
5. Apps recomendadas

Deploying Django

Deploying Django

Virtualenv + PIP + Fabric

Virtualenv

- Independencia completa de las librerías del sistema.
- Cada proyecto tiene su juego de librerías en la versión que necesita
 - Sin miedo a actualizar
 - Sin miedo a migrar
 - etc...

Virtualenv

- Independencia completa de las librerías del sistema.
- Cada proyecto tiene su juego de librerías en la versión que necesita
 - Sin miedo a actualizar
 - Sin miedo a migrar
 - etc...

crear

```
$ virtualenv --no-site-packages mi_entorno
```

Virtualenv

- Independencia completa de las librerías del sistema.
- Cada proyecto tiene su juego de librerías en la versión que necesita
 - Sin miedo a actualizar
 - Sin miedo a migrar
 - etc...

crear

```
$ virtualenv --no-site-packages mi_entorno
```

activar

```
$ source mi_entorno/bin/activate
```

PIP

- Gestor de paquetes del siglo 21 :D
- Permite especificar un fichero de REQUIREMENTS con un listado de paquetes a instalar.

PIP

- Gestor de paquetes del siglo 21 :D
- Permite especificar un fichero de REQUIREMENTS con un listado de paquetes a instalar.

```
Django==1.2.1  
psycopg2  
feedparser==4.1  
stripogram==1.5  
GChartWrapper==0.8
```

```
pip install -E mi_entorno -r REQUIREMENTS
```

Fabric

“ Repetition leads to boredom, boredom to horrifying mistakes, horrifying mistakes to God-I-wish-I-was-still-bored

Fabric

“Repetition leads to boredom, boredom to horrifying mistakes, horrifying mistakes to God-I-wish-I-was-still-bored

fabfile.py

```
env.user = "example"  
env.hosts = ["example.com"]  
def deploy():  
    run('svn up /home/example/')  
    sudo('/etc/init.d/lighttpd restart')
```

Fabric

“Repetition leads to boredom, boredom to horrifying mistakes, horrifying mistakes to God-I-wish-I-was-still-bored

fabfile.py

```
env.user = "example"  
env.hosts = ["example.com"]  
def deploy():  
    run('svn up /home/example/')  
    sudo('/etc/init.d/lighttpd restart')
```

USO

```
fabric deploy
```

Magia avanzada

1. Context Processors
2. Middleware
3. Caching
4. Deploying Django
5. Apps recomendadas



Apps

django-registration



<http://bitbucket.org/ubernostrum/django-registration/>

django-registration

- Sistema completo para la gestión de usuarios.
- Desde el registro, activación etc...
- DRY!!
- Instalación muy sencilla.
- Recomendado ++



<http://bitbucket.org/ubernostrum/django-registration/>

django-registration



<http://bitbucket.org/ubernostrum/django-registration/>

django-registration

/activate/complete/
/activate/<activation_key>/
/register/
/register/complete/
/register/closed/
/login/
/logout/
/password/change/
/password/change/done/
/password/reset/
/password/reset/confirm/<token>
/password/reset/complete/
/password/reset/done/
/reactivate/



<http://bitbucket.org/ubernostrum/django-registration/>

django-socialregistration

<http://github.com/flashingpumpkin/django-socialregistration>

django-socialregistration

- Configuración sencilla
- Autenticación con:
 - twitter, facebook, oauth, openid
- Integración perfecta con contrib.auth



<http://github.com/flashingpumpkin/django-socialregistration>

dajaxproject

<http://dajaxproject.com>

dajaxproject

- Comunicación AJAX muy sencilla
- CrossBrowsing
- Muy útil si no sabemos demasiado JS



<http://dajaxproject.com>

django-locale-url

<http://code.google.com/p/django-localeurl/>

django-locale-url

- Nos permite especificar el idioma en la url

www.example.com/en/

www.example.com/es/

- Mejoras en el SEO
 - Cada página tiene una única url.
- Configuración no tan sencilla, pero interesante.



