Variables de plantilla

En las plantillas de Django, puede representar variables colocándolas entre {{ }} llaves:

Ejemplo

template.html:

```
<!DOCTYPE html>
<html>
<body>

<h1>Hola {{ firstname }}, cómo estás?</h1>

<p>En views.py puedes ver como crear una variable.</p>
<p>En template.html puedes ver cómo usar una variable.</p>

</body>
</html>
```

views.py

```
from django.http import HttpResponse
from django.template import loader

def testing(request):
  template = loader.get_template('template.html')
  context = {
    'firstname': 'Mihifidem',
  }
  return HttpResponse(template.render(context, request))
```

127.0.0.1:8000/members/testing

Hola Mihifidem, cómo estás?

En views.py puedes ver como crear una variable..

En template.html puedes ver cómo usar una variable.

Crear variable en vista

La variable firstname del ejemplo anterior se envió a la plantilla a través de una vista:

views.py:

```
from django.http import HttpResponse
from django.template import loader

def testing(request):
 template = loader.get_template('template.html')
 context = {
   'firstname': 'Linus',
 }
 return HttpResponse(template.render(context, request))
```

Como puede ver en la vista anterior, creamos un objeto llamado context, lo llenamos con datos y lo enviamos como el primer parámetro en la función **template.render().**

Crear variables en el template

También puede crear variables directamente en la plantilla, utilizando la etiqueta **{% with %}** de plantilla:

Ejemplo

template.html:

```
<!DOCTYPE html>
<html>
<body>

{% with firstname="Tobias" %}
<h1>Hello {{ firstname }}, how are you?</h1>

</body>
</html>
```

views.py

```
from django.http import HttpResponse
from django.template import loader

def testing(request):
  template = loader.get_template('template.html')
  return HttpResponse(template.render())
```

127.0.0.1:8000/members/testing

Hello Tobias, how are you?

Datos desde un modelo

El ejemplo anterior mostró un enfoque sencillo sobre cómo crear y usar variables en una plantilla.

Normalmente, la mayoría de los datos externos que desea utilizar en una plantilla provienen de un modelo.

Hemos creado un modelo en los capítulos anteriores, llamado Members, usaremos este modelo.

Para obtener datos del modelo Members, tendremos que importarlo en el archivo de vistas y extraer datos de él en la vista:

views.py:

```
from django.http import HttpResponse, HttpResponseRedirect
from django.template import loader
from .models import Members

def testing(request):
 mymembers = Members.objects.all().values()
 template = loader.get_template('template.html')
 context = {
  'mymembers': mymembers,
 }
 return HttpResponse(template.render(context, request))
```

Ahora podemos usar los datos en la plantilla:

template.html:

```
<!DOCTYPE html>
<html>
<body>

<ul>
  {% for x in mymembers %}
    <li>{{ x.firstname }}</li>
  {% endfor %}
</ul>

<p>In views.py you can see how to import and fetch members from the database.</p>

</body>
</html>
```

views.py

```
from django.http import HttpResponse, HttpResponseRedirect
from django.template import loader
from .models import Members

def testing(request):
  mymembers = Members.objects.all().values()
  template = loader.get_template('template.html')
  context = {
    'mymembers': mymembers,
  }
  return HttpResponse(template.render(context, request))
```

127.0.0.1:8000/members/testing

- Emil
- Tobias
- Linus
- Lene
- Stalikken

In views.py you can see how to import and fetch members from the database.

Usamos la etiqueta **{% for %}** de plantilla de Django para recorrer los miembros.

Etiquetas de plantilla

En las plantillas de Django, puede realizar la lógica de programación como ejecutar sentencias **if** y bucles **for**.

Estas palabras clave **if** y **for**, se denominan "etiquetas de plantilla" en Django.

Para ejecutar etiquetas de plantilla, las encerramos entre **{% %}** llaves.

