Desarrollo de aplicaciones web con Bottle

Antonio Sarasa Cabezuelo

HTML

- HTML (HyperText Markup Language) es un lenguaje diseñado para la publicación de contenidos en Internet. Permite formatear textos, incluir imágenes y animaciones, así como para enlazar con otros contenidos del mismo o de otro servidor.

HTML

- Es un lenguaje interpretado por los navegadores de tal forma que la apariencia de las páginas puede variar entre diferentes navegadores, versiones y plataformas.

HTML

- El lenguaje está estructurado mediante etiquetas normalmente emparejadas y encerrando un contenido. Las etiquetas modifican la cualidad o formato del contenido mostrado en la pantalla del navegador.

HTML

- El lenguaje está estructurado mediante etiquetas normalmente emparejadas y encerrando un contenido. Las etiquetas modifican la cualidad o formato del contenido mostrado en la pantalla del navegador.:

```
<etiqueta>
    contenido
</etiqueta>
```

HTML

- Las etiquetas pueden estar anidadas ordenadamente:

```
<etiqueta1>
    <etiqueta2>
      contenido
    </etiqueta2>
</etiqueta1>
```

HTML

- Algunas etiquetas sólo están integradas por la etiqueta inicial y no se cierran. Es decir no existe <etiqueta/>
- Las letras de las etiquetas pueden ser mayúsculas o minúsculas indistintamente.

HTML

- La estructura básica de un documento HTML es:

```
<!DOCTYPE html>
<HTML>
    <HEAD>
        <TITLE>Título de la página</TITLE>
    </HEAD>
    <BODY>
        Aquí aparecen todas las etiquetas y contenidos
        que se visualizan en el navegador.
    </BODY>
</HTML>
```

HTML

- Las partes de un documento HTML:
  - La primera línea es la declaración del tipo de documento que indica la versión de HTML que se está usando.
  - Todo documento HTML empieza con la etiqueta <HTML> indicando el inicio del documento y acaba con la etiqueta </HTML> para indicar el final del mismo.

HTML

- Las partes de un documento HTML:
  - El documento en sí está dividido en 2 partes:
    - El encabezado, encerrado por las etiquetas: <HEAD> </HEAD>.
    - El cuerpo, encerrado por las etiquetas: <BODY> </BODY>.
  - En el encabezado se encuentra toda la información del documento que no se visualiza en el área principal del navegador, como el título de la página que se encuentra entre las etiquetas <TITLE> </TITLE>.

HTML

- Las partes de un documento HTML:
  - Dentro del cuerpo se halla todo el contenido que se muestra en el área principal del navegador, como texto, imágenes, etc.
  - HTML no interpreta los espacios en blanco o los saltos de línea del documento fuente, para lo cual existen caracteres especiales al efecto.

HTML

- Observar que si se quiere incluir un comentario dentro de la página HTML, se utilizaran los dos símbolos: <!— (para abrir el comentario) y --> (para cerrarlo).

HTML

- Instrucciones de formateo:

Instrucción	Significado
<B>**Texto**</B>	Muestra el contenido "Texto" en negrita.
<I>*Texto*</I>	Muestra el contenido "Texto" en cursiva.
<PRE>Texto</PRE>	Muestra el contenido como si hubiese sido escrito por una máquina de escribir con una fuente de espacio fijo (tipo *Courier*) respetando espacios y saltos de línea.
<BLOCKQUOTE> Texto </BLOCKQUOTE>	Destaca una cita dentro del texto general añadiendo márgenes izquierdo y derecho.

HTML

- Instrucciones de formateo:

Instrucción	Significado
<P>	Efectúa un salto de línea y deja una línea en blanco.
 	Efectúa un salto de línea únicamente. Este elemento no se cierra
<HR>	Dibuja una línea horizontal. Este elemento no se cierra.
<ADDRESS> *Nombre autor* </ADDRESS>	Indica el nombre del autor del documento formateando el texto en cursiva y alineado a la izquierda.

HTML

- Instrucciones de formateo:

Instrucción	Significado
<H4> Texto Cabecera </H4>	Las etiquetas <Hx>, con x desde 1 hasta 6, muestran diferentes tamaños de letras y un salto de línea variable. La cabecera tipo 1 es la más grande, decreciendo el tamaño de la fuente al aumentar el número x.

HTML

- Listas

  Se pueden mostrar listas de contenidos existiendo 3 tipos:

Estructura	Significado
<UL> <LI> Palabra 1 <LI> Palabra 2 <LI> Palabra 3 <LI> ….. </UL>	Presenta una lista sin que sus elementos estén precedidos de un número secuencial.

HTML

Estructura	Significado
<OL> <LI> Palabra 1 <LI> Palabra 2 <LI> Palabra 3 <LI> …. </OL>	Presenta una lista cuyos elementos están precedidos de un número secuencial.

HTML

Estructura	Significado
<DL> <DT> Término 1 <DD> Def. término 1 <DT> Término 2 <DD> Def. término 2 </DL>	Se utiliza para glosarios o definiciones de términos. Cada bloque de texto está formado por 2 etiquetas. <DT> que indica la palabra que deseamos definir, y <DD> que es la definición propiamente dicha.

HTML

- Observar que se pueden anidar unas listas dentro de otras.

HTML

- Enlaces:
  - Permiten conectar contenidos del mismo o de diferentes servidores de manera muy sencilla e intuitiva. Usualmente, los enlaces tienen la siguiente estructura:

    <A HREF="Destino"> Representación del enlace </A>

HTML

- Enlaces:
  - Se pueden diferenciar 3 tipos de enlaces:
    - Enlaces dentro de una misma página: Son utilizados para poder saltar dentro de una misma página. La primera línea es la marca en la página (también llamada ancla) e indica el sitio exacto adonde se desea saltar; la segunda línea es el enlace, que nos llevará al ancla.

```
<A NAME="marca"> </A>

<A HREF="#marca"> </A>
```

HTML

- Enlaces:
  - Se pueden diferenciar 3 tipos de enlaces:
    - Enlaces a otras páginas: Son utilizados para poder "enlazar" a páginas o alguna parte concreta de la página(mediante un ancla) del mismo servidor o a páginas de diferentes servidores.

