		ASIGNATURA tecnologías para la programación y el diseño web i	CURSO 2	GRUPO 01
CALIFICACIÓN	EVALUACIÓN	APELLIDOS	NOMBRE	ID
		OBSERVACIONES Ejercicio 3: sistema de plantillas y estilos	FECHA 30/03/2016	FECHA ENTREGA 22/04/2016

Ejercicio 3

A continuación se muestran unos enlaces y un tutorial recomendados para el desarrollo de una aplicación web, la conexión a una base de datos y la definición de plantillas (templates) para la presentación de la información.

- Python Web Framework `http://bottlepy.org/docs/dev/`
- World Wide Web consortium `http://www.w3.org`
- Web Design and Applications `http://www.w3.org/standards/webdesign/`

Bottle: Python Web Framework

Bottle is a fast, simple and lightweight `WSGI` micro web-framework for `Python`. It is distributed as a single file module and has no dependencies other than the `Python Standard Library`.

- **Routing**: Requests to function-call mapping with support for clean and dynamic URLs.
- **Templates**: Fast and pythonic built-in template engine and support for `mako`, `jinja2` and `cheetah` templates.
- **Utilities**: Convenient access to form data, file uploads, cookies, headers and other HTTP-related metadata.
- **Server**: Built-in HTTP development server and support for `paste`, `fapws3`, `bjoern`, `Google App Engine`, `cherrypy` or any other WSGI capable HTTP server.

Example: “Hello World” in a bottle

```
from bottle import route, run, template

@route('/hello/<name>')
def index(name='World'):
    return template('<b>Hello {{name}}</b>!', name=name)

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

Run this script or paste it into a Python console, then point your browser to `http://localhost:8080/hello/world`. That’s it. Try different routes: `http://localhost:8080/hello/juan`

SimpleTemplate Engine

Bottle comes with a fast, powerful and easy to learn built-in template engine called SimpleTemplate or stpl for short. It is the default engine used by the view() and template() helpers but can be used as a stand-alone general purpose template engine too. This document explains the template syntax and shows examples for common use cases.

Basic API Usage:

SimpleTemplate implements the BaseTemplate API:

```
>>> from bottle import SimpleTemplate
>>> tpl = SimpleTemplate('Hello {{name}}!')
>>> tpl.render(name='World')
u'Hello World!'
```

In this document we use the template() helper in examples for the sake of simplicity:

```
>>> from bottle import template
>>> template('Hello {{name}}!', name='World')
u'Hello World!'
```

Just keep in mind that compiling and rendering templates are two different actions, even if the template() helper hides this fact. Templates are usually compiled only once and cached internally, but rendered many times with different keyword arguments.

		ASIGNATURA tecnologías para la programación y el diseño web i	CURSO 2	GRUPO 01
CALIFICACIÓN	EVALUACIÓN	APELLIDOS	NOMBRE	ID
		OBSERVACIONES Ejercicio 3: sistema de plantillas y estilos	FECHA 30/03/2016	FECHA ENTREGA 22/04/2016

1. SIMPLETEMPLATE SYNTAX

Python is a very powerful language but its whitespace-aware syntax makes it difficult to use as a template language. SimpleTemplate removes some of these restrictions and allows you to write clean, readable and maintainable templates while preserving full access to the features, libraries and speed of the Python language.

Warning: The SimpleTemplate syntax compiles directly to python bytecode and is executed on each SimpleTemplate.render() call. Do not render untrusted templates! They may contain and execute harmful python code.

1.1. INLINE EXPRESSIONS

You already learned the use of the {{...}} syntax from the “Hello World!” example above, but there is more: any python expression is allowed within the curly brackets as long as it evaluates to a string or something that has a string representation:

```
>>> template('Hello {{name}}!', name='World')
u'Hello World!'
>>> template('Hello {{name.title() if name else "stranger"}}!', name=None)
u'Hello stranger!'
>>> template('Hello {{name.title() if name else "stranger"}}!', name='mArC')
u'Hello Marc!'
```

The contained python expression is executed at render-time and has access to all keyword arguments passed to the SimpleTemplate.render() method. HTML special characters are escaped automatically to prevent XSS attacks. You can start the expression with an exclamation mark to disable escaping for that expression:

