# Práctica 12.1: Datos dinámicos en las plantillas

Tenemos diferentes piezas en su lugar: el modelo `Post` está definido en `models.py`, tenemos a `post_list` en `views.py` y la plantilla agregada. Pero, nos falta cómo hacer que realmente aparezcan nuestros posts en nuestra plantilla HTML. Porque eso es lo que queremos hacer: tomar algún contenido (modelos guardados en la base de datos) y mostrarlo adecuadamente en nuestra plantilla.

Esto es exactamente lo que las *views* se supone que hacen: conectar modelos con plantillas. En nuestra *view* `post_list` necesitaremos tomar los modelos que deseamos mostrar y pasarlos a una plantilla. Así que básicamente en una *view* decidimos qué (modelo) se mostrará en una plantilla.

Necesitamos abrir nuestro archivo `blog/views.py`. Hasta ahora la *view* `post_list` se ve así:

```
from django.shortcuts import render

def post_list(request):
    return render(request, 'blog/post_list.html', {})
```

Recuerdas cuando hablamos de incluir código en diferentes archivos. Ahora tenemos que incluir el modelo que definimos en el archivo `models.py`. Agregaremos la línea `from .models import Post` de la siguiente forma:

```
from django.shortcuts import render
from .models import Post
```

El punto antes de `models` indica el *directorio actual* o la *aplicación actual*. Como `views.py` y `models.py` están en el mismo directorio, simplemente usamos `.` y el nombre del archivo (sin `.py`). Ahora importamos el nombre del modelo (`Post`).

Para tomar publicaciones reales del modelo `Post`, necesitamos algo llamado `QuerySet` (conjunto de consultas).

## QuerySet

Ya debes estar familiarizado con la forma en que funcionan los QuerySets. Hablamos de ellos en la práctica anterior.

Entonces ahora nos interesa obtener una lista de entradas del blog que han sido publicadas y ordenadas por `published_date` (fecha de publicación), ya hicimos eso en el la práctica anterior.

```
Post.objects.filter(published_date__lte=timezone.now()).order_by('published_date')
```

Ahora pondremos este bloque de código en el archivo `blog/views.py`, agregándole a la función `def post_list(request)`, pero no olvides primero agregar `from django.utils import timezone`:

```
from django.shortcuts import render
from django.utils import timezone
from .models import Post

def post_list(request):
    posts = Post.objects.filter(published_date__lte=timezone.now()).order_by('published_date')
    return render(request, 'blog/post_list.html', {})
```

La última parte es pasar el QuerySet posts al template (veremos cómo mostrarla en la siguiente parte).

Observa que creamos una *variable* en nuestro QuerySet: `posts`. Tómala como el nombre de nuestro QuerySet. De aquí en adelante vamos a referirnos al QuerySet con ese nombre.

En la función `render` ya tenemos el parámetro `request` (todo lo que recibimos del usuario vía Internet) y el archivo `'blog/post_list.html'` como plantilla. El último parámetro, `{}`, es un campo en el que podemos agregar algunas cosas para que la plantilla las use. Necesitamos nombrarlos (los seguiremos llamando `'posts'` por ahora). Se debería ver así: `{'posts': posts}`. Observa que la parte que va antes de `:` es una cadena; necesitas ponerlo entre comillas: `''`.

Finalmente nuestro archivo `blog/views.py` debería verse así:

```
from django.shortcuts import render
from django.utils import timezone
from .models import Post

def post_list(request):
    posts = Post.objects.filter(published_date__lte=timezone.now()).order_by('published_date')
    return render(request, 'blog/post_list.html', {'posts': posts})
```

Ahora regresemos a nuestra plantilla y mostremos este QuerySet.

> Si quieres leer un poco más acerca de QuerySets visitando la documentación de Django.

# Práctica 12.2 - Plantillas de Django

Hora de mostrar algunos datos. Django nos provee las útiles **template tags** para ello.

## ¿Qué son las template tags?

Verás, en HTML no puedes realmente poner código Python, porque los navegadores no lo entienden. Ellos sólo saben HTML. Sabemos que HTML es algo estático, mientras que Python es mucho más dinámico.

**Django template tags** nos permite transferir cosas de Python como cosas en HTML, así que tu puedes construir sitios web dinámicos más rápido y fácil.

