Templates

Antonio Espín Herranz

Contenidos

- Utilización de plantillas.
- Etiquetas.
- Uso de las plantillas en las vistas.
- Herencia entre plantillas.

Plantillas

- Es la forma eficiente de trabajar con las vistas.
- En vez de volcar directamente el HTML en la vista utilizamos una plantilla con el fin de separar, por un lado, el código Python y por otro la presentación.
- Cuando está todo junto implica tocar el código Python cuando cambia algo de la presentación (HTML).
- Además, esta separación también permite separar la parte de programación de la parte de presentación.
  - Habitualmente perfiles distintos dentro del mundo del desarrollo software.

Plantillas

- Son **documentos genéricos** con una vista similar a HTML donde vamos a inyectar valores en **variables** (a través de un **Contexto**) y podremos utilizar **cierta lógica de programación** mediante el uso de **etiquetas**.

Ejemplo

```
<html>
<head><title>Orden de pedido</title></head>
<body>

<h1>Orden de pedido</h1>

<p>Estimado: {{ nombre }},</p>

<p>Gracias por el pedido que ordeno de la {{ empresa }}.
El pedido junto con la mercancía se enviaran el
{{ fecha|date:"F j, Y" }}.</p>

<p>Esta es la lista de productos que usted ordeno:</p>

<ul>
{% for pedido in lista_pedido %}
   <li>{{ pedido }}</li>
{% endfor %}
</ul>

{% if garantía %}
   <p>La garantía será incluida en el paquete.</p>
   {% else %}
      <p>Lamentablemente no ordeno una garantía, por lo
      que los daños al producto corren por su cuenta.</p>
{% endif %}

<p>Sinceramente <br /> {{ empresa }}</p>
</body>
</html>
```

Dentro de la plantilla:
{{nombre}} → **Variable**
{{fecha|date ...}} → **Filtros**
{%for .... %}
{%endfor%} → **Etiqueta**

Las variables se rellenan
Con un **Contexto** que se
Puede inicializar con un
Diccionario de Python
Y se pasa desde la **vista**.

Plantillas

- Dentro del módulo: **django.template** disponemos de la clase **Template** y **Context**

- **Template** tiene un constructor donde recibe el código de la plantilla.

- Estas se pueden probar desde una **consola**.

  python manage.py shell

```
In [1]: from django import template
In [2]: t=template.Template('Hola {{nombre}}')
In [3]: c = template.Context({'nombre':'Adrian'})
In [4]: print(t.render(c))
Hola Adrian
```

- El método **render**, sustituye el valor de la variable nombre con el contexto.

Plantillas

- La misma plantilla se puede utilizar con distintos contextos siempre y cuando esos contextos tengan las variables asociadas a la plantilla.

```
from django.template import Template, Context
t = Template('Hola, {{ nombre }}')
print (t.render(Context({'nombre': 'Juan'})))
Hola, Juan
print (t.render(Context({'nombre': 'Julia'})))
Hola, Julia
print (t.render(Context({'nombre': 'Paty'})))
Hola, Paty
```

Plantilla

- Dentro de un contexto se puede añadir datos simples o estructuras más complejas.
- Dentro de una estructura más compleja (por ejemplo, una **clase**, un **diccionario**) ***el acceso a los campos será a través del punto***.

```
from django.template import Template, Context
persona = {'nombre': 'Silvia', 'edad': '43'}
t = Template('{{ persona.nombre }} tiene {{ persona.edad }} años.')
c = Context({'persona': persona})
t.render(c)
'Silvia tiene 43 años.'
```

Plantilla

- En las plantillas también es posible llamar a métodos, pero ojo, tienen que ser ***métodos que no tengan parámetros y la llamada se realiza sin paréntesis***.

```
from django.template import Template, Context
t = Template('{{ var }} {{var.upper }}– {{ var.isdigit }}')
t.render(Context({'var': 'hola'}))
'hola HOLA False
 t.render(Context({'var': '123'}))
123 123 True
```

Plantillas

- También se pueden utilizar listas:

```
from django.template import Template, Context
t = Template('Fruta 2 es {{ frutas.2 }}.')
c = Context({'frutas': ['manzana', 'plátano', 'pera']})
 t.render(c)
'Fruta 2 es pera.'
```

En Resumen

- **En las plantillas se pueden utilizar**:
  - Variables simples.
  - Clases
  - Diccionarios
  - Listas
  - Y llamadas a métodos (*que no tengan parámetros*)

- En todos estos casos se utiliza el punto, incluso en las listas, indicando la posición.

