



# Django 2.0

Introducción

# ¿Qué es **django** ?



Django es un framework para desarrollo web, escrito en Python; el cual fomenta un desarrollo rápido y un diseño limpio y pragmático. Desarrollado por experimentados desarrolladores; se encarga de gran parte de las complicaciones del desarrollo web.

# ¿Por qué utilizar Django?



Con Django, se pueden llevar las aplicaciones web desde el concepto hasta el lanzamiento en cuestión de horas, por lo que puedes concentrarte en escribir tus aplicaciones sin necesidad de reinventar la rueda. Es gratis y de código abierto.

# Ventajas de utilizar Django

- ❖ Ridículamente rápido.
- ❖ Completamente cargado.
- ❖ Tranquilo y seguro.
- ❖ Extremadamente escalable.
- ❖ Increíblemente versátil.





**¿Qué necesito para comenzar?**



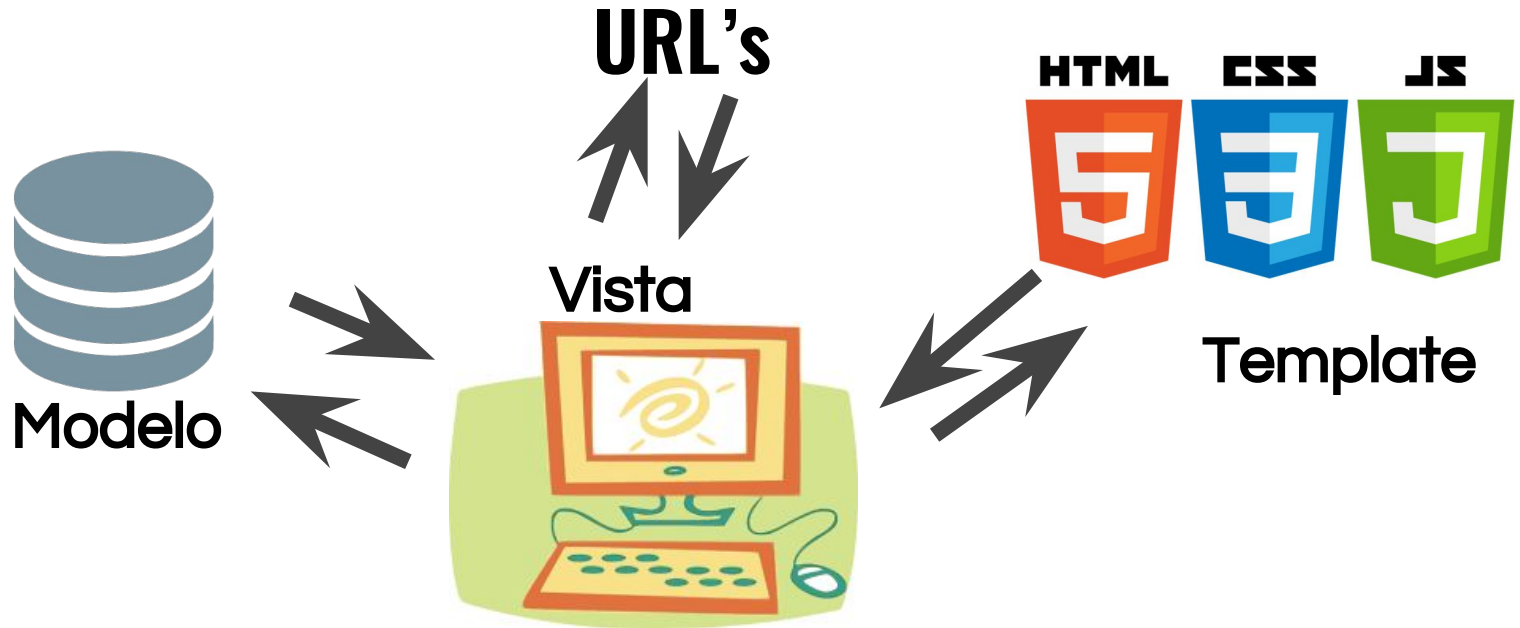
# MVT(Modelo Template Vista)

El modelo MVT es un patrón de diseño basado en los siguientes elementos:

- **Modelo:** Contiene toda la información sobre los datos, es decir, cómo acceder a estos, cómo validarlos, cuál es el comportamiento que tienen, y las relaciones entre ellos. Se utiliza el ORM(Object Relational Mapping) de Django para esto.