Ejemplo

template.html:

```
<!DOCTYPE html>
<html>
<body>

{% if greeting == 1 %}
  <h1>Hello</h1>
{% else %}
  <h1>Bye</h1>
{% endif %}

<p>In views.py you can see what the greeting variable looks
like.</p>

</body>
</html>
```

views.py

```
from django.http import HttpResponse
from django.template import loader

def testing(request):
  template = loader.get_template('template.html')
  context = {
    'greeting': 1,
  }
  return HttpResponse(template.render(context, request))
```

127.0.0.1:8000/members/testing

Hello

In views.py you can see what the greeting variable looks like.

Código Django

Las etiquetas de plantilla son una forma de decirle a Django que aquí viene algo más que HTML simple.

Las etiquetas de plantilla nos permiten hacer algo de programación en el servidor antes de enviar HTML al cliente.

template.html:

```
<!DOCTYPE html>
<html>
<body>

<ul>
  {% for x in mymembers %}
    <li>{{ x.firstname }}</li>
  {% endfor %}
</ul>

<p>In views.py you can see how to import and fetch members from the database.</p>

</body>
</html>
```

views.py

```
from django.http import HttpResponse, HttpResponseRedirect
from django.template import loader
from .models import Members

def testing(request):
  mymembers = Members.objects.all().values()
  template = loader.get_template('template.html')
  context = {
    'mymembers': mymembers,
  }
  return HttpResponse(template.render(context, request))
```

127.0.0.1:8000/members/testing

- Emil
- Tobias
- Linus
- Lene
- Stalikken

In views.py you can see how to import and fetch members from the database.

Referencia de etiqueta

Una lista de todas las etiquetas de plantilla:

Tag	Description
autoescape	Specifies if autoescape mode is on or off
block	Specifies a block section
comment	Specifies a comment section
csrf_token	Protects forms from Cross Site Request Forgeries
cycle	Specifies content to use in each cycle of a loop
debug	Specifies debugging information
extends	Specifies a parent template
filter	Filters content before returning it
firstof	Returns the first not empty variable
for	Specifies a for loop
if	Specifies a if statement
ifchanged	Used in for loops. Outputs a block only if a value has changed since the last iteration
include	Specifies included content/template
load	Loads template tags from another library
lorem	Outputs random text
now	Outputs the current date/time
regroup	Sorts an object by a group
resetcycle	Used in cycles. Resets the cycle
spaceless	Removes whitespace between HTML tags
templatetag	Outputs a specified template tag
url	Returns the absolute URL part of a URL
verbatim	Specifies contents that should not be rendered by the template engine
widthratio	Calculates a width value based on the ration between a given value and a max value
with	Specifies a variable to use in the block

Etiqueta if else

if

Una declaración **if** evalúa una variable y ejecuta un bloque de código si el valor es verdadero.

Ejemplo

template.html

```
<!DOCTYPE html>
<html>
<body>

{% if greeting == 1 %}
  <h1>Hello</h1>
{% endif %}

<p>In views.py you can see what the greeting variable looks
like.</p>

</body>
</html>
```

views.py

```
from django.http import HttpResponse
from django.template import loader

def testing(request):
  template = loader.get_template('template.html')
  context = {
    'greeting': 1,
  }
  return HttpResponse(template.render(context, request))
```

elif

La palabra clave elif dice "si las condiciones anteriores no fueron ciertas, intente con esta condición".

Ejemplo

```
{% if greeting == 1 %}
 <h1>Hello</h1>
{% elif greeting == 2 %}
 <h1>Welcome</h1>
{% endif %}
```

Else

La palabra clave else captura cualquier cosa que no esté capturada por las condiciones anteriores.

Ejemplo

```
{% if greeting == 1 %}
  <h1>Hello</h1>
{% elif greeting == 2 %}
  <h1>Welcome</h1>
{% else %}
  <h1>Goodbye</h1>
{% endif %}
```

Operadores

Los ejemplos anteriores usan el ==operador, que se usa para verificar si una variable es igual a un valor, pero hay muchos otros operadores que puede usar, o incluso puede eliminar el operador si solo desea verificar si una variable no está vacía :

Ejemplo

```
{% if greeting %}
  <h1>Hello</h1>
{% endif %}
```

==

Es igual a.