```
>>> template('Hello {{name}}!', name='<b>World</b>')
u'Hello &lt;b&gt;World&lt;/b&gt;!'
>>> template('Hello {{!name}}!', name='<b>World</b>')
u'Hello <b>World</b>!'
```

1.2. EMBEDDED PYTHON CODE

The template engine allows you to embed lines or blocks of python code within your template. Code lines start with % and code blocks are surrounded by <% and %> tokens:

```
% name = "Bob"  # a line of python code
<p>Some plain text in between</p>
<%
  # A block of python code
  name = name.title().strip()
%>
<p>More plain text</p>
```

Embedded python code follows regular python syntax, but with two additional syntax rules:

- **Indentation is ignored**. You can put as much whitespace in front of statements as you want. This allows you to align your code with the surrounding markup and can greatly improve readability.
- Blocks that are normally indented now have to be closed explicitly with an end keyword.

```
<ul>
  % for item in basket:
    <li>{{item}}</li>
  % end
</ul>
```

Both the % and the <% tokens are only recognized if they are the first non-whitespace characters in a line. You don’t have to escape them if they appear mid-text in your template markup. Only if a line of text starts with one of these tokens, you have to escape it with a backslash. In the rare case where the backslash + token combination appears in your markup at the beginning of a line, you can always help yourself with a string literal in an inline expression:

```
This line contains % and <% but no python code.
\% This text-line starts with the '%' token.
\<% Another line that starts with a token but is rendered as text.
{{'\\%'}} this line starts with an escaped token.
```

		ASIGNATURA tecnologías para la programación y el diseño web i	CURSO 2	GRUPO 01
CALIFICACIÓN	EVALUACIÓN	APELLIDOS	NOMBRE	ID
		OBSERVACIONES Ejercicio 3: sistema de plantillas y estilos	FECHA 30/03/2016	FECHA ENTREGA 22/04/2016

If you find yourself to escape a lot, consider using custom tokens.

1.3. WHITESPACE CONTROL

Code blocks and code lines always span the whole line. Whitespace in front of after a code segment is stripped away. You won't see empty lines or dangling whitespace in your template because of embedded code:

```
<div>
 % if True:
  <span>content</span>
 % end
</div>
```

This snippet renders to clean and compact html:

```
<div>
  <span>content</span>
</div>
```

But embedding code still requires you to start a new line, which may not what you want to see in your rendered template. To skip the newline in front of a code segment, end the text line with a double-backslash:

```
<div>\\
 %if True:
<span>content</span>\\
 %end
</div>
```

This time the rendered template looks like this:

```
<div><span>content</span></div>
```

This only works directly in front of code segments. In all other places you can control the whitespace yourself and don't need any special syntax.

2. TEMPLATE FUNCTIONS

Each template is preloaded with a bunch of functions that help with the most common use cases. These functions are always available. You don't have to import or provide them yourself. For everything not covered here there are probably good python libraries available. Remember that you can import anything you want within your templates. They are python programs after all.

Changed in version 0.12: Prior to this release, include() and rebase() were sytnax keywords, not functions.

include(sub_template, **variables)

Render a sub-template with the specified variables and insert the resulting text into the current template. The function returns a dictionary containing the local variables passed to or defined within the sub-template:

```
% include('header.tpl', title='Page Title')
Page Content
% include('foother.tpl')
```

rebase(name, **variables)

Mark the current template to be later included into a different template. After the current template is rendered, its resulting text is stored in a variable named base and passed to the base-template, which is then rendered. This can be used to wrap a template with surrounding text, or simulate the inheritance feature found in other template engines:

```
% rebase('base.tpl', title='Page Title')
<p>Page Content ...</p>
```

This can be combined with the following base.tpl:

```
<html>
<head>
```

		ASIGNATURA tecnologías para la programación y el diseño web i	CURSO 2	GRUPO 01
CALIFICACIÓN	EVALUACIÓN	APELLIDOS	NOMBRE	ID
		OBSERVACIONES Ejercicio 3: sistema de plantillas y estilos	FECHA 30/03/2016	FECHA ENTREGA 22/04/2016

