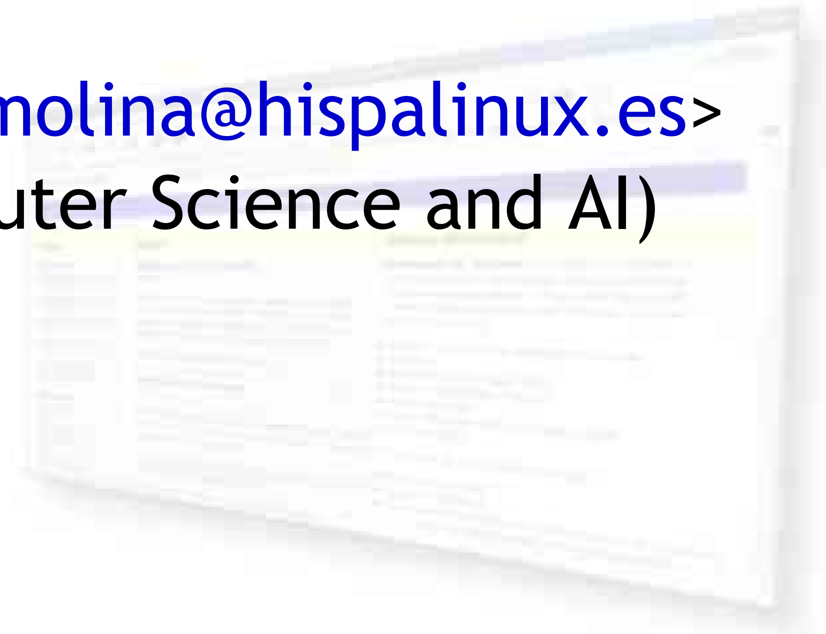
Programación Web con Python

Miguel Hernández Martos <enlavin@gcubo.org>
Ingeniería y Control Remoto S.A.

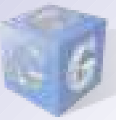
Daniel Molina Cabrera <daniel.molina@hispalinux.es>
DECSAI (Department of Computer Science and AI)



Programación web

- Protocolo HTTP
- Carencia de estado en el cliente
- Interfaces de usuario definidas con HTML
- Generalmente multiplataforma

Alternativa 1: CGI



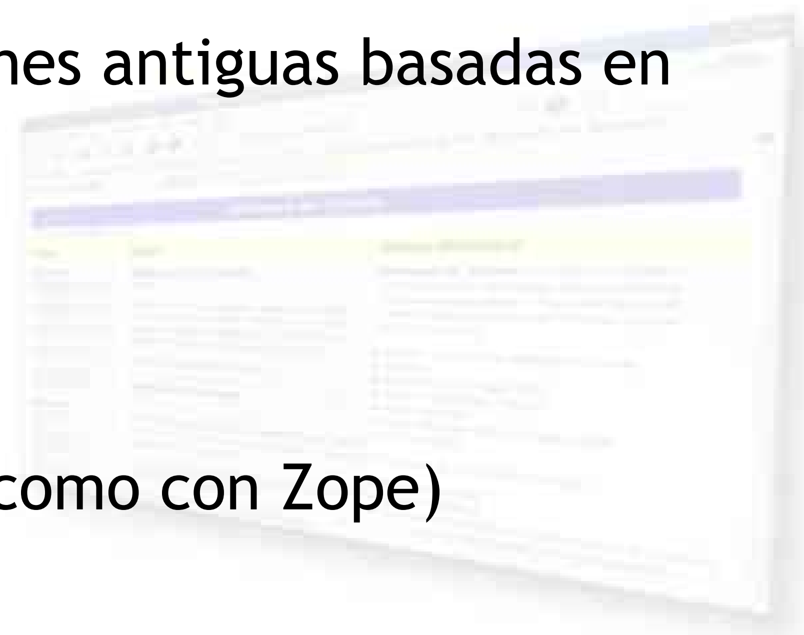
- Common Gateway Interface
- A favor
  - Tecnología veterana y muy probada
  - Bastante simple
  - Muy fácil encontrar servicios de hosting
- En contra
  - Menor eficiencia que otras alternativas
  - Nos hace programar quick-n-dirty

Alternativa 2: Zope

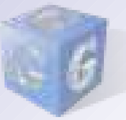
- A favor
  - Muy alto nivel
  - Diseño orientado a objetos
- En contra
  - Elevada curva de aprendizaje
  - Edición por web
  - DTML
  - Consume muchos recursos
  - Escasa documentación avanzada

Alternativa 3: Webware

- A favor
  - Más rápido que CGI y más simple que Zope
  - Completo framework orientado a objetos
  - Acepta diferentes modelos de desarrollo
    - Servlets (con o sin plantillas)
    - PSP: Python Servlets Pages
  - Integración sencilla de soluciones antiguas basadas en CGI (CGIKit)
- En contra
  - No es tan estándar como CGI
  - No es fácil encontrar hosting (como con Zope)
- Estupendo, póngame 2 :)



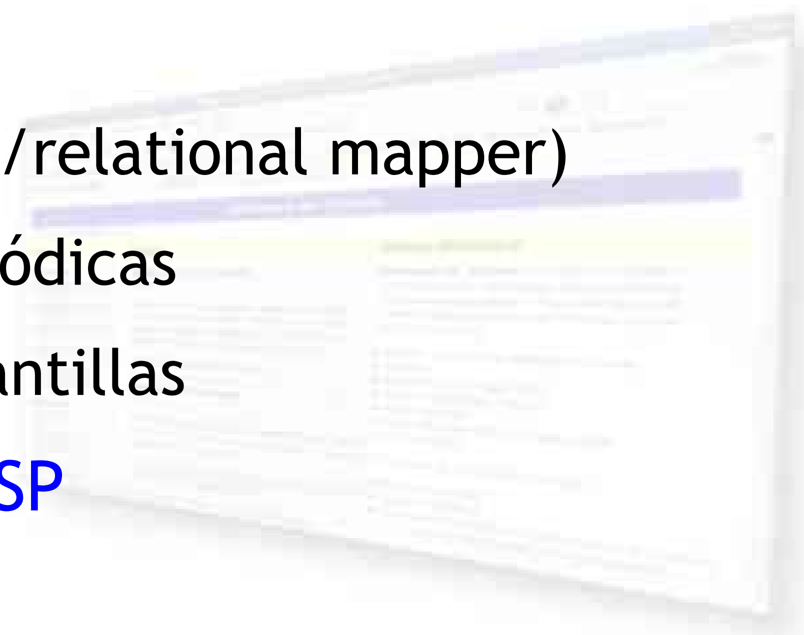
¿Qué es Webware?