Ejemplo de enlace
<A HREF="pagina.htm"> Representación del enlace </A>
<A HREF="pagina.htm#marca"> Representación del enlace </A>
<A HREF="http://dominio/pag.htm"> Representación del enlace </A>

HTML

- Enlaces:
  - Se pueden diferenciar 3 tipos de enlaces:
    - Enlaces a una dirección e-mail: Son usados para facilitar el envío de correo electrónico.

```
<A HREF="mailto:email@dominio">
        Representación del enlace
</A>
```

HTML

- Imágenes

  Para incluir imágenes se utiliza la etiqueta <IMG> (no tiene etiqueta de cierre)y se indica el nombre del fichero que contiene la imagen a mostrar.

  <IMG SRC="imagen.gif" ALT="descripción“ ALIGN=alineamiento WIDTH=ancho HEIGHT=alto>

HTML

- Imágenes

Los atributos de la etiqueta <IMG> son:

Atributo	Significado
SRC	Indica el camino y el nombre de fichero donde se encuentra la imagen que se quiere incluir. Ésta puede residir en el mismo o en otro servidor. Es el único que siempre debe aparecer en la etiqueta.
ALT	Se utiliza para mostrar un texto emergente cada vez que se coloca sobre la imagen el puntero del ratón.

HTML

- Imágenes

Los atributos de la etiqueta <IMG> son:

Atributo	Significado
ALIGN	Se emplea para alinear la imagen en la página HTML. Los valores posibles son: TOP, MIDDLE, BOTTOM, RIGHT y LEFT.
WIDTH HEIGHT	Se usan para modificar el tamaño de la imagen en la página HTML. El tamaño se expresa en pixels (por ejemplo, WIDTH = 200) o con un porcentaje del tamaño de la imagen (por ejemplo, WIDTH = 60%). Si no se incluyen estos atributos, la imagen aparece con las mismas dimensiones con las que ha sido creada.

HTML

- Fondos

  Se puede cambiar el fondo de una página con un color uniforme o mediante una imagen tápiz. Para ello se introduce un atributo en la etiqueta BODY.

Etiqueta	Significado
<BODY BGCOLOR="#XXYYZZ" TEXT="#XXYYZZ“ LINK="#XXYYZZ" VLINK="#XXYYZZ" ALINK="#XXYYZZ">	BGCOLOR: Color de fondo de la página. en formato hexadecimal. TEXT : Color del texto. LINK : Color de los enlaces. VLINK: Color de los enlaces visitados. ALINK: Color de los enlaces activos (cuando se pulsan).

HTML

- Fondos

  Se puede cambiar el fondo de una página con un color uniforme o mediante una imagen tápiz. Para ello se introduce un atributo en la etiqueta BODY.

Etiqueta	Significado
<BODY BACKGROUND="Archivo">	BACKGROUND: Permite incluir una imagen en formato gif o jpg, de fondo en la página. "Archivo" hace referencia a la imagen. Además, se pueden añadir todos los modificadores vistos en el modo anterior para establecer los colores del texto y de los enlaces.

HTML

- Fuente de un texto

  Para establecer las características de la fuente de un texto, se usa la etiqueta <FONT>:

  <FONT FACE="fuente" SIZE="tamaño« COLOR="#XXYYZZ">Texto</FONT>

HTML

- Fuente de un texto

Los atributos de la etiqueta <FONT> son:

Atributo	Significado
FACE	Se elige el tipo o tipos de fuente para el "Texto" contenido entre la etiqueta de inicio y de cierre.
COLOR	Selecciona el color de la fuente en formato hexadecimal.

HTML

- Fuente de un texto

Los atributos de la etiqueta <FONT> son:

Atributo	Significado
SIZE	Se cambia el tamaño de la fuente asignando un valor de 1 a 7 (1 es la más pequeña). El tamaño normal es el 3. Se puede cambiar relativamente respecto a este tamaño normal escribiendo por ejemplo, <FONT SIZE=+1>. También se puede cambiar el tipo de fuente normal de todo el documento incluyendo a continuación de la etiqueta BODY la etiqueta: <BASEFONT SIZE=5>.

HTML

- Alineación de textos e imágenes

  Cualquier texto o imagen incluidos en una página se puede alinear usando llas etiquetas CENTER, LEFT, RIGHT como por ejemplo:

  <CENTER>Texto o Imagen</CENTER>

  Además en el caso de las imágenes puede usarse el modificador ALIGN de la etiqueta IMG.

HTML

- Tablas

  Se pueden usar tablas para distribuir y presentar contenidos dentro de una página. El esquema básico:

  <TABLE> Inicio de la tabla

  <TR> Primera fila

  <TD> Contenido </TD> Primera columna de la primera fila

  <TD> Contenido </TD> Segunda columna de la primera fila

  </TR> Final de la primera fila

  <TR> Segunda fila

  <TD> Contenido </TD> Primera columna de la segunda fila

  <TD> Contenido </TD> Segunda columna de la segunda fila

  </TR> Final de la segunda fila

  ……

  </TABLE> Final de la tabla

HTML

- Tablas

Etiqueta	Significado
<TABLE BORDER=borde WIDTH=largo HEIGHT=ancho BGCOLOR=#XXYYZZ CELLSPACING=sep_celdas CELLPADDING=sep_cont> Filas </TABLE>	Indican el comienzo y final de una tabla. • BORDER establece la anchura del borde de la tabla (BORDER=0 indica que la tabla no tiene bordes) • WIDTH, HEIGHT fijan el largo y ancho de la tabla respectivamente. Se expresan en pixels o porcentaje de la dimensión de la pantalla. • BGCOLOR establece el color de fondo de la tabla en formato hexadecimal. • CELLSPACING fija la separación entre las celdas de la tabla en pixels. Por defecto 2 pixels • CELLPADDING determina la separación entre el borde y el contenido dentro de cada celda en pixels. Por defecto 1 pixel.