Ejemplo

```
{% if greeting == 2 %}
  <h1>Hello</h1>
{% endif %}
```

!=

No es igual a.

Ejemplo

```
{% if greeting != 1 %}
  <h1>Hello</h1>
{% endif %}
```

<

Es menos que.

Ejemplo

```
{% if greeting < 3 %}
  <h1>Hello</h1>
{% endif %}
```

<=

Es menor o igual que.

Ejemplo

```
{% if greeting <= 3 %}
  <h1>Hello</h1>
{% endif %}
```

>

Es mayor que.

Ejemplo

```
{% if greeting > 1 %}
  <h1>Hello</h1>
{% endif %}
```

>=

Es mayor o igual a.

Ejemplo

```
{% if greeting >= 1 %}
  <h1>Hello</h1>
{% endif %}
```

and

Para comprobar si más de una condición es verdadera.

Ejemplo

```
{% if greeting == 1 and day == "Friday" %}
  <h1>Hello Weekend!</h1>
{% endif %}
```

or
Para comprobar si una de las condiciones es verdadera.

Ejemplo

```
{% if greeting == 1 or greeting == 5 %}
  <h1>Hello</h1>
{% endif %}
```

and/or
Combinar and y or.

Ejemplo

```
{% if greeting == 1 and day == "Friday" or greeting == 5 %}
```

Los paréntesis no están permitidos en declaraciones **if** en Django, por lo que cuando combina operadores **and** y **or**, es importante saber que los paréntesis se agregan para **and** pero no para **or**.

Lo que significa que el ejemplo anterior es leído por el intérprete de esta manera:

```
{% if (greeting == 1 and day == "Friday") or greeting == 5 %}
```

in
Para comprobar si un elemento determinado está presente en un objeto.

Ejemplo

```
{% if 'Banana' in fruits %}
  <h1>Hello</h1>
{% else %}
  <h1>Goodbye</h1>
{% endif %}
```

not in
Para comprobar si un elemento determinado no está presente en un objeto.

Ejemplo

```
{% if 'Banana' not in fruits %}
  <h1>Hello</h1>
{% else %}
  <h1>Goodbye</h1>
{% endif %}
```

is

Comprueba si dos objetos son iguales.

Este operador es diferente del operador ==, porque el operador == verifica los valores de dos objetos, pero el operador **is** verifica la identidad de dos objetos.

En la vista tenemos dos objetos, x y, con los mismos valores:

Ejemplo

views.py:

```
from django.http import HttpResponse
from django.template import loader

def testing(request):
  template = loader.get_template('template.html')
  context = {
    'x': ['Apple', 'Banana', 'Cherry'],
    'y': ['Apple', 'Banana', 'Cherry'],
  }
  return HttpResponse(template.render(context, request))
```

Los dos objetos tienen el mismo valor, pero ¿es el mismo objeto?

Ejemplo

```
{% if x is y %}
  <h1>YES</h1>
{% else %}
  <h1>NO</h1>
{% endif %}
```

NO

In views.py you can see what x and y variables look like.

Probemos el mismo ejemplo con el ==operador en su lugar:

Ejemplo

```
{% if x == y %}
  <h1>YES</h1>
{% else %}
  <h1>NO</h1>
{% endif %}
```

YES

In views.py you can see what x and y variables look like

¿Cómo pueden dos objetos ser iguales? Bueno, si tiene dos objetos que apuntan al mismo objeto, entonces el operador is se evalúa como verdadero:

Lo demostraremos usando la etiqueta **{% with %}**, que nos permite crear variables en la plantilla:

Ejemplo

```
{% with var1=x var2=x %}
  {% if var1 == var2 %}
    <h1>YES</h1>
  {% else %}
    <h1>NO</h1>
  {% endif %}
{% endwith %}
```

not is
Para comprobar si dos objetos no son iguales.

