# Documentación Django (Phyton)

# Tabla de contenidos

# Introducción

Tabla de contenidos

Django es un framework web de alto nivel de desarrollo web de código abierto, escrito en Python, que respeta el patrón de diseño conocido como modelo–vista–controlador (MVC).

El modelo-vista-controlador es un patrón de arquitectura de software, que separa los datos y la lógica de negocio de una aplicación de su representación y el módulo encargado de gestionar los eventos y las comunicaciones.

- **Modelo**: Es la representación de la información con la cual el sistema opera, por lo tanto gestiona todos los accesos a dicha información, tanto consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación (lógica de negocio).
- **Vista**: Presenta el modelo en un formato adecuado para interactuar (usualmente la interfaz de usuario) y también puede encargarse de filtrar la entrada de datos y enviarla al modelo.
- **Controlador**: Responde a eventos (usualmente acciones del usuario) e invoca peticiones al modelo cuando se hace alguna solicitud de información (por ejemplo, editar un documento o un registro en una base de datos).

## Instalación en Linux

Tabla de contenidos

Python viene instalado por defecto en la mayoría de distribuciones GNU/Linux. Para comprobar si lo tenemos instalado, abrimos una terminal y escribimos:

```
python3 --version
```

En el caso de que no lo tengamos instalado, podemos instalarlo desde los repositorios oficiales de nuestra distribución. En el caso de Ubuntu, escribimos:

```
sudo apt install python3
```

Una vez verificado que tenemos instalado Python podemos empezar a instalar Django primero debemos instalar python3.10-venv para poder crear entornos virtuales. Para ello, escribimos:

```
sudo apt install python3.10-venv
```

Una vez instalado, creamos un entorno virtual llamado mysite y lo activamos:

> [!NOTE] El entorno virtual se puede llamar como queramos, pero es recomendable llamarlo como el proyecto que vamos a crear. Y un entorno virtual es un entorno aislado donde podemos instalar paquetes de Python sin afectar al sistema operativo.
>
> Ademas es recomendable crear un entorno virtual para cada proyecto que vayamos a crear.
>
> Al crear un entorno virtual, se crea un directorio con el nombre del entorno virtual. En nuestro caso, se creará un directorio llamado mysite. Hay que tener en cuenta que el entorno virtual se crea en el directorio donde nos encontremos en la terminal y es donde se instalaran las bibliotecas de Django.

```
$ python3 -m venv mysite
$ source mysite/bin/activate
(mysite)$ pip install django
(mysite)$ python -m django --version
4.2.7
```

## Explicación de los comandos

- **python3 -m venv mysite**: Creamos un entorno virtual llamado mysite.
- **source mysite/bin/activate**: Activamos el entorno virtual esto hay que hacerlo cada vez que queramos trabajar con Django, hay que tener en cuenta que la ruta varia dependiendo de como llames a tu entorno virtual en mi caso lo e llamado mysite pues es mysite/bin/activate si fuera por ejemplo otroNombre seria otroNombre/bin/activate.
- **pip install django**: Instalamos Django.
- **python -m django --version**: Comprobamos la versión de Django instalada.

Si en la terminal nos aparece (mysite) delante del nombre de nuestro usuario, significa que el entorno virtual está activado. En caso contrario, debemos activarlo con el comando source mysite/bin/activate. Para desactivar el entorno virtual, escribimos deactivate.