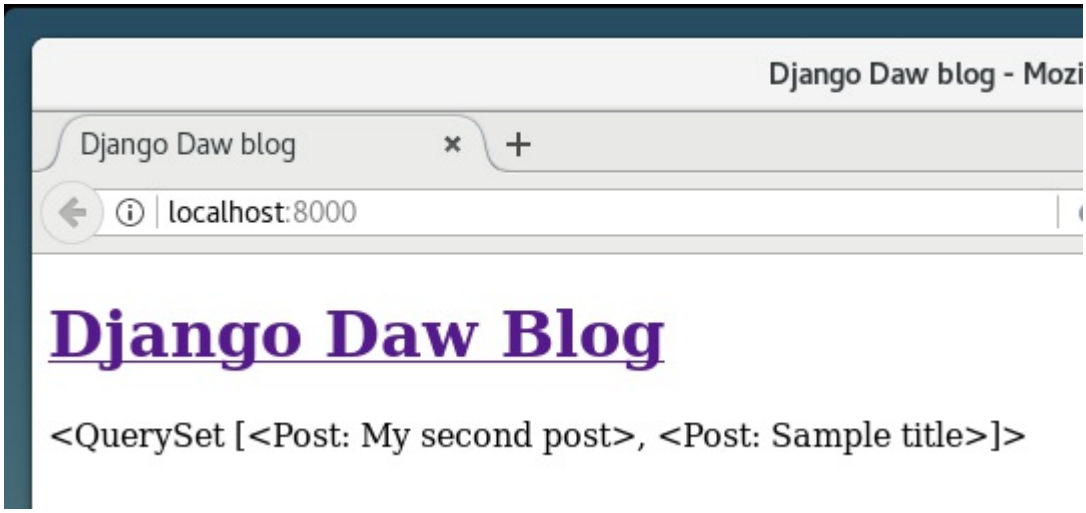
## Mostrar la plantilla post list

En la parte anterior dimos a nuestra plantilla una lista de posts en la variable `posts`. Ahora lo mostraremos en HTML.

Para imprimir una variable en una plantilla de Django, utilizamos llaves dobles con el nombre de la variable dentro, así:

```
{{ posts }}
```

Prueba esto en tu plantilla `blog/templates/blog/post_list.html`. Reemplaza todo desde el segundo `<div>` hasta el tercer `</div>` con `{{ posts }}`, guarda el archivo y actualiza la página para ver los resultados:



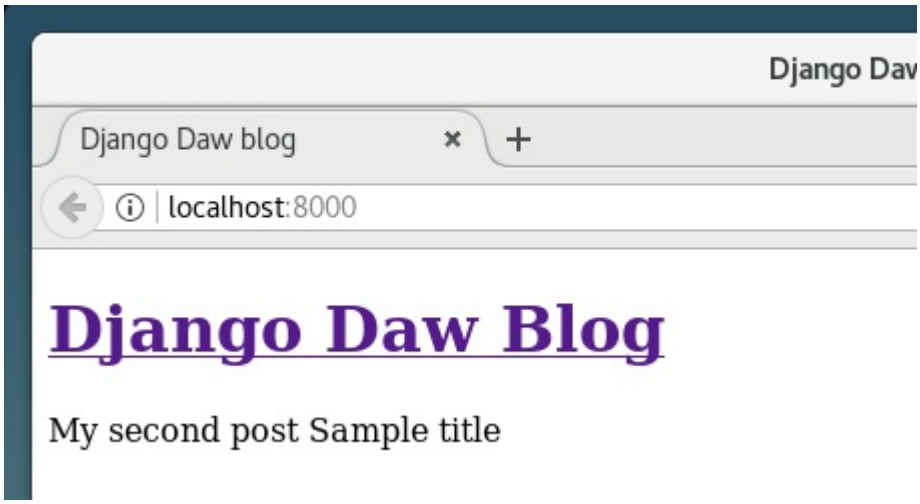

Como puedes ver, todo lo que obtenemos es esto:

```
<QuerySet [<Post: Mi segundo post>, <Post: Mi primer post>]>
```

Esto significa que Django lo entiende como una lista de objetos. Recuerda de la práctica de **Introducción a Python** que podemos mostrar listas con bucles. En una plantilla de Django, lo haces de esta manera:

```
{% for post in posts %}
   {{ post }}
{% endfor %}
```

Prueba esto en tu plantilla.