**Vista:** Contiene la lógica que accede al modelo y la delega a la plantilla apropiada: puedes pensar en esto como un puente entre el modelos y las plantillas.



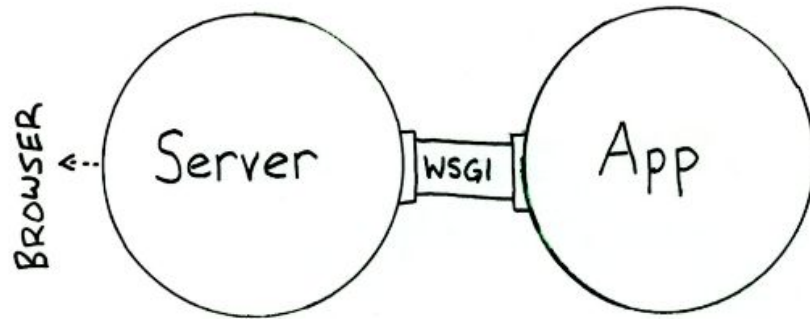
**Template:** Es la capa de presentación. Esta capa contiene las decisiones relacionadas a la presentación: como algunas cosas son mostradas sobre una página web o otro tipo de documento.





# WSGI(Web Server Gateway Interface)

WSGI es una interface simple y universal entre los servidores web y las aplicaciones web o frameworks, la cual nos permite tener un servidor local para realizar pruebas de nuestros proyectos de Django.



# Instalación de Django

- Django como casi cualquier archivo de python es otro módulo podemos instalarlo con pip.

```
$ pip install django
```

- En el la línea de comandos verificar que este instalado

```
$ python -c "import django;print(django.get_version())"
```

- Las versiones que trabajan la mayoría de los proyectos actuales son:
  - » 3.4 >= python
  - » 2.0.4 >= django (Actual 2.0.4)
- Cualquier error resultará en: "No module named django".



**¡COMENCEMOS!**

# Creando un nuevo proyecto de Django

- Con el siguiente comando podemos crear el contenedor del proyecto

```
$ django-admin startproject misitio
```

- Tenemos que evitar nombres que confundan al intérprete como ***django*** o ***test***, que crearía conflictos
- Podemos ver que **startproject** creó:

```
kubos@kubos:~/Desktop$ tree misitio/
misitio/
├── manage.py
└── misitio
    ├── __init__.py
    ├── settings.py
    ├── urls.py
    └── wsgi.py
```