Ejemplo

```
{% if x is not y %}
  <h1>YES</h1>
{% else %}
  <h1>NO</h1>
{% endif %}
```

Etiqueta FOR bucles

Un ciclo for se usa para iterar sobre una secuencia, como recorrer elementos en una matriz, una lista o un diccionario.

Ejemplo

Recorra los elementos de una lista:

template.html

```
<!DOCTYPE html>
<html>
<body>

{% for x in fruits %}
  <h1>{{ x }}</h1>
{% endfor %}

<p>In views.py you can see what the fruits variable looks like.</p>

</body>
</html>
```

views.py

```
from django.http import HttpResponse
from django.template import loader

def testing(request):
  template = loader.get_template('template.html')
  context = {
    'fruits': ['Apple', 'Banana', 'Cherry'],
  }
  return HttpResponse(template.render(context, request))
```

127.0.0.1:8000/members/testing

Apple

Banana

Cherry

In views.py you can see what the fruits variable looks like.

Ejemplo

Recorra una lista de diccionarios:

template.html

```
<!DOCTYPE html>
<html>
<body>

{% for x in cars %}
  <h1>{{ x.brand }}</h1>
  <p>{{ x.model }}</p>
  <p>{{ x.year }}</p>
{% endfor %}

<p>In views.py you can see what the cars variable looks like.</p>

</body>
</html>
```

views.py

```
from django.http import HttpResponse
from django.template import loader

def testing(request):
  template = loader.get_template('template.html')
  context = {
    'cars': [
      {
        'brand': 'Ford',
        'model': 'Mustang',
        'year': '1964',
      },
      {
        'brand': 'Ford',
        'model': 'Bronco',
        'year': '1970',
      },
      {
        'brand': 'Volvo',
        'model': 'P1800',
        'year': '1964',
      }]
    }
  return HttpResponse(template.render(context, request))
```

Datos de un modelo

Los datos en un modelo son como una tabla con filas y columnas.

El modelo Miembros que creamos anteriormente tiene cinco filas y cada fila tiene tres columnas:

id	firstname	lastname
1	emilio	refsnes
2	tobías	refsnes
3	Linus	refsnes
4	lene	refsnes
5	Stalikken	refsnes

Cuando obtenemos datos del modelo, vienen como un objeto QuerySet, con un formato similar al del ejemplo de autos anterior: una lista con diccionarios:

```
<QuerySet [
 {
  'id': 1,
  'firstname': 'Emil',
  'lastname': 'Refsnes'
 },
 {
  'id': 2,
  'firstname': 'Tobias',
  'lastname': 'Refsnes'
 },
 {
  'id': 3,
  'firstname': 'Linus',
  'lastname': 'Refsnes'
 },
 {
  'id': 4,
  'firstname': 'Lene',
  'lastname': 'Refsnes'
 },
 {
  'id': 5,
  'firstname': 'Stalikken',
  'lastname': 'Refsnes'
 }
]>
```

Ejemplo

Recorra los elementos obtenidos de una base de datos:

template.html

```
<!DOCTYPE html>
<html>
<body>

{% for x in members %}
  <h1>{{ x.id }}</h1>
  <p>
    {{ x.firstname }}
    {{ x.lastname }}
  </p>
{% endfor %}

<p>In views.py you can see how to import and fetch members from the
database.</p>

</body>
</html>
```

views.py

```
from django.http import HttpResponse, HttpResponseRedirect
from django.template import loader
from .models import Members

def testing(request):
  mymembers = Members.objects.all().values()
  template = loader.get_template('template.html')
  context = {
    'members': mymembers,
  }
  return HttpResponse(template.render(context, request))
```

1
Emil Refsnes

2
Tobias Refsnes

3
Linus Refsnes

4
Lene Refsnes

5
Stalikken Refsnes

In views.py you can see how to import and fetch members from the database

Reversed

La palabra clave reversed se usa cuando desea hacer el bucle en orden inverso.