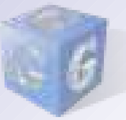
- Escrito y pensado en Python
- Orientado a objetos
- Arquitectura modular
- Open Source
- Multiplataforma
  - Linux/*nix
  - Windows NT/2k/XP/2003

Módulos Webware

- La funcionalidad de Webware se organiza en módulos independientes
  - **WebKit**: servidor de aplicaciones
  - **PSP**: Python Server Pages, parecido a JSP
  - **UserKit**: manejo de usuarios
  - **MiddleKit**: middleware (object/relational mapper)
  - **TaskKit**: gestión de tareas periódicas
  - **Cheetah**: motor externo de plantillas
- Nos centraremos en WebKit y PSP

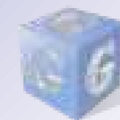


WebKit



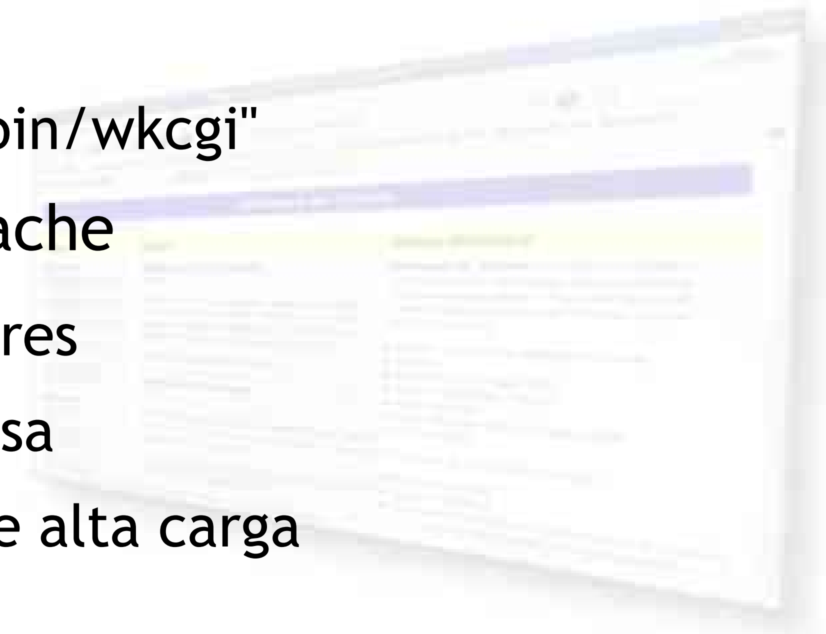
- Servidor de aplicaciones rápido y sencillo
- Utiliza hebras para la concurrencia
  - Gestión más simple de la persistencia
  - Funciona bien en Linux/*NIX y Windows
- Estable y maduro
- Acepta diferentes modelos de desarrollo web
  - Servlets
  - Python Server Pages

Arquitectura



Comunicando con el servidor web: Adaptadores

- Los adaptadores conectan a WebKit con el servidor web
  - **OneShot.cgi**, **WebKit.cgi**: escritos en python
  - **wkcgi**: versión en C de WebKit.cgi
    - Más rápido que WebKit.cgi
    - ScriptAlias /wk "/usr/lib/cgi-bin/wkcgi"
  - **mod_webkit**: módulo para apache
    - El más rápido de los adaptadores
    - Configuración un poco más liosa
    - Merece la pena en entornos de alta carga

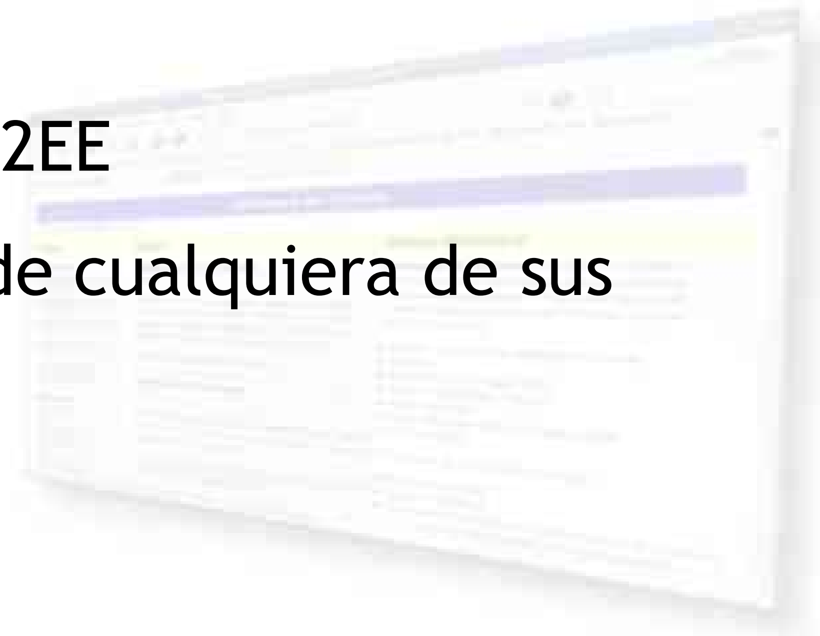


Organización de directorios

```
webware/ Cache/                          Cache de clasess, servlets, etc
         Configs/Application.config      Configuración de las aplicaciones instaladas
                 AppServer.config        Configuración del servidor de aplicaciones
         ErrorMsgs/                      Mensajes de error que se van generando
         Logs/                           Logs del servidor
         Context1/                       Contextos (pueden llamarse como se quiera)
         Context2/
         .
         .
         Sessions/                       Datos persistentes de las sesiones
         AppServer                       Script para lanzar el servidor de aplicaciones
         AppServer.bat                   Lo mismo pero en Win32
         Launch.py                       Lanzador del servidor de aplicaciones
         NTService.py                    Wrapper para instalar servicio en Win32
         WebKit.cgi                      Adaptador CGI
         OneShot.cgi                     Adaptador CGI para depuración
```

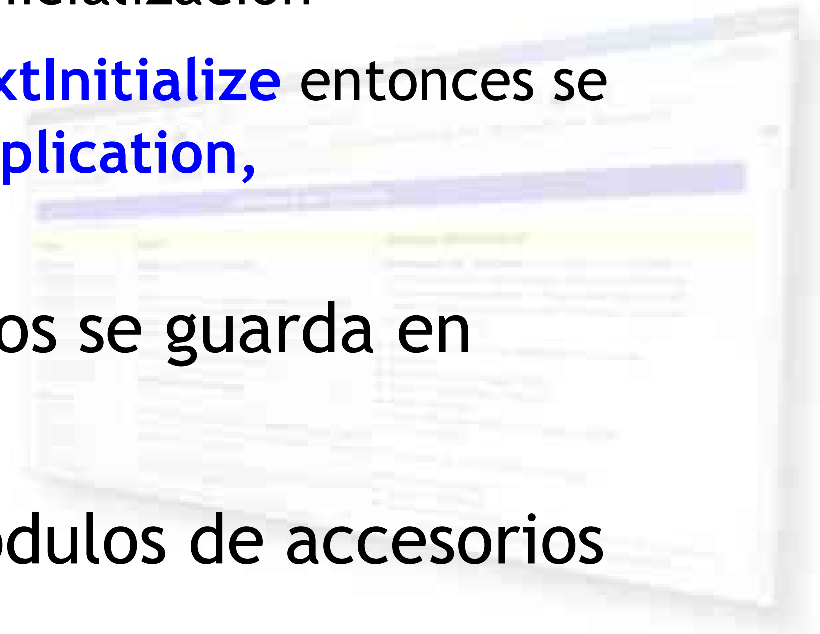
Servlets

- Objetos que se ejecutan dentro de WebKit
- Se cargan en memoria en la primera petición
- Más rápidos que los CGI
- Pueden mantener estado en el servidor de aplicaciones
- Equivalentes a los servlets de J2EE
- Heredan de **WebKit.Servlet** o de cualquiera de sus descendientes
  - **WebKit.HTTPServlet**
  - **WebKit.Page**