```
  <title>{{title or 'No title'}}</title>
</head>
<body>
  {{base}}
</body>
</html>
```

Accessing undefined variables in a template raises NameError and stops rendering immediately. This is standard python behavior and nothing new, but vanilla python lacks an easy way to check the availability of a variable. This quickly gets annoying if you want to support flexible inputs or use the same template in different situations. These functions may help:

defined(name)

Return True if the variable is defined in the current template namespace, False otherwise.

get(name, default=None)

Return the variable, or a default value.

setdefault(name, default)

If the variable is not defined, create it with the given default value. Return the variable.

Here is an example that uses all three functions to implement optional template variables in different ways:

```
% setdefault('text', 'No Text')
<h1>{{get('title', 'No Title')}}</h1>
<p> {{ text }} </p>
% if defined('author'):
  <p>By {{ author }}</p>
% end
```

3. Listado completo del ejercicio

Código principal de la aplicación `todo.py`:

		ASIGNATURA tecnologías para la programación y el diseño web i	CURSO 2	GRUPO 01
CALIFICACIÓN	EVALUACIÓN	APELLIDOS	NOMBRE	ID
		OBSERVACIONES Ejercicio 3: sistema de plantillas y estilos	FECHA 30/03/2016	FECHA ENTREGA 22/04/2016

```
import sqlite3
from bottle import route, run, debug, template, request, static_file, error

@route('/')
@route('/todo')
def todo_list():
    conn = sqlite3.connect('todo.db')
    c = conn.cursor()
    c.execute("SELECT id, task FROM todo WHERE status LIKE '1';")
    result = c.fetchall()
    c.close()
    output = template('templates/make_table', rows=result, title='Show open tasks')
    return output

@route('/new', method='GET')
def new_item():
    if request.GET.get('save','').strip():
        new = request.GET.get('task', '').strip()
        conn = sqlite3.connect('todo.db')
        c = conn.cursor()
        c.execute("INSERT INTO todo (task,status) VALUES (?,?)", (new,1))
        new_id = c.lastrowid
        conn.commit()
        c.close()
        return template('templates/message',
            message='The new task was inserted into the database, the ID is %s' % new_id,
            title='New task added!')
    else:
        return template('templates/new_task.tpl', title='New task')

@route('/edit/:no', method='GET')
def edit_item(no):
    if request.GET.get('save','').strip():
        edit = request.GET.get('task','').strip()
        status = request.GET.get('status','').strip()
        if status == 'open':
            status = 1
        else:
            status = 0
        conn = sqlite3.connect('todo.db')
        c = conn.cursor()
        c.execute("UPDATE todo SET task = ?, status = ? WHERE id LIKE ?", (edit,status,no))
        conn.commit()
        return template('templates/message', message='The item number %s was successfully updated' % no, title='Task updated')
    else:
        conn = sqlite3.connect('todo.db')
        c = conn.cursor()
        c.execute("SELECT task FROM todo WHERE id LIKE ?", (str(no),))
        cur_data = c.fetchone()
        return template('templates/edit_task', old = cur_data, no = no, title='Edit task '+ cur_data[0])

@route('/item:item#[0-9]+#')
def show_item(item):
        conn = sqlite3.connect('todo.db')
        c = conn.cursor()
        c.execute("SELECT task FROM todo WHERE id LIKE ?", (item,))
        result = c.fetchone()
        c.close()
        if not result:
            return template('templates/message', message='This item number does not exist!', title='Error')
        else:
            return template('templates/message', message='Task: %s' % result[0] , title='Show task')

@route('/help')
def help():
    return static_file('help.html', root='.')

@route('/css/:filename', name='css')
def server_static(filename):
    return static_file(filename, root='./css')

@error(403)
def mistake403(code):
    return template('templates/message', message='There is a mistake in your url!', title='Error')

@error(404)
def mistake404(code):
    return template('templates/message', message='Sorry, this page does not exist!', title='Error')

debug(True)
run(host='localhost', port=8080)
#remember to remove reloader=True and debug(True) when you move your application from development to a productive environment
```

Template `make_table.tpl`:

		ASIGNATURA tecnologías para la programación y el diseño web i	CURSO 2	GRUPO 01
CALIFICACIÓN	EVALUACIÓN	APELLIDOS	NOMBRE	ID
		OBSERVACIONES Ejercicio 3: sistema de plantillas y estilos	FECHA 30/03/2016	FECHA ENTREGA 22/04/2016