# Creación de un proyecto

Tabla de contenidos

En primer lugar vamos a ver como crear un proyecto Django. Para ello, nos situamos en el directorio donde queremos crear el proyecto y escribimos:

```
(mysite)$ django-admin startproject HelloWorld
```

Esto creará un directorio llamado HelloWorld con la siguiente estructura:

```
HelloWorld/
    manage.py
    HelloWorld/
        __init__.py
        settings.py
        urls.py
        asgi.py
        wsgi.py
```

## Explicación de los ficheros al crear un proyecto Django

Tabla de contenidos

- **manage.py**: Es un script que ayuda con la gestión del sitio. Con él podemos arrancar un servidor de desarrollo, crear aplicaciones, crear migraciones de la base de datos, etc.
- El directorio **HelloWorld/** es un paquete de Python para nuestro proyecto.
- **settings.py**: Contiene la configuración del proyecto.
- **urls.py**: Contiene las definiciones de las URLs del proyecto.
- **wsgi.py**: Es un punto de entrada para los servidores web compatibles con WSGI para servir el proyecto.
- **asgi.py**: Es un punto de entrada para los servidores web compatibles con ASGI para servir el proyecto.

**Configuración del fichero settings.py**

En el fichero settings.py podemos configurar nuestro proyecto. En este fichero podemos configurar la base de datos, el idioma, la zona horaria, etc.

Por defecto el fichero settings.py viene configurado de la siguiente manera:

- **DATABASES**: Configuración de la base de datos que se va a utilizar en el proyecto. Por defecto se utiliza una base de datos sqlite llamada db.sqlite3.
- **INSTALLED_APPS**: La lista de las aplicaciones que tiene instalada el proyecto, por ejemplo vemos que se ha incluido la aplicación polls (polls.apps.PollsConfig). También tenemos una aplicación que nos permite tener un panel de control de la aplicación (django.contrib.admin). Y otras cuantas aplicaciones…
- **DEBUG**: True: Si está activo los errores que se produzcan en la aplicación se verán con todo lujo de detalles en el navegador. Si tenemos la aplicación en producción debería ser False.
- **ALLOWED_HOSTS**: Es una lista de cadenas que especifica los nombres de host válidos para el sitio.

**Configuración del fichero urls.py**

En el fichero urls.py podemos configurar las URLs de nuestro proyecto. En este fichero podemos añadir las URLs de las aplicaciones que vayamos creando.

## Ejecución del servidor de desarrollo

Tabla de contenidos

Para arrancar el servidor de desarrollo, nos situamos en el directorio donde se encuentra el fichero manage.py y escribimos:

```
(mysite)$ python manage.py runserver
```

> [!CAUTION] Si al ejecutar el comando de **python manage.py runserver** nos aparece en la consola un mensaje de error como este:
>
> You have 18 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.
>
> Esto significa que tenemos que aplicar las migraciones de la base de datos. Para ello, escribimos:
>
> ```
> (mysite)$ python manage.py migrate
> ```

> Las migraciones son como una versión controlada de tu base de datos, y Django las usa para crear, modificar y eliminar tablas y sus campos.
>
> Cuando creas modelos o cambias tus modelos existentes, Django genera estas migraciones. Sin embargo, estos cambios no se reflejan en tu base de datos hasta que apliques las migraciones.
>
> El mensaje te está indicando que debes aplicar estas migraciones para que tu proyecto funcione correctamente.

Ahora si todo a ido bien, podemos acceder a nuestro servidor de desarrollo desde un navegador web en la dirección http://127.0.0.1:8000/ en mi caso esa es la dirección, pero puede variar en función de la configuración de nuestro equipo. Si todo ha ido bien, veremos una página de bienvenida de Django.

Ahora bien si queremos que el servidor de desarrollo sea accesible desde cualquier dirección IP y ademas en un puerto determinado, escribimos:

```
(mysite)$ python manage.py runserver 0.0.0.0:8000
```

**Primeros pasos con Django (Entrar con el usuario admin)**

Para entrar en el panel de administración de Django, debemos crear un superusuario. Para ello, escribimos:

```
(mysite)$ python manage.py createsuperuser
```

> [!NOTE] Tienes que estar situado en la carpeta donde se encuentra el fichero manage.py para poder ejecutar el comando.

Esto nos pedirá un nombre de usuario, una dirección de correo electrónico y una contraseña. Una vez creado el superusuario, podemos acceder al panel de administración de Django en la dirección http://127.0.0.1:8000/admin/ en mi caso esa es la dirección, pero puede variar en función de la configuración de nuestro equipo. Si todo ha ido bien, veremos una página de login de Django.