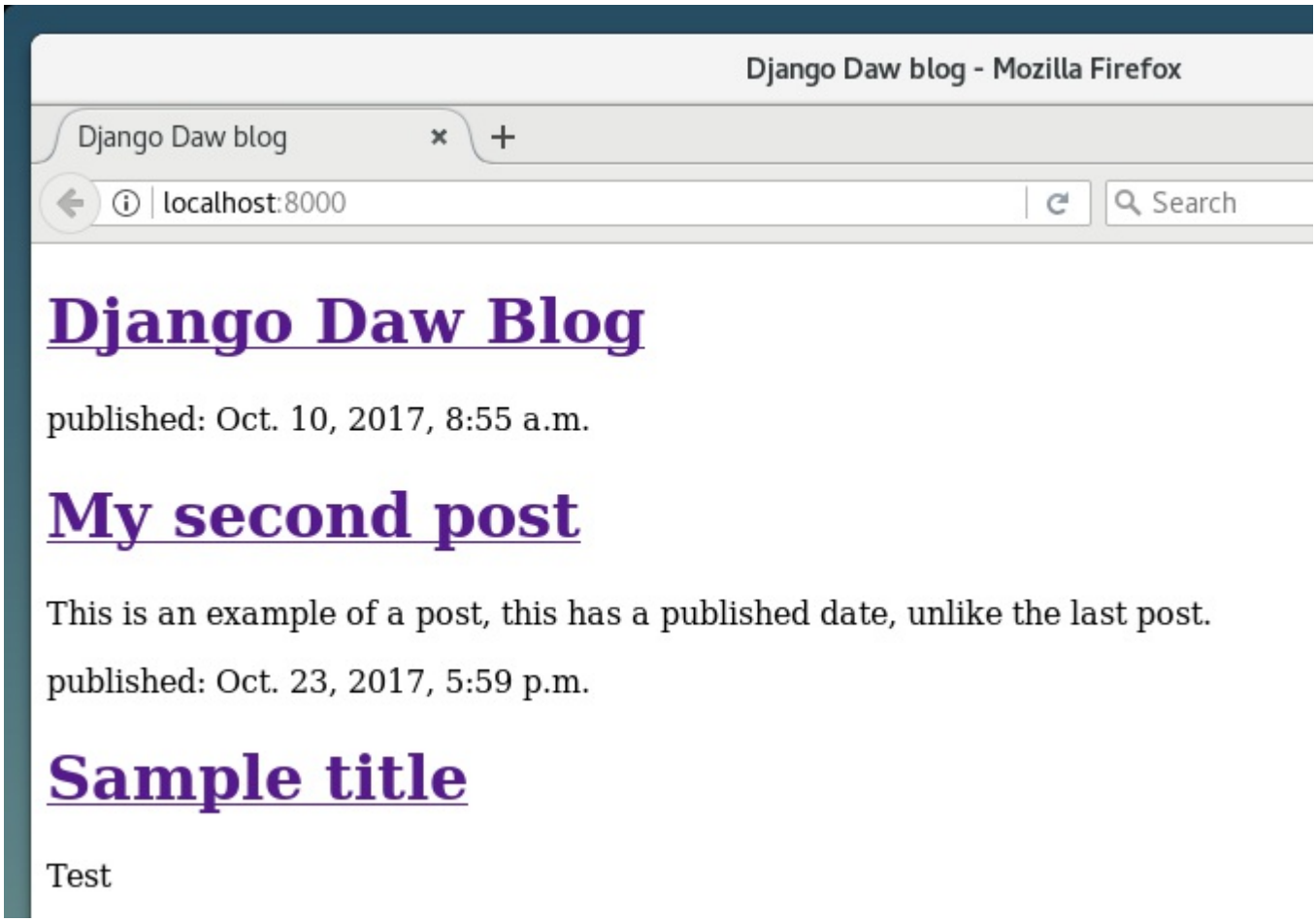



Funciona. Pero queremos que se muestren cómo los posts estáticos que creamos anteriormente en la práctica de **Introducción a HTML**. Puedes mezclar HTML y template tags. Nuestro body se verá así:

```
<div>
    <h1><a href="/">Django DAW Blog</a></h1>
</div>

{% for post in posts %}
    <div>
        <p>published: {{ post.published_date }}</p>
        <h1><a href="">{{ post.title }}</a></h1>
        <p>{{ post.text|linebreaksbr }}</p>
    </div>
{% endfor %}
```

Todo lo que pones entre `{% for %}` y `{% endfor %}` se repetirá para cada objeto en la lista. Actualiza tu página:




¿Has notado que utilizamos una notación diferente esta vez `{{ post.title }}` ó `{{ post.text }}`? Estamos accediendo a datos en cada uno de los campos definidos en nuestro modelo `Post`. Además, el `|linebreaksbr` está dirigiendo el texto de los posts a través de un filtro para convertir saltos de línea en párrafos.

## Una cosa más

Sería bueno ver si tu sitio web seguirá funcionando en la Internet pública,intentemos desplegándolo en PythonAnywhere nuevamente. Aquí te esta una recapitulación para ayudarte...

- Primero, sube tu código a GitHub

```
$ git status
[...]
$ git add --all .
$ git status
[...]
$ git commit -m "Modified templates to display posts from database."
[...]
$ git push
```

- Luego, identifícate en PythonAnywhere y ve a tu **consola Bash** (o empieza una nueva), y ejecuta:

```
$ cd ~/<tu-pythonanywhere-username>.pythonanywhere.com
$ git pull
[...]
```

- Finalmente, ve a la pestaña Web y presiona **Reload** en tu aplicación web. Tu actualización debería poder verse.

Ahora sigue adelante, trata de agregar un nuevo post usando el panel de administrador de Django (¡recuerda añadir published_date!) y luego actualiza tu página para ver si aparece tu nuevo post.