```
<html>
% include('templates/head.tpl', title='Show open tasks')
<body>
%#template to generate a HTML table from a list of tuples (or list of lists, or tuple of tuples or ...)
<p>The open items are as follows:</p>
<table>
%for row in rows:
  <tr>
  %for col in row:
    <td>{{col}}</td>
  %end
  <td class="centered"><a href="/edit/{{row[0]}}">edit</a></td>
</tr>
%end
</table>

<p>Create a new task <a href="/new">here</a></p>

% include('templates/footer.tpl')
</body>
</html>
```

Template `edit_task.tpl`:

```
%#template for editing a task
%#the template expects to receive a value for "no" as well a "old", the text of the selected ToDo item
<html>
% include('templates/head.tpl', title='Edit task '+ no)

<body>
<p>Edit the task with ID = {{no}}</p>
<form action="/edit/{{no}}" method="get">
<input type="text" name="task" value="{{old[0]}}" size="100" maxlength="100" placeholder="Type the description here...">
<select name="status">
<option>open</option>
<option>closed</option>
</select>
<br/>
<input type="submit" name="save" value="save">
</form>

% include('templates/footer.tpl')

</body>
</html>
```

Template `new_task.tpl`:

```
%#template for the form for a new task
<html>
% include('templates/head.tpl', title=title)
<body>
<p>Add a new task to the ToDo list:</p>
<form action="/new" method="get">
<input type="text" size="100" maxlength="100" name="task" placeholder="Type a new task here...">
<input type="submit" name="save" value="save">
</form>
% include('templates/footer.tpl')
</body>
</html>
```

Template `footer.tpl`:

```
<footer>
<p><a href="/">Show all tasks...</a> | <a href="/help">Help</a></p>
</footer>
```

Template `message.tpl`:

		ASIGNATURA tecnologías para la programación y el diseño web i	CURSO 2	GRUPO 01
CALIFICACIÓN	EVALUACIÓN	APELLIDOS	NOMBRE	ID
		OBSERVACIONES Ejercicio 3: sistema de plantillas y estilos	FECHA 30/03/2016	FECHA ENTREGA 22/04/2016

```
%#template for the form for a new task
<html>
% include('templates/head.tpl', title=title)
<body>
<p>{{message}}</p>
% include('templates/footer.tpl')
</body>
</html>
```

Template `head.tpl`:

```
<head>
<title>{{title}}</title>
<link rel="stylesheet" type="text/css" href="/css/styles.css">
<link href='http://fonts.googleapis.com/css?family=Sintony:400,700' rel='stylesheet' type='text/css'>
</head>
```

Hoja de estilos CSS `styles.css`:

ASIGNATURA	CURSO	GRUPO
tecnologías para la programación y el diseño web i	2	01

CALIFICACIÓN	EVALUACIÓN	APELLIDOS	NOMBRE	ID
		OBSERVACIONES Ejercicio 3: sistema de plantillas y estilos	FECHA 30/03/2016	FECHA ENTREGA 22/04/2016

```
/* TYPOGRAPHY
/* ------------------------------------------------- */

html, body * {
      font-family: 'Sintony', sans-serif;
}

/* LAYOUT
/* ------------------------------------------------- */

body {
      font-size: 10pt;
      line-height: 12pt;
      margin: 20px auto;
}

@media all {
  body {
    width: 800px;
    background: url('https://cdn0.iconfinder.com/data/icons/seo-smart-pack/128/grey_new_seo-43-512.png') 10px 10px no-repeat;
    background-size: 150px;
  }
}

@media (max-width:800px) {
  body {
    width: 80%;
    background: url('https://cdn0.iconfinder.com/data/icons/seo-smart-pack/128/grey_new_seo-43-512.png') no-repeat;
    background-attachment: fixed;
    background-position: center;
    background-size: 450px;
  }
}

footer {margin-top: 50px;}


/* COMMON STYLES
/* ------------------------------------------------- */

a {
      text-decoration: none;
      color: black;
}

a:hover{
      font-weight: bold;
}

pre {
      font-family: monospace, courier;
      padding:5px;
      border: 1px dashed #d1d1d1;
      background-color: #f1f1f1;
      font-size: 9pt;
      line-height: 9pt;
}
div.step {
      padding: 10px;
}
div.step:hover {
      background-color: #f9f9f9;
}