Ejemplo

```
{% for x in members reversed %}
  <h1>{{ x.id }}</h1>
  <p>
    {{ x.firstname }}
    {{ x.lastname }}
  </p>
{% endfor %}
```

empty

La palabra clave empty se puede usar si desea hacer algo especial si el objeto está vacío.

Ejemplo

```
<ul>
  {% for x in emptytestobject %}
    <li>{{ x.firstname }}</li>
  {% empty %}
    <li>No members</li>
  {% endfor %}
</ul>
```

La palabra clave empty también se puede utilizar si el objeto no existe:

Ejemplo

```
<ul>
  {% for x in myobject %}
    <li>{{ x.firstname }}</li>
  {% empty %}
    <li>No members</li>
  {% endfor %}
</ul>
```

Variables de bucle

Django tiene algunas variables que están disponibles para ti dentro de un ciclo:

forloop.counter
forloop.counter0
forloop.first
forloop.last
forloop.parentloop
forloop.revcount
forloop.revcount0

forloop.counter
La iteración actual, comenzando en 1.

Ejemplo

```
<ul>
  {% for x in fruits %}
    <li>{{ forloop.counter }}</li>
  {% endfor %}
</ul>
```

forloop.contador0
La iteración actual, comenzando en 0.

Ejemplo

```
<ul>
  {% for x in fruits %}
    <li>{{ forloop.counter0 }}</li>
  {% endfor %}
</ul>
```

forloop.first

Le permite probar si el ciclo está en su primera iteración.

Ejemplo

Dibuja un fondo azul para la primera iteración del ciclo:

```
<ul>
 {% for x in fruits %}
   <li
    {% if forloop.first %}
      style='background-color:lightblue;'
    {% endif %}
   >{{ x }}</li>
 {% endfor %}
</ul>
```

forloop.last

Le permite probar si el ciclo está en su última iteración.

Ejemplo

Dibuja un fondo azul para la última iteración del bucle:

```
<ul>
 {% for x in fruits %}
   <li
    {% if forloop.last %}
      style='background-color:lightblue;'
    {% endif %}
   >{{ x }}</li>
 {% endfor %}
</ul>
```

forloop.revcount

La iteración actual si comienza al final y cuenta hacia atrás, terminando en 1.

Ejemplo

```
<ul>
 {% for x in fruits %}
   <li>{{ forloop.revcounter }}</li>
 {% endfor %}
</ul>
```

forloop.revcount0

La iteración actual si comienza al final y cuenta hacia atrás, terminando en 0.

Ejemplo

```
<ul>
  {% for x in fruits %}
    <li>{{ forloop.revcounter0 }}</li>
  {% endfor %}
</ul>
```

Etiqueta de comentario de Django

Comentarios

Los comentarios le permiten tener secciones de código que deben ignorarse.

Ejemplo

```
<h1>Welcome Everyone</h1>
{% comment %}
  <h1>Welcome ladies and gentlemen</h1>
{% endcomment %}
```

Comentario Descripción

Puede agregar un mensaje a su comentario, para ayudarlo a recordar por qué escribió el comentario, o como mensaje para otras personas que lean el código.

Ejemplo

Añade una descripción a tu comentario:

```
<h1>Welcome Everyone</h1>
{% comment "this was the original welcome message" %}
   <h1>Welcome ladies and gentlemen</h1>
{% endcomment %}
```

Comentarios más pequeños

También puede usar las {# ... #}etiquetas al comentar el código, lo que puede ser más fácil cuando se trata de comentarios más pequeños:

Ejemplo

Comente la palabra Todos:

```
<h1>Welcome{# Everyone#}</h1>
```

Comentar en Vistas

Las vistas se escriben en Python y los comentarios de Python se escriben con el #carácter:

Ejemplo
Comente una sección en la vista:

```
from django.http import HttpResponse
from django.template import loader

def testing(request):
  template = loader.get_template('template.html')
  #context = {
  # 'var1': 'John',
  #}
  return HttpResponse(template.render())
```

Etiqueta cycle

Cycles

La etiqueta cycle le permite realizar diferentes tareas para diferentes iteraciones.

