

Contenidos a Trabajar

1. Etiquetas de plantillas incorporadas
2. Filtros

Etiquetas de plantilla incorporados

En esta sección veremos algunas de las etiquetas más utilizadas en Django sin contar las utilizadas para herencia. Para mayor referencia por favor diríjase al sitio oficial de Django (referencia al final) o a alguno de los libros sugeridos:

autoescape

Controla el comportamiento actual del auto-escape. Esta etiqueta toma como argumento tanto a: on y off y determina si el auto-escapeo están dentro del bloque. Cuando el auto-escapeo está activado, todas las variables contenidas que contenga HTML serán escapadas antes de mostrar el resultado de la salida (pero después de que cualquier filtro se haya aplicado). Esto es equivalente a manualmente aplicar el filtro escape a cada variable. La única excepción son las variables que están marcadas como “safe” para autoescape, ya sea por la clave que pobló la variable, o porque se ha aplicado el filtro safe o escape.

Forma de usarlo:

```
{% autoescape on %}  
{{ body }}  
{% endautoescape %}
```

comment

Ignora todo lo que aparece entre {% comment %} y {% endcomment %}. Como nota opcional, se puede insertar la primera etiqueta. Por ejemplo, es útil para comentar fuera del código para documentar, porqué el código fue deshabilitado.

Ejemplo de su uso:

```
<p>Renderizar texto con {{ fecha_publicacion|date:"c" }}</p>  
{% comment "Nota opcional" %}  
<p>Comentado fuera del texto {{ creado|date:"c" }}</p>  
{% endcomment %}
```

csrf_token

Esta etiqueta es usada para protección CSRF. Para más información sobre esta protección visite el sitio de Django que le dejamos a continuación:

<https://docs.djangoproject.com/en/3.2/ref/csrf/>

cycle

Rota una cadena de texto entre diferentes valores, cada vez que aparece la etiqueta. Dentro de un bucle, el valor rota entre los distintos valores disponibles en cada iteración del bucle:

```
{% for o in some_list %}  
<tr class="{% cycle row1,row2 %}">  
...  
</tr>  
{% endfor %}
```

Fuera de un bucle, hay que asignar un nombre único la primera vez que se usa la etiqueta, y luego hay que incluirlo ese nombre en las sucesivas llamadas:

```
<tr class="{% cycle row1,row2,row3 as rowcolors %}">...</tr>  
<tr class="{% cycle rowcolors %}">...</tr>  
<tr class="{% cycle rowcolors %}">...</tr>
```

filter

Filtra el contenido de una variable. Los filtros pueden ser encadenados sucesivamente (La salida de uno es la entrada del siguiente), y pueden tener argumentos, como en la sintaxis para variables.

He aquí un ejemplo:

```
{% filter escape|lower %}
```

Este texto será escapado y aparecerá en minúsculas

```
{% endfilter %}
```


firstof

Presenta como salida la primera de las variables que se le pasen que evalúe como no falsa. La salida será nula si todas las variables pasadas valen False.

He aquí un ejemplo:

```
{% firstof var1 var2 var3 %}  
Equivale a:  
{% if var1 %}  
  {{ var1 }}  
{% else %} {% if var2 %}  
  {{ var2 }}  
{% else %} {% if var3 %}  
  {{ var3 }}  
{% endif %} {% endif %} {% endif %}
```

for

Itera sobre cada uno de los elementos de un lista o array. Por ejemplo, para mostrar una lista de libros, cuyos títulos estén en la lista_libros, podríamos hacer esto:

```
<ul>  
{% for libro in lista_libros %}  
<li>{{ libro.titulo }}</li>  
{% endfor %}  
</ul>
```

También se puede iterar la lista en orden inverso usando {% for obj in list reversed %}.

Se pueden usar cualquier número de valores, separándolos por comas. Asegúrate de no poner espacios entre los valores, sólo comas.

Dentro de un bucle, la propia sentencia for crea una serie de variables. A estas variables se puede acceder únicamente dentro del bucle:

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.

for ... empty

La etiqueta for toma una cláusula opcional `{% empty %}` cuando el texto es mostrado, si el array esta vacío o no puede ser encontrado.

```
<ul>
{% for atleta in lista_atletas %}
<li>{{ atleta.nombre }}</li>
{% empty %}
<li>Lo sentimos, no hay atletas en esta lista.</li>
{% endfor %}
</ul>
```

El ejemplo anterior es equivalente a (pero más corto, limpio y posiblemente más rápido) a lo siguiente:

```
<ul>
{% if lista_atletas %}
{% for atleta in lista_atletas %}
<li>{{ atleta.nombre }}</li>
{% endfor %}
{% else %}
<li>Lo sentimos, no hay atletas en esta lista.</li>
{% endif %}
</ul>
```

If

La etiqueta `{% if %}` evalúa una variable. Si dicha variable se evalúa como una expresión “verdadera” (Es decir, que el valor exista, no esté vacía y no es el valor booleano False), se muestra el contenido del bloque:

```
{% if lista_atletas %}  
Número de atletas: {{ lista_atletas|length }}  
{% else %}  
No hay atletas.  
{% endif %}
```

Si la lista `lista_atletas` no está vacía, podemos mostrar el número de atletas con la expresión `{{ lista_atletas|length }}` Además, como se puede ver en el ejemplo, la etiqueta `if` puede tener un bloque opcional

`{% else %}` que se mostrará en el caso de que la evaluación de falso.