/* FORMS
/* ------------------------------------------------- */

input {
      font-size: 10pt;
      line-height: 12pt;
      border: none;
      outline: none;
}


/* TABLES
/* ------------------------------------------------- */

table {
      font-size: 10pt;
      line-height: 12pt;
      border-collapse: collapse;
      border: none;
      width: 100%;
}

tr {
      border-top: 1px solid rgba(0,0,0,.15);
      border-bottom: 1px solid rgba(0,0,0,.15);
}

td {
      padding: 4px 10px;
      opacity: .35;
}

td.centered {
      text-align: center;
}

td.centered a {
      opacity: 0;
}

tr:hover td {
      background: #f1f1f1;
      opacity: 1;
}

tr:hover a {opacity: 1;}
```

		ASIGNATURA tecnologías para la programación y el diseño web i	CURSO 2	GRUPO 01
CALIFICACIÓN	EVALUACIÓN	APELLIDOS	NOMBRE	ID
		OBSERVACIONES Ejercicio 3: sistema de plantillas y estilos	FECHA 30/03/2016	FECHA ENTREGA 22/04/2016

Entregables y tareas opcionales

Después de seguir la guía/tutorial los alumnos deberán entregar sus trabajos en un paquete comprimido ZIP por correo electrónico antes de la fecha indicada a través del campus online. En caso de dudas pueden contactar en la siguiente dirección: `jsanz.eps@ceu.es`. Incluir tanto el informe creado (formato PDF) como el fichero de base de datos, como las plantillas y hojas de estilo creadas., es decir, la carpeta que contiene la aplicación enyregada con las modificaciones a la misma realizadas por el alumno.

1. Antes de comenzar el ejercicio

- Crear la DB desde el fichero todo-db.py
- Comprobar las opciones de ejecución del servidor y las rutas definidas
- Ejecutar la aplicación y comenzar a jugar con ella

2. Algunas cosas Incluir templates

- Crear una plantilla footer (footer.tpl) con enlaces en el pie de la página
- Incluir la plantilla footer en el resto de plantillas

```
% include('footer.tpl')
```

- Crear una plantilla message (message.tpl) para mostrar mensajes con ella
- Cambiar todos los return de las funciones para devolver mensajes con dicha plantilla, p.ej.:

```
@error(403)
def mistake403(code):
    return '<p>There is a mistake in your url!</p>'

@error(403)
def mistake403(code):
    return template('templates/message', message='There is a mistake in your url!', title='Error 403')
```

- Crear una plantilla head (head.tpl) con el título y otras opciones

```
<head>
  <title>{{title}}</title>
</head>
```

- Incluir la plantilla head en las otras plantillas
- Mover todas las plantillas a la carpeta templates

3. Aplicar algunos estilos

- Aplicar estilos al elemento p (hoja de estilos en línea: inline style sheet)

```
font-size: 10pt;
line-height: 12pt;
```

- Aplicar estilos al elemento table (hoja de estilos en línea: inline style sheet)

```
font-size: 10pt;
line-height: 12pt;
border-collapse: collapse;
border: none;
width: 100%;
```

		ASIGNATURA tecnologías para la programación y el diseño web i	CURSO 2	GRUPO 01
CALIFICACIÓN	EVALUACIÓN	APELLIDOS	NOMBRE	ID
		OBSERVACIONES Ejercicio 3: sistema de plantillas y estilos	FECHA 30/03/2016	FECHA ENTREGA 22/04/2016

3.1. Crear el elemento style en la sección head del documento html (hoja de estilos interna: internal style sheet)

- Mover las reglas CSS de los atributos style de los elementos al elemento style creado
- Aplicar estilos al elemento body

```
font-size: 10pt;
line-height: 12pt;
```

3.2. Crear un fichero CSS hoja de estilos externo (external style sheet CSS file)

- Mover las reglas CSS desde el elemento style al fichero CSS
- Crear ruta y función para crgar ficheros CSS estáticos

