

# Servicios y Aplicaciones Telemáticas (2014-15)

## Grado en Ingeniería Tecnologías de Telecomunicación (URJC)

Jesús M. González Barahona, Gregorio Robles Martínez

<http://cursosweb.github.io>  
GSyC, Universidad Rey Juan Carlos

17 de marzo de 2015





©2002-2015 Jesús M. González Barahona, Gregorio Robles y Jorge Ferrer.

Algunos derechos reservados. Este artículo se distribuye bajo la licencia  
“Reconocimiento-CompartirIgual 3.0 España” de Creative Commons, disponible en  
<http://creativecommons.org/licenses/by-sa/3.0/es/deed.es>

Este documento (o uno muy similar) está disponible en  
<http://cursosweb.github.io>

- 1 Presentación de la asignatura
- 2 Cookies HTTP
- 3 REST: Representational State Transfer
- 4 Arquitectura modelo-vista-controlador
- 5 Introducción a XML
- 3 Hojas de estilo CSS
- 6 Prácticas: Introducción a Python
  - Transparencias principales
  - Consideraciones adicionales
- 7 Prácticas: Introducción a Django

# Presentación de la asignatura

# Datos, datos, datos

- Profesores:
  - ① Jesús M. González Barahona (jgb @ gsync.urjc.es)
  - ② Gregorio Robles (grex @ gsync.urjc.es)
- Grupo de Sistemas y Comunicaciones (GSyC)
- Despachos: 003 Biblioteca y 110 Departamental III
- Horario: M (11:00-13:00) y J (11:00-13:00)
- Tutoría: X de 16:00 a 19:00 (en los propios Laboratorios)
- Aula 314, Aulario III (esporádicamente)
- Laboratorio 209 Laboratorios III (habitualmente)

Campus virtual: <http://campusonline.urjc.es/>

GitHub: <http://cursosweb.github.io/>

¿De qué va todo esto?



# Entendiendo cómo funciona la web

# En concreto...

- Cómo se construyen los sistemas reales que se usan en Inet
- Qué tecnologías se están usando
- Qué esquemas de seguridad hay
- Cómo encajan las piezas
- En la medida de lo posible, “manos en la masa”

# Ejemplos

- ¿Qué es una aplicación web?
- ¿Cómo construir un servicio REST?
- Acaba con la magia de los servicios web
- ¿Qué es el web 2.0?
- ¿Cómo se hace un mashup?
- ¿Cómo puedo interaccionar con los servicios más populares?
- ¿Qué es HTML5?



# Fundamentos filosóficos de la asignatura

A person with short dark hair, seen from the back, is sitting at a desk. They are wearing a blue long-sleeved shirt. In front of them is a laptop and two large monitors. The laptop screen shows a blue guitar. The top monitor displays lines of code in a dark-themed editor. The bottom monitor shows a blurred image. The text "La programación es el lenguaje de la tecnología" is overlaid in orange on the lower half of the image.

**La programación es el  
lenguaje de la tecnología**

# Lenguaje de Programación: Python



Primer Mandamiento:  
Amarás Python por encima de (casi) todo.

# Plataforma: Django



## DESARROLLO WEB

### Primera Generación

HTML

CGI

### Segunda Generación

PHP

JSP

PERL

### Tercera Generación

RAILS

**DJANGO**

SYMFONY

Segundo Mandamiento:  
No tomarás el nombre de Django en vano.

# Metodología

- Objetivo principal: conceptos básicos de construcción de sitios web modernos
- Clases de teoría y de prácticas, pero...
- Teoría en prácticas, prácticas en teoría
- Uso de resolución de problemas para aprender
- Fundamentalmente, entender lo fundamental

# Fundamentos filosóficos de la asignatura



## Aprender no puede ser aburrido

# Las Clases

- Empezamos en punto
- 10 minutos con un tema motivacional
  - Gadgets tecnológicos
  - Aplicaciones
  - Cuestiones interesantes
  - ...
- Generalmente, explicación de los conceptos más importantes y luego realización de ejercicios
- No hay descanso
- Ejercicios para hacer fuera de clase (y entregar)

# Fundamentos filosóficos de la asignatura

El estudiante es el centro del aprendizaje



# Evaluación

- Teoría (obligatorio): nota de 0 a 4.
- Práctica final (obligatorio): nota de 0 a 2.
- Opciones y mejoras práctica final: nota de 0 a 3
- Prácticas incrementales: 0 a 1
- Ejercicios en foro/GitHub: nota de 0 a 2
- Nota final: Suma de notas, moderada por la interpretación del profesor
- Mínimo para aprobar:
  - Aprobado en teoría (2) y práctica final (1), y
  - 5 puntos de nota final en total



# Evaluación (2)

- Evaluación teoría: prueba escrita
- Evaluación prácticas incrementales (evaluación continua):
  - entre 0 y 1 (sobre todo las extensiones)
  - es muy recomendable hacerlas
- Evaluación práctica final
  - posibilidad de examen presencial para práctica final
  - ¡tiene que funcionar en el laboratorio!
  - enunciado mínimo obligatorio supone 1, se llega a 2 sólo con calidad y cuidado en los detalles
- Opciones y mejoras práctica final:
  - permiten subir la nota mucho
- Evaluación ejercicios (evaluación continua):
  - preguntas y ejercicios en foro/GitHub
- Evaluación extraordinaria:
  - prueba escrita (si no se aprobó la ordinaria)
  - nueva práctica final (si no se aprobó la ordinaria)

# Prácticas finales

Ejemplos del pasado:

- Servicio de apoyo a la docencia
- Sitio de intercambio de fotos
- Aplicación web de autoevaluación docente
- Agregador de blogs (canales RSS)
- Agregador de microblogs (Identi.ca, Twitter)

# Ejemplos de prácticas finales de otros años

- Fernando Yustas:

<https://www.youtube.com/watch?v=TUUMVEaBzeg>

- Miguel Ariza: <https://www.youtube.com/watch?v=fVBx9cGPjWs>

(puedes buscar en YouTube por muchos más ejemplos)

Aquí se enseñan cómo son las cosas  
que se usan en el mundo real

Las buenas noticias son...  
que no son tan difíciles



# Cookies HTTP

# Cookies

- Forma de mantener estado en un protocolo sin estado (HTTP)
- Forma de almacenar datos en el lado del cliente
- Dos versiones principales:
  - Versión 0 (Netscape)
  - Version 1 (RFC 2109 / RFC 2965)

# Cabeceras HTTP para cookies (version 0)

- Set-Cookie: De servidor a navegador

```
Set-Cookie: statusmessages="deleted"; Path=/  
Expires=Wed, 31-Dec-97 23:59:59 GMT
```

- Nombre de la cookie: statusmessages
- Valor: "deleted"
- Path: / (todo el sitio del que se recibió)
- Expira: 31-Dec-97 23:59:59 GMT

- Cookie: De navegador a servidor

```
Cookie: statusmessages="deleted"
```

- Nombre de la cookie: statusmessages
- Valor: "deleted"

RFC 2965 (version 1) tiene: "Cookie", "Cookie2" y "Set-Cookie2"

# Estructura de Set-Cookie

- Nombre y valor (obligatorios)
  - Uso normal: "nombre=valor"
  - También valor nulo: "nombre="
- Fecha de expiración
  - "Expires=fecha"
  - Si no tiene, cookie no persistente (sólo en memoria)
- Path: camino para el que es válida
  - "Path=/camino"
  - Prefijo de caminos válidos
- Domain: dominio para el que es válida
  - El servidor ha de estar en el dominio indicado
  - Si no se indica, servidor que sirve la cookie
- Seguridad: se necesita SSL para enviar la cookie
  - Campo "Secure"
- Campos separados por ";"



# Estructura de Cookie

- Lista de pares nombre - valor
- Cada par corresponde a un "Set-Cookie"
- Se envían las cookies válidas para el dominio y el path de la petición HTTP
- Si no se especificó dominio en "Set-Cookie", el del servidor
- Si no se especificó camino en "Set-Cookie", todo

Cookie: user=jgb; last=5; edited=False

# Límites para las cookies

- Originalmente: 20 cookies del mismo dominio
- La mayoría de los navegadores: 30 o 50 cookies del mismo dominio
- Cada cookie: como mucho 4 Kbytes

# Gestión de sesión en HTTP

- Mediante cookies: normalmente, identificador de sesión en la cookie

Set-Cookie: session=ab34cd-34fd3a-ef2365

- Reescritura de urls: se añade identificador a la url

http://sitio.com/path;session=ab34cd-34fd3a-ef2365

- Campos escondidos en formularios HTML

```
<form method="post" action="http://sitio.com/path">  
  <input type="hidden" name="session" value="ab34cd-34fd3a-ef2365">  
  ...  
  <input type="submit">  
</form>
```

# Referencias

- Persistent Client State HTTP Cookies  
(especificación original de Netscape)  
[http://curl.haxx.se/rfc/cookie\\_spec.html](http://curl.haxx.se/rfc/cookie_spec.html)
- RFC 2109: HTTP State Management Mechanism  
<http://tools.ietf.org/html/rfc2109>
- RFC 2965: HTTP State Management Mechanism  
<http://tools.ietf.org/html/rfc2965>

# REST: Representational State Transfer

# REST: El estilo arquitectural de la web

- REST: REpresentational State Transfer
- Estilo arquitectural para sistemas distribuidos
  - Ampliamente extendido en la web estática (ficheros y directorios)
  - Parcialmente extendido en la web programática (aplicaciones y servicios web)
- Basado en un conjunto de normas...