# Archivos importantes

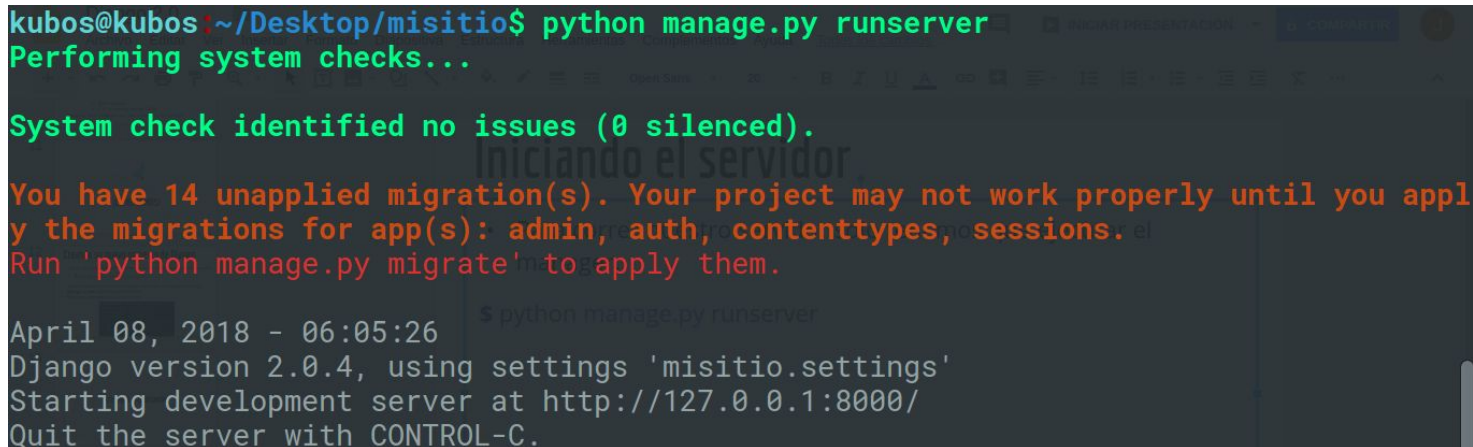


- **urls.py**: Declaración de las URL's de este proyecto Django; una "tabla de contenidos" de tu sitio Django.
- **wsgi.py**: Punto de entrada para servir tu proyecto mediante servidores web compatibles con WSGI.
- **settings.py**: Configuración de este proyecto Django.
- **manage.py**: Permite interactuar con el proyecto de varias maneras
- **\_\_init\_\_.py**: Indica que la carpeta es un paquete, el archivo está vacío.

# Iniciando el servidor

- Para correr nuestro servidor solo tenemos que ejecutar el manager:

\$ python manage.py runserver

A terminal window screenshot showing the output of the command 'python manage.py runserver'. The prompt is 'kubos@kubos:~/Desktop/misitio\$'. The output includes 'Performing system checks...', 'System check identified no issues (0 silenced).', a warning about 14 unapplied migrations for 'admin', 'auth', 'contenttypes', and 'sessions', and the start of the Django development server at 'http://127.0.0.1:8000/'.

```
kubos@kubos:~/Desktop/misitio$ python manage.py runserver
Performing system checks...

System check identified no issues (0 silenced).

You have 14 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.

April 08, 2018 - 06:05:26
Django version 2.0.4, using settings 'misitio.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

- Ingresando en la dirección que nos brinda podemos ver nuestro proyecto corriendo en un servidor local

django

[View release notes](#) for Django 2.0



The install worked successfully! Congratulations!

You are seeing this page because `DEBUG=True` is in your settings file and you have not configured any URLs.



[Django Documentation](#)

Topics, references, & how-to's



[Tutorial: A Polling App](#)

Get started with Django

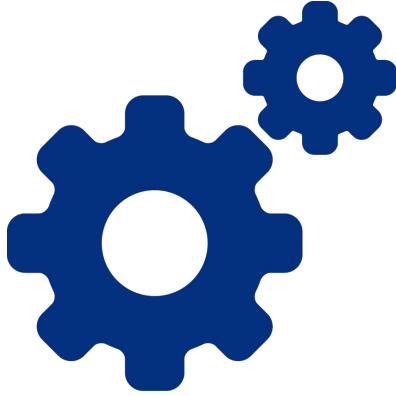


[Django Community](#)

Connect, get help, or contribute

- Iniciado el servidor de desarrollo de Django, un servidor web ligero escrito puramente en Python. **Nota: Jamás usar para ambientes de producción.**





**¡A configurar nuestro proyecto!**

# Configurando allowed hosts en `settings.py`

La lista `ALLOWED_HOSTS` nos ayudará a indicar con qué direcciones vamos a poder acceder a nuestro sitio en el servidor local, estas deben ser una dirección IP o nombre de dominio.

```
ALLOWED_HOSTS = ['127.0.0.1', 'localhost']
```

# Configurando idioma y zona horaria en **settings.py**

En cuanto al idioma con el cual se a mostrar nuestro sitio vamos a ingresar el español de méxico y de igual forma la zona horaria:

```
LANGUAGE_CODE = 'es-mx'  
TIME_ZONE = 'America/Mexico_City'
```

# Configurando directorio de templates en **settings.py**

Para que django pueda ubicar el lugar donde se van a localizar los templates hay que indicarselo en la lista que se encuentra en templates de la siguiente forma:

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

Habiendo realizado las configuraciones anteriores nuestro proyecto está listo para utilizarse. Ahora veremos cómo migrar los modelos que Django nos ofrece para registrar usuarios, los comandos que utilizaremos son los siguientes:

Para realizar las migraciones(si ya existe información en los modelos):