La etiqueta cycle toma argumentos, la primera iteración usa el primer argumento, la segunda iteración usa el segundo argumento, etc.

```
{% cycle 'lightblue' 'pink' 'yellow' 'coral' 'grey' %}
```

Si desea tener un nuevo color de fondo para cada iteración, puede hacerlo con la etiqueta cycle:

Ejemplo

template.html

```
<!DOCTYPE html>
<html>
<body>

<ul>
  {% for x in members %}
    <li style='background-color:{% cycle 'lightblue' 'pink'
'yellow' 'coral' 'grey' %}'>
      {{ x.firstname }}
    </li>
  {% endfor %}
</ul>

<p>In views.py you can see how to import and fetch members from the
database.</p>

</body>
</html>
```

views.py

```
from django.http import HttpResponse, HttpResponseRedirect
from django.template import loader
from .models import Members

def testing(request):
  mymembers = Members.objects.all().values()
  template = loader.get_template('template.html')
  context = {
    'members': mymembers,
  }
  return HttpResponse(template.render(context, request))
```

127.0.0.1:8000/members/testing

- Emil
- Tobias
- Linus
- Lene
- Stalikken

In views.py you can see how to import and fetch members from the database.

Si el ciclo llega al final de los argumentos, comienza de nuevo:

Ejemplo

```
<ul>
 {% for x in members %}
   <li style='background-color:{% cycle 'lightblue' 'pink' %}'>
    {{ x.firstname }}
   </li>
 {% endfor %}
</ul>
```

Ciclo de argumentos como variable

En el primer ejemplo, los valores de los argumentos se mostraban directamente en el ciclo, pero también puede mantener los valores de los argumentos en una variable y usarlos más adelante:

Ejemplo

Almacene los valores de color en una variable llamada bgcolor y utilícelo como color de fondo más adelante en el bucle:

```
<ul>
 {% for x in members %}
   {% cycle 'lightblue' 'pink' 'yellow' 'coral' 'grey' as bgcolor silent %}
   <li style='background-color:{{ bgcolor }}'>
    {{ x.firstname }}
   </li>
 {% endfor %}
</ul>
```

¿Te diste cuenta de la **silent** palabra clave? Asegúrese de agregar esto, de lo contrario, los valores de los argumentos se mostrarán dos veces en la salida:

Ejemplo
El mismo ejemplo anterior, pero sin la palabra silent clave:

```
<ul>
 {% for x in members %}
   {% cycle 'lightblue' 'pink' 'yellow' 'coral' 'grey' as bgcolor %}
   <li style='background-color:{{ bgcolor }}'>
     {{ x.firstname }}
   </li>
 {% endfor %}
</ul>
```

Ciclo de reinicio

Puede forzar el reinicio del ciclo usando la etiqueta {% resetcycle %}:

Ejemplo
Reiniciar el ciclo después de 3 ciclos:

```
<ul>
 {% for x in members %}
   {% cycle 'lightblue' 'pink' 'yellow' 'coral' 'grey' as bgcolor silent %}
   {% if forloop.counter == 3 %}
     {% resetcycle %}
   {% endif %}
   <li style='background-color:{{ bgcolor }}'>
     {{ x.firstname }}
   </li>
 {% endfor %}
</ul>
```

Etiqueta extends

La etiqueta extends le permite agregar una plantilla principal para la plantilla actual.