# REST: 1. Cada recurso debe tener una URL

Un recurso puede ser cualquier tipo de información que quiere hacerse visible en la web: documentos, imágenes, servicios, gente

```
http://example.com/profiles
```

```
http://example.com/profiles/juan
```

```
http://example.com/posts/2007/10/11/ventajas_rest
```

```
http://example.com/posts?f=2007/10/11&t=2007/11/17
```

```
http://example.com/shop/4554/
```

```
http://example.com/shop/4554/products
```

```
http://example.com/shop/4554/products/15
```

```
http://example.com/orders/juan/15
```

# REST: 1. Cada recurso debe tener una URL

Dos tipos de recursos:

- Colecciones, como `http://example.com/resources/`
- Elementos, como `http://example.com/resources/142`

Los métodos tienen un significado ligeramente diferente se trate de una colección o un elemento.



## REST: 2. Los recursos tienen hiperenlaces a otros recursos

Los formatos más usados (XHTML, Atom, ...) ya lo permiten. También aplica a los formatos inventados por nosotros mismos

```
<profiles self="http://example.com/profiles">
  <profile type="moderator"
    self="http://example.com/profiles/123">
    <name>Pepito Pérez</name>
    <company ref="http://example.com/companies/321">
      Compañía XYZ</company>
    </profile>
  ...
</profiles>
```

## REST: 3. Conjunto limitado y estándar de métodos

- GET: Obtener información (quizá de caché)
  - No cambia estado, idempotente
- PUT: Actualizar o crear un recurso conocida su URL
  - Cambia estado, idempotente
- POST: Crear un recurso conociendo la URL de un constructor de recursos
  - Cambia estado, NO idempotente
- DELETE: Borrar un recurso conocida su URL
  - Cambia estado, idempotente

## REST: 4. Múltiples representaciones por recurso

Las representaciones muestran el estado actual del recurso en un formato dado

```
GET /profile/1234
```

```
Host: example.com
```

```
Accept: text/html
```

```
GET /profile/1234
```

```
Host: example.com
```

```
Accept: text/x-vcard
```

```
GET /profile/1234
```

```
Host: example.com
```

```
Accept: application/mi-vocabulario-propio+xml
```

## REST: 5. Comunicación sin estado

- En REST hay estado, pero sólo en los recursos, no en la aplicación (o sesión).
- Una petición del cliente al servidor debe contener TODA la información necesaria para entender la misma.
- No puede basarse en información previa asociada a la comunicación (por ejemplo, información de sesión).
- No puede haber datos de sesiones del cliente en el servidor. El servidor sólo guarda y maneja el estado de los recursos que aloja.
- Es el cliente el que debe guardar su estado (y enviárselo al servidor). Sin embargo, las cookies no son una buena implementación desde el punto de vista REST.
- Esto permite escalabilidad (servidores más sencillos). Implica mayor ancho de banda, pero facilita las *cachés*.

# REST: Ventajas de REST

- Maximiza la reutilización
  - Todos los recursos tienen identificadores = hacen más grande la Web
  - La visibilidad de los recursos que forman una aplicación/servicio permite usos no previstos
- Minimiza el acoplamiento y permite la evolución
  - La interfaz uniforme esconde detalles de implementación
  - El uso de hipertexto permite que sólo la primera URL usada para acceder accope un sistema a otro
- Elimina condiciones de fallo parcial
  - Fallo del servidor no afecta al cliente
  - El estado siempre puede ser recuperado como un recurso
- Escalado sin límites
  - Los servicios pueden ser replicados en cluster, cacheados y ayudados por sistemas intermediarios
- Unifica los mundos de las aplicaciones web y servicios web. El mismo código puede servir para los dos fines.

# Funcionalidades de HTTP no tan conocidas

- Códigos de respuesta estandarizados: los entiende un navegador, los proxies, o nuestras propias aplicaciones
  - Information: 1xx, Success 2xx, Redirection 3xx, Client Error 4xx, Server Error 5xx
- Negociación de contenido: la misma URL puede mandar la representación más adecuada al cliente (HTML, Atom, texto)
- Redirecciones
- Cacheado (incluyendo mecanismos para especificar el periodo de validez y expiraciones)
  - Dos tipos: browser, proxy
  - Cabecera HTTP: Cache-Control

# Funcionalidades de HTTP no tan conocidas (2)

- Compresión
- División de la respuesta en partes (Chunking)
- GET condicional (sólo se envía la respuesta si ha habido cambios)
  - Permite ahorrar ancho de banda, procesado en el cliente y (posiblemente) procesado en el servidor
  - Dos mecanismos:
    - ETag y If-None-Match: identificador asociado al estado de un recurso. Ejemplo: hash de la representación en un formato dado
    - Last-Modified y If-Modified-Since: fecha de última actualización

# Ejemplo de interacción HTTP: Petición

GET / HTTP/1.1

Host: www.ejemplo.com

User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1;  
en-US; rv:1.8.1.3)...

Accept: text/xml,application/xml,application/xhtml+xml,  
text/html;q=0.9...

Accept-Language: en-us,en;q=0.5

Accept-Encoding: gzip,deflate

Accept-Charset: ISO-8859-1,utf-8;q=0.7,\*;q=0.7

If-None-Match: "4c083-268-423f1dc5"

If-Modified-Since: "4c083-268-423f1dc5"

Keep-Alive: 300

Connection: keep-alive

Cookie: [...]

Cache-Control: max-age=0



# Ejemplo de interacción HTTP: Respuesta

HTTP/1.x 200 OK

Date: Thu, 17 May 2007 14:06:30 GMT

Server: Microsoft-IIS/6.0

ETag: "4c083-268-423f1dc6"

Last-Modified: Mon, 21 Mar 2005 19:17:26 GMT

Cache-Control: private

Content-Type: text/html; charset=utf-8

Content-Length: 16850

# Mecanismos de seguridad

- Autenticación:
  - HTTP Basic
  - HTTP Digest
  - OAuth
- Importante: los mecanismos de autenticación de HTTP son extensibles
- Cifrado y firma digital: SSL

# Límites actuales de REST en la web

- Obliga a los desarrolladores a pensar diferente
- Carencia de soporte de PUT y DELETE en navegadores y cortado en cortafuegos
- Mecanismos de seguridad limitados (en comparación con WS-Security): está mejorando rápidamente
- La negociación de contenidos no funciona del todo bien en la práctica
- Algunos piden un método más: PATCH
- Poca documentación

# Ejemplos RESTful

- En la web
  - Todos los sitios web estáticos
  - Servicios web de sólo lectura
  - Servicios web Google (GData)
  - Aplicaciones web hechas con Ruby Rails 1.2+
- Estándares basados en REST
  - Atom Publishing Protocol
  - WebDAV
  - Amazon S3
  - GData (basado en Atom Publishing Protocol)
  - OpenSearch (basado en Atom Publishing Protocol)

# Conclusiones

- REST es un estilo arquitectural para aplicaciones distribuidas
- Introduce restricciones que producen propiedades deseables a cambio
  - Estas propiedades han permitido la expansión con éxito de la web
  - Las restricciones también pueden aplicarse a las aplicaciones y servicios web desarrollados para mantener las propiedades
- Los protocolos de la web, HTTP y URL, facilitan el desarrollo de arquitecturas RESTful y permiten explotar sus propiedades
- REST no es el único estilo arquitectural
  - Es posible eliminar restricciones si no nos importa perder propiedades
  - Hay otros estilos arquitecturales con otras propiedades. Ejemplo: RPC

# Referencias

- “RESTful Web Services”, Leonard Richardson, Sam Ruby, O'Reilly Press
- “Architectural Styles and the Design of Network-based Software Architectures” (Capítulo 5)  
[http://www.ics.uci.edu/~fielding/pubs/dissertation/rest\\_arch\\_style.htm](http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm)
- Atom Publishing Protocol:  
<http://tools.ietf.org/html/rfc5023>

# Arquitectura modelo-vista-controlador

# ¿Qué es la arquitectura MVC?

- Patrón de arquitectura que separa la lógica de aplicación de la interfaz de usuario
- Modelo: Conocimiento (datos).
- Vista: Representación del modelo (filtrado, actualización)
- Controlador: Enlace entre el usuario y la aplicación. Permite que las vistas sean presentadas al usuario.
- Ejemplo: HTML (modelo), CSS (vista), navegador (controlador)



# MVC en aplicaciones web (1)

*The model is any of the logic or the database or any of the data itself. The view is simply how you lay the data out, how it is displayed. [...]*

*The controller in a web app is a bit more complicated, because it has two parts. The first part is the web server (such as a servlet container) that maps incoming HTTP URL requests to a particular handler for that request. The second part is those handlers themselves, which are in fact often called “controllers”. [...]*

“The Importance of Model-View Separation”, Terence Parr  
(continúa)

# MVC en aplicaciones web (2)

- Modelo:  
Descripción de la base de datos
- Vista:  
Interfaz de usuario (HTML) que presenta el modelo al usuario
- Controlador:  
Recibe indicaciones del usuario, indica cambios al modelo, elige vista para mostrar resultados

# MVC en Django

- Vista: Qué data se muestra al usuario (no cómo se muestra) funciones en `views.py`
- Vistas delegan la presentación en plantillas (templates)
- Modelo: Descripción de los datos clases en `models.py`
- Controlador: maquinaria de Django incluyendo `urls.py`
- Django como “plataforma MTV”: modelo, plantilla (template), vista

# Referencias

- MVC, Xerox PARC 1978-79:  
`http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html`
- “The Importance of Model-View Separation. A Conversation with Terence Parr”, by Bill Venners  
`http://www.artima.com/lejava/articles/stringtemplate.html`

# Introducción a XML

# Introducción

- XML: Extensible Markup Language
- Norma promovida por el W3C (1998)
- Es un dialecto simplificado de SGML (Standardized General Markup Language)
- Pretende ser razonablemente simple
- Se está usando en varios nuevos estándares: MathML, SMIL, (Synchronized Multimedia Integration Language), DocBook/XML, XHTML, RSS, etc.