## Instalación del IDE PyCharm

Tabla de contenidos

Ahora que tenemos todo lo necesario para desarrollar nuestra aplicacion web con Django vamos a instalar PyCharm que es un IDE multiplataforma creado por JetBrains. Es uno de los mejores IDEs para proyectos que utilizan el lenguaje de programación Python, primero debemos descargarlo desde la página oficial de JetBrains. Una vez descargado, nos situamos en el directorio donde se encuentra el fichero descargado y escribimos:

```
$ tar -xzf pycharm-2023.2.5.tar.gz
```

Esto creará un directorio llamado pycharm-2023.2.5 con la siguiente estructura:

```
pycharm-2023.2.5/
    bin/
    lib/
    jbr/
    license/
    plugins/
    help/
    debug-eggs/
```

Ahora para ejecutar PyCharm, nos situamos en el directorio pycharm-2023.2.5/bin y escribimos:

```
$ ./pycharm.sh
```

Esto nos abrirá una ventana de bienvenida de PyCharm, donde podemos crear un proyecto nuevo o abrir uno existente.

Es recomendable crear un alias para ejecutar PyCharm desde cualquier directorio. Para ello, abrimos una terminal y escribimos:

```
$ sudo nano ~/.bashrc
```

Esto nos abrirá el fichero .bashrc en el editor de texto nano. Ahora debemos añadir la siguiente línea al final del fichero:

```
alias pycharm="{tu sitio de instalacion}/pycharm-2023.2.5/bin/pycharm.sh"
```

Ahora guardamos los cambios y cerramos el editor de texto. Para que los cambios surtan efecto, escribimos:

```
$ source ~/.bashrc
```

Ahora podemos ejecutar PyCharm desde cualquier directorio escribiendo pycharm en la terminal.

### Configuración de PyCharm

Tabla de contenidos

Una vez abierto PyCharm, nos aparecerá una ventana de bienvenida. En ella, podemos crear un proyecto nuevo o abrir uno existente. En nuestro caso, vamos a abrir el proyecto anteriormente creado **mysyte**. Para ello, abrimos la carpeta contenedora del proyecto.

Una vez abierto el proyecto, vamos a configurar el intérprete de Python. Para ello, nos situamos en **File > Settings > Project: mysite > Python Interpreter**. En el desplegable, seleccionamos el intérprete de Python que hemos instalado anteriormente.

> [!IMPORTANT] Es importante que el intérprete de Python que seleccionemos sea el que hemos instalado en el entorno virtual. De lo contrario, no nos reconocerá las bibliotecas de Django y asegurarse de encontrar el ejecutable python que hay dentro de la carpeta bin dentro de la carpeta creada al crear el entorno virtual.

Si no aparece en el desplegable activamos el entorno virtual anteriormente creado y nos vamos a la carpeta mysite donde lo hayas creado y hay es donde aparece instalado Django **dentro de la carpeta bin**, debemos añadirlo. Para ello, sin movernos de la ventana Python Interpreter le damos a **Add Interpreter** que debe aparecer al lado. En la ventana que se nos abre, seleccionamos **Add Local Interpreter** y pulsamos en **Existing**. En la siguiente ventana, seleccionamos el intérprete de Python que ya existe y tenemos instalado de antes. En mi caso la ruta en donde e creado mi entorno virtual y instalado Django es en **../Documentos/Pycharm/mysite/bin/python** que es lo que necesitamos para configurar el intérprete y que nos reconozca las bibliotecas de Django.

Ahora vamos a crear un perfil para ello con que ejecutemos el programa nos saldra una ventana para añadir un perfil en el que tendremos que poner el host y el puerto en el que queremos que se ejecute el servidor de desarrollo de Django.

Lo guardamos y ya podemos ejecutar el servidor de desarrollo de Django desde PyCharm.

# Primeros pasos con Python (