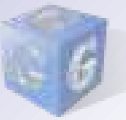


Contextos

- Los servlets se guardan en contextos
- Un contexto es un módulo Python
  - Como cualquier módulo contiene **__init__.py**
    - Se ejecuta antes que cualquier servlet
    - Puede ponerse ahí código de inicialización
    - Si contiene una función **contextInitialize** entonces se llamará a **contextInitialize(application, path_of_context)**
- La configuración de los contextos se guarda en **Application.config**
- Nota: mejor poner todos los módulos de accesorios fuera de los contextos



Hello World!



- Ejemplo más simple de servlet

```
from WebKit.Servlet import Servlet
class HelloWorld(Servlet):
    def respond(self, trans):
        trans.response().write('Hello, world!')
```

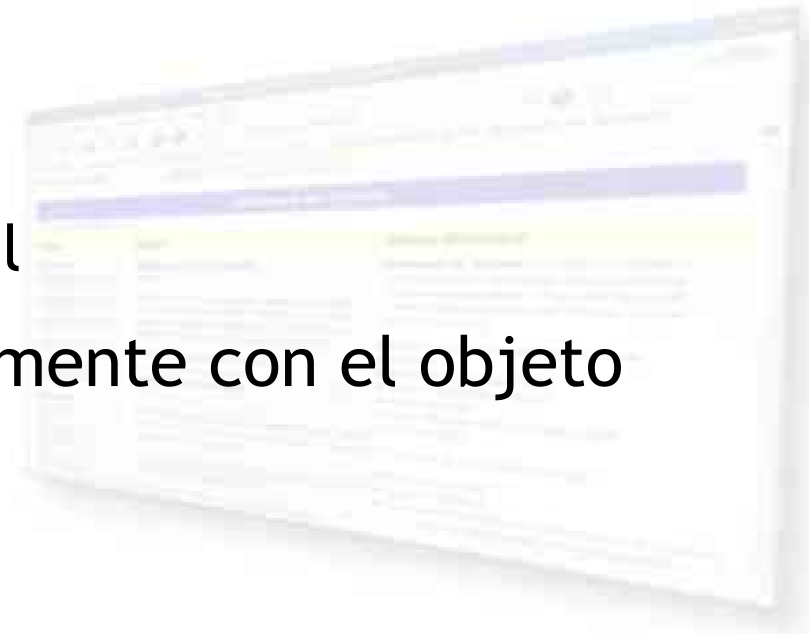
- Por defecto se genera **text/html**

Cómo se ejecuta un servlet

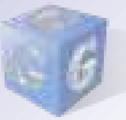
- El usuario hace una petición del tipo:
  - http://localhost/wk/Contexto/Servlet
- Se activa el contexto "Contexto" y el servlet "Servlet"
  - No es necesario indicar la extensión.
- Se reutiliza una instancia de "Servlet" del pool de instancias, o si el pool está vacío se crea una nueva
- Se crea un objeto "Transaction"
- Se ejecutan en secuencia los métodos:
  - Servlet.awake(transaction)
  - Servlet.respond(transaction)
  - Servlet.sleep(transaction)
- Se devuelve la instancia de "Servlet" al pool de instancias

Qué contiene el objeto “Transaction”

- Una transacción asocia los siguientes objetos
  - Request: petición del cliente
  - Response: respuesta (cabeceras, contenido)
  - Servlet
  - Session
    - Cookies
    - Parámetros en las url
  - Application: controlador global
- No es habitual trabajar directamente con el objeto “Transaction”



Hello World! 0.2

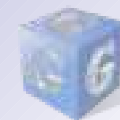


- Otro ejemplo

```
from WebKit.Page import Page
class HelloWorld(Page):
    def writeContent(self):
        self.writeln('Hello, world!')
```

- **Importante**: se hereda de **WebKit.Page**

Estructura de WebKit.Page



writeDocType()

writeTitle()

writeHeadParts()

writeHTML()

writeHead()

writeStyleSheet()

writeBody()

writeBodyParts()

writeContent()

Peticiones: HTTPRequest

- Se deriva de la clase “Request”
- Encapsula la información enviada por el navegador
  - Campos GET/POST
    - **.field(name, [default])**
    - **.hasField(name)**
    - **.fields()**
  - Cookies:
    - **.cookie(name, [default])**
    - **.hasCookie(name)**
    - **.cookies()**
  - Cuando da lo mismo si es GET/POST/Cookie se usa
    - **.value(name, [default])**
    - **.hasValue(name)**
    - **.values()**
  - Se usa también para pasar variables a los CGI
  - Rutas en el servidor
  - más cosas ...

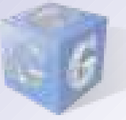
Respuestas: HTTPResponse

- Se deriva de la clase “Response”
- Encapsula la información que se devuelve al navegador
  - **.writeln(text)** – envía texto al navegador
    - Por defecto el texto se acumula en un buffer hasta que se termina de generar el contenido o se vacía explícitamente
  - **.setHeader(name, value)** – cambia/añade una cabecera
  - **.flush()** – vacia los buffers y envia el contenido
    - Cuando se están enviando ficheros grandes
    - Cuando se quieren mostrar resultados parciales
  - **.sendRedirect(url)** – manda una cabecera “Location:”

Métodos útiles de HTTPServlet

- Para acceder a la transacción y sus objetos
  - **.transaction(), .reponse(), .request(), .session(), .application()**
- Para generar contenido
  - **.write()** – equivalent to .response().write()
  - **.writeln()** – adds a newline at the end
- Utilidades varias
  - **.htmlEncode()**
  - **.urlEncode()**
- Controlar el flujo de la navegación
  - **.forward()**
  - **.includeURL()**
  - **.callMethodOfServlet()**