# Características

- Lenguaje de marcado: etiquetas de componentes según su semántica
- Puede especificarse que un documento ha de estar organizado de cierta forma
- Estructura jerárquica
- Sintaxis básica:
  - Marcas:  
`<p> ... </p>`  
`<p/>`
  - Atributos:  
`<article lang="es">`

# Fichero XML sencillo

```
<?xml version="1.0" ?>
```

```
<document>
```

```
Este es el documento
```

```
</document>
```



# Ejemplo más complejo

Estructura muy simplificada de DocBook:

```
<?xml version="1.0" ?>
<article>
  <artheader>
    <title>El lenguaje XML</title>
    <author>Tim Ray</author>
  </artheader>
  <sect1>
    <para>...</para>
  </sect1>
  <sect2>
    <para>...</para>
  </sect2>
</article>
```

# Definición y validación

- XML ofrece la posibilidad de definir lenguajes (también llamados vocabularios) de etiquetas
- La definición de un lenguaje XML especifica:
  - Elementos del lenguaje
  - Anidamiento de los elementos
  - Atributos para cada elemento
  - Atributos por defecto, atributos obligatorios
- El objetivo es poder validar que un documento XML tiene la información que se espera y con la estructura correcta.

# Definición y validación

- Sistemas de definir lenguajes XML
  - DTD (Document Type Definition): Estándar clásico
  - XML Schema: Nuevo estándar. Muy completo y complejo
  - RelaxNG: Alternativa al nuevo estándar. Más sencillo
- Cada documento XML debe tener una referencia a su definición al principio del fichero (si tiene una)

# Ejemplos de lenguajes

## DocBook XML:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE article PUBLIC
    "-//Norman Walsh//DTD DocBk XML V3.1//EN"
    "/usr/lib/sgml/dtd/docbook-xml/docbookx.dtd">
```

## LaEspiral Document:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE article PUBLIC
    "-//laespiral.org//DTD LE-document 1.1//EN"
    "http://.../xml/styles/LE-document-1.1.dtd">
```

# Especificación (ejemplo)

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE joke [
  <!ELEMENT joke (start, end+)>
  <!ATTLIST joke title CDATA #IMPLIED>
  <!ELEMENT start (#PCDATA)>
  <!ELEMENT end (#PCDATA)>
]>

<joke title="Van dos''">
  <start>Van dos por una calle</start>
  <end>y se cae el del medio</end>
  <end>y luego vienen</end>
</joke>
```

# Bibliotecas para proceso XML

Tipos de bibliotecas disponibles:

- DOM
  - Document Object Model
  - Lee todo el documento, y permite recorrer el árbol de elementos
  - Permite modificar el documento XML
- SAX
  - Standard API for XML
  - Procesa el documento según está disponible
  - Es muy rápido y requiere menos memoria, pero es sólo para lectura
- Otras: variantes de DOM y SAX más sencillas pero no estándar

# Usos típicos de XML en aplicaciones web

## Gestión de interfaz de usuario en navegador:

- Manipulación de árbol DOM de la página HTML
- Realizada normalmente mediante JavaScript
- Interfaces de usuario mucho más ricas
- Normalmente: combinación con código de aplicación web

## Intercambio de datos entre aplicaciones web:

- Fuente para distintos formatos de documento (ej: documentos en Docbook servidos en HTML, ePub, etc.)
- Canales (feeds) de una web
- Interfaz para bases de datos (XPath, etc.)
- Web semántica (RDF, RDFa)

# DOM de documento HTML en JavaScript (1)

- `node.parentNode`: nodo padre
- `node.childNodes[1]`: segundo nodo hijo
- `node.firstChild`, `x.lastChild`: primer, último nodo hijo
- `document.getElementsByTagName(tag)`:  
vector con todos los elementos con etiqueta tag  
`< tag > XXX < /tag >`
- `document.getElementById(id)`:  
elemento con identificador id  
`< tagid =" id" > XXX < /tag >`



## DOM de documento HTML en JavaScript (2)

- `x.nodeValue`: valor del nodo
- `x.innerHTML`: HTML que “cuelga” del nodo
- `node.setAttribute('attr', val)`:  
asignar a atributo 'attr' a valor val
- `document.createElement(tag)`  
crear nodo tag
- `document.createTextNode`  
crear nodo con texto
- `node.appendChild(newnode)`  
añade newnode como hijo de node
- `node.removeChild(remnode)`  
elimina hijo remnode de node

# DOM de documento HTML en JavaScript: ejemplo (1)

```
<html>
  <head>
    <script type="text/javascript" charset="utf-8">
      function change()
      {
        document.getElementById('another').innerHTML =
          'Changed!';
      }
    </script>
  </head>
```

# DOM de documento HTML en JavaScript: ejemplo (2)

```
<body>
  <h1>Simple DOM demo</h1>
  <p>First sentence</p>
  <p id="sentence">Second sentence, with id "sentence"</p>
  <p id="another">Another sentence, with id "another"</p>
  <p>Text of the sentence with id "sentence":
    <script type="text/javascript" charset="utf-8">
      document.write(
        document.getElementById('sentence').innerHTML
      );
    </script>
  </p>
```

# DOM de documento HTML en JavaScript: ejemplo (3)

```
<p>Text of the second sentence:
  <script type="text/javascript" charset="utf-8">
    document.write(
      document.getElementsByTagName('p')[1].innerHTML
    );
  </script>
</p>
<form>
  <input type="button" onclick="change()"
    value="Change" />
</form>
</body>
</html>
```

# Preparación de documentos

- Los documentos se escriben en un lenguaje XML (por ejemplo Docbook)
- Usando hojas de estilo XSL (Extensible Style Language) pueden obtenerse diversos formatos
- Ventajas
  - Ayuda en la estructuración del texto
  - Simplifica el tratamiento automático
  - Simplifica la validación
  - Se separa el contenido de la presentación

# Canales (feeds) de una web

- Introducido por Netscape en 1999, para definir “canales” (feeds)
- Permite a los sitios web publicar una lista de las novedades del sitio
- Esta lista es accedida por aplicaciones lectores de feeds
- Los usuarios pueden subscribirse a los feeds y recibir avisos de los cambios
- Lenguajes XML empleados
  - RSS 0.91 (Rich Site Summary)
  - RSS 0.9 y 1.0 (RDF Site Summary)
  - RSS 2.0 (Really Simple Syndication)
  - Atom 1.0

# Ejemplo de canal RSS (1)

```
<?xml version="1.0" encoding="UTF-8"?>
<rss version="2.0">
<channel>
  <title>Menéame: publicadas</title>

  <link>http://meneame.net</link>
  <image>
    <title>Menéame: publicadas</title>
    <link>http://meneame.net</link>
    <url>http://meneame.net/img/mnm/eli-rss.png</url>
  </image>
  <description>Sitio colaborativo de publicación y comunicación
    blogs</description>
```

## Ejemplo de canal RSS (2)

```
<pubDate>Sat, 22 Nov 2008 13:45:02 +0000</pubDate>
<item>
  <title>¿Qué pasa cuando un familiar te pide que le montes
    PC?</title>
  <link>http://feedproxy.google.com/...-montes-pc</link>
  <pubDate>Sat, 22 Nov 2008 12:55:03 +0000</pubDate>
  <description>&lt;p&gt;Un breve relato que viene a ...
  </description>
</item>
<item>...</item>
<item>...</item>
</channel>
```



# Atom

- Creado como evolución de RSS 2.0
- Mejoras que introduce:
  - Identificación del tipo de contenido de los elementos summary y content
  - Incorpora XML namespaces para permitir extender Atom con otros vocabularios XML
- Mucho más apropiado para su interpretación por parte de programas:
  - Esto está provocando su uso habitual en servicios web RESTful

# Ejemplo de canal Atom (1)

```
<?xml version="1.0" encoding="utf-8"?>
<feed xmlns="http://www.w3.org/2005/Atom">
  <title>Ejemplo</title>
  <subtitle>Muy interesante</subtitle>
  <link href="http://example.org/">
  <updated>2003-12-13T18:30:02Z</updated>
  <author>
    <name>John Doe</name>
    <email>johndoe@example.com</email>
  </author>
  <id>urn:uuid:60a76c80-d399-11d9-b93C-0003939e0af6</id>
```

## Ejemplo de canal Atom (2)

```
<entry>
  <title>Atom-Powered Robots Run Amok</title>
  <link href="http://example.org/2003/12/13/atom03"/>
  <id>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efa6a</id>
  <updated>2003-12-13T18:30:02Z</updated>
  <summary type="text">Some text.</summary>
</entry>
</feed>
```

# Tipos de contenido en Atom