```
$ python manage.py makemigrations
```

Para migrar(crear los modelos/tablas en la base de datos):

```
$ python manage.py migrate
```

```
kubos@kubos:~/Escritorio/django_julio2018/Lunes/misitio$ python manage.py makemigrations
No changes detected
kubos@kubos:~/Escritorio/django_julio2018/Lunes/misitio$ python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying sessions.0001_initial... OK
kubos@kubos:~/Escritorio/django_julio2018/Lunes/misitio$
```

# Creando un super usuario



Un super usuario, es aquel usuario que tiene permisos para acceder al administrador de django, crear nuevos registros(usuarios o información de otros modelos), actualizarlos así como también eliminarlos. Para crear un super usuario utilizaremos el siguiente subcomando:

```
$ python manage.py createsuperuser
```

```
kubos@kubos:~/Escritorio/django_julio2018/Lunes/misitio$ python manage.py createsuperuser
Username (leave blank to use 'kubos'):: jorgechavez
Email address: jorgechavez.proteco@gmail.com
Password:
Password (again):
Superuser created successfully.
kubos@kubos:~/Escritorio/django_julio2018/Lunes/misitio$
```

Ya teniendo nuestro superusuario, podemos ingresar al administrador que nos ofrece django en nuestro navegador preferido con ayuda de la siguiente url:

**localhost:8000/admin**



Identificarse | Sitio de administración de Django - Google Chrome

Identificarse | Sitio de administración de Django

localhost:8000/admin/login/?next=/admin/

### Administración de Django

Nombre de usuario:

Contraseña:

IDENTIFICARSE

Administración del sitio | Sitio de administración de Django - Google Chrome

Administración del sitio

localhost:8000/admin/

## Administración de Django

BIENVENIDO, **JORGECHAVEZ**. [VIEW SITE](#) / [CAMBIAR CONTRASEÑA](#) / [CERRAR SESIÓN](#)

### Administración del sitio

#### AUTENTICACIÓN Y AUTORIZACIÓN

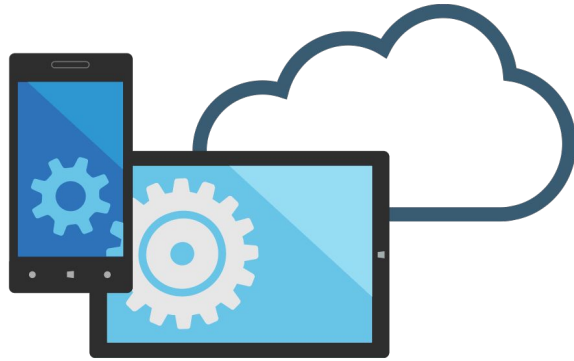
<b>Grupos</b>	<a href="#">+ Agregar</a>	<a href="#">✎ Modificar</a>
<b>Usuarios</b>	<a href="#">+ Agregar</a>	<a href="#">✎ Modificar</a>

#### Recent actions

#### My actions

Ninguna disponible

# ¡A crear nuestra primera app!



# Creando una aplicación(App vs Proyecto)

¿Cuál es la diferencia entre un proyecto y una aplicación? Una **app** es una aplicación web que hace algo, por ejemplo, un sistema de inicio de sesión, una base de datos de registros o una aplicación de encuesta simple.

Un **proyecto** es un conjunto de configuraciones y aplicaciones para un sitio web determinado. Un proyecto puede contener aplicaciones múltiples. Una aplicación puede estar en varios proyectos.

# Creando una aplicación

Para crear una aplicación vamos a ejecutar el siguiente subcomando:

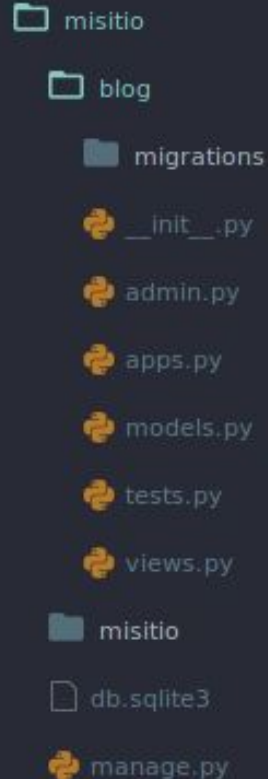
```
$ python manage.py startapp blog
```

**\*Puede ser cualquier nombre(referente a lo que hará la app)**



# Estructura de una aplicación

## FOLDERS



**admin.py:** Aquí se registrarán los modelos escritos en el archivo models.py.

**views.py:** Aquí se escribirán las vistas las cuales tendrán el funcionamiento de la aplicación y renderizan los templates.

**test.py:** Utilizado para realizar pruebas.

**apps.py:** Sirve para configurar algunos atributos de la aplicación.

# Registrando nuestra aplicación en `settings.py`

Para obtener acceso a la aplicación en cualquier otra parte de nuestro sitio, debemos agregar nuestra aplicación a la lista de aplicaciones de Django, que se encuentra en el archivo `settings.py` de la siguiente manera:

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
    'blog',  
]
```