HTML

- Tablas

Etiqueta	Significado
<TR>Celdas</TR>	Señalan el inicio y el final de una fila. Hay que repetirlas tantas veces como filas deseemos obtener.

HTML

- Tablas

Etiqueta	Significado
<TD ALIGN="alineam_horiz" VALIGN="alineam_vert" COLSPAN=extender_horiz ROWSPAN=extender_vert BGCOLOR=#XXYYZZ> Contenido </TD>	Inicio y el final de una celda con contenido. Hay que repetirlas tantas veces como columnas deseemos obtener. • ALIGN establece el alineamiento horizontal del contenido en la celda. Los valores posibles son: CENTER, RIGHT y LEFT. • VALIGN establece el alineamiento vertical del contenido en la celda. Los valores posibles son: TOP, BOTTOM y MIDDLE. • COLSPAN indica la extensión de la celda en nº de columnas. Así, una celda puede ocupar varias columnas. • ROWSPAN indica la extensión de la celda en nº de filas. Así, una celda puede ocupar varias filas. • BGCOLOR establece el color de fondo de la celda. Se expresa en formato hexadecimal. Si hemos incluido este atributo en <TABLE>, cambiará el color por defecto de la celda.

HTML

- Tablas

Etiqueta	Significado
<TH> Contenido </TH>	Establece el inicio y el final de una celda de tipo cabecera. Se distingue de <TD> en que muestra el texto en negrita y centrado. Se pueden aplicar los mismos atributos que en la etiqueta <TD>.

HTML

- Observar que se pueden anidar unas tablas dentro de otras.

HTML

- Formularios

  Los formularios permiten al usuario introducir información y enviarla al servidor. Su estructura es la siguiente:

  <FORM ACTION="acción_URL" METHOD="método“

  ENCTYPE="tipo_codificado">

  Cuerpo del formulario con los diferentes elementos para la introducción de datos

  y los botones de envío o de borrado.

  </FORM>

HTML

- Los atributos de la etiqueta <FORM> son:

Atributo	Significado
ACTION	Indica la acción que se debe realizar. Normalmente es una *URL* que apunta a un CGI, Servlet o página dinámica.
METHOD	Método utilizado para el envío de la información. Puede tomar los valores de "GET" (por defecto) o "POST". Con "GET" el contenido introducido se añade a la URL como si fuera parte de ésta y con "POST" la información va en el cuerpo de datos separadamente.

HTML

- Los atributos de la etiqueta FORM son:

Atributo	Significado
ENCTYPE	Denota el tipo de codificación utilizada para enviar los datos. Por ejemplo, "TEXT/PLAIN" consigue que las respuestas sean recibidas como un fichero de texto, perfectamente legible y sin codificar.

HTML

- Los elementos del cuerpo son de 5 tipos:
  - **Introducción por medio de texto:** el usuario debe completar la información que desea enviar al servidor. Tiene la estructura siguiente:

    <INPUT TYPE="xxx" NAME="yyy" VALUE="zzz" SIZE="aaa" MAXLENGTH="bbb">

    Observar que la etiqueta <INPUT> no tiene cierre.

HTML

- Los elementos del cuerpo son de 5 tipos:
  - Los atributos de la etiqueta <INPUT> son:

Atributo	Significado
TYPE	Indica el tipo de elemento utilizados para la introducción de información. Los valores posibles son: a) text para establecer el elemento como un 'edit' de texto, y b)password es igual que el anterior, pero al escribir sólo se muestran asteriscos.
NAME	Establece el nombre del elemento para poder relacionarlo con su contenido.
VALUE	Fija el contenido inicial del elemento.
SIZE	Establece la longitud del elemento en la pantalla.

HTML

- Los elementos del cuerpo son de 5 tipos:
  - Los atributos de la etiqueta <INPUT> son:

Atributo	Significado
MAXLENGTH	Establece el número máximo de caracteres que se pueden introducir.

HTML

- Los elementos del cuerpo son de 5 tipos:
  - Cuando es necesario introducir un texto que pueda alcanzar una gran longitud u ocupar varias líneas, como un comentario, es conveniente utilizar un elemento de texto de múltiples líneas(etiqueta <TEXTAREA>) en vez de <INPUT>:

    <TEXTAREA NAME="yyy" ROWS="número" COLS="número">

    contenido inicial

    </TEXTAREA>

HTML

- Los elementos del cuerpo son de 5 tipos:
  - Los atributos de la etiqueta <TEXTAREA> son:

Atributos	Significado
NAME	Establece el nombre del elemento para poder relacionarlo con su contenido.
ROWS	Representa el número de filas.
COLS	Representa el número de columnas.

HTML

- Los elementos del cuerpo son de 5 tipos:
  - 2. Introducción por medio de menús: El usuario escoja entre varias opciones.

```
<SELECT NAME="yyy" MULTIPLE SIZE="aaa">
    <OPTION VALUE="bbb" selected> literal_opción 1 </OPTION>
    <OPTION VALUE="ccc"> literal_opción 2 </OPTION>
    <OPTION VALUE="ddd" selected> literal_opción 3</OPTION>
</SELECT>
```

HTML

- Los elementos del cuerpo son de 5 tipos:
  - **Introducción por medio de menús:** El usuario escoja entre varias opciones. La etiqueta <SELECT> se usa para señalar el inicio y el final del elemento menú y las etiquetas <OPTION> delimitan el inicio y final de las diferentes opciones:

```
<SELECT NAME="yyy" MULTIPLE SIZE="aaa">
    <OPTION VALUE="bbb" selected> literal_opción 1 </OPTION>
    <OPTION VALUE="ccc"> literal_opción 2 </OPTION>
    <OPTION VALUE="ddd" selected> literal_opción 3</OPTION>
</SELECT>
```

HTML

- Los elementos del cuerpo son de 5 tipos:
  - Los atributos de la etiqueta <SELECT> son:

Atributo	Significado
NAME	Establece el nombre del elemento para poder relacionarlo con su contenido.
MULTIPLE	Permite que se puedan seleccionar múltiples opciones.
SIZE	Fija el número de opciones visibles en la pantalla sin necesidad de desplazarse. Sólo funciona conjuntamente con MULTIPLE. Si es mayor que 1, aparecerá una lista; en caso contrario, se verá una lista desplegable.