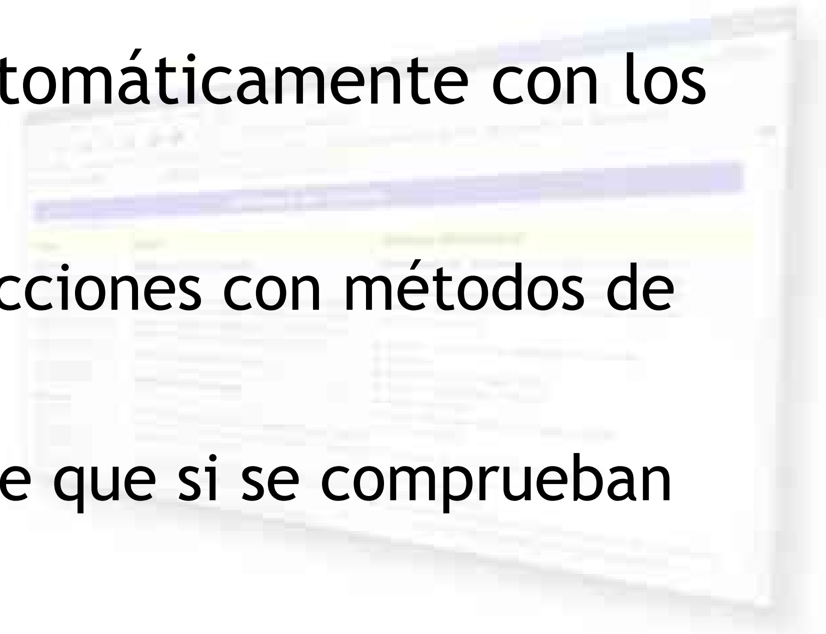
Formularios



- Los formularios pueden procesarse a mano consultando los valores **.field()** en **writeContent()**

o bien

- Pueden procesarse algo más automáticamente con los "Actions" de WebKit
  - Asocian diferentes botones o acciones con métodos de un servlet
  - Código más limpio y reutilizable que si se comprueban uno por uno los parámetros



Actions

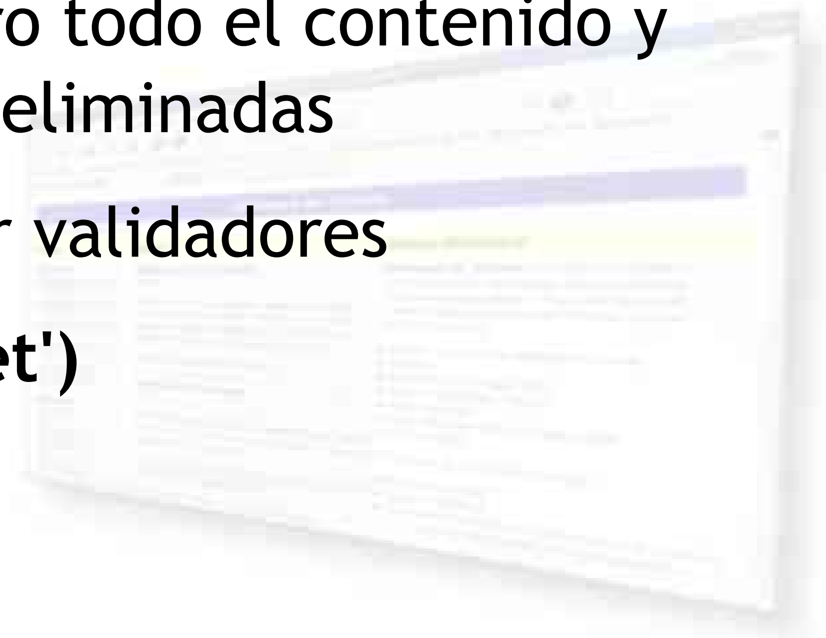
- Crear un botón en un formulario mas o menos así
  - <input name=”_action_add” type=”submit” value=”Add item”>
- Subclasificar un método .actions()

```
def actions(self):
    return Page.actions() + ['add']
```

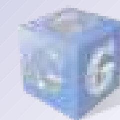
- El método .respond() revisa los campos **_action_ACCION**, donde **ACCION** debe corresponder con un método devuelto por **.actions()**
- Si se encuentra un campo con el nombre adecuado se llamará a **handleAction(ACCION)** en vez de a **writeHTML()**

Forwards

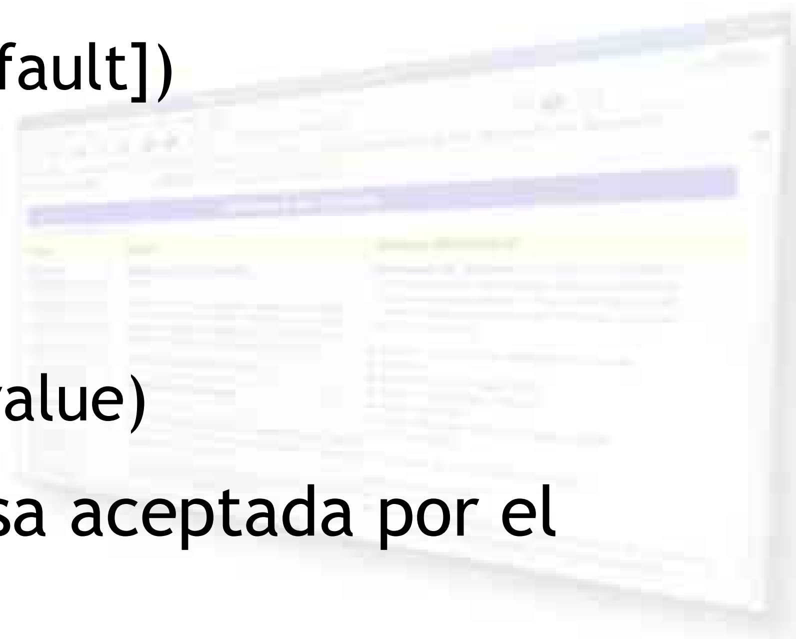
- Es como una redirección de la que ni el servidor web ni el cliente tienen constancia
- Se encapsula un nuevo Request en la transacción y se le pasa al nuevo servlet
- Cuando el servlet llamado termina se devuelve el control al servlet llamador, pero todo el contenido y las cabeceras del llamador son eliminadas
- Puede usarse para implementar validadores
- Se usa con **self.forward('Servlet')**