Operadores booleanos

Las etiquetas `if` pueden usar operadores lógicos como `and`, `or` y `not` para evaluar expresiones más complejas:

```
{% if lista_atletas and lista_entrenadores %}
```

Los atletas y los 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 hay algunos entrenadores

```
{% endif %}
```

```
{% if lista_atletas and not lista_entrenadores %}
```

Hay algunos atletas y absolutamente ningún entrenador.

```
{% endif %}
```


La etiqueta if no admite, sin embargo, mezclar los operadores and y or dentro de la misma comprobación, porque la orden de aplicación de los operadores lógicos sería ambigua. Por ejemplo, el siguiente código es inválido:

```
{% if lista_atletas and lista_entrenadores or lista_animadoras %}
```

Para combinar operadores and y or, puedes usar sentencias if anidadas, como en el siguiente ejemplo:

```
{% if lista_atletas %}  
{% if lista_entrenadores or lista_animadoras %}  
¡Tenemos atletas, y ya sea entrenadores o porristas!  
{% endif %}  
{% endif %}
```

Es perfectamente posible usar varias veces un operador lógico, siempre que sea el mismo siempre. Por ejemplo, el siguiente código es válido:

```
{% if lista_atletas or lista_entrenadores or lista_animadoras or lista_profesores %}
```

Filtros

Puedes usar filtros en las expresiones if.

Por ejemplo:

```
{% if messages|length >= 100 %}  
¡Hoy tienes montones de mensajes!  
{% endif %}
```

include

Carga una plantilla y la representa usando el contexto actual. Es una forma de “incluir” una plantilla dentro de otra.

El nombre de la plantilla puede o bien ser el valor de una variable o estar escrita en forma de cadena de texto, rodeada ya sea con comillas simples o comillas dobles, a gusto del lector.

El siguiente ejemplo incluye el contenido de la plantilla “foo/bar.html”:

```
{% include “foo/bar.html” %}
```

Este otro ejemplo incluye el contenido de la plantilla cuyo nombre sea el valor de la variable `template_name`:

```
{% include template_name %}
```

lorem

Muestra en orden aleatorio el texto en Latin “lorem ipsum”. Esto puede ser útil para proveer datos en las plantillas.

Uso:

```
{% lorem [count] [method] [random] %}
```

now

Muestra la fecha, escrita de acuerdo a un formato indicado. Esta etiqueta fue inspirada por la función `date()` de PHP(), y utiliza el mismo formato que esta (<http://php.net/date>). La versión Django tiene, sin embargo, algunos extras.

He aquí un ejemplo:

```
Es el {% now “jS F Y H:i” %}
```


Se pueden escapar los caracteres de formato con una barra invertida, si se quieren incluir de forma literal. En el siguiente ejemplo, se escapa el significado de la letra “f” con la barra invertida, ya que de otra manera se interpretaría como una indicación de incluir la hora. La “o”, por otro lado, no necesita ser escapada, ya que no es un carácter de formato:

Es el `{% now "jS o¥f F" %}`

El ejemplo mostraría: ‘ ‘Es el 4th of September’ ’ .

url

Devuelve una URL absoluta (Es decir, una URL sin la parte del dominio) que coincide con una determinada vista, incluyendo sus parámetros opcionales. De esta forma se posibilita realizar enlaces sin violar el principio DRY, codificando las direcciones en las plantillas:

`{% url 'algunnombreurl' v1 v2 %}`

El primer argumento es el nombre del patrón URL o name. El resto de los parámetros son opcionales y deben ir separados con comas, convirtiéndose en parámetros posicionales o por nombre que se incluirán en la URL. Deben estar presentes todos los argumentos que se hayan definido como obligatorios en el URLconf. No es posible mezclar argumentos posicionales y argumentos.

Por ejemplo, supongamos que tenemos una vista, VistaDetallesCliente, y que en el URLconf se la indica que acepta un parámetro, el identificador del cliente. La línea del URL podría ser algo así:

`url(r'^cliente/(?P<pk>[09]+)/$', VistaDetallesCliente.as_view(),
name='detallescliente'),`

Si este URLconf fuera incluido en el URLconf del proyecto bajo un directorio, como en este ejemplo:

`('^clientes/', include('project_name.app_name.urls'))`

Podríamos crear un enlace a esta vista, en nuestra plantilla, con la siguiente etiqueta:

`{% url 'detallescliente' cliente.id %}`

La salida de esta etiqueta será `/clientes/cliente/123/`.

Para una referencia completa, por favor diríjase a la documentación oficial en <https://docs.djangoproject.com/en/3.2/ref/templates/builtins/> o algunos de los libros recomendados por el docente.