HTML

- Los elementos del cuerpo son de 5 tipos:
  - Los atributos de la etiqueta <OPTION> son:

Atributo	Significado
VALUE	Establece el valor que se asigna a la variable.
SELECTED	Si se incluye en alguna de las opciones, esta opción será seleccionada por defecto.

HTML

- Los elementos del cuerpo son de 5 tipos:
  - **Introducción por medio de botones:** El usuario confirme una opción determinada con Sí o No. Se utiliza un elemento de tipo checkbox con la estrctura:

    <INPUT TYPE="checkbox" NAME="yyy" CHECKED>

Atributo	Significado
NAME	Establece el nombre del elemento para poder relacionarlo con el contenido.
CHECKED	Especifica el elemento que aparece seleccionado por defecto.

HTML

- Los elementos del cuerpo son de 5 tipos:
  - Si se quiere que el usuario decida entre varias posibilidades, se utiliza el elemento radiobutton:

<INPUT TYPE="radio" NAME="yyy" VALUE="zzz" CHECKED>

Atributo	Significado
NAME	Nombre del elemento para poder relacionarlo con el contenido. Para asociar varios botones de radio a una única variable se pone a todos el mismo NAME.
VALUE	Especifica el valor que se asignará a la variable.
CHECKED	Fija la única opción seleccionada por defecto.

HTML

- Los elementos del cuerpo son de 5 tipos:
  - **Botones de envío y de borrado:** Sirven para que el usuario pulse en ellos y envíe la información al servidor o borre los datos introducidos en la página.

    <INPUT TYPE="tipo" VALUE="zzz">

Atributo	Significado
TYPE	Establece el tipo de botón. Para el envío de datos escribiremos "SUBMIT" y para borrar los datos de la página "RESET".
VALUE	Especifica el texto que queremos que aparezca en el botón.

HTML

- Los elementos del cuerpo son de 5 tipos:
  - **Elemento invisible:** Sirven para guardar alguna variable oculta en el formulario para almacenarla en la página activa.

```
<INPUT TYPE=HIDDEN NAME="yyy" VALUE="zzz">
```

Atributo	Significado
NAME	Establece el nombre del elemento para poder relacionarlo con su contenido.
VALUE	Especifica el valor que se asignará a la variable.

HTML

- Marcos
  - Un marco es una ventana independiente dentro de la ventana general del navegador.
  - Cada marco tiene sus propiedades diferenciadas: bordes y barras de desplazamiento propias. Usando marcos cada página se dividirá, en la práctica, en varias páginas independientes.

HTML

- Marcos
  - Para crear marcos se necesita un documento HTML específico, que llamado documento de definición de marcos donde se especifica el tamaño y la posición de cada marco y el documento HTML que contendrá. Este documento no contendrá la etiqueta <BODY>

HTML

- Marcos
  - La estructura general de un marco es:

```
<HTML>
        <HEAD>
                <TITLE>Título de la página</TITLE>
        </HEAD>
         <FRAMESET ROWS="xxx, yyy" COLS="vvv, zzz">
                <FRAME NAME=“marco1" SRC=“pagina1.html">
                <FRAME NAME=“marco2" SRC="página2.html">
                <NOFRAMES>
                        MENSAJE
                </NOFRAMES>
        </FRAMESET>
</HTML>
```

HTML

- Marcos
  - <FRAMESET>:Se usa para definir la distribución de los marcos en la página principal mediante columnas (COLS) y filas (ROWS). Son anidables, por lo que se pueden incluir unos marcos dentro de otros. Los formatos admitidos:
    - Porcentual: Se indica el porcentaje que ocupará del espacio total disponible.
    - Absoluto: Se especifica el tamaño en pixels.
    - Sobre el espacio sobrante: Un asterisco (*) indica que el marco debe ocupar todo el espacio sobrante. Si se pone en varios marcos, se repartirá el espacio equitativamente, y si se quiere que uno tenga más espacio, se debe poner un número delante del asterisco. Así, un marco con un espacio de 3* será tres veces más grande que su compañero con sólo 1*.

HTML

- Marcos
  - <FRAME>: Define las características de un marco, y sus atributos son:

Atributo	Significado
NAME	Asigna el nombre al marco para que podamos referirnos a él.
SRC	Indica la URL del documento HTML que ocupará el marco.
SCROLLING	Establece la aparición de barras de desplazamiento en el marco. Su valor por defecto es AUTO (el navegador decide si se ven o no) .Los otros valores son YES y NO.
NORESIZE	Si lo escribimos, el usuario no podrá cambiar el tamaño del marco.

HTML

- Marcos
  - <FRAME>: Define las características de un marco, y sus atributos son:

Atributo	Significado
FRAMEBORDER	Establece el borde del marco. Para eliminar el borde su valor debe ser 0.
MARGINWIDTH	Fija los márgenes horizontales dentro de un marco y se expresa en pixels.
MARGINHEIGHT	Fija los márgenes verticales dentro de un marco y se expresa en pixels.

HTML

- Marcos
  - <NOFRAMES> : Se utilizan para mostrar un mensaje o una página alternativa cuando el navegador del usuario no soporta marcos.

HTML

- HTML no admite directamente determinados caracteres por compatibilidad internacional, y hay que sustituirlos por códigos:

Código	Resultado
á, Á, é, É, etcétera...	á, Á, é, É, í, Í, ó, Ó, ú y Ú
&ñtilde	ñ
&iquest;	¿
&iexcl;	¡
&deg;	º
&ordf;	ª
&euro;	€
&copy;	©
&reg;	®
	(espacio en blanco, que no puede ser usado para saltar de línea)

Bottle

- Bottle es una framework web para Python que no tiene dependencias externas con otras librerías.
- Para usarla basta tener el archivo bottle.py en el mismo directorio en el que se crea la aplicación web.