Esto significa que puede tener una página maestra que actúe como principal para todas las demás páginas:

Ejemplo
mymaster.html:

```
<html>
<body>

<h1>Welcome</h1>

{% block mymessage %}
{% endblock %}

</body>
</html>
```

template.html:

```
{% extends 'mymaster.html' %}

{% block mymessage %}
  <p>This page has a master page</p>
{% endblock %}
```

views.py

```
from django.http import HttpResponse
from django.template import loader

def testing(request):
  template = loader.get_template('template.html')
  return HttpResponse(template.render())
```

Welcome

This page has a master page.

Pones marcadores de posición en la plantilla maestra, diciéndole a Django dónde poner qué contenido.

Django usa la etiqueta **{% block %}** para crear marcadores de posición:

master.html:

```
<!DOCTYPE html>
<html>
<body>

{% block myheading %}
{% endblock %}

{% block mymessage %}
{% endblock %}

<p>Check out the two templates to see what they look like,
and views.py to see the reference to the child template
and to see how the members variable looks like.</p>

</body>
</html>
```

Plantillas que usan la plantilla maestra, usan la {% block %}etiqueta para crear contenido que se mostrará en el marcador de posición con el mismo nombre:

template.html:

```
{% extends 'mymaster.html' %}

{% block myheading %}
  <h1>Members</h1>
{% endblock %}

{% block mymessage %}
  <ul>
    {% for x in members %}
      <li>{{ x.firstname }}</li>
    {% endfor %}
  </ul>
{% endblock %}
```

views.py

```
from django.http import HttpResponse
from django.template import loader
from .models import Members

def testing(request):
 mymembers = Members.objects.all().values()
 template = loader.get_template('template.html')
 context = {
  'members': mymembers,
 }
 return HttpResponse(template.render(context, request))
```

Members

- Emil
- Tobias
- Linus
- Lene
- Stalikken

Check out the two templates to see what they look like, and views.py to see the reference to the child template and to see how the members variable looks like.

Etiqueta include

La etiqueta include le permite incluir una plantilla dentro de la plantilla actual.

Esto es útil cuando tiene un bloque de contenido que es el mismo para muchas páginas.

Ejemplo
footer.html:

```
<p>You have reach the bottom of this page, thank you for your time.</p>
```

template.html:

```
<h1>Hello</h1>

<p>This page contains a footer in a template.</p>

{% include 'footer.html' %}
```

Variables en Incluir

Puede enviar variables a la plantilla utilizando la palabra clave **with**.

En el archivo de inclusión, hace referencia a las variables utilizando la sintaxis de nombre de **{{** variable : **}}**

Ejemplo
mymenu.html:

```
<div>HOME | {{ me }} | ABOUT | FORUM | {{ sponsor }}</div>
```

template.html:

```
<!DOCTYPE html>
<html>
<body>

{% include mymenu.html with me="OSCAR" sponsor="Mihifidem" %}

<h1>Welcome</h1>

<p>This is my webpage</p>

</body>
</html>
```

views.py

```
from django.http import HttpResponse
from django.template import loader

def testing(request):
  template = loader.get_template('template.html')
  return HttpResponse(template.render())
```

HOME | OSCAR | ABOUT | FORUM | Mihifidem

Welcome

This is my web site.

Check out mymenu.html to see the HTML content of the include.

Etiqueta Filter

Filtrar un valor

Con el carácter de canalización **|**seguido de un nombre de filtro, puede ejecutar un valor a través de un filtro antes de devolverlo.

El nombre del filtro define qué hará el filtro con el valor.

Ejemplo
Devuelve la variable **firstname** con mayúsculas:

```
<h1>Hello {{ firstname|upper }}, how are you?</h1>
```

Filtros Múltiples

Puede agregar más de un filtro agregando |caracteres verticales seguidos de los nombres de los filtros:

Ejemplo
Devuelve el primer carácter de la variable **firstname**, en minúsculas:

```
<h1>Hello {{ firstname|first|lower }}, how are you?</h1>
```

La etiqueta filter

La etiqueta filter le permite ejecutar una sección completa de código a través de un filtro y devolverlo de acuerdo con las palabras clave del filtro.