```
<content type="text">
```

```
    Contenido del feed
```

```
</content>
```

```
<content type="html">
```

```
    Contenido del <em>feed</em>;
```

```
</content>
```

```
<content type="xhtml" xmlns:xhtml="http://www.w3.org/1999/xhtml">
```

```
    <xhtml:div>
```

```
        Contenido del <xhtml:em>feed</xhtml:em>
```

```
    </xhtml:div>
```

```
</content>
```

# Web semántica: RDF

- RDF: Resource Description Framework
  - Mecanismo uniforme para especificar la semántica de un recurso web, y su relación con otros.
  - Cada documento RDF es una serie de tripletes.
- Modelo de metadatos basado en declaraciones sobre recursos
- Sujeto (recurso), predicado (relación o aserción entre sujeto y objeto) y objeto (o valor)
- Una colección de declaraciones RDF constituyen un grafo orientado y etiquetado
- Es habitual que el recurso sea un URI, el objeto una cadena Unicode, y el predicado un URI

# Otras tecnologías de la familia XML

- XLink: enlaces entre documentos XML
- XSLT: conversión automática de XML a otros formatos
- XPath: localización dentro de un documento (“query” sobre XML)

# XML Namespaces

- Permiten agrupar varios vocabularios XML en un único documento
- Resuelven el problema de colisiones entre etiquetas con el mismo nombre

# Ejemplo: Problema

Introducir en el canal de una web información geográfica

- Existen dos lenguajes XML distintos para cada información:
  - RSS o Atom: Permiten dar información del canal y sus items
  - geoRSS: Permite especificar longitud y latitud



# Ejemplo: Solución

```
<rss xmlns:georss="http://www.georss.org/georss"
    version="2.0">
<channel>
  <item>
    <title>En Málaga colocar publicidad en los parabrisas de
      los coches se multará con hasta 750 euros</title>
    <georss:point>36.7196745 -4.4200359</georss:point>
    ...
```

# Uso de Namespaces en la práctica

- El parser empleado debe soportar namespaces: la mayoría lo hacen ya
- Problema: el mecanismo de validación original de XML, los DTDs, no proporciona ninguna solución para validar documentos XML con namespaces
  - Solución: usar XML Schema o RelaxNG

# Alternativa a XML: JSON

- JavaScript Object Notation
- Definido en RFC 4627
- Internet media type: application/json.
- Permite representar estructuras de datos
- Específicamente pensado para aplicaciones AJAX
- Es un subconjunto de JavaScript (y de python)

# Ejemplo de JSON

```
{  
  "firstName": "John",  
  "lastName": "Smith",  
  "address": {  
    "streetAddress": "21 2nd Street",  
    "city": "New York",  
    "state": "NY",  
    "postalCode": 10021  
  },  
  "phoneNumbers": [  
    "212 732-1234",  
    "646 123-4567"  
  ]  
}
```

# Uso en la práctica de JSON

Existen intérpretes para casi cualquier lenguaje:

- JavaScript: La función `eval()` es capaz de interpretarlo directamente
- Python: `python-cjson`, `python-json`, ...
- Java: JSON Tools, `xstream`, `Restlet`, ...

Extensiones:

- JSOINT: Equivalente a XSLT para JSON
- JSONP: convierte el contenido en activo al incluir la invocación a una función JavaScript.

# Ventajas e Inconvenientes

## Ventajas

- Requiere menos caracteres para la misma información (consume menos ancho de banda)
- Es fácil escribir un intérprete rápido (incluso en un navegador)
- Fácil de leer por personas

## Inconvenientes

- No tiene mecanismos de definición de lenguajes y validación
- No hay equivalentes a Atom, XSLT, ...

# Referencias

- Introducción al XML, por Jaime Villate:  
<http://gsyc.escet.urjc.es/actividades/linuxprog/xml/xml-notas.html>
- Especificación de XML:  
<http://www.w3.org/TR/xml-spec.html>
- Especificación anotada de XML:  
<http://www.xml.com/xml/pub/axml/axmlintro.html>
- Python and XML Processing:  
<http://pyxml.sourceforge.net/topics/>
- SIG for XML Processing in Python:  
<http://www.python.org/sigs/xml-sig/>

# Referencias II

- Python/XML HOWTO:  
<http://py-howto.sourceforge.net/xml-howto/>
- XML Linking Language (XLink):  
<http://www.w3.org/TR/xlink/>
- XML Path Language (XPath):  
<http://www.w3.org/TR/xpath>
- XPath Tutorial, por Miloslav Nic y Jiri Jirat:  
<http://www.zvon.org/xxl/XPathTutorial/General/examples.html>
- XML Schema:  
<http://www.w3.org/XML/Schema>



## Referencias III


- Using W3C XML Schema, por Eric van der Vlist:  
<http://www.xml.com/pub/a/2000/11/29/schemas/part1.html>
- XSL Transformations (XSLT):  
<http://www.w3.org/TR/xslt>
- Página DOM del W3C:  
<http://www.w3.org/DOM/>
- Página de SAX: <http://www.megginson.com/SAX/>
- Página de RDF: <http://www.w3.org/RDF/>
- RSS 1.0:  
<http://groups.yahoo.com/group/rss-dev/files/specification.html>

# Referencias IV

- XML Namespaces:  
<http://www.w3.org/TR/REC-xml-names/>
- XML Schema:  
<http://www.w3.org/XML/Schema>
- Relax NG:  
<http://relaxng.org>
- Atom (RFC4287):  
<http://tools.ietf.org/html/rfc4287>
- W3C DOM -Introduction:  
<http://www.quirksmode.org/dom/intro.html>
- JSON:  
<http://json.org/>

# Prácticas: Introducción a Python

# Python

A close-up of Yoda from Star Wars, looking slightly to the right with a serious expression. He is wearing his characteristic grey robe over a red garment. His hands are clasped in front of him.

**Python el lenguaje de los  
verdaderos maestros es**

# Transparencias principales

# Transparencias principales

(las que veremos en clase)

# Hola Mundo

- Desde la shell, accede al intérprete de Python:

```
$ python  
>>>
```

- Y ya podemos introducir instrucciones en Python:

```
>>> print "hola mundo"  
hola mundo
```

- A partir de ahora obviaremos generalmente los >>> del intérprete.

# Más ejemplos

- Podemos usar Python como calculadora (ojo:  $3/2=1$ )
- Verás que Python es sensible mayúsculas
- En Python hay diferentes tipos de datos
- Los comentarios se indican con #

```
print "hola mundo"      # esto es un comentario
euros = 415
pesetas = euros * 166.386
print str(euros) + " euros son " + str(pesetas) + " pesetas"
```

# Sangrado y separadores de sentencias

- ¡En Python NO hay llaves ni begin-end para encerrar bloques de código!
- Un mayor nivel de sangrado indica que comienza un bloque, y un menor nivel indica que termina un bloque.
- Ejemplo:

# Ejemplo de dos funciones en Python

```
def a_centigrado(faren):  
    """Convierte grados fahrenheit en grados centígrados"""  
    return (faren - 32) * (5.0/9)
```

```
def a_fahrenheit(cels):  
    """Convierte grados centígrados en grados fahrenheit"""  
    return (cels * 1.8) + 32
```



# Condicional

Sentencia if:

```
entero = 3
if entero:
    print 'verdadero'
else:
    print 'falso'
```

Nótese como el caracter `:` introduce cada bloque de sentencias. Si hay `:`, entonces la siguiente línea estará indentada.

# Cadenas

- No existe tipo char
- Comilla simple o doble  
`print "hola" o print 'hola'`  
`print 'me dijo "hola"'`  
más legible que `print 'me dijo \'hola\''`
- Puede haber caracteres especiales  
`print "hola\nque tal"`
- El operador `+` concatena cadenas, y el `*` las repite un número entero de veces

# Listas

- Tipo de datos predefinido en Python, va mucho más allá de los arrays
- Es un conjunto **indexado** de elementos, no necesariamente homogéneos
- Sintaxis: Identificador de lista, mas índice entre corchetes
- Cada elemento se separa del anterior por un caracter ,

```
colores = ['rojo', 'amarillo']  
colores.append('verde')  
print colores  
print colores[2]  
print len(colores)
```

```
cosas = ['uno', 2, 3.0]
```

# Más sobre listas

- El primer elemento tiene índice 0.
- Un índice negativo accede a los elementos empezando por el final de la lista. El último elemento tiene índice -1.
- Pueden referirse **rodajas** (*slices*) de listas escribiendo dos índices entre el caracter :
- La rodaja va desde el **primero, incluido**, al **último, excluido**.
- Si no aparece el primero, se entiende que empieza en el primer elemento (0)
- Si no aparece el segundo, se entiende que termina en el último elemento (incluido).

# Ejemplos de listas

```
lista = [0, 1, 2, 3, 4]
print lista          # [0, 1, 2, 3, 4]
print lista[1]       # 1
print lista[0:2]      # [0, 1]
print lista[3:]       # [3, 4]
print lista[-1]       # 4
print lista[:-1]      # [0, 1, 2, 3]
print lista[:-2]      # [0, 1, 2]
```

¡La misma sintaxis se aplica a las cadenas!