- El contexto se inicializará con un diccionario.
- Donde el valor de la clave puede ser una lista, un objeto …
- Para estructuras anidadas utilizar el punto hasta llegar al campo.

Errores

- Por ejemplo, si dentro de una plantilla se hace referencia a **una variable que no se encuentra en el contexto** *no se produce un error*, se rellena con un blanco.

Etiquetas

- Django proporciona un conjunto de etiquetas que se pueden utilizar dentro de las plantillas. Las etiquetas se encierran entre **{% … %}**
- Dentro de la etiqueta **if** se pueden utilizar los siguientes operadores:

  ==, !=, <, >, <=, >=, in (para una subcadena, lista, conjunto, queryset) , not in

- **if / else**

```
{% if es_fin_de_semana %}
<p>¡Bienvenido fin de semana!</p>
{% endif %}
```

- La etiqueta {% else %} es opcional:

```
{% if es_fin_de_semana %}
<p>¡Bienvenido fin de semana!</p>
{% else %}
<p>De vuelta al trabajo.</p>
{% endif %}
```

- Dentro de la etiqueta **if** se pueden utilizar **and** / **or / not.**
- Dentro de la **misma sentencia no puede haber and y or**.
- No se pueden utilizar paréntesis en las condiciones.

EJEMPLOS

```
{% if lista_atletas and lista_entrenadores %}
Atletas y Entrenadores están disponibles
{% endif %}

{% if not lista_atletas %}
No hay atletas
{% endif %}

{% if lista_atletas or lista_entrenadores %}
Hay algunos atletas o algunos entrenadores
{% endif %}

{% if not lista_atletas or lista_entrenadores %}
No hay atletas o no hay entrenadores.
{% endif %}

{% if lista_atletas and not lista_entrenadores %}
Hay algunos atletas y absolutamente ningún entrenador.
{% endif %}
```

Etiquetas

- Etiqueta **for**: para **iterar** por las **colecciones**.

```
<ul>
    {% for atleta in lista_atletas %}
        <li>{{ atleta.nombre }}</li>
    {% endfor %}
</ul>
```

- Se puede iterar al revés:

```
{% for atleta in lista_atletas reversed %}
...
{% endfor %}
```

Etiquetas

- La etiqueta **for** se puede anidar:
- Relaciones 1 → N

```
{% for pais in paises %}
    <h1>{{ pais.nombre }}</h1>
    <ul>
        {% for ciudad in pais.lista_ciudades %}
            <li>{{ ciudad }}</li>
        {% endfor %}
    </ul>
{% endfor %}
```

Bucles anidados

- Si dentro del modelo tenemos una relación de muchos a muchos, por ejemplo:

```
class Autor(models.Model):
    nombre = models.CharField(…)
    …

class Libro(models.Model):
    titulo = models.CharField()
    autores = models.ManyToManyField(Autor)
```

Bucles Anidados II

- En una plantilla, pintamos una lista de libros y dentro de cada libro se pintan los autores de cada libro en una sublista.

```
<ul>
{% for libro in libros %}
    <li>{{libro.titulo}}</li>
    <li>
        <ul>
            {% for autor in libro.autores.all %}
                <li>{{autor.nombre}}</li>
            {% endfor %}
        </ul>
    </li>
{% endfor %}
</ul>
```

Etiquetas

- Para **comprobar si la colección tiene o no elementos**, podemos hacer:

```
{% if lista_atletas %}
    {% for atleta in lista_atletas %}
        <p>{{ atleta.nombre }}</p>
    {% endfor %}
{% else %}
    <p>No hay atletas. Únicamente programadores.</p>
{% endif %}
```