Sesiones: Manteniendo el estado

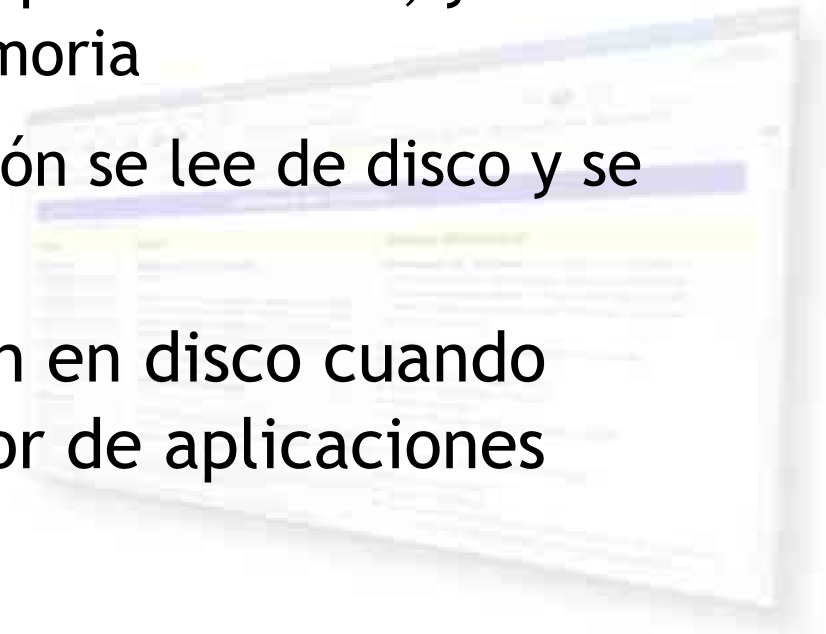


- Las sesiones sirven para guardar datos entre peticiones
- Las sesiones caducan después de un cierto tiempo
  - Por defecto 30min, aunque es configurable
- Se proporcionan los siguientes métodos
  - self.session().value(name [, default])
  - self.session().hasValue(name)
  - self.session().values()
  - self.session().setValue(name, value)
- "value" puede ser cualquier cosa aceptada por el módulo cPickle



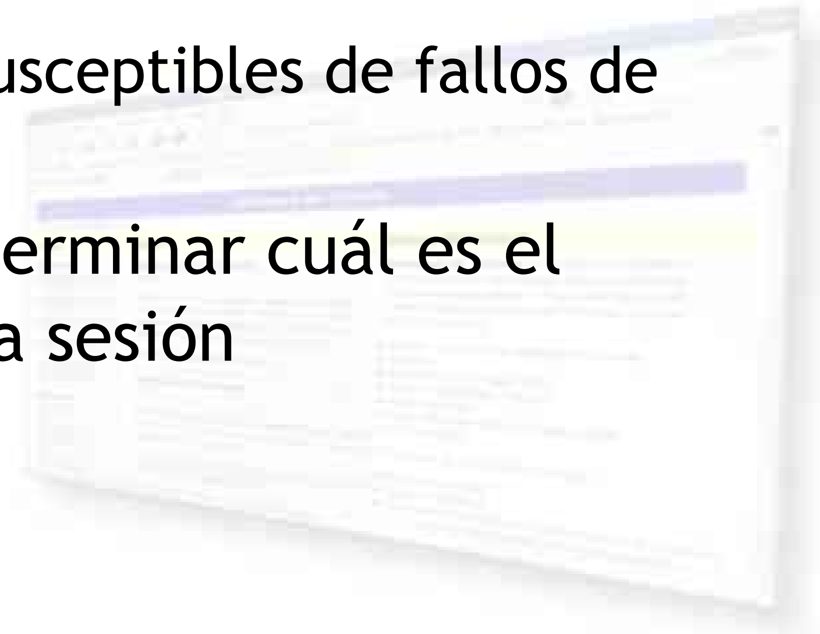
Sesiones: Dónde se guardan

- La variable **SessionStore** controla el comportamiento de la caché de sesiones
  - **Memory**: las sesiones activas se mantienen todas en memoria
  - **Dynamic**: se establece un umbral de antigüedad por encima del cual las sesiones se pasan a disco, y en caso contrario se mantienen en memoria
  - **File**: para cada petición la sesión se lee de disco y se guarda en disco
- Las sesiones siempre se guardan en disco cuando finaliza la ejecución del servidor de aplicaciones



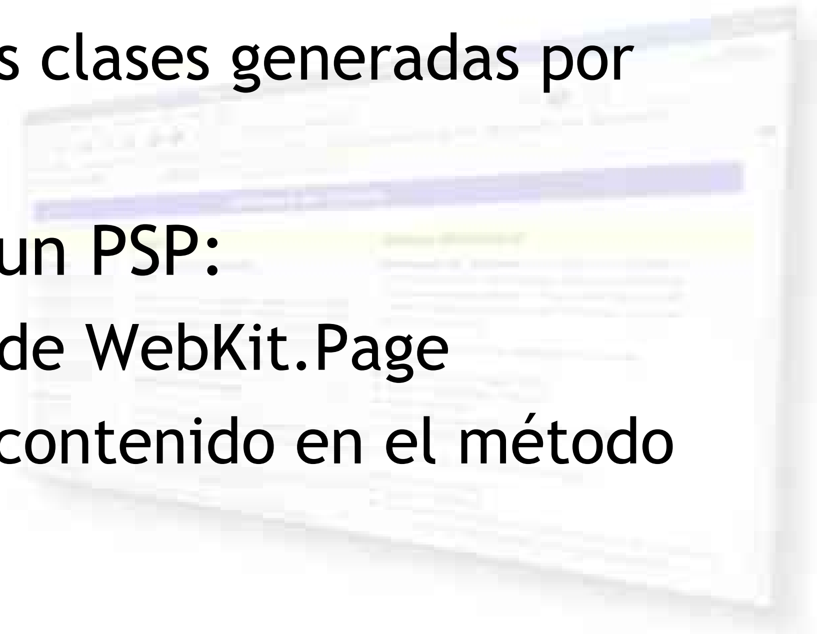
Identificación de las sesiones

- Cada sesión se identifica por un identificador aleatorio
- Por defecto dicho identificador se almacena en una cookie en el navegador
- Si se cambia **UseAutomaticPathSessions** a 1
  - El identificador de sesión se pasa por la URL
  - Ya no se requiere ninguna cookie
  - Las urls son más complejas y susceptibles de fallos de seguridad
- WebKit por ahora no puede determinar cuál es el mejor sistema para mantener la sesión

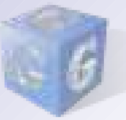


PSP: Python Server Pages

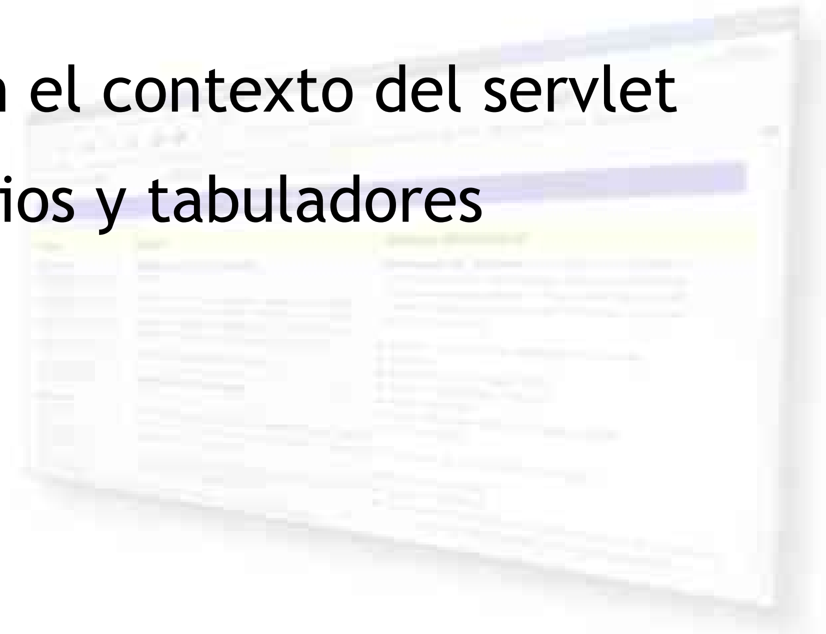
- Algunas características de PSP
  - Sintaxis inspirada en JSP
  - Basado en Python
  - Acceso al API de WebKit
  - Flexible PSP Base Class framework
  - Se pueden añadir métodos a las clases generadas por PSP
- Cuando se hace una petición a un PSP:
  - Se compila una clase derivada de WebKit.Page
  - Por defecto se incluye todo el contenido en el método **writeHTML()**