<http://code.google.com/p/django-localeurl/>

django-piston

<http://bitbucket.org/jespern/django-piston/wiki/Home>

django-piston

- Framework para crear APIs RESTful
- Muy fácil instalación
- Serialización a:
 - JSON, YAML, Python Pyckle, XML ...
- OAuth
- Recomendado +++++

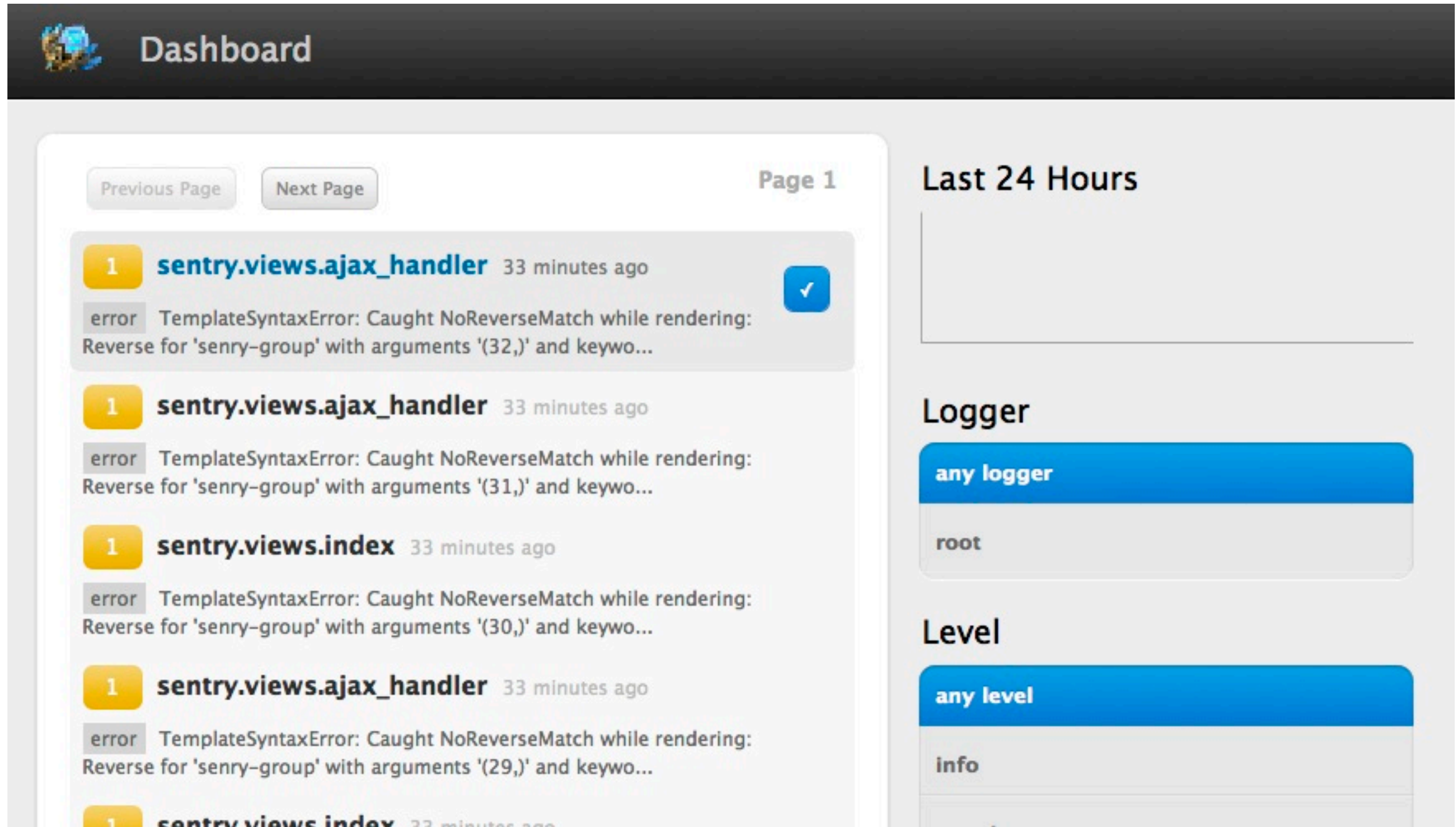


<http://bitbucket.org/jespern/django-piston/wiki/Home>

django-sentry

<http://github.com/dcramer/django-sentry>

django-sentry



The screenshot displays the Django-Sentry dashboard interface. At the top, a dark header contains the Sentry logo and the word "Dashboard". Below this, the main content area is divided into two columns. The left column, titled "Page 1", shows a list of error logs. Each log entry includes a yellow square with the number "1", the file path (e.g., `sentry.views.ajax_handler`), the time ("33 minutes ago"), and a blue square with a white checkmark. The error message for each entry is "error TemplateSyntaxError: Caught NoReverseMatch while rendering: Reverse for 'senry-group' with arguments '(32,)' and keywo...". The right column contains three sections: "Last 24 Hours" (empty), "Logger" (with a blue button labeled "any logger" and a dropdown menu showing "root"), and "Level" (with a blue button labeled "any level" and a dropdown menu showing "info").

Dashboard

Previous Page Next Page Page 1