- Pero django proporciona **empty** para tal cometido:

```
{% for atleta in lista_atletas %}
    <p>{{ athlete.nombre }}</p>
{% empty %}
    <p>No hay atletas. Únicamente programadores.</p>
{% endfor %}
```

Etiquetas

- **Dentro** del bucle **for** se puede utilizar la **variable**: **forloop** que proporciona información en cada vuelta del bucle.
  - **forloop.counter**: representa el índice, empieza en 1
  - **forloop.counter0**: igual que la anterior, pero empieza en 0.
  - **forloop.revcounter**: representa el índice, descendente hasta 1
  - **forloop.revcounter0**: igual que la anterior, pero hasta 0.

```
{% for objeto in lista %}
    <p>{{ forloop.counter }}: {{ objeto }}</p>
{% endfor %}
```

Etiquetas

- **forloop.first:**
  - Es un valor booleano asignado a **True si es la primera vez que se pasa por el bucle**.
  - Cuando queremos hacer algo distinto en la 1ª iteración.

- **forloop.last:**
  - Es un valor booleano asignado a **True si es la última pasada por el bucle**. Un uso común es para esto es poner un carácter pipe entre una lista de enlaces:

```
{% for enlace in enlaces %} {{ enlace }}
    {% if not forloop.last %} | {% endif %}
{% endfor %}
```

- Se evita colocar la separación después del último elemento.

Enlace1 | Enlace2 | Enlace3 | Enlace4

Etiquetas

- **forloop.parentloop**
  - Es para bucles anidados y hace referencia al bucle padre.

```
{% for pais in paises %}
    <table>
    {% for ciudad in pais.lista_ciudades %}
        <tr>
        <td>pais #{{ forloop.parentloop.counter  }}</td>
        <td>City #{{ forloop.counter  }}</td>
        <td>{{ ciudad }}</td>
        </tr>
    {% endfor %}
    </table>
{% endfor %}
```

Resumen: forloop

Variable	Descripción
forloop.counter	El número de vuelta o iteración actual (usando un índice basado en 1).
forloop.counter0	El número de vuelta o iteración actual (usando un índice basado en 0).
forloop.revcounter	El número de vuelta o iteración contando desde el fin del bucle (usando un índice basado en 1).
forloop.revcounter0	El número de vuelta o iteración contando desde el fin del bucle (usando un índice basado en 0).
forloop.first	True si es la primera iteración.
forloop.last	True si es la última iteración.
forloop.parentloop	Para bucles anidados, es una referencia al bucle externo.

Etiquetas

- **ifequal/ifnotequal**
  - Para realizar comparaciones entre dos valores:

```
{% ifequal usuario actual_usuario %}
    <h1>¡Bienvenido!</h1>
{% endifequal %}

{% ifequal seccion 'noticias' %}
    <h1>Noticias</h1>
{% endifequal %}

{% ifequal seccion "comunidad" %}
    <h1>Comunidad</h1>
{% endifequal %
```

Se puede comparar
Contra cadenas o vars.
También soporta else

Etiquetas

{% **ifequal** seccion 'noticias' %}

    <h1>Noticias</h1>

{% **else** %}

    <h1>No hay noticias nuevas</h1>

{% **endifequal** %}

- *ifequal solo soporta entero, número reales y cadenas pero no listas o diccionarios.*

Etiquetas

- **now**
  - {% now "jS F Y H:i" %}
  - Fecha y hora.

- **url**
  - Devuelve una **URL** absoluta. Sin incluir la parte del dominio que coincide con una determinada vista.
  - **{% url 'algunnombredeurl‘ v1 v2 %}**

Ejemplo

- En la **template** tenemos:
  - Inicio
  - Administración
  - Facturación

- En **urls.py**:
  - path('admin/', admin.site.urls),
  - path('', dpii.views.index, name='**index**'),
  - path('facturas/', dpii.views.vistaFacturas, name='**facturas**'),

Etiquetas

- Comentarios:
  - De una línea:

    {# Esto es un comentario #}

  - De un bloque:

    {% **comment** %}

    Este es un comentario

    que abarca varias líneas

    {% **endcomment** %}

Etiquetas

- Para definir variables tenemos la etiqueta with.