```
cadena = "tortilla"
print cadena[-1]
```

# Bucles

Sentencia for:

```
>>> amigos = ['ana', 'jacinto', 'guillermo', 'jennifer']
>>> for invitado in amigos:
...     print invitado, len(invitado)
...
ana 3
jacinto 7
guillermo 9
jennifer 8
```

# Diccionarios

- Es un conjunto **desordenado** de elementos
- Cada elemento del diccionario es un par clave-valor.
- Se pueden obtener valores a partir de la clave, pero no al revés.
- Longitud variable
- Elementos heterogéneos
- Hace las veces de los *registros* en otros lenguajes
- Atención: Se declaran con {}, se refieren con []

# Más sobre diccionarios

```
países = {'de': 'Alemania', 'fr': 'Francia', 'es': 'España'}  
print países; print países["fr"]
```

```
extensiones = {}  
extensiones['py'] = 'python'  
extensiones['txt'] = 'texto plano'  
extensiones['ps'] = 'PostScript'
```

```
for país in países: # iteramos por el diccionario  
    print país, países[país]
```

```
del países['fr'] # borra esta llave (y su valor)  
print len(países) # devuelve el número de elementos en el diccionario  
países.clear() # vacía el diccionario
```



# Sobre los diccionarios

- Asignar valor a una clave existente reemplaza el antiguo
- Una clave de tipo cadena es sensible a mayúsculas/minúsculas
- Pueden añadirse entradas nuevas al diccionario
- Los diccionarios se mantienen desordenados
- Los valores de un diccionario pueden ser de cualquier tipo
- Las claves pueden ser enteros, cadenas y algún otro tipo
- Pueden borrarse un elemento del diccionario con `del`
- Pueden borrarse todos los elementos del diccionario con `clear()`

# Python - Características principales

Parémonos a ver las características de Python. Python es:

- de alto nivel
- interpretado (no compilado)
- orientado a objetos (todo son objetos)
- dinámicamente tipado (frente a estáticamente tipado)
- fuertemente tipado (frente a débilmente tipado)
- sensible a mayúsculas/minúsculas

# Utilizando un editor o un IDE (I)

- Usar el intérprete de Python para programar es tedioso
- Es mejor utilizar cualquier editor de texto (p.ej., gedit) o un IDE (como Eclipse)
- Lo que crearemos son ficheros de texto plano.
- Se puede añadir información en la parte superior del fichero para indicar a la shell que es un fichero en Python y con caracteres UTF-8 (y así añadir eñes y tildes, p.ej.).

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

print "¡Hola Mundo!"
```

# Utilizando un editor o un IDE (y II)

- Si lo guardamos como `hola.py`, se ejecuta desde la línea de comandos como:

```
$ python hola.py
```

- Podemos darle permisos de ejecución al fichero:

```
$ chmod +x hola.py
```

- Y entonces, se ejecuta desde la línea de comandos como:

```
$ ./hola.py
```

# Ficheros

- `open(nombre_fichero,modo)` devuelve un objeto fichero modo:
  - `w`: Escritura. Destruye contenido anterior
  - `r`: Lectura. Modo por defecto
  - `r+`: Lectura y Escritura
  - `a`: Append
- `write(cadena)` escribe la cadena en el fichero
- `read()` devuelve el contenido del fichero
- `readlines()` devuelve una lista con cada línea del fichero
- `close()` cierra el fichero

# Ejemplos de uso de ficheros

```
#!/usr/bin/python
fich=open("/tmp/prueba","w")
fich.write("lunes\n")
fich.close()
```

```
fich=open("/tmp/prueba","a")
fich.write("martes\n")
fich.close()
```

```
fich=open("/etc/hosts","r")
maquinas=fich.readlines()
fich.close()
```

```
for maquina in maquinas:
    print maquina,
```

# Un programa en Python

```
def sum(sumando1, sumando2):  
    """Sums two integer/floats  
  
    Returns integer/float."""  
  
    return sumando1 + sumando2  
  
if __name__ == "__main__":  
    primero = int(raw_input("Please enter an integer/float: "))  
    segundo = int(raw_input("Please enter another integer/float: "))  
    print sum(primeros, segundos)
```

Ejecución:

```
$ python suma.py
```

# El atributo `__name__` de un módulo

Los módulos son objetos, con ciertos atributos predefinidos.

El atributo `__name__`:

- si el módulo es importado (con `import`), contiene el nombre del fichero, sin trayecto ni extensión
- si el módulo es un programa que se ejecuta sólo, contiene el valor `__main__`

Puede escribirse ejecución condicionada a cómo se use el módulo:

```
if __name__ == "__main__":  
    ...
```



# Importar módulos

- `import nombre-módulo`  
permite acceder a los símbolos del módulo con la sintaxis `nombre-módulo.X`
- `from nombre-módulo import a, b, c`  
incorpora los símbolos `a`, `b`, `c` al espacio de nombres, siendo accesibles directamente (sin cualificarlos con el nombre del módulo)
- `from nombre-módulo import *`  
incorpora los símbolos del módulo al espacio de nombres, siendo accesibles directamente (sin cualificarlos con el nombre del módulo).

# PEP 8 (Python Enhancement Proposal #8)

- Guía de estilo para programar en Python
- Es (todavía) más estricta que el propio intérprete
- Mejora la legibilidad y mantenibilidad del código
- Parcialmente, se puede comprobar con pep8

```
$ pep8 pepe.py
pepe.py:248:30: E225 missing whitespace around operator
pepe.py.py:248:80: E501 line too long (97 > 79 characters)
[...]
```

# Consideraciones adicionales

## Consideraciones Adicionales

(transparencias de referencia)

# Ámbito de las variables

- Las variable declaradas fuera de una función son globales

```
#!/usr/bin/python
numero = 5
def f(parametro):
    return parametro + numero
print f(3)      # 8
```

- Las variable declaradas dentro de una función son locales

```
#!/usr/bin/python
def f(parametro):
    numero = 5
    return parametro + numero
print f(3)
print numero    # ERROR: numero es de ambito local
```

# Más sobre ámbito de variables

- Dentro de una función se puede ver una variable global pero no modificar

```
#!/usr/bin/python
numero = 5
def f(parametro):
    numero = numero-1    #ERROR: no se puede modificar variable global
    return parametro + numero

print f(3)
```

- A menos que se use la sentencia global

```
#!/usr/bin/python
numero = 5
def f(parametro):
    global numero    # permite modificar una variable global
    numero = numero-1
    return parametro + numero

print f(3)    # 7
print numero    # 4
```

# Más sobre ámbito de variables

- Un poco más complicado:

```
#!/usr/bin/python
numero = 5
def f(parametro):
    numero = 4    # ahora numero es variable local
    return parametro + numero

print f(3)    # 7
print numero  # 5
```

# Definición de variables

Python es

- fuertemente tipado (frente a débilmente tipado)
- sensible a mayúsculas/minúsculas

En Python la declaración de variables es implícita  
(no hay declaración explícita)

- Las variables “nacen” cuando se les asigna un valor
- Las variables “desaparecen” cuando se sale de su ámbito

## Sangrado y separadores de sentencias (II)

- Las sentencias se terminan al acabarse la línea (salvo casos especiales donde la sentencia queda “abierta”: en mitad de expresiones entre paréntesis, corchetes o llaves).
- El caracter `\` se utiliza para extender una sentencia más allá de una línea, en los casos en que no queda “abierta”.
- El caracter `:` se utiliza como separador en sentencias compuestas. Ej.: para separar la definición de una función de su código.
- El caracter `;` se utiliza como separador de sentencias escritas en la misma línea.



# Tuplas

Tipo predefinido de Python para una lista inmutable.

Se define de la misma manera, pero con los elementos entre paréntesis.

Las tuplas no tienen métodos: no se pueden añadir elementos, ni cambiarlos, ni buscar con `index()`.

Sí puede comprobarse la existencia con el operador `in`.

```
>>> tupla = ("a", "b", "mpilgrim", "z", "example")
```

```
>>> tupla[0]
```

```
'a'
```

```
>>> 'a' in tupla
```

```
1
```

```
>>> tupla[0] = "b"
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in ?
```

```
TypeError: object doesn't support item assignment
```

## Utilidad de las tuplas:

- Son más rápidas que las listas
- Pueden ser una clave de un diccionario (no así las listas)
- Se usan en el formateo de cadenas

`tuple(lista)` devuelve una tupla con los elementos de la lista `lista`

`list(tupla)` devuelve una lista con los elementos de la tupla `tupla`

# Funciones predefinidas

- `abs()` valor absoluto
- `float()` convierte a float
- `int()` convierte a int
- `str()` convierte a string
- `round()` redondea
- `raw_input()` acepta un valor desde teclado

# Operadores

En orden de precedencia decreciente:

```
+x, -x, ~x      Unary operators
x ** y         Power
x * y, x / y, x % y    Multiplication, division, modulo
x + y, x - y     Addition, subtraction
x << y, x >> y    Bit shifting
x & y           Bitwise and
x | y           Bitwise or
x < y, x <= y, x > y, x >= y, x == y, x != y,
x <> y, x is y, x is not y, x in s, x not in s
                                Comparison, identity,
                                sequence membership tests
not x           Logical negation
x and y         Logical and
lambda args: expr      Anonymous function
```

- La declaración implícita de variables como en perl puede provocar resultados desastrosos

```
#!/usr/bin/perl
$sum_elementos= 3 + 4 + 17;
$media=suma_elementos / 3;    # deletreamos mal la variable
print $media;    # y provocamos resultado incorrecto
```

- Pero Python no permite referenciar variables a las que nunca se ha asignado un valor.

```
#!/usr/bin/python
sum_elementos = 3 + 4 + 17
media = suma_elementos / 3    # deletreamos mal la variable
print media    # y el compilador nos avisa con un error
```

adi

# Operaciones sobre cadenas

- `join()` devuelve una cadena que engloba a todos los elementos de la lista.
- `split()` devuelve una lista dividiendo una cadena
- `upper()` devuelve la cadena en mayúsculas
- `lower()` devuelve la cadena en minúsculas

Estas funciones de biblioteca, como todas, podemos encontrarlas en la *python library reference* (disponible en el web en muchos formatos)

# Más sobre cadenas

```
#!/usr/bin/python
```

```
cadena = "más vale pájaro en mano"  
print cadena.split()  
print cadena.upper()
```

```
otra_cadena = "que,cocodrilo,en,tobillo"  
print otra_cadena.split(',')
```

```
lista = ['rojo', 'amarillo', 'verde']  
print lista.join()
```