1 **sentry.views.ajax_handler** 33 minutes ago ☒

error TemplateSyntaxError: Caught NoReverseMatch while rendering: Reverse for 'senry-group' with arguments '(32,)' and keywo...

1 **sentry.views.ajax_handler** 33 minutes ago

error TemplateSyntaxError: Caught NoReverseMatch while rendering: Reverse for 'senry-group' with arguments '(31,)' and keywo...

1 **sentry.views.index** 33 minutes ago

error TemplateSyntaxError: Caught NoReverseMatch while rendering: Reverse for 'senry-group' with arguments '(30,)' and keywo...

1 **sentry.views.ajax_handler** 33 minutes ago

error TemplateSyntaxError: Caught NoReverseMatch while rendering: Reverse for 'senry-group' with arguments '(29,)' and keywo...

1 **sentry.views.index** 33 minutes ago

Last 24 Hours

Logger

any logger

root

Level

any level

info

<http://github.com/dcramer/django-sentry>

django-debug-toolbar

The logo for Django Debug Toolbar, featuring the letters 'DjDT' in a stylized font. The 'D' and 'j' are light green, while the 'D' and 'T' are white.

Django Debug Toolbar

<http://github.com/robhudson/django-debug-toolbar>

django-command-extensions

<http://code.google.com/p/django-command-extensions/>

django-command-extensions

- Extensión para “manage.py”
 - Muchas utilidades para el día a día
 - Una de las mejores: graph_models
 - Usando GraphViz genera un modelo relacional de los modelos de nuestro proyecto.