  {% with ‘valor‘ as var %}

  ….

  {% endwith %}

Filtros

- **Filtros**:
  - Se aplican a las variables de las plantillas.
  - Se utiliza un pipe como en Linux.
  - Se puede encadenar más de 1 filtro.

{{nombre**|lower**}}

- Convierte la variable nombre a minúsculas.

{{ mi_lista**|first|upper** }}

- Toma el primer elemento de la lista y lo encadena con el siguiente filtro para convertirlo a mayúsculas.

{{valor**|floatformat:"2"**}}

- Para formatear a 2 decimales.

{% with 'valor' **|add:otra_var** as var %}

- **add** concatena si son cadenas, si son números los suma.

{% endwith %}

{{valor|escape}}

- Para escapar el contenido de HTML, por ejemplo: **< <**

Filtros

- {{ fecha|date:"F j, Y" }}
  - Formatea la fecha. Los formatos en la siguiente página.

- Para **ordenar**:
- {{ valor |**dictsort**:"nombre" }}

  - Si el valor es:
  - [{'nombre': 'zed', 'edad': 19},
  - {'nombre': 'amy', 'edad': 22},
  - {'nombre': 'joe', 'edad': 31},]

  - La salida será:
  - [{'nombre': 'amy', 'edad': 22},
  - {'nombre': 'joe', 'edad': 31},
  - {'nombre': 'zed', 'edad': 19},]

Para ordenar **DESC**
{{ valor |**dictsortreversed**:"nombre" }}

Formato	Descripción	Salida
a	'a.m.' o 'p.m.'.	'a.m.' o 'p.m.
A	'AM' o 'PM'.	'AM'
b	El nombre del mes, en forma de abreviatura de tres letras minúsculas.	'jan'
d	Día del mes, dos dígitos que incluyen rellenando con cero por la izquierda si fuera necesario.	'01' a '31'
D	Día de la semana, en forma de abreviatura de tres letras.	'Fri'
f	La hora, en formato de 12 horas y minutos, omitiendo los minutos si estos son cero.	'1', '1:30'
F	El mes, en forma de texto	'January'
g	La hora, en formato de 12 horas, sin rellenar por la izquierda con ceros.	'1' a '12'
G	La hora, en formato de 24 horas, sin rellenar por la izquierda con ceros.	'0' a '23'
h	La hora, en formato de 12 horas.	'01' a '12'
H	La hora, en formato de 24 horas.	'00' a '23'
i	Minutos.	'00' a '59'
j	El día del mes, sin rellenar por la izquierda con ceros.	'1' a '31'
l	El nombre del día de la semana.	'Friday'
L	Booleano que indica si el año es bisiesto.	True o False
m	El día del mes, rellenando por la izquierda con ceros si fuera necesario.	'01' a '12'
M	Nombre del mes, abreviado en forma de abreviatura de tres letras.	'Jan'
n	El mes, sin rellenar con ceros	'1' a '12'
N	La abreviatura del mes siguiendo el estilo de la Associated Press.	'Jan.', 'Feb.' , 'March', 'May'
O	Diferencia con respecto al tiempo medio de Grennwich (*Greenwich Mean Time* - GMT)	'+0200'
P	La hora, en formato de 12 horas, más los minutos, recto si estos son cero y con la indicación a.m./p.m.	'1 a.m.', '1:30 p.m.' , 'midnight',