Ejemplo
Devuelve la variable **firstname** con mayúsculas:

```
{% filter upper %}
  <h1>Hello everyone, how are you?</h1>
{% endfilter %}
```

Para agregar varios filtros, separe las palabras clave con el |carácter de barra vertical:

Ejemplo
Filtros separados con el carácter de canalización:

```
{% filter upper|linenumbers %}Hello!
my name is
Emil.
What is your name?{% endfilter %}
```

1. HELLO! 2. MY NAME IS 3. EMIL. 4. WHAT IS YOUR NAME?

Referencia de filtro

Una lista de todas las palabras clave de filtro:

Keyword	Description
add	Adds a specified value.
addslashes	Adds a slash before any quote characters, to escape strings.
capfirst	Returns the first letter in uppercase.
center	Centers the value in the middle of a specified width.
cut	Removes any specified character or phrases.
date	Returns dates in the specified format.
default	Returns a specified value if the value is False.
default_if_none	Returns a specified value if the value is None.
dictsort	Sorts a dictionary by the given value.
dictsortreversed	Sorts a dictionary reversed, by the given value.
divisibleby	Returns True if the value can be divided by the specified number, otherwise it returns False.
escape	Escapes HTML code from a string.
escapejs	Escapes JavaScript code from a string.
filesizeformat	Returns a number into a file size format.
first	Returns the first item of an object (for Strings, the first character is returned).
floatformat	Rounds floating numbers to a specified number of decimals, default one decimal.
force_escape	Escapes HTML code from a string.
get_digit	Returns a specific digit of a number.
iriencode	Convert an IRI into a URL friendly string.
join	Returns the items of a list into a string.
json_script	Returns an object into a JSON object surrounded by <script></script> tags.
last	Returns the last item of an object (for Strings, the last character is returned).
length	Returns the number of items in an object, or the number of characters in a string.
length_is	Returns True if the length is the same as the specified number
linebreaks	Returns the text with instead of line breaks, and <p> instead of more than one line break.
linebreaksbr	Returns the text with instead of line breaks.
linenumbers	Returns the text with line numbers for each line.
ljust	Left aligns the value according to a specified width
lower	Returns the text in lower case letters.
make_list	Converts a value into a list object.
phone2numeric	Converts phone numbers with letters into numeric phone numbers.
pluralize	Adds a 's' at the end of a value if the specified numeric value is not 1.
pprint	
random	Returns a random item of an object
rjust	Right aligns the value according to a specified width
safe	Marks that this text is safe and should not be HTML escaped.

safeseq	Marks each item of an object as safe and the item should not be HTML escaped.
slice	Returns a specified slice of a text or object.
slugify	Converts text into one long alphanumeric-lower-case word.
stringformat	Converts the value into a specified format.
striptags	Removes HTML tags from a text.
time	Returns a time in the specified format.
timesince	Returns the difference between two datetimes.
timeuntil	Returns the difference between two datetimes.
title	Upper cases the first character of each word in a text, all other characters are converted to lower case.
truncatechars	Shortens a string into the specified number of characters.
truncatechars_html	Shortens a string into the specified number of characters, not considering the length of any HTML tags.
truncatewords	Shortens a string into the specified number of words.
truncatewords_html	Shortens a string into the specified number of words, not considering any HTML tags.
unordered_list	Returns the items of an object as an unordered HTML list.
upper	Returns the text in upper case letters.
urlencode	URL encodes a string.
urlize	Returns any URLs in a string as HTML links.
urlizetrunc	Returns any URLs in a string as HTML links, but shortens the links into the specified number of characters.
wordcount	Returns the number of words in a text.
wordwrap	Wrap words at a specified number of characters.
yesno	Converts Booleans values into specified values.
i18n	
l10n	
tz	