Hola Mundo

```
from bottle import route, run

@route('/hola')
def hola():
    return "<h1>¡Hola Mundo!</h1>"

run(host='0.0.0.0', port=8080)
```

Hola Mundo

- Observar:
  - La primera línea importa los módulos route y run de Bottle:
    - El módulo run es usado para ejecutar una aplicación en el servidor de desarrollo.
    - El módulo route indica a la aplicación en que URL hay que solicitar la petición. En las aplicaciones bottle se implementa llamando a una función Python por cada URL solicitada, retornando los resultados de la función al usuario.

```
from bottle import route, run
```

Hola Mundo

- Observar:
  - A continuación se define el patrón de la ruta que se va asociar a la función definida posteriormente.

```
@route('/hola')
```

  - La función retorna una cadena HTML que será mostrada en el navegador.

```
def hola():
    return "<h1>¡Hola Mundo!</h1>"
```

Hola Mundo

- Observar:
  - La línea con la instancia sobre run indica que se ejecute en el servidor de desarrollo. Además el parámetro host='0.0.0.0' indica que el contenido puede ser servido en cualquier ordenador, y el parámetro puerto indica el puerto que se va a usar.

```
run(host='0.0.0.0', port=8080)
```

  - Una vez ejecutado el programa se introduce la siguiente url en el navegador:

http://localhost:8080/hola

Hola Mundo

- Observar:
  - El servidor se puede parar tecleando en el terminal “CTRL-C”

Rutas

- Se pueden asociar más de una ruta a una función:

```
@route('/')
@route('/hola/<name>')
def greet(name='Pepito'):
    return template('Hola {{name}}, ¿qué tal estás?', name=name)
```

Rutas

- Se pueden definir rutas dinámicas que encajen con más de una URL a la vez. Para ello se usan términos parametrizados que van entre “<” y “>”. Así en el ejemplo anterior /hola/<name> aceptará diferentes peticiones de acuerdo al valor que tome la variable “name”.

```
@route('/wiki/<pagename>')              # encaja con /wiki/Learning_Python
def show_wiki_page(pagename):
    ...

@route('/<action>/<user>')              # encaja /follow/defnull
def user_api(action, user):
```

Rutas

- Se pueden usar filtros para definir rutas más específicas. En este caso se usa la sintaxis <nombre:filtro> o bien <nombre:filtro:configuración>.
- Algunos filtros:
  - **:int** encaja con digitos y convierte el valor a entero.
  - **:float** es similar a :int para números decimales.
  - **:path** encaja todos los caracteres incluyendo el caracter "/" y puede ser usado para encajar más de un segment de una ruta.
  - **:re** permite especificar una expresión regular para el campo configuración. El valor encajado no es modificado .

Rutas

```
@route('/object/<id:int>')
def callback(id):
    assert isinstance(id, int)

@route('/show/<name:re:[a-z]+>')
def callback(name):
    assert name.isalpha()

@route('/static/<path:path>')
def callback(path):
    return static_file(path, ...)
```

Métodos de petición HTTP

- El protocolo HTTP define varios métodos para realizar peticiones: GET, POST, PUT, DELETE or PATCH. Por defecto se usa GET.
- Para usarlos se añade la palabra al método route() o bien se utilizan las palabras clave: get(), post(), put(), delete() or patch()

Métodos de petición HTTP

```
from bottle import get, post, request # or route

@get('/login') # or @route('/login')
def login():
    return '''
        <form action="/login" method="post">
            Username: <input name="username" type="text" />
            Password: <input name="password" type="password" />
            <input value="Login" type="submit" />
        </form>
    '''

@post('/login') # or @route('/login', method='POST')
def do_login():
    username = request.forms.get('username')
    password = request.forms.get('password')
    if check_login(username, password):
        return "<p>Your login information was correct.</p>"
    else:
        return "<p>Login failed.</p>"
```

Métodos de petición HTTP

- En este ejemplo la URL /login es enlazada a dos funciones diferentes, una para una petición GET y otra para una petición POST. La primera muestra un formulario HTML al usuario y el segundo es invocado cuando se envía un formulario y se chequea la información que el usuario ha entrado en el formulario.

Archivos estáticos

- Los archivos estáticos tales como imágenes no son servidos automáticamente, debiendo añadir una ruta y una función para controlar que los archivos son servidos y dónde se encuentran.

```
from bottle import static_file
@route('/static/<filename>')
def server_static(filename):
    return static_file(filename, root='/path/to/your/static/files')
```

Archivos estáticos

- En el ejemplo anterior se sirven los archivos que se encuentran directamente en el directorio /path/to/your/static/files
- Para servir archivos en subdirectorios se debe usar un filtro:

```
@route('/static/<filepath:path>')
def server_static(filepath):
    return static_file(filepath, root='/path/to/your/static/files')
```

Archivos estáticos

```
from bottle import static_file
@route('/images/<filename:re:.*\.png>')
def send_image(filename):
    return static_file(filename, root='/path/to/image/files', mimetype='image/png')

@route('/static/<filename:path>')
def send_static(filename):
    return static_file(filename, root='/path/to/static/files')
```

Archivos estáticos

- Se puede forzar a que un archivo sea descargado o sugerir un nombre al usuario de la siguiente manera:

```
@route('/download/<filename:path>')
def download(filename):
    return static_file(filename, root='/path/to/static/files', download=filename)
```

Páginas de error

- Cuando se produce un error Bottle muestra información sobre el mismo. Esta información puede ser sobreescrita usando la palabra reservada error()

```
from bottle import error
@error(404)
def error404(error):
    return 'Nothing here, sorry'
```

Páginas de error

- El único parámetro que se pasa al gestor de errores es una instancia de HTTPError.
- Se puede leer una petición, escribir una respuesta y retornar cualquier tipo de datos excepto instancias HTTPError.
- Los gestores de errores solo son usados para aplicaciones que retornan o lanzan excepciones del tipo HTTPError.