# Operaciones sobre diccionarios

- `len(d)` devuelve el número de elementos de `d`
- `d.has_key(k)` devuelve 1 si existe la clave `k` en `d`, 0 en caso contrario
- `k in d` equivale a: `d.has_key(k)`
- `d.items()` devuelve la lista de elementos de `d`
- `d.keys()` devuelve la lista de claves de `d`



# Recogiendo datos del usuario con `raw_input`

```
#!/usr/bin/python
entero = int(raw_input("Please enter an integer: "))
if entero < 0:
    entero = 0
    print 'Negative changed to zero'
elif entero == 0:
    print 'Zero'
elif entero == 1:
    print 'Single'
else:
    print 'More'
```

No existe `switch/case`

# Más sobre listas

- `append()` añade un elemento al final de la lista
- `insert()` inserta un elemento en la posición indicada

```
>>> lista
['a', 'b', 'mpilgrim', 'z', 'example']
>>> lista.append("new")
>>> lista
['a', 'b', 'mpilgrim', 'z', 'example', 'new']
>>> lista.insert(2, "new")
>>> lista
['a', 'b', 'new', 'mpilgrim', 'z', 'example', 'new']
```

# Más sobre listas

- `index()` busca en la lista un elemento y devuelve el índice de la primera aparición del elemento en la lista. Si no aparece se eleva una excepción.
- El operador `in` devuelve 1 si un elemento aparece en la lista, y 0 en caso contrario.

```
>>> lista
['a', 'b', 'new', 'mpilgrim', 'z', 'example', 'new']
>>> lista.index("example")
5
>>> lista.index("new")
2
>>> lista.index("c")
Traceback (innermost last):
  File "<interactive input>", line 1, in ?
ValueError: list.index(x): x not in list
>>> "c" in lista
0
```

# Más sobre listas

- `remove()` elimina la primera aparición de un elemento en la lista. Si no aparece, eleva una excepción.
- `pop()` devuelve el último elemento de la lista, y lo elimina. (Pila)
- `pop(0)` devuelve el primer elemento de la lista, y lo elimina. (Cola)

```
>>> lista
['patatas', 'bravas', 'alioli', 'huevo', 'tortilla', 'chorizo']
>>> lista.remove("alioli")
>>> lista
['patatas', 'bravas', 'huevo', 'tortilla', 'chorizo']
>>> lista.remove("oreja")
Traceback (innermost last):
  File "<interactive input>", line 1, in ?
ValueError: list.remove(x): orija not in list
>>> lista.pop()
'chorizo'
>>> lista
['patatas', 'bravas', 'alioli', 'huevo', 'tortilla']
```

# Más sobre listas

- El operador + concatena dos listas, devolviendo una nueva lista
- El operador \* concatena repetitivamente una lista a sí misma

```
>>> lista = ['patatas', 'bravas', 'alioli']
>>> lista = lista + ['huevo', 'tortilla']
>>> lista
['patatas', 'bravas', 'alioli', 'huevo', 'tortilla']
>>> lista += ['chorizo']
>>> lista
['patatas', 'bravas', 'alioli', 'huevo', 'tortilla', 'chorizo']
>>> lista = [1, 2] * 3
>>> lista
[1, 2, 1, 2, 1, 2]
```

# Más sobre listas

- `sort()` ordena una lista. Puede recibir opcionalmente un argumento especificando una función de comparación, lo que enlentece notable su funcionamiento
- `reverse()` invierte las posiciones de los elementos en una lista.

Ninguno de estos métodos devuelve nada, simplemente alteran la lista sobre la que se aplican.

```
>>> li = ['a', 'b', 'new', 'mpilgrim', 'z', 'example', 'new', 'two', 'elements']
>>> li.sort()
>>> li
['a', 'b', 'elements', 'example', 'mpilgrim', 'new', 'new', 'two', 'z']
>>> li.reverse()
>>> li
['z', 'two', 'new', 'new', 'mpilgrim', 'example', 'elements', 'b', 'a']
```

# Asignaciones múltiples y rangos

- Pueden hacerse también tuplas de variables:

```
>>> tupla = ('a', 'b', 'e')
>>> (primero, segundo, tercero) = tupla
>>> primero
'a'
```

- La función `range()` permite generar listas al vuelo:

```
>>> range(7)
[0, 1, 2, 3, 4, 5, 6]
>>> (MONDAY, TUESDAY, WEDNESDAY, THURSDAY,
... FRIDAY, SATURDAY, SUNDAY) = range(7)
>>> MONDAY
0
>>> SUNDAY
6
```

# Mapeo de listas

- Se puede mapear una lista en otra, aplicando una función a cada elemento de la lista:

```
>>> li = [1, 9, 8, 4]
>>> [elem*2 for elem in li]
[2, 18, 16, 8]
>>> li
[1, 9, 8, 4]
>>> li = [elem*2 for elem in li]
>>> li
[2, 18, 16, 8]
```



# Filtrado de listas

- Sintaxis:

`[expresión-mapeo for elemento in lista-orig if condición-filtrado]`

- Ejemplos:

```
>>> li = ["a", "mpilgrim", "foo", "b", "c", "b", "d", "d"]
>>> [elem for elem in li if len(elem) > 1]      1
['mpilgrim', 'foo']
```

# Control de flujo

Sentencia while:

```
>>> a, b = 0, 1
>>> while b < 1000:
...     print b,
...     a, b = b, a+b
...
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
```

Nótese el efecto de un caracter , al final de un print

Nótese otro modelo de asignación múltiple

- break sale de un bucle:

```
#!/usr/bin/python
a=10
while a > 0:
    print a,
    a=a-1
```

equivale a

```
#!/usr/bin/python
a=10
while 1:
    print a,
    if a==1:
        break
    a=a-1
```

# Uso de bibliotecas

- Llamada al shell

```
#!/usr/bin/python
import os
os.system('ls -l')
```

- Argumentos de linea de comandos

```
#!/usr/bin/python
import sys
print sys.argv[1:]
```

Las funciones de biblioteca podemos encontrarlas en la *python library reference* (disponible en el web en muchos formatos)

# Excepciones

- Un programa sintácticamente correcto puede dar errores de ejecución

```
#!/usr/bin/python
while 1:
    x=int(raw_input("Introduce un n°"))
    print x
```

- Definimos una acción para determinada excepción

```
#!/usr/bin/python
while 1:
    try:
        x=int(raw_input("Introduce un n°:"))
        print x
    except ValueError:
        print ("Número incorrecto")
```

- Se puede indicar una acción para cualquier excepción pero es *muy* desaconsejable (enmascara otros errores)
- El programador puede levantar excepciones

```
#!/usr/bin/python
try:
    x=int(raw_input("Introduce un nº:"))
    print x
except :      # para cualquier excepción
    print ("Número incorrecto")

raise SystemExit
print "nunca se ejecuta"
```

# Objetos en Python

Todo son objetos, en sentido amplio:

- Cualquier objeto puede ser asignado a una variable o pasado como parámetro a una función
- Algunos objetos pueden no tener ni atributos ni métodos
- Algunos objetos pueden no permitir que se herede de ellos

Ejemplos de objetos Python: Strings, listas, funciones, módulos...



Todos los objetos tienen:

- **Identidad:**

- Nunca cambia.
- El operador `is` compara la identidad de dos objetos.
- La función `id()` devuelve una representación de la identidad (actualmente, su dirección de memoria).

- **Tipo:**

- Nunca cambia.
- La función `type()` devuelve el tipo de un objeto (que es otro objeto)

- **Valor:**

- Objetos inmutables: su valor no puede cambiar
- Objetos mutables: su valor puede cambiar

**Contenedores:** objetos que contienen referencias a otros objetos (ej.: tuplas, listas, diccionarios).

# Cadenas de documentación

- No son obligatorias pero sí muy recomendables (varias herramientas hacen uso de ellas).
- La cadena de documentación de un objeto es su atributo `__doc__`
- En una sola línea para objetos sencillos, en varias para el resto de los casos.
- Entre triples comillas-dobles (incluso si ocupan una línea).
- Si hay varias líneas:
  - La primera línea debe ser una resumen breve del propósito del objeto. Debe empezar con mayúscula y acabar con un punto
  - Una línea en blanco debe separar la primera línea del resto
  - Las siguientes líneas deberían empezar justo debajo de la primera comilla doble de la primera línea

De una sola línea:

```
def kos_root():  
    """Return the pathname of the KOS root directory."""  
    global _kos_root  
    ...
```

De varias:

```
def complex(real=0.0, imag=0.0):  
    """Form a complex number.  
  
    Keyword arguments:  
    real -- the real part (default 0.0)  
    imag -- the imaginary part (default 0.0)  
  
    """  
    if imag == 0.0 and real == 0.0: return complex_zero
```

# Documentando el código (tipo Javadoc)

- Permite documentar el código -generalmente las funciones- dentro del propio código
- Genera la documentación del código en formatos legibles y navegables (HTML, PDF...)
- Se basa en un lenguaje de marcado simple
- PERO... hay que mantener la documentación al día cuando se cambia el código

## Ejemplo

```
def interseccion(m, b):  
    """  
    Devuelve la interseccion de la curva  $M\{y=m*x+b\}$  con el eje X.  
    Se trata del punto en el que la curva cruza el eje X ( $M\{y=0\}$ ).  
  
    @type m: número  
    @param m: La pendiente de la curva  
    @type b: número  
    @param b: La intersección con el eje Y  
  
    @rtype: número  
    @return: la intersección con el eje X de la curva  $M\{y=m*x+b\}$ .  
    """  
    return -b/m
```

# Prácticas: Introducción a Django

# Enfoques comunes de desarrollo web

- Frameworks de desarrollo web
  - PHP, JavaEE, Python+HttpServer...
- Entornos de desarrollo web completos
  - Django (Python), <http://djangoproject.org>
  - Ruby on Rails (Ruby), <http://rubyonrails.org/>
  - CakePHP (PHP), <http://cakephp.org/>
  - Grails (Groovy, sobre JVM), <http://grails.org/>
  - RIFE (Java), <http://rifers.org/>
- Plataformas extensibles
  - CMS: Joomla, Drupal...
  - Portal: Plone/Zope, Liferay Portal...
  - Plataformas de propósito específico: Moodle, Wordpress...