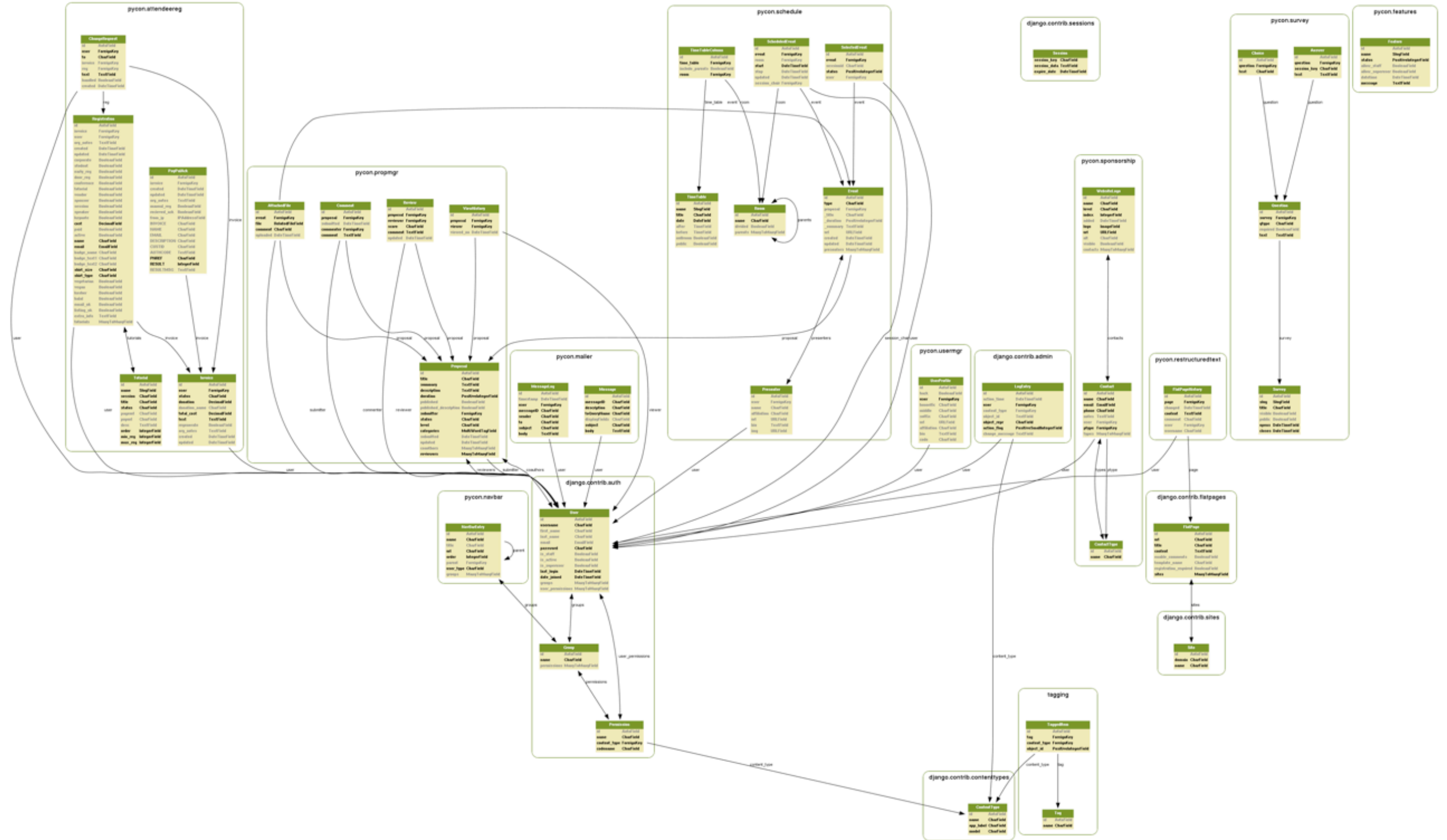
<http://code.google.com/p/django-command-extensions/>

django-command-extensions

django-command-extensions

django-command-extensions

django-command-extensions



Textmate + Django

<https://bitbucket.org/bkerr/django-textmate-bundles>

Textmate + Django

- Shortcuts para crear:
 - Modelos
 - Fields
 - Filters
 - TemplateTags
 - Forms
 - Admin