	Además, se usarán las cadenas de texto especiales 'midnight' y 'noon' para la medianoche y el mediodía respectivamente.	'noon' , '12:30 p.m.'
r	La fecha en formato RFC 822.	'Thu, 21 Dec 2000 16:01:07 +0200'
s	Los segundos, rellenos con ceros por la izquierda de ser necesario.	'00' a '59'
S	El sufijo inglés para el día del mes (dos caracteres).	'st', 'nd', 'rd' o 'th'
t	Número de días del mes.	28 a 31
T	Zona horaria	'EST', 'MDT'
w	Día de la semana, en forma de dígito.	'0' (Domingo) a '6' (Sábado)
W	Semana del año, siguiente la norma ISO-8601, con la semana empezando el lunes.	1, 23
y	Año, con dos dígitos.	'99'
Y	Año, con cuatro dígitos.	'1999'
z	Día del año	0 a 365
Z	Desfase de la zona horaria, en segundos. El desplazamiento siempre es negativo para las zonas al oeste del meridiano de Greenwich, y positivo para las zonas que están al este.	-43200 a 43200

Uso de plantillas en las vistas

- Al utilizar las plantillas podemos separar por un lado la presentación del código de Python.
- La solución más optima es guardar las plantillas en ficheros independientes.
- Django proporciona plantillas predefinidas que podemos adaptar a nuestro sitio Web.
- En las plantillas podremos aplicar herencia para evitar duplicidad en el código.

Plantillas

- En una vista podríamos hacer:

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

def fecha_actual(request):
    ahora = datetime.datetime.now()
    # Manera simple de usar plantillas del sistema de archivos.
    # Esto es malo, porque no toma en cuenta los archivos no encontrados.
    fp = open('/home/djangouser/templates/miplantilla.html')
    t = Template(fp.read())
    fp.close()
    html = t.render(Context({'fecha_actual': ahora}))
    return HttpResponse(html)
```

- Esta forma no es óptima, hay que estar gestionando el fichero → **para ello tenemos cargadores de plantillas**.

Plantillas

- Cuando arrancamos el servidor y entramos en http://localhost:8000 se muestra por defecto la siguiente página:

Plantillas
Sustituir la página principal de Django

- Crear en la carpeta de la **aplicación**: una carpeta llamada **templates** para almacenar las plantillas.
  - Puede haber otra carpeta **templates** a nivel de proyecto, pero sería para el panel de administración.

- En el fichero de **settings.py** indicar la ruta de las carpetas de las plantillas (*uso del cargador de plantillas*).

- Crear una plantilla, mejor utilizar herencia para tener una plantilla padre con las cosas comunes. **index.html**

- En **urls.py** crear un mapeo:
  - **path(‘’, miapp.views.index, name='index’),**

- En **views.py** crear una vista que renderize la plantilla:

```
def index(request):
    return render(request, “index.html”, contexto)
```

Plantillas

- **Cargadores de plantillas**:
  - Necesitan tener una ubicación de donde se encuentran las plantillas.
  - En el fichero de **settings.py** al crear el proyecto tendremos:

```
TEMPLATES = [{
'BACKEND': 'django.template.backends.django.DjangoTemplates',
'DIRS': [],   → AQUÍ SE INDICAN LAS RUTAS.
'APP_DIRS': True,
'OPTIONS': {
'context_processors': [
'django.template.context_processors.debug',
'django.template.context_processors.request',
'django.contrib.auth.context_processors.auth',
'django.contrib.messages.context_processors.messages',
],},},]
```

Plantillas

- La ruta se suele indicar a partir a partir de una variable, previamente inicializada (***settings.py***)

  BASE_DIR = os.path.dirname(

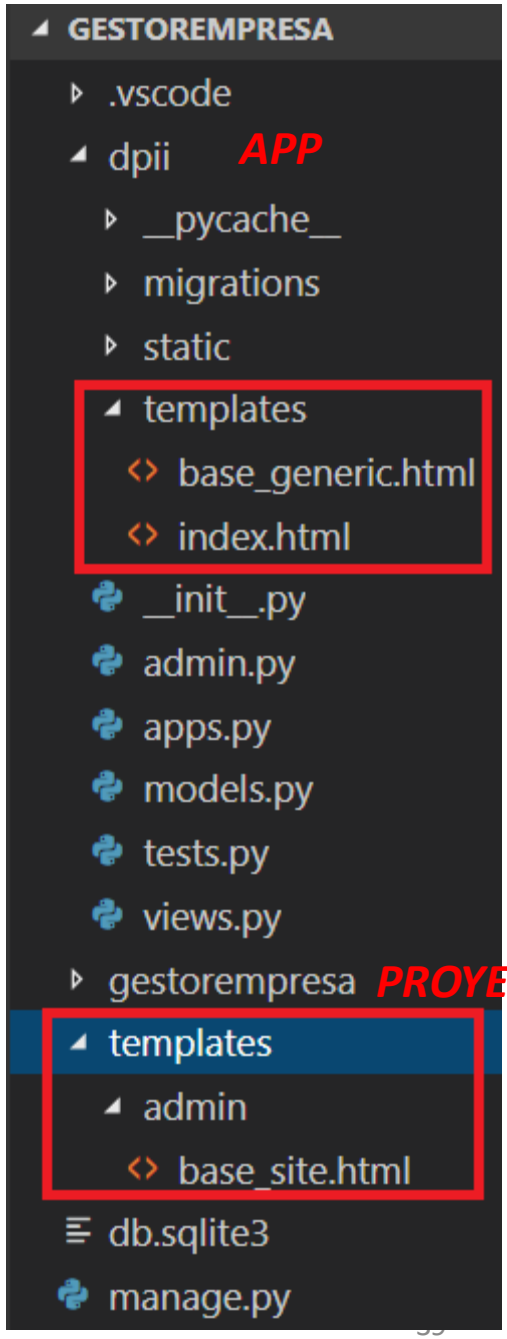
  os.path.dirname(os.path.abspath(__file__)))

- Tener en cuenta que podemos tener plantillas a nivel de proyecto → (***lo utiliza el panel de administración***).

- Y a nivel de **aplicación para nuestras vistas**.

- Ambas se guardan en una carpeta **templates**.

- **'DIRS'**: [os.path.join(BASE_DIR, **'templates'**)],



Plantillas

- Dentro del fichero: **settings.py** localizar la variable **TEMPLATES**:

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')],
        'APP_DIRS': True,
```

Render

- Utilizando **render**() no es necesario cargar la plantilla previamente.

- **La vista se reduce bastante**:

```
from django.shortcuts import render

def fecha_actual(request):
    ahora = datetime.datetime.now()
    return render(request, 'fecha_actual.html',
      {'fecha_actual': ahora})
```

Render II

- La llamada a **render** también se puede hacer **con 2 parámetros** (si no necesitamos pasar información de la vista a la plantilla).
  - Por ejemplo, para **templates** que solo muestran contenido estático:

- def galeria(request):
  - return render(request, 'galeria.html')

Plantillas

- Básicamente en una **vista** haremos:
  - **Recuperar datos** del **modelo**.
  - **Cargar** un **contexto (*un diccionario*)** con toda la información que necesitamos presentar.
  - Llamar a **render** indicando el nombre de la plantilla y los valores del contexto.

- Si tenemos muchas plantillas habrá que **organizarlas en directorios**, que cuelguen de templates y esto se indicará a render:
  - return **render**(request, '**horas/**fecha_actual.html', {'fecha_actual': ahora})

include

- Es una posibilidad que ofrecen las plantillas para reutilizar código.
- Consiste en incrustar una plantilla dentro de otra plantilla.
- Ahorramos código (copy / paste).
- {% include template_name %}
  - Se utilizan comillas simples o doble y pueden venir de otro directorio.
  - **{% include 'includes/nav.html' %}**

Ejemplo

mipagina.html

```
<html>
<body>
 {% include "includes/nav.html" %}
    <h1>{{ titulo }}</h1>
</body>
</html>
```

includes/nav.html

```
<div id="nav">
    Tu estas en: {{ seccion_actual }}
</div>
```

Herencia entre plantillas

- **Es la mejor estrategia** para organizar el código.

- La herencia de plantillas permite construir una **plantilla base** ‘‘**esqueleto**’’ que **contenga** todas las **partes comunes** del sitio y **definir** ‘‘**bloques**’’ que las plantillas hijas puedan **sobrescribir**.