Marcas PSP

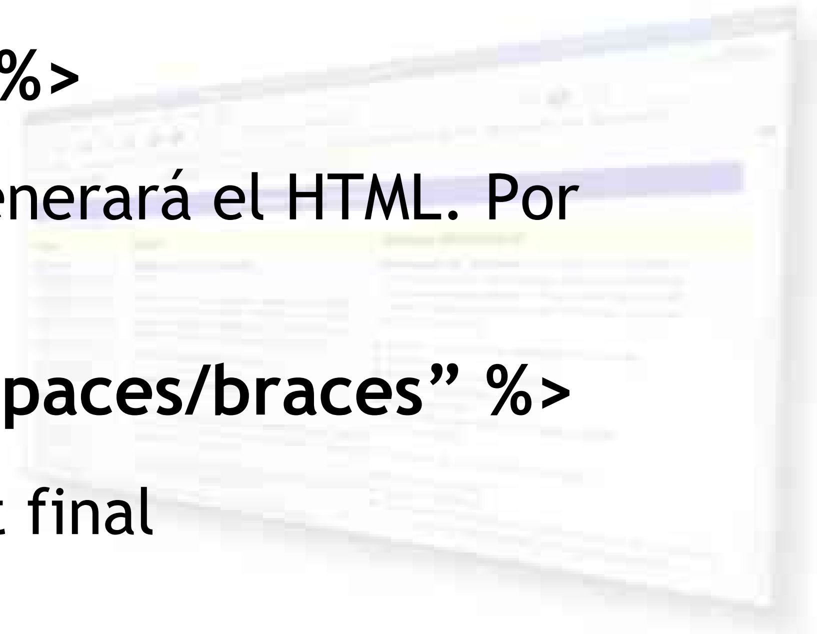


- **<%= expr %>**
  - Imprime el resultado en el HTML
- **<% bloque_de_código %>**
  - Bloque de código a ejecutar en el contexto del servlet
  - Cuidado especial con los espacios y tabuladores



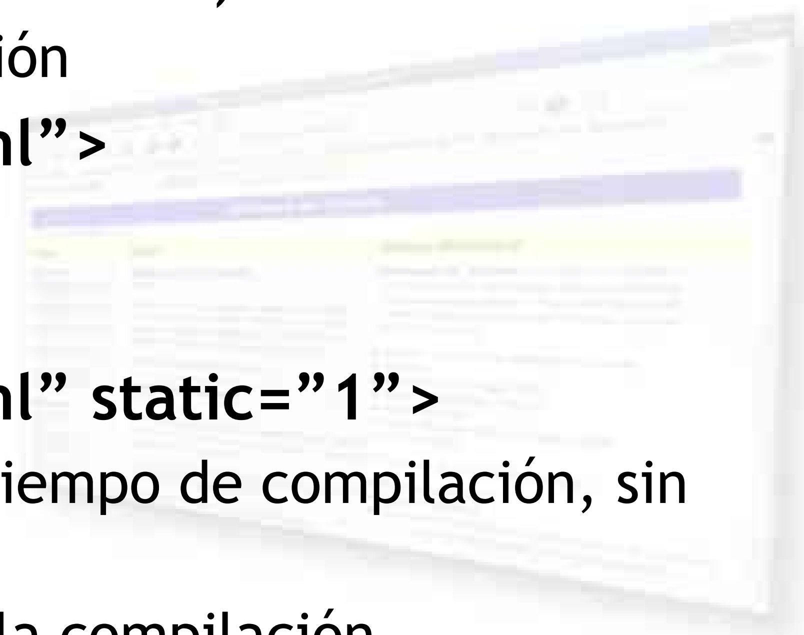
Directivas PSP

- **<%@ page imports=”modulos”%>**
  - Lista de módulos a importar
- **<%@ page extends=”ClaseBase”%>**
  - Clase de la que hereda la página
- **<%@ page method=”metodo” %>**
  - Método del servlet donde se generará el HTML. Por defecto será **writeHTML()**
- **<%@ page indentType=”tabs/spaces/braces” %>**
  - Tipo de indentación del servlet final



Directivas PSP

- **<%@ include file="myinclude.psp"%>**
  - Incluye un fichero en tiempo de compilación. Se parsea como PSP
  - Si el fichero cambia durante la ejecución no se verán los cambios
- **<psp:include path="myinclude">**
  - Equivalente **self.includeURL('myinclude')**
  - Se carga el fichero en cada petición
- **<psp:insert file="myinclude.html">**
  - Se lee de disco sin parsear PSP
  - Se lee de disco en cada petición
- **<psp:insert file="myinclude.html" static="1">**
  - Se incluye el fichero tal cual en tiempo de compilación, sin parsear PSP
  - Solo se lee el fichero una vez en la compilación



Bloques de código en PSP

- Bloques automáticos
  - <% for i in range(10): %> define automáticamente un bloque que termina con <% end %>
- Bloques manuales

  <% for i in range(10):

  j = i $%> define automáticamente un bloque que indenta al mismo nive que la siguiente instrucción del bucle y termina con cualquier tag <% %> (p.ej. <% pass %>)
- Llaves
  - Se pueden emplear llaves { } para la indentación y eliminar ambigüedades debidas al formateo del HTML

TaskKit

- Facilita la ejecución de tareas periodicamente
  - Tareas de limpieza temporal
  - Creación de contenido semiestático
- Clases
  - TaskKit.Task
  - TaskKit.Scheduler

Cheetah

¿Qué tenemos?

- Servlet: Estructura de Clases para generar datos dinámicos, y generar la salida “en código”.
- PSP: Páginas HTML con Python empotrado para generar páginas dinámicas.

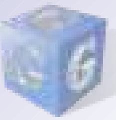
¿Que problemas da?

- Servlet: Código de salida “HTML” a fuego en el código.
- PSP: Puede conducir a exceso de código en las páginas.

¿Otra opción?

- Generar mediante Servlet la información dinámica.
- Utilizar plantillas para generar la salida.