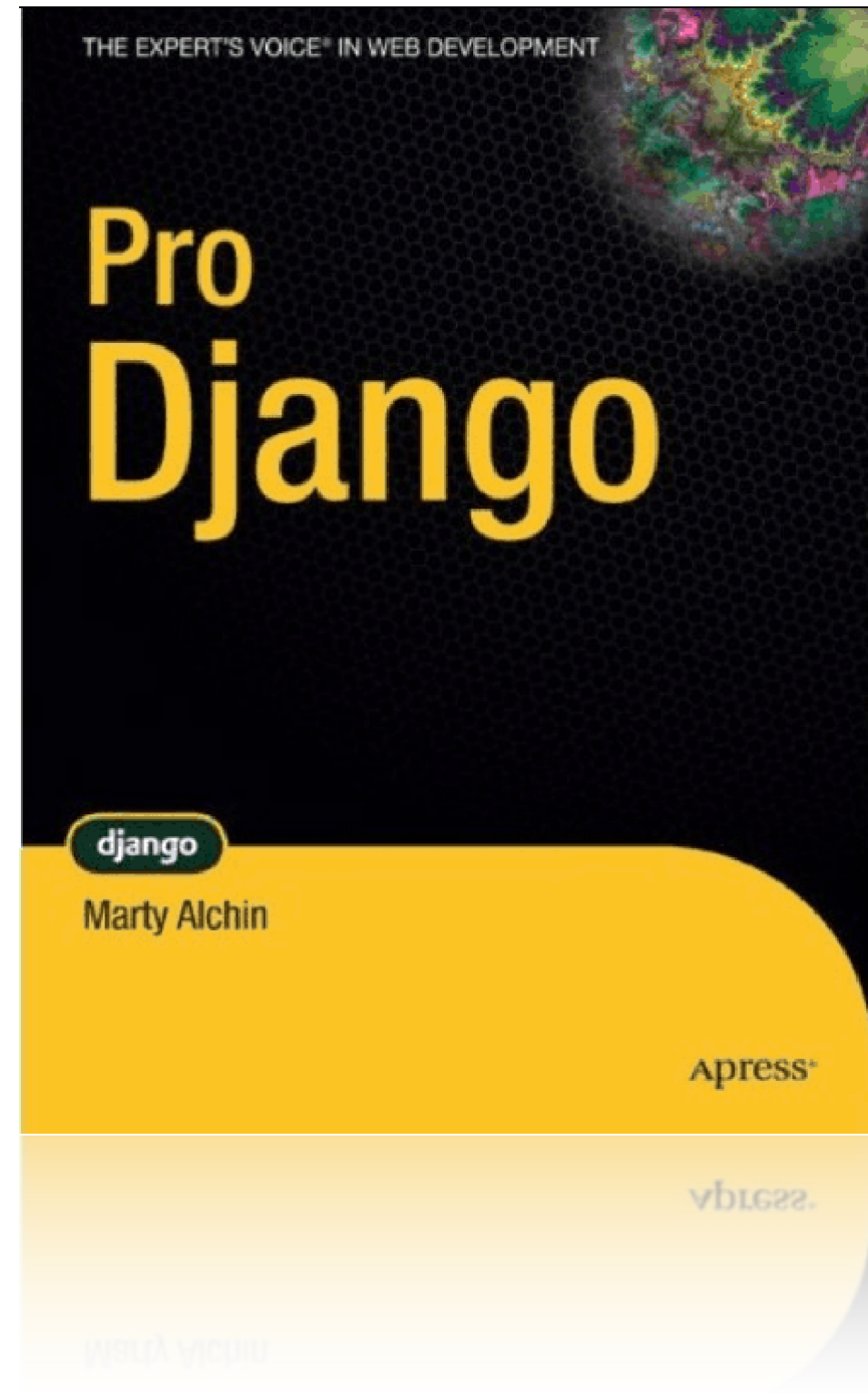
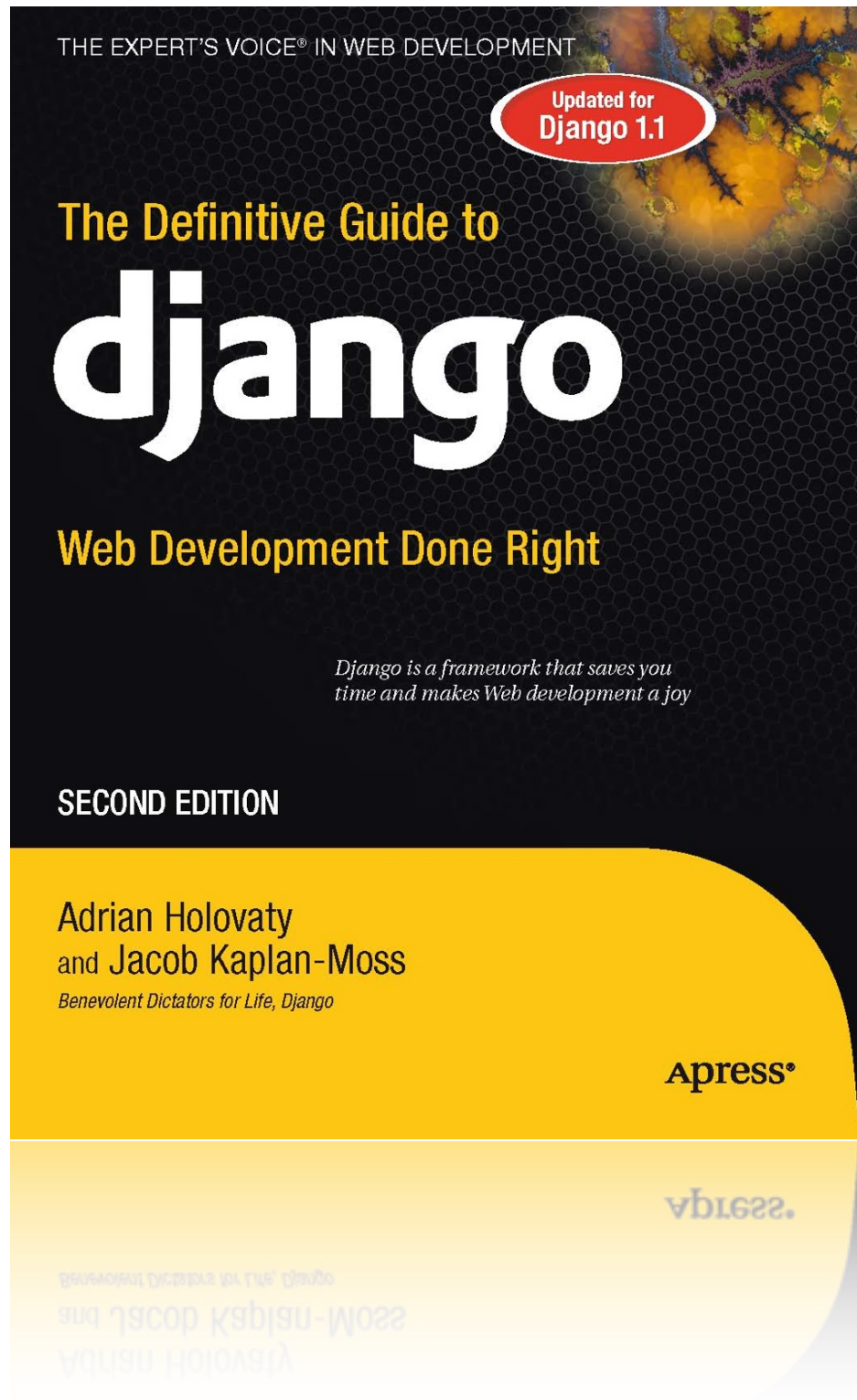


Durante el curso lo hemos utilizado.

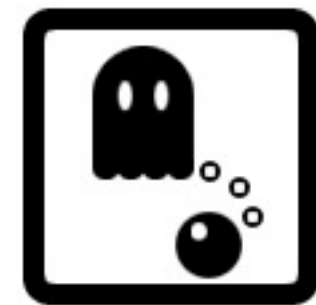
<https://bitbucket.org/bkerr/django-textmate-bundles>



La punta del iceberg



<http://www.djangobook.com/>



django

Disfruta programando.

Jorge Bastida [neo]

neo2001@gmail.com
@jorgebastida

Jaime Irurzun [etox]

jaime.irurzun@gmail.com
@jaimeirurzun