Ejemplo: La plantilla Base

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>
    {% block title %}{% endblock %}
</title>
</head>
<body>
<h1>Mi sitio Web</h1>
{% block content %}{% endblock %}
{% block footer %}
    <hr>
    <p>Gracias por visitar nuestro sitio web.</p>
{% endblock %}
</body>
</html>
```

Se definen **3 bloques** que se pueden Sustituir en las plantillas:

- **title**
- **content**
- **footer**

Este fichero se puede almacenar en:
.. app/templates/base.html

Ejemplo: La plantilla Hija

```
{% extends "base.html" %}
{% block title %}La fecha actual{% endblock %}
{% block content %}
    <p>Hoy es: {{ fecha_actual }}</p>
{% endblock %}
```

- El bloque **footer** tiene la opción de sobrescribirlo, si no lo hace, toma el contenido de la plantilla base.

- *La estructura del HTML se hereda de la plantilla padre.*
  - *En esta también se podrá añadir contenido estático como CSS / JavaScript.*
  - *Enlaces a frameWorks de diseño deberían ir en la plantilla Padre. Por ejemplo:* ***bootstrap***

Ejemplo: Plantilla Base - **Head**

```
<!DOCTYPE html>
<html lang="es">
<head>
{% block title %}<title>TITULO DE LA PAGINA</title>{% endblock %}

<!-- Etiquetas Meta: encoding / viewport -->
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1">

<!-- Referencias a ficheros externos CSS / JS -->
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>

<!-- Contenido estático de CSS -->
{% load static %}
<link rel="stylesheet" href="{% static 'css/estilos.css' %}">

</head>
```

Niveles de herencia

- La herencia **no afecta** al comportamiento del **contexto**.

- Se aconseja para un **sitio Web tener 3 niveles** de herencia.
  - Una **plantilla base.html** que contenga el aspecto principal del sitio.
  - Una **plantilla base_SECTION.html** para cada ‘‘**sección**’’ de tu sitio (por ej. base_fotos.html y base_foro.html).
    - Estas plantillas **heredan de base.html** e incluyen secciones específicas de estilo y diseño.
  - Una **plantilla individual** para cada tipo de página, tales como páginas de formulario o galería de fotos. Estas plantillas heredan de la plantilla solo la sección apropiada

Niveles de herencia II

- Al menos tener 2 niveles de herencia.
- En la plantilla BASE se define la estructura principal con los enlaces a la CSS, el menú (se puede definir un menú base y si este cambia en la plantilla hija se puede sustituir).
- Hay que definir todas las posibles partes que puedan cambiar cuando vayamos a lanzar la plantilla hija.

Tener en cuenta

- La etiqueta {% **extends** ‘ …’ %} tiene que ser la **primera etiqueta** de la plantilla (si se utiliza).
  - El argumento podría ser una variable.

- Las **plantillas hija no están obligadas a rellenar todos los bloques** de la plantilla padre (esos bloques pueden tener un contenido por defecto).

- **En la misma plantilla NO puede haber más de 1 bloque con el mismo nombre**.

- Si tenemos **dos plantillas con código repetido** … este código debería estar en la **plantilla padre**.

- Se puede acceder al contenido de la plantilla padre para **AÑADIR** más contenido **SIN SOBRESCRIBIR**. Para ello se dispone de la variable **{{block.super}}**

Ejemplo

 # Template: A.html

```
<html>
<head></head>
<body>
{% block hello %} HELLO {% endblock %}
</body>
</html>
```

 # Template B.html

```
{% extends "A.html" %}
{% block hello %}World{% endblock %}
```

Renderiza Template B: **SOBRESCRIBE**

```
<html>
<head></head>
<body>World </body>
</html>
```

 # Template C

```
{% extends "A.html" %}
{% block hello %}{{ block.super }}
World{% endblock %}
```

 # Renderiza Template C: AÑADE

```
<html> <head></head>
<body>HELLO  World </body>
</html>
```