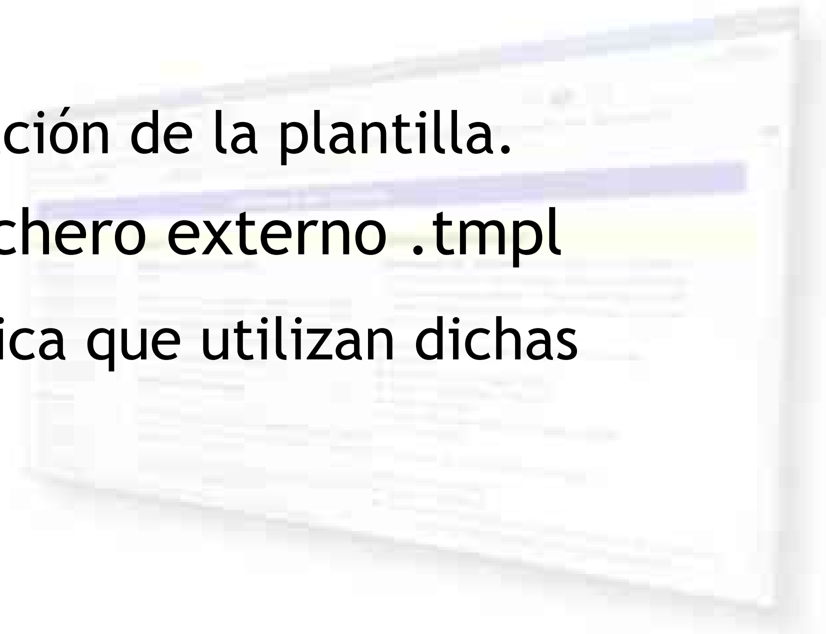
¿Qué ofrece Cheetah?



- Mayor separación Código/Presentación.
- Estructura de Plantillas (Páginas HTML) con un sublenguaje de acceso a datos en Python.
- Puede utilizarse integrado en WebWare y como módulo independiente.
- Lenguaje Fácil de aprender.
- Sistema Avanzado de Caché.
- Admite división jerárquica de las plantillas.

Cheetah: ¿Cómo Funciona?

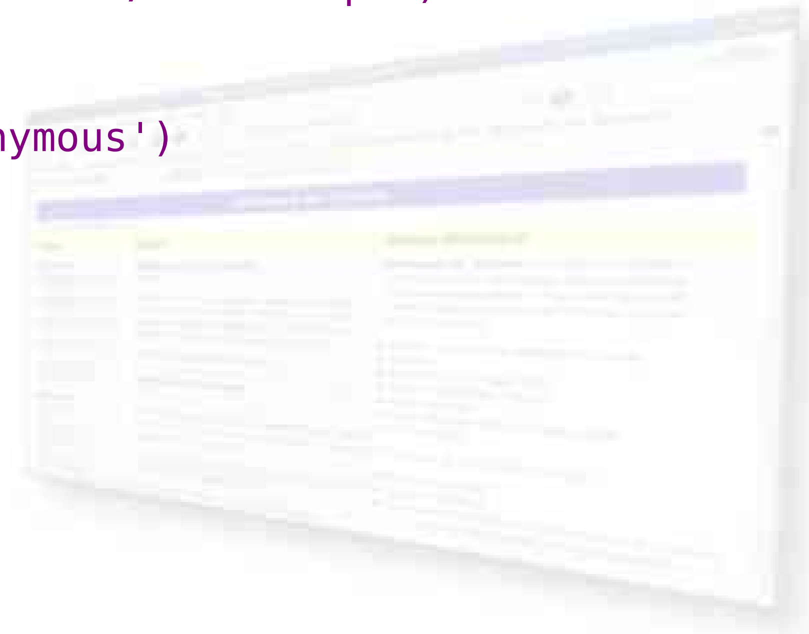
- Estructura de las páginas en dos partes:
  - Parte dinámica en Servlet:
    - Realiza las operaciones que desee (acceso a Base de Datos, identificación del usuario ...).
    - Guardar los datos que afectan a la presentación (usuario, theme elegido, ...) en variables de comunicación con la plantilla.
    - Devolver como salida la aplicación de la plantilla.
  - Parte de Presentación en un fichero externo .tmpl
    - Código HTML con parte dinámica que utilizan dichas variables para generarse.



Cheetah: Ejemplo

- Hello.py

```
from WebKit.Page import Page
from Cheetah.Template import Template
class Hello(Page):
      def __init__(self):
          Page.__init__(self)
          self.template = Template(file='Templates/hello.tmpl')
    def writeContent(self):
         user.self.request.field('user','anonymous')
          self.template.user = user
         self.writeln(self.template)
```



Hello.tmpl

- Hello.tmpl

```
<html>
<head>
<title>Bienvenido $user</title>
</head>
<body>
<h1>Bienvenido, $user, a esta primera prueba</h1>
</body>
</html>
```

Cheetah: Identificadores de Plantilla

- Los identificadores que empiezan por $ son variables, se sustituyen
- Admite condionales #if ... #end if.
- Admite bucles #for,#foreach y #while
- Admite encadenar plantillas dentro de otras
  - #include para incluir plantillas
  - #include raw para incluir ficheros sin analizar
- Admite bucles #foreach

Cheetah: Ejemplos de Plantillas

- Bucle if

```
#if $country in ('Argentina', 'Uruguay', 'Peru', 'Colombia','Costa Rica',
   'Venezuela', 'Mexico')

<H1>Hola, senorita!</H1>

#else

<H1>Hey, baby!</H1>

#end if
```

- Recorrido de un vector

```
<TABLE>

#for $client in $service.clients

<TR>

<TD>$client.surname, $client.firstname</TD>

<TD><A HREF="mailto:$client.email" >$client.email</A></TD>

</TR>

#end for

</TABLE>
```

Cheetah: Ejemplos de Plantillas

- Acceso a un diccionario

```
<PRE>
#for $key, $value in $dict.items()
$key: $value
#end for
</PRE>
```

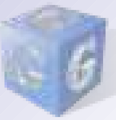
- Acceso a valores de un diccionario

```
nombre: $user['name']
Apellidos: $user['apellidos']
```