Generación de contenido

- Los tipos de datos que se pueden devolver desde Bottle son los siguientes:
  - Diccionarios: Los diccionarios de Python son transformados en cadenas JSON y retornadas al navegador con el Content-Type de la cabecera configurada a application/json.
  - Cadena vacía, Falso, None y valores booleanos. Producen una salida vacía con el Content-Length de la cabecera a 0.
  - Cadenas Unicode son automaticamente codificadas con el codec especificado en el Content-Type de la cabecera (utf8 por defecto) y es tratada como una cadena de bytes normal.

Generación de contenido

- Los tipos de datos que se pueden devolver desde Bottle son los siguientes:
  - Cadenas de Bytes. Bottle retorna una cadena entera y añade un Content-Length de la cabecera basado en la longitud de la cadena.
  - Instancias de HTTPError o HTTPResponse. Tiene el mismo efecto que lanzar una excepción. En caso de un HTTPError, el gestor de errores es aplicado.
  - Objetos de archivo. Todas las cosas que tienen un método read() son tratados como un archive.

Codificación

- Bottle usa el parámetro charset del Content-Type de la cabecera para decidir como codificar las cadenas unicode. Por defecto es text/html; charset=UTF8 and puede ser cambiado usando el atributo Response.content_type o configurando el atributo Response.charset

```
from bottle import response
@route('/iso')
def get_iso():
    response.charset = 'ISO-8859-15'
    return u'This will be sent with ISO-8859-15 encoding.'

@route('/latin9')
def get_latin():
    response.content_type = 'text/html; charset=latin9'
    return u'ISO-8859-15 is also known as latin9.'
```

Codificación

- Alguna veces Python codifica los nombres de forma diferente a los nombres que son soportados por la especificación HTTP. En estos casos primero se configure la cabecera Response.content_type y a continuación se configure el atributo Response.charset

Cookies

- Es un trozo de texto almacenado perfil de usuario del navegador. Se puede acceder mediante Request.get_cookie() y se puede configurar nuevas cookies con el método Request.set_cookie()

```
@route('/hello')
def hello_again():
    if request.get_cookie("visited"):
        return "Welcome back! Nice to see you again"
    else:
        response.set_cookie("visited", "yes")
        return "Hello there! Nice to meet you"
```

Cookies

- El método Request.set_cookie() tiene un conjunto de parámetros:
  - max_age: Máxima edad en segundos(Por defecto None)
  - expires: Tiempo en que expira(Por defecto None)
  - domain: Dominio es es permitido para leer la cookie(Por defecto el dominio actual)
  - path: Limita la cookie a un camino dado.
  - secure: Limita la cookie a conexiones Https(Por defecto a inactivo)
  - httponly: Previene la lectura de la cookie desde javascript.(Por defecto inactivo).

Cookies

- Observaciones:
  - Las cookies están limitadas a 4kb de texto en muchos navegadores.
  - Muchos usuarios configuran sus navegadores para no aceptar cookies.
  - Las cookies son almacenadas en el lado del cliente y no son encriptadas

Cookies

- Las cookies pueden ser firmadas criptográficamente de forma que no pueden ser manipuladas, para lo cual se debe proporcionar una clave de firma via un argumento que sea una clave secreta.

```
@route('/login')
def do_login():
    username = request.forms.get('username')
    password = request.forms.get('password')
    if check_login(username, password):
        response.set_cookie("account", username, secret='some-secret-key')
        return template("<p>Welcome {{name}}! You are now logged in.</p>", name=user
    else:
        return "<p>Login failed.</p>"

@route('/restricted')
def restricted_area():
    username = request.get_cookie("account", secret='some-secret-key')
    if username:
        return template("Hello {{name}}. Welcome back.", name=username)
    else:
        return "You are not logged in. Access denied."
```

Objeto Request

- Las cookies, las cabeceras http , los campos de los formularios HTML y otros datos solicitados se pueden obtener a través del objeto global request.

```
from bottle import request, route, template

@route('/hello')
def hello():
    name = request.cookies.username or 'Guest'
    return template('Hello {{name}}', name=name)
```

Objeto request

- Bottle usa un tipo especial de diccionario para almacenar los datos de los formularios y las cookies denominado FormsDict:
  - Todos los valores en el diccionario son accesibles como atributos retornando cadenas unicode. Incluso si no tiene valor se devuelve la cadena vacía.

```
name = request.cookies.name

# is a shortcut for:

name = request.cookies.getunicode('name') # encoding='utf-8' (default)

# which basically does this:

try:
    name = request.cookies.get('name', '').decode('utf-8')
except UnicodeError:
    name = u''
```

Objeto request

Cookies

- Las cookies son pequeños trozos de texto almacenados en los navegadores de los clientes que son enviados a los servidores en cada petición.
- Todas las cookies del clientes son accesible mediante BaseRequest.cookies. En el siguiente ejemplo se muestra un simple contador basado en las cookies.

```
from bottle import route, request, response
@route('/counter')
def counter():
    count = int( request.cookies.get('counter', '0') )
    count += 1
    response.set_cookie('counter', str(count))
    return 'You visited this page %d times' % count
```

Objeto request

Cookies

- Un método alternativo para acceder a las cookies es BaseRequest.get_cookie()

Objeto request

Cabeceras http

- Todas las cabeceras enviadas por el cliente son almacenadas en un diccionario sensitivo a mayúsculas y minúsculas denominado WSGIHeaderDict que es accesible desde el atributo BaseRequest.headers

```
from bottle import route, request
@route('/is_ajax')
def is_ajax():
    if request.headers.get('X-Requested-With') == 'XMLHttpRequest':
        return 'This is an AJAX request'
    else:
        return 'This is a normal request'
```

Objeto request

Variables Query

- La cadena de consulta como /forum?id=1&page=5, usadas para enviar pares de clave/valor al servidores, se puede acceder a sus valores mediante el atributo BaseRequest.query (un FormsDict), y mediante BaseRequest.query_string se puede acceder a la cadena entera.

```
from bottle import route, request, response, template
@route('/forum')
def display_forum():
    forum_id = request.query.id
    page = request.query.page or '1'
    return template('Forum ID: {{id}} (page {{page}})', id=forum_id, page=page)
```