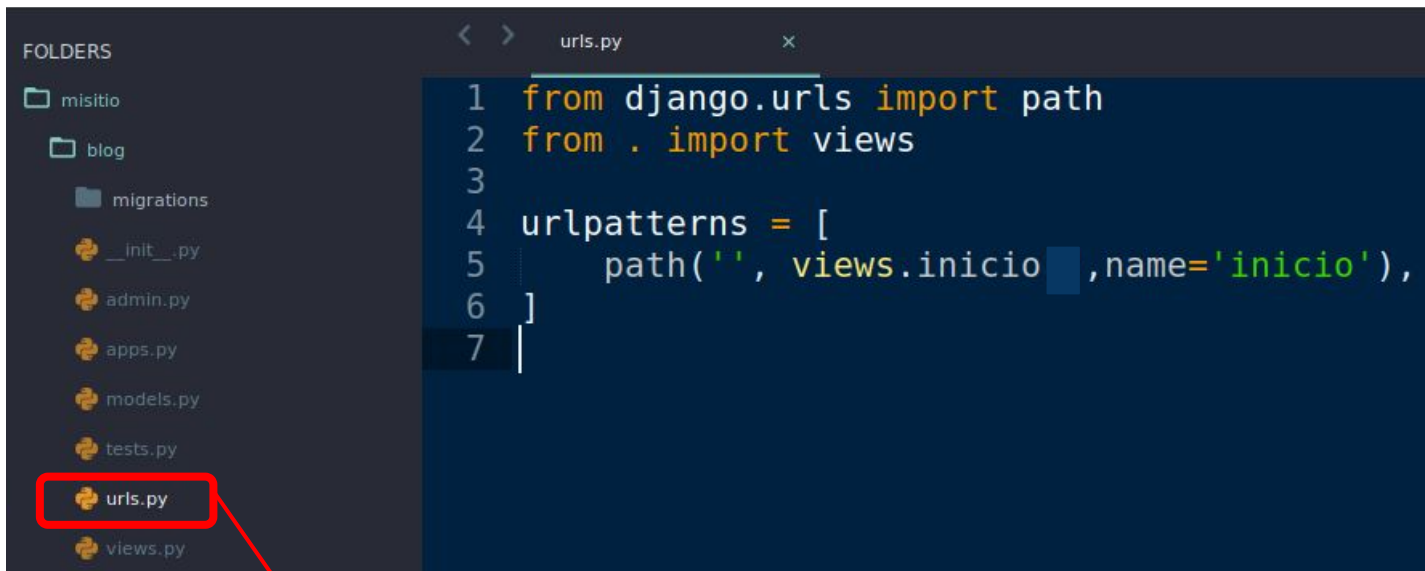
**¡A configurar las urls!**

# URL's(Uniform Resource Locators)

Las URLs o localizadores uniformes de recursos, nos van a permitir cargar nuestras páginas(las vistas que a su vez renderizan los templates) de acuerdo al nombre que les asignemos, es recomendable crear un archivo **urls.py** por cada aplicación que se cree en el proyecto y posteriormente incluirlas en el archivo **urls.py** general.