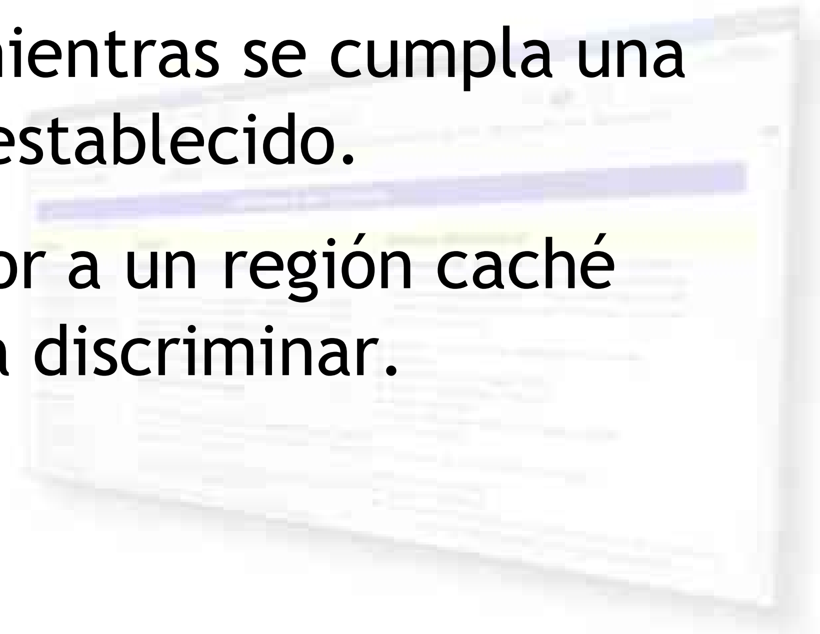
ó

```
nombre: $user.name
Apellidos: $user.apellidos
```

Cheetah: Cache



- Cheetah admite un sistema de caché muy avanzado.
- Así, mientras se mantenga las mismas variables de interacción devolvería la misma caché.
- Se puede guardar en caché regiones de ficheros mediante **#cache .... #end cache**
- Las cachés se pueden guardar mientras se cumpla una condición o durante un tiempo establecido.
- Se puede asignar un identificador a un región caché (por usuario, por ejemplo), para discriminar.



Cheetah: Ejemplo de uso de Caché

```
#cache
Parte estática, no será refrescado
$a $b $c
#end cache

#cache timer='30m', id='cache1'
#* Actualizado cada media hora *#
#for $cust in $customer $cust.name:
     $cust.street - $cust.city
#end for
#end cache

#cache id='sidebar', test=($isDBUpdated or $someOtherCondition)
#*Actualizado mientras no se modifique la Base de Datos *#
... left sidebar HTML ...
#end cache
```

Cheetah: Uso desde WebWare

Hay varios modos de utilizarlo:

- Utilizando el *template* dentro del Servlet
  - En este caso el servlet no es una plantilla, sino que lo crea y le pasa los parámetros indicados.

```
template = Template(file="cabecera.tmpl")

print (template)
```

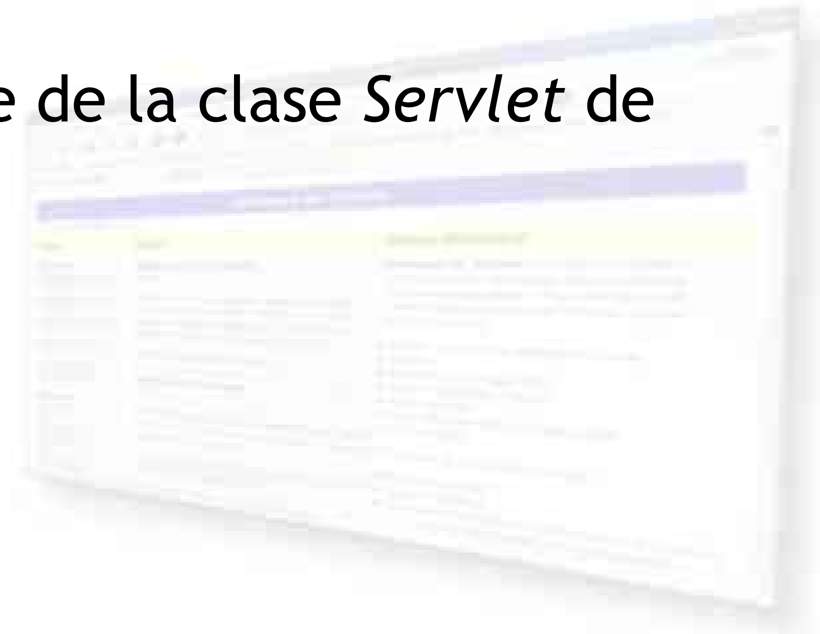
  - Haciendo que el Servlet herede de la clase *Servlet* de *Cheetah.Servlet*

```
from Cheetah.Servlet import Servlet

class Hello(Servlet):

      .

      .
```



Cheetah: compilando las páginas

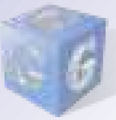
- Cheetah permite compilar las páginas:
  - De esta forma se permite generar automáticamente por un *tmpl* el código *.py* que genera la misma salida.

cheetah compile <fichero.tmpl>

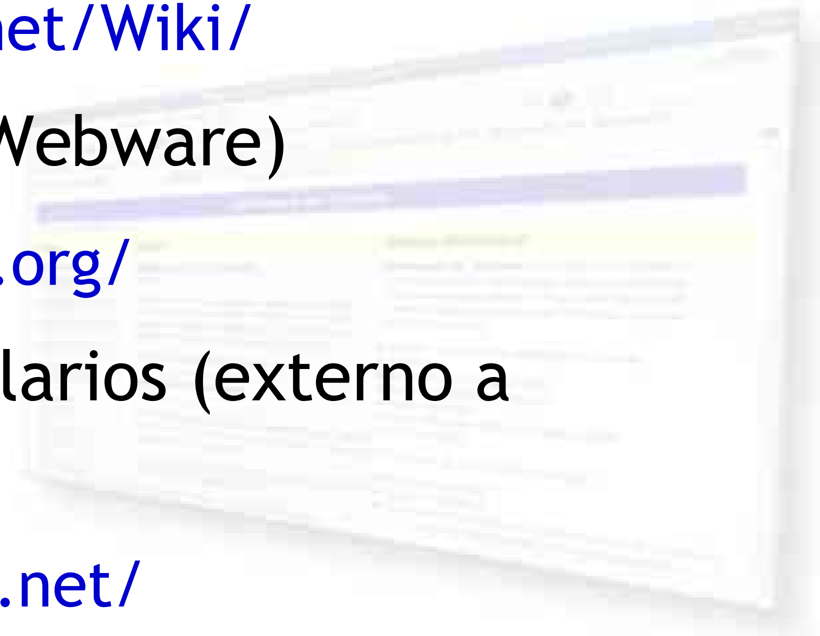
  - Así, simplemente haciendo un import de dicha clase se generaría la salida.
  - También se puede generar las plantillas .tmpl a un fichero estático HTML.

cheetah fill <fichero.tmpl>

Referencias



- Página principal del proyecto Webware
  - http://webware.sourceforge.net
- Documentación introductoria
  - http://webware.sourceforge.net/Papers/
- Wiki del proyecto
  - http://webware.sourceforge.net/Wiki/
- Motor de plantillas (externo a Webware)
  - http://www.cheetahtemplate.org/
- Validador automático de formularios (externo a Webware)
  - http://funformkit.sourceforge.net/



EOC: End of Charla

- ¿Preguntas?