# ¿Qué es Django?

- Entorno integrado de desarrollo de aplicaciones web
- Herramientas para gestionar la aplicación
- Framework (armazón) para presentación de la aplicación
- Acceso a base de datos (correspondencia objeto-relacional)
- Seguridad (XSS, SQL Injection, ...)
- Componentes listos para usar (gestión de usuarios, sesiones, interfaz administración,...)
- Cache, internacionalización, plantillas, etc.

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



# Django: conceptos principales

- Objetivo principal: desarrollo muy rápido
  - Entorno integrado y completo
  - Cambios en caliente
  - Descripciones de error muy descriptivas
  - Convenciones preferible a configuración
  - Evitar duplicación a toda costa (DRY, don't repeat yourself)
- Desarrollo dirigido por el modelo
  - Se comienza por el diseño del modelo de datos

# Preparativos

- Usaremos la versión 1.7.5
- Disponible para Linux, \*BSD, Windows, MacOS, etc.
- Descarga e instalación en \$DJANGO
- Descompresión desde la *shell*:

```
tar xvzf Django-1.7.5.tar.gz
```

<http://docs.djangoproject.com/en/dev/topics/install/>

<http://www.djangoproject.com/download/1.7.5/tarball/>

# Preparativos (y II)

- Preparación de entorno (necesario si no se ha instalado en path “habituales”, como p.ej. en el laboratorio):  

```
export DJANGO=/home/al-...-.../....../.../Django-1.7.5/  
# Camino al directorio con fuentes descomprimidas  
export PATH=$DJANGO/django/bin:$PATH  
export PYTHONPATH=$DJANGO:$PYTHONPATH
```
- Comprobación (debe responder 1.7.5):  

```
django-admin.py --version
```
- Incluir los tres exports anteriores en el fichero `/.bashrc` para que se ejecuten con cada nuevo terminal.
- Abrir un nuevo terminal y comprobar que funciona (ver paso anterior).

# Armazón para proyecto y aplicación

- Creación (primero proyecto, luego aplicación)  
\$ cd dir-practica  
\$ django-admin.py startproject myproject  
\$ cd myproject  
\$ python manage.py startapp myfirstapp
- Más opciones de manage.py  
\$ python manage.py --help
- Ejecución de la aplicación (<http://localhost:1234>)  
\$ python manage.py runserver 1234

# Ficheros creados en el almacén

- Raíz:
  - `manage.py`: herramienta para gestionar el proyecto
- Proyecto:
  - `__init__.py`: fichero vacío, directorio debe ser considerado un paquete Python
  - `settings.py`: configuración del proyecto
  - `urls.py`: URLs de las aplicaciones del proyecto
  - `wsgi.py`: fichero para servir Django mediante WSGI (p.ej. con Apache)
- Aplicación:
  - `__init__.py`: fichero vacío, directorio debe ser considerado un paquete Python
  - `models.py`: definición de las clases del modelo de datos
  - `views.py`: vistas (código invocado para cada recurso)
  - `tests.py`: fichero para la implementación de tests

# Fichero settings.py

- Fichero de configuración, en Python
- Configuración de la base de datos (usaremos SQLite3)

```
ENGINE = 'django.db.backends.sqlite3'
```

```
NAME = 'myproject.sqlite'
```

```
USER = ''
```

```
PASSWORD = ''
```

```
HOST = ''
```

```
PORT = ''
```

- Aplicaciones instaladas

```
INSTALLED_APPS = (  
    'myfirstapp',  
)
```

- Otros: zona horaria, codificación, directorio de plantillas...

# Declaración de URLs

- En el fichero `urls.py`
- Usa expresiones regulares para asociar URLs (sin parámetros) a vistas
- Ejemplo:

```
urlpatterns = patterns('',
    url(r'^$', 'myfirstapp.views.say_main',),
    url(r'^hello', 'myfirstapp.views.say_hello',),
    url(r'^bye/(.*)', 'myfirstapp.views.say_bye_to',),
    url(r'^num/(?P<num>[\d]+)', 'myfirstapp.views.say_num',),
)
```

[http://es.wikipedia.org/wiki/Expresi%C3%B3n\\_regular](http://es.wikipedia.org/wiki/Expresi%C3%B3n_regular)

<https://docs.djangoproject.com/en/dev/topics/http/urls/>

# Views

- Código invocado para una URL o conjunto de URLs
- Debe ser un método (o un objeto)
- Los métodos se definen en el fichero myfirstapp/views.py
- Ejemplo:

```
from django.http import HttpResponse
def say_main(request):
    return HttpResponse('<h1>My Application</h1>')
def say_hello(request):
    return HttpResponse('Hello!')
def say_bye_to(request, name):
    return HttpResponse('Bye %s'%name)
def say_number(request, number=0):
    return HttpResponse('Number: %s'%number)
```



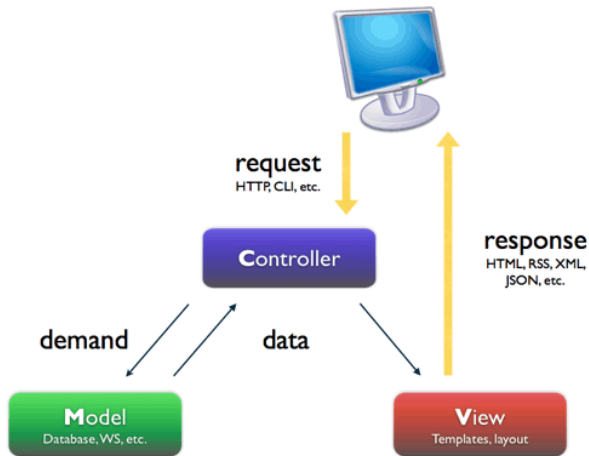
# Gestión de datos persistentes

- Django hace corresponder un objeto Python con cada tabla
- Cada aplicación tiene su `models.py`
  - Una clase por cada entidad (tabla) del modelo
  - Un campo por cada dato (columna) de la entidad
  - Ejemplo:

```
class MyFirstAppData(models.Model):  
    name = models.CharField(max_length=200)  
    birthday = models.DateTimeField()
```

- Creación de tablas  
\$ `python manage.py syncdb`

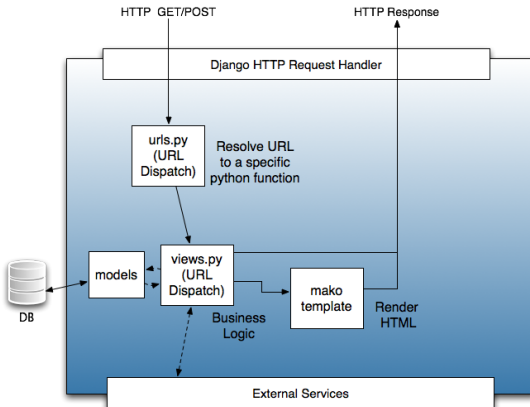
# Model View Controller (el tradicional)



Fuente:

<http://djangoexamples.blogspot.com.es/2013/05/about-django-mvc.html>

# Model View Template (el de Django)



Fuente: <http://archive.cloudera.com/cdh4/cdh/4/hue/sdk/sdk.html>

# Definición del modelo

- Tipos de campos:
  - CharField(maxlength)
  - TextField()
  - IntegerField()
  - DateField()
  - BooleanField()
- Relaciones:
  - ForeignKey(othermodel)
  - ManyToManyField('self', symmetrical=False)

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

# Otras acciones de gestión del proyecto

- Ejecución en el contexto de Python con acceso al código de la aplicación  
`% python manage.py shell`
- Validación de modelos de datos  
`% python manage.py validate`
- Exportación de datos de la base de datos  
`% python manage.py dumpdata`
- Importación de datos en la base de datos  
`% python manage.py loaddata`

# Seguridad

- Django viene con alguna protección de seguridad por defecto
- Por ejemplo, CSRF: *Cross-site request forgery* o falsificación de petición en sitios cruzados
- Si el POST/PUT no incluye un csrf\_token creado con anterioridad por el servidor, da un error de seguridad
- Se evita con el @csrf\_exempt para un método concreto en views.py, de la siguiente manera:

```
@csrf_exempt
def vista(request):
    if request.method == "POST":
        return HttpResponse("Has enviado " + request.body)
```

# Consultas a la base de datos

- Métodos para realizar consultas a la base de datos
- Acceso a entradas de la base de datos mediante el objeto 'objects' (ej. `MyFirstAppData.objects`)
- Métodos:
  - `MyFirstAppData.save()`
  - `MyFirstAppData.objects.all()`
  - `MyFirstAppData.objects.filter(campo=valor)`
  - `MyFirstAppData.objects.get(campo=valor)`  
Excepción si no lo encuentra