# Creando el archivo `urls.py` para la aplicación



The screenshot shows a code editor with a dark theme. On the left, a file explorer under the heading 'FOLDERS' shows a project structure with a 'blog' folder containing 'migrations' and several Python files. The file 'urls.py' is highlighted with a red rectangle, and a red arrow points from it to the text below. The main editor area shows the content of 'urls.py' with line numbers 1 through 7. The code imports 'path' from 'django.urls' and 'views' from the current directory. It then defines a list 'urlpatterns' containing a single path for 'views.inicio' with the name 'inicio'.

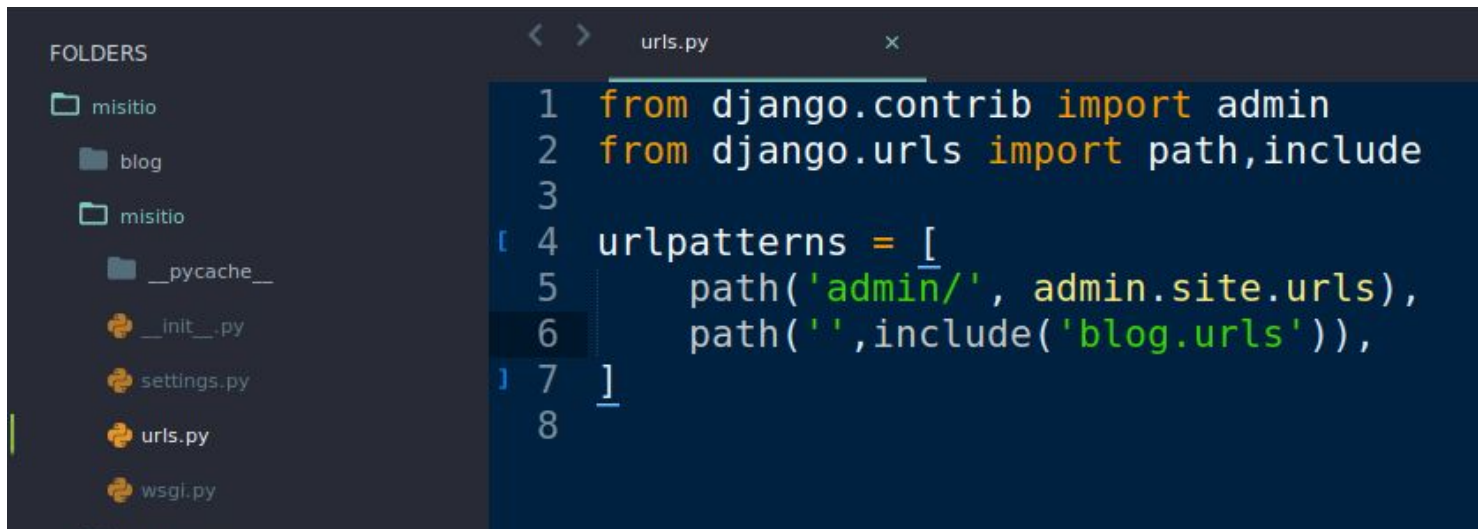
```
1 from django.urls import path
2 from . import views
3
4 urlpatterns = [
5     path('', views.inicio, name='inicio'),
6 ]
7
```

Contenido  
del archivo

Archivo `urls.py` creado(no esta predeterminado al crear la aplicación)

# Incluyendo las urls de mi aplicación

Ya tenemos las urls de nuestra aplicación, ahora tenemos que incluirlas en nuestro archivo urls.py general de la siguiente forma:



The screenshot shows a code editor with a dark theme. On the left, a sidebar titled 'FOLDERS' displays the project structure: a 'misitio' folder containing a 'blog' subfolder, another 'misitio' folder, and files like '\_\_pycache\_\_', '\_\_init\_\_.py', 'settings.py', 'urls.py' (highlighted), and 'wsgi.py'. The main editor area shows the 'urls.py' file with the following Python code:

```
1 from django.contrib import admin
2 from django.urls import path, include
3
4 urlpatterns = [
5     path('admin/', admin.site.urls),
6     path('', include('blog.urls')),
7 ]
8
```

En la páginas anteriores vimos la creación del archivo `urls.py` y su contenido, pero hemos escrito algo que aún no hemos creado, ¿Te diste cuenta? ¡Sí!, son las vistas. En Django existen vistas basadas en funciones y basadas en clases, comenzaremos por ver vistas con funciones.