```
@route('/css/:filename', name='css')
def server_static(filename):
    return static_file(filename, root='./css')
```

3.3. Crear asombrosas reglas CSS

- Usar un font web en la aplicación
- Cargar la font como un recurso externo

```
<link href='http://fonts.googleapis.com/css?family=Sintony:400,700' rel='stylesheet' type='text/css'>
```

- Aplicar ese font en las reglas CSS

```
font-family: 'Sintony', sans-serif;
```

- Usar pseudo-clases CSS

```
a:hover{
  font-weight: bold;
}
```

- El camino a la interacción...

Preguntas de repaso sobre protocolos de red, desarrollo básico de aplicaciones web, sistemas de plantillas y aplicación de estilos

1. Acceder a la dirección `http://localhost:8080/help` y comprobar todo el trabajo realizado en la estructura de archivos de la aplicación (carpetas utilizadas por la aplicación) y el propio fichero de aplicación (todo.py). Observar la complejidad innecesaria del HTML en el fichero de aplicación (todo.py) del ejercicio 2 anteriormente realizado y compararlo con el resultado simplificado al delegar la generación de HTML completamente a las plantillas. ¿Cuáles son las ventajas, en cuanto a facilidad de desarrollo, de utilizar adecudamente el sistema de plantillas que ofrece el framework Bottle?

2. Como se ha visto en las sesiones teóricas, existen tres (3) formas de proporcionar información sobre los estilos a un navegador. Después de probar las dos primeras (inline style sheet y internal style sheet) tal como se describe en la página de ayuda de nuestra aplicación o en esta guía, intenta enumerar algunos beneficios e inconvenientes para cada método.

3. Se pide escribir un pequeño informe en el que se permita comprobar cómo se realiza la conversación entre el cliente y el servidor (ciclo petición-respuesta) en la aplicación democssweb utilizando el inspector web. Para ver el menú de desarrollo en Safari puede ser necesario activarlo desde la pestaña de configuración avanzada en el menú preferencias de la aplicación:

		ASIGNATURA tecnologías para la programación y el diseño web i	CURSO 2	GRUPO 01
CALIFICACIÓN	EVALUACIÓN	APELLIDOS	NOMBRE	ID
		OBSERVACIONES Ejercicio 3: sistema de plantillas y estilos	FECHA 30/03/2016	FECHA ENTREGA 22/04/2016

    1. Los tiempos de carga y tamaño de la respuesta enviada por el servidor
    2. Los encabezados HTTP de la petición
    3. Los encabezados HTTP de la respuesta
    4. Información de los recursos adicionales transferidos para completar la petición.

4. Comparar este informe con los resultados ofrecidos por el framework Bottle y el inspector web cuando se ejecuta la aplicación `democssweb` si se asigna la hoja de estilos `styles.css` y se configura una fuente web (`Sintony`) para su utilización. Para ello es necesario añadir el elemento link que hace referencia al fichero styles.css y a la fuente web, ambos en la plantilla que genera la sección head del documento (`head.tpl`). El contenido final de esta plantilla puede encontrarse en esta guía.
5. Tras acceder a la dirección `http://localhost:8080/todo` intente analizar la respuesta que se muestra en pantalla utilizando el inspector web. ¿Cuáles son las diferencias entre la respuesta enviada por el servidor y la respuesta (árbol DOM) que construye el navegador?
6. Intenta explicar el comportamiento/apariencia de los elementos de table de acuerdo a las reglas CSS que se muestran en el fichero style.css.
7. Intenta explicar el comportamiento/apariencia del elemento body (background) de acuerdo a las reglas CSS proporcionadas en el fichero style.css.
8. A continuación se muestran algunas tareas en las que el alumno puede trabajar por su cuenta y que podrá incluir en la entrega en el paquete comprimido (ZIP):
    1. Intente crear diferentes estilos y defina las reglas CSS necesarias en su propio fichero de estilos `mystyles.css` que vinculará en la plantilla `head.tpl` a continuación del fichero de estilos (`styles.css`) proporcionado
    2. Intente crear diferentes ficheros de estilos externos para componer la apariencia en pantalla en función de diferentes resoluciones o dispositivos utilizados para presentar la aplicación, utilizando el atributo `media` en el elemento `link`.