Objeto request

Formularios HTML

- Considerar un formulario típico de html:

```
<form action="/login" method="post">
    Username: <input name="username" type="text" />
    Password: <input name="password" type="password" />
    <input value="Login" type="submit" />
</form>
```

Objeto request

Formularios HTML

- El atributo action especifica la URL que recibirá los datos del formulario, el método define se usa GET o POST(si usa GET los valores del formulario serán accesibles a través de la URL y por tanto del método BaseRequest.query)
- Los campos del formulario transmtidos mediante POST son almacenados en BaseReques.forms como un FormsDict

Objeto request

Formularios HTML

```
from bottle import route, request

@route('/login')
def login():
    return '''
        <form action="/login" method="post">
            Username: <input name="username" type="text" />
            Password: <input name="password" type="password" />
            <input value="Login" type="submit" />
        </form>
    '''

@route('/login', method='POST')
def do_login():
    username = request.forms.get('username')
    password = request.forms.get('password')
    if check_login(username, password):
        return "<p>Your login information was correct.</p>"
    else:
        return "<p>Login failed.</p>"
```

Objeto request

Formularios HTML

Existen un conjutno de atributos para acceder a los datos del formulario, que se resumen en la siguiente tabla:

Attribute	GET Form fields	POST Form fields	File Uploads
`BaseRequest.query`	yes	no	no
`BaseRequest.forms`	no	yes	no
`BaseRequest.files`	no	no	yes
`BaseRequest.params`	yes	yes	no
`BaseRequest.GET`	yes	no	no
`BaseRequest.POST`	no	yes	yes

Objeto request

Archivos cargados

Para subir un fichero al servidor se codifica el formulario de una forma diferente, añadiendo el atributo enctype="multipart/form-data" a la etiqueta <form>. Además se añade las etiquetas <input type="file" /> para poder seleccionar el fichero.

```
<form action="/upload" method="post" enctype="multipart/form-data">
  Category:      <input type="text" name="category" />
  Select a file: <input type="file" name="upload" />
  <input type="submit" value="Start upload" />
</form>
```

Objeto request

Archivos cargados

Bottle almacena los archivos cargados en BaseRequest.files como instancias FileUpload con metadaos acerca del archivo subido. Supongamos que se quiere guardar un archivo al disco:

```
@route('/upload', method='POST')
def do_upload():
    category   = request.forms.get('category')
    upload     = request.files.get('upload')
    name, ext = os.path.splitext(upload.filename)
    if ext not in ('.png','.jpg','.jpeg'):
        return 'File extension not allowed.'

    save_path = get_save_path_for_category(category)
    upload.save(save_path) # appends upload.filename automatically
    return 'OK'
```

Objeto request

Archivos cargados

FileUpload.filename contiene el nombre del archivo sobre el sistema de archivos cliente, pero se encuentra modificado para prevenir fallos. De manera que si se necesita el nombre no modificado tal como lo ha enviado el cliente, se puede acceder a través de FileUpload.raw_filename.

Objeto request

Archivos cargados

El método FileUpload.save se usa para guardar un archivo en disco. Además se puede acceder al objeto archivo directamente mediante FileUpload.file

Objeto request

JSON

Algunas aplicaciones envían contenido del tipo application/json al servidor.
Este contenido se encuentra en el atributo BaseRequest.json

Plantillas

- Bottle dispone de un motor de plantillas. Para usar una plantilla se puede usar la función template() o bien el decorador view(). Se debe proporcionar el nombre de la plantilla y las variables que se desean pasar como argumentos clave/valor

```
@route('/hello')
@route('/hello/<name>')
def hello(name='World'):
    return template('hello_template', name=name)
```

Plantillas

- En este ejemplo se carga la plantilla hello_template.tpl y se pasa como argumento la variable name.
- En general las plantillas se almacenan en la carpeta ./views/ o en aquellas carpetas especificadas en la lista bottle.TEMPLATE_PATH

Plantillas

- El decorador view() permite retornar un diccionario con las variables de la plantilla:

```
@route('/hello')
@route('/hello/<name>')
@view('hello_template')
def hello(name='World'):
    return dict(name=name)
```

Plantillas

- Las plantillas son almacenadas en memoria , y las modificaciones no tendrán efecto hasta que se limpie la cache usando el método bottle.TEMPLATES.clear()

Plugins

- Para desinstalar un plugin se usa uninstall() con argumento el nombre del componente instalado

```
sqlite_plugin = SQLitePlugin(dbfile='/tmp/test.db')
install(sqlite_plugin)

uninstall(sqlite_plugin) # uninstall a specific plugin
uninstall(SQLitePlugin)  # uninstall all plugins of that type
uninstall('sqlite')      # uninstall all plugins with that name
uninstall(True)          # uninstall all plugins at once
```

Plugins

- Los plugins pueden ser instalado y eliminados las veces que sea necesario.
- Solo afectan a determinadas rutas, aunque se puede hacer explicito a quien deber afectar mediante el método apply de route:

```
sqlite_plugin = SQLitePlugin(dbfile='/tmp/test.db')

@route('/create', apply=[sqlite_plugin])
def create(db):
    db.execute('INSERT INTO ...')
```

Plugins

- Se puede desactivar mediante el método skip de route:

```
sqlite_plugin = SQLitePlugin(dbfile='/tmp/test1.db')
install(sqlite_plugin)

dbfile1 = '/tmp/test1.db'
dbfile2 = '/tmp/test2.db'

@route('/open/<db>', skip=[sqlite_plugin])
def open_db(db):
    # The 'db' keyword argument is not touched by the plugin this time.

    # The plugin handle can be used for runtime configuration, too.
    if db == 'test1':
        sqlite_plugin.dbfile = dbfile1
    elif db == 'test2':
        sqlite_plugin.dbfile = dbfile2
    else:
        abort(404, "No such database.")

    return "Database File switched to: " + sqlite_plugin.dbfile
```

Modelo MVC

- El modelo MVC establece que en toda aplicación web se debe separar la funcionalidad de la interface de usuario, para lo cual se definen 3 componentes:
  - Modelo: Representación de los datos y responsable de almacenar, consultar y modificar los datos.
  - Vista: Describe como la información debería ser mostrado al usuario. Es usado para formatear y controlar la presentación de los datos.
  - Controlador: Es aquella que realiza el procesamiento decidiendo como responder a las peticiones de usuario.