# Vistas basadas en funciones

Para crear una vista, basta con crear la función y pasar como parámetro el **request**(solicitud que nos va a permitir enviar y recibir información del servidor siendo nosotros los clientes) esta función va a retornar una función la cuál se encargará de renderizar, es decir interpretar la solicitud y junto con ella verificar si hay una consulta a la base de datos, si la hay la realizará, en caso contrario nos enviará el **template**.

**¡Escribamos nuestra primera vista!**

## FOLDERS

misitio

blog

migrations

\_\_init\_\_.py

admin.py

apps.py

models.py

tests.py

urls.py

views.py



views.py

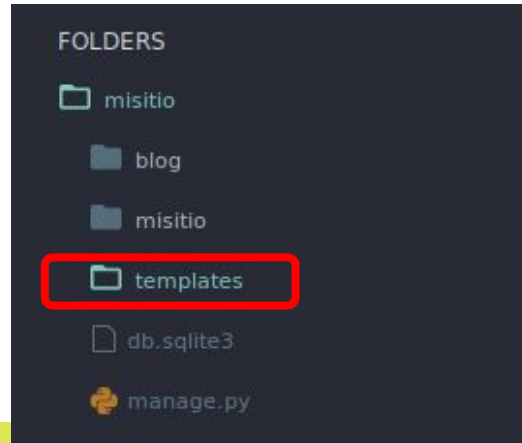


```
1 from django.shortcuts import render
2
3 # Create your views here.
4
5 def inicio(request):
6     return render(request, 'blog/index.html')
7
8
```

# ¿Templates?



Sí, viste bien. No hemos creado ningún template pero ya tenemos configurada la lista. Vamos a proceder a crear un nuevo directorio llamado templates dentro de nuestro proyecto, de forma que nuestras carpetas se verán así:



# Creando nuestro primer template

En cuanto a los templates, se recomienda tener una carpeta contenedora de los templates de cada aplicación que creemos, en este caso crearemos una carpeta llamada blog dentro de templates, y dentro colocaremos nuestro primera archivo HTML, con el nombre de *index.html*

## FOLDERS

misitio

blog

misitio

templates

blog

index.html

db.sqlite3

manage.py

index.html

```
1 <!DOCTYPE html> <!--Indica la versión de HTML, en este caso la 5-->
2 <html lang="es"> <!--Inicio del documento HTML con lenguaje español-->
3 <head> <!--Etiqueta para incluir metadatos,hojas de estilo entre otro-->
4     <meta charset="UTF-8"> <!--Incluyendo la codificación UTF-8 -->
5     <title>Mi blog</title> <!--Título de la pestaña en el navegador-->
6 </head> <!--Fin de la etiqueta de metadatos-->
7 <body> <!--Inicia el cuerpo de nuestro sitio web-->
8     <p>Hola mundo de Django 2.0</p> <!--Parrafo-->
9 </body><!--Termina el cuerpo de nuestro sitio web-->
10 </html> <!--Fin del documento HTML-->
```



# HTML

Los templates se escriben con HTML5, el cual es un lenguaje de marcado de hipertexto, lo que quiere decir que usaremos etiquetas.



# ¡A ver nuestro sitio!



# Se ve feo, pero funciona



Hasta este momento ya hemos configurado el proyecto, creado una aplicación y configurado las urls, sin embargo, si notas algo extraño, eso no se ve nada bonito. Para darle estilo a nuestro sitio y que se vea cool, vamos a configurar lo que se conoce como archivos estáticos(CSS, imágenes, documentos etc.)

Esta historia continuará...