# Acceso al modelo desde las vistas

- Las vistas pueden usarse para leer y modificar al modelo

```
from django.http import HttpResponseRedirect,HttpResponseNotFound
from content.models import Pages
```

```
def show_content(request, resource):
    try:
        record = Pages.objects.get(name=resource)
        return HttpResponseRedirect(record.page)
    except Pages.DoesNotExist:
        return HttpResponseRedirect(
            'Page not found: /%s.' % resource
        )
```



# La shell de Django

Acceso a la API de los objetos de nuestro proyecto

```
% python manage.py shell
>>> from myproject.myfirstapp.models import MyFirstAppData
>>> MyFirstAppData.objects.all()
[]
>>> p = MyFirstAppData(name="Jesus",
                        birthday="2009-05-05")
>>> p.save()
>>> p.id
1
>>> MyFirstAppData.objects.filter(name="Jesus")
...
>>> MyFirstAppData.objects.get(pk=1).name
u'Jesus'
```

# Usuarios

- INSTALLED\_APPS (en settings.py) ha de incluir:
  - django.contrib.auth
  - django.contrib.contenttypes
- Hay que crear las tablas pertinentes (manage.py syncdb)

# Admin site

Versión simple:

- INSTALLED\_APPS (en settings.py) ha de incluir:
  - django.contrib.admin
  - django.contrib.sessions (dependencia del anterior)
  - ...y lo necesario para usuarios
- Hay que crear las tablas pertinentes (manage.py syncdb)
- Enganche en urls.py

```
from django.contrib import admin
admin.autodiscover()

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

## Admin site (2)

Ahora, proporcionemos interfaz para nuestra tabla Pages:

- Crea en el directorio de la aplicación Django de gestión de contenidos el fichero `admin.py`
- Registra en él los modelos a manejar:

```
from django.contrib import admin
from cms_users.content.models import Pages
```

```
admin.site.register(Pages)
```

- Prueba que ahora puedes manejar esta tabla desde el sitio de administración

# Acceso a información de usuario y logout

- Accedemos a información del objeto User, que tenemos en `HttpRequest`
- En `views.py`:

```
def show_content(request, resource):  
    if request.user.is_authenticated():  
        logged = 'Logged in as ' + request.user.username  
    else:  
        logged = 'Not logged in.'
```

- Para logout, utilizamos view predefinida. En `urls.py`:  
`(r'^logout', 'django.contrib.auth.views.logout'),`

# Plantillas (templates)

- Ficheros de texto que pueden generar cualquier formato basado en texto (HTML, XML, CSV, etc.)
- Contienen:
  - Texto (que queda igual)
  - Variables (reemplazadas por su valor cuando se evalúan)
  - Filtros (modifican variables cuando se evalúan)
  - Etiquetas (controlan la lógica de la evaluación de la plantilla)
  - Comentarios {# Comentario #}
- Pueden extender (heredar de) otras plantillas
- Se colocan en los directorios de plantillas (TEMPLATE\_DIRS en settings.py)

# Plantillas: variables y filtros

- Variables:

```
{{ variable }}
```

- Filtros:

```
{{ variable|filtro|otrofiltro }}
```

- Filtro con argumentos:

```
{{ variable|filtro:30 }}
```

- Ejemplos de filtros:

```
{{ value|default:"nothing" }}
```

```
{{ value|length }}
```

```
{{ text|striptags }}
```

```
{{ text|truncatewords:30 }}
```

```
{{ text|escape|linebreaks }}
```

```
{{ list|join:", " }}.
```

# Plantillas: etiquetas

- for

```
{% for athlete in athlete_list %}  
    <li>{{ athlete.name }}</li>  
{% endfor %}
```

- if

```
{% if athlete_list %}  
    Number of athletes: {{ athlete_list|length }}  
{% else %}  
    No athletes.  
{% endif %}
```



# Plantillas: etiquetas (2)

- ifequal, ifnotequal

```
{% ifequal athlete.name coach.name %}
```

```
...
```

```
{% endifequal %}
```

```
{% ifnotequal athlete.name "Joe" %}
```

```
...
```

```
{% endifnotequal %}
```

- block, extends: Herencia

# Ejemplo de plantilla

```
{% extends "base.html" %}
{% block title %}{% section.title %}{% endblock %}
{% block content %}
<h1>{% section.title %}</h1>
{% for story in story_list %}
<h2>
    <a href="{% story.get_absolute_url %}">
        {% story.headline|upper %}
    </a>
</h2>
<p>{% story.tease|truncatewords:"100" %}</p>
{% endfor %}
{% endblock %}
```

# Plantillas: uso en vistas

Directorios con plantillas: `TEMPLATE_DIRS` en `settings.py`

```
from django.template.loader import get_template
from django.template import Context

def show_annotated_content(request, resource):
    ...
    template = get_template("annotated.html")
    return HttpResponse(template.render(
        Context({'user': user,
                 'resource': resource,
                 'page': page})))
```

# Login

- Utilizamos view predefinida (entiende GET y POST)
- En urls.py:  

```
url(r'^login', 'django.contrib.auth.views.login'),
```
- Necesita una plantilla registration/login.html:
  - En settings.py:  

```
TEMPLATE_DIRS = ('templates')
```
  - Creación de templates/registration/login.html

Info detallada: “User authentication in Django”

## templates/registration/login.html

```
<html><body>
<form method="post" action="/login">
{% csrf_token %}
<table>
  <tr><td>Username</td>
    <td>{{ form.username }}</td></tr>
  <tr><td>Password</td>
    <td>{{ form.password }}</td></tr>
</table>
<input type="submit" value="login" />
</form>
</body></html>
```

# Plantillas: uso en urls.py

```
from django.conf.urls.defaults import *
from django.views.generic.simple import direct_to_template

urlpatterns = patterns('',

    url(r'^about$', direct_to_template, {
        'template': 'about.html'
    }),

)
```

# Modelos: relación muchos a uno (ForeignKey)

```
class Manufacturer(models.Model):
    # ...

class Car(models.Model):
    manufacturer = models.ForeignKey(Manufacturer)
    # ...

# Creating
m = Manufacturer(name='Seat')
c = Car(name='Toledo')
m.save(); c.save()

# Relationship
c.manufacturer = m

# Obtaining
c.manufacturer
c.manufacturer.id
```

# Modelos: relación muchos a muchos (ManyToManyField)

```
class Topping(models.Model):  
    # ...  
class Pizza(models.Model):  
    # ...  
    toppings = models.ManyToManyField(Topping)
```



## Modelos: relación muchos a muchos (ManyToManyField) (2)

```
pb = Pizza(name='Barbecue')
pq = Pizza(name='4 Cheese')
b = Topping(name='Barbecue sauce')
m = Topping(name='Mozzarella')
pb.save(); pq.save(); b.save(); m.save()
pb.toppings.add(b, m)
pq.toppings.add(m)
pq.toppings.create(name='Rocheport')
m.pizza_set.all()
pb.toppings.all()
Pizza.objects.filter(toppings__name='Mozzarella')
```

# Ficheros estáticos con Django

- Los ficheros estáticos no se deberían servir con Django...
- (lo hace mucho mejor un servidor web como Apache o Cherokee)
- ...pero se pueden servir
- `django.views.static.serve()`

```
(r'^css/(?P<path>.*)$', 'django.views.static.serve',  
    {'document_root': 'sfiles/css'}),
```

# Generador de canales

- Django viene con módulos para generar canales RSS y Atom
- View de alto nivel que genera el canal (feed):

```
(r'^feeds/(?P<url>.*)/$', 'django.contrib.syndication.views.  
    {'feed_dict': feeds})),
```

- Hay que proporcionar un diccionario con la correspondencia canal a objeto Feed:

```
feeds = {  
    'latest': LatestEntries,  
    'categories': LatestEntriesByCategory,  
}
```

# Generador de canales: objetos Feed

- Representan los datos de un canal:

```
from django.contrib.syndication.feeds import Feed
from content.models import Pages
```

```
class LatestEntries(Feed):
    title = "My CMS contents"
    link = "/feed/"
    description = "Contents of my CMS."

    def items(self):
        return Pages.objects.order_by('-pub_date')[:5]
```

- También hay que definir plantillas (templates) para <title> y <description> de cada ítem del canal RSS

# Internacionalización

- Cadenas de traducción en código Python

```
from django.utils.translation import ugettext as _
```

```
def my_view(request):  
    output = _("Welcome to my site.")  
    return HttpResponse(output)
```

```
def my_view(request, m, d):  
    output = _('Today is %(month)s, %(day)s.') %  
        {'month': m, 'day': d}  
    return HttpResponse(output)
```

# Internacionalización (2)

- Cadenas de traducción en plantillas

```
<title>{% trans "This is the title." %}</title>
```

```
{% blocktrans %}
```

```
This string will have {{ value }} inside.
```

```
{% endblocktrans %}
```

# Internacionalización (3)

- Traducciones en los lenguajes requeridos

```
django-admin.py makemessages -l es
```

- Activar el soporte para locale en Django

```
MIDDLEWARE_CLASSES = (  
    'django.contrib.sessions.middleware.SessionMiddleware',  
    'django.middleware.locale.LocaleMiddleware',  
    'django.middleware.common.CommonMiddleware',  
)
```

# Referencias

- Documentación de Django  
<http://docs.djangoproject.com/en/dev>
- Libro de Django  
<http://www.djangobook.com>
- Documentación de Python  
<http://www.python.org/doc/>
- Tutorial sobre Django (e introducción a Django)  
<http://docs.djangoproject.com/en/dev/intro>