Ejemplo de MVC

Creación de un modelo

Se va a usar como componente para la persistencia de los datos una base de datos SQLite. Para ello se creará una base de datos con un conjunto de datos.

Ejemplo de MVC

Creación de un modelo

```
import sqlite3
db = sqlite3.connect('libreria.sqlite3')
db.execute("CREATE TABLE libros (id INTEGER PRIMARY KEY, item CHAR(100) NOT NULL, cantidad INTEGER NOT NULL)")
db.execute("INSERT INTO libros (item,cantidad) VALUES ('El Quijote', 4)")
db.execute("INSERT INTO libros (item,cantidad) VALUES ('Dracula', 2)")
db.execute("INSERT INTO libros (item,cantidad) VALUES ('Guerra y Paz', 30)")
db.execute("INSERT INTO libros (item,cantidad) VALUES ('Hamlet', 1)")
db.execute("INSERT INTO libros (item,cantidad) VALUES ('Frankestein', 4)")
db.commit()
```

Ejemplo de MVC

Creación de un modelo

#	id	item	cantidad
1	1	El Quijote	4
2	2	Dracula	2
3	3	Guerra y Paz	30
4	4	Hamlet	1
5	5	Frankestein	4

Ejemplo de MVC

Creación del controlador

En el controlador se implementa la funcionalidad principal de la aplicación:

- Conectarse a la base de datos.
- Obtener los datos de la tabla.
- Llamar a la vista para generar la vista, y mostrar los datos formateados al usuario.

Ejemplo de MVC

Creación del controlador

```
import sqlite3
from bottle import route, run, template

@route('/libreria')
def mostrar_libros():
    db = sqlite3.connect('libreria.sqlite3')
    c = db.cursor()
    c.execute("SELECT item,cantidad FROM libros")
    data = c.fetchall()
    c.close()
    output = template('mostrar_libros', rows=data)
    return output

run(host='0.0.0.0', port=8080)
```

Ejemplo de MVC

Creación del controlador

Observar que la línea output = template('mostrar_libros', rows=data) llama a la vista para formatear los datos. La vista es una plantilla denominada "mostrar_libros.tpl" que formatea los datos que le son pasados a través de la variable rows que toma como valor los datos recuperados en la consulta.

Ejemplo de MVC

Creación de la vista

- La aplicación buscará una plantilla que coincida con el nombre dado en la función template() y finalizado con la extensión .tpl
- El archivo puede mezclar HTML con trozos de código en Python.

Ejemplo de MVC

Creación de la vista

```
<h1>Libros que se encuentran en la librería</h1>
<table>
<tr><th>Item</th><th>Cantidad</th></tr>
%for row in rows:
    <tr>
    %for col in row:
        <td>{{col}}</td>
    %end
    </tr>
%end
</table>
```

Ejemplo de MVC

Creación de la vista

Observar:

- Se usa un bucle para crear una tabla que se rellena con los datos del modelo.
- La variable rows que tenía los datos del modelo es accesible desde la plantilla.
- Las líneas en Python van precedidas por "%", siendo posible acceder a la variables desde HTML usando la sintaxis "{{var}}"

Ejemplo de MVC

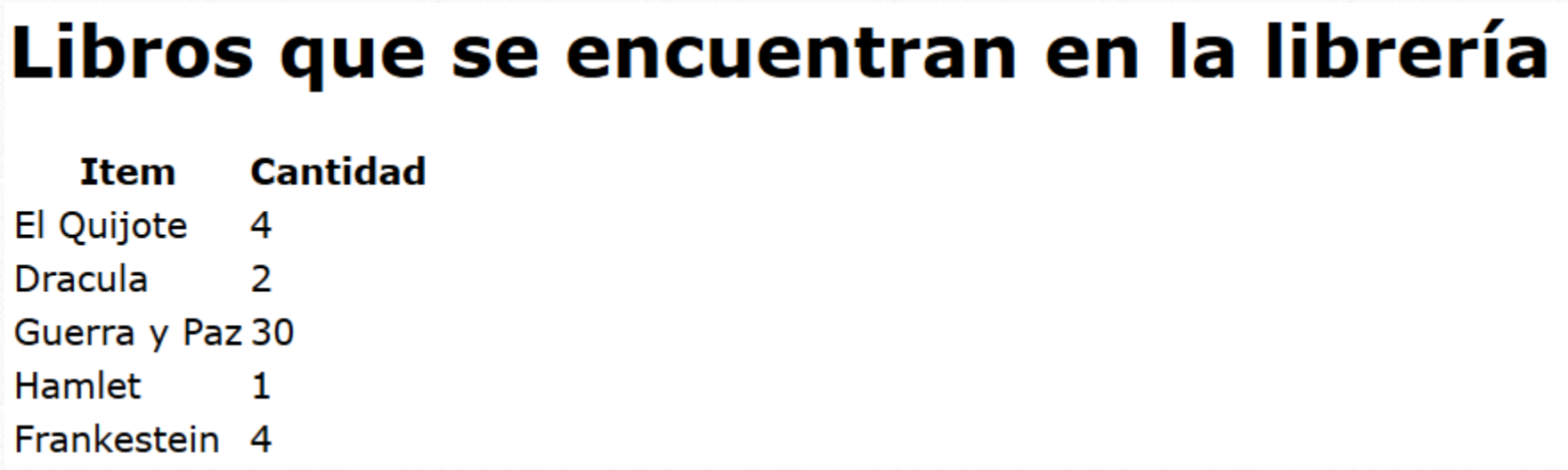
Ejecución

Se ejecuta el programa que implementa el controlador, y se visita la url:

http://localhost:8080/libreria

Ejemplo de MVC

Ejecución



Libros que se encuentran en la librería

Item	Cantidad
El Quijote	4
Dracula	2
Guerra y Paz	30
Hamlet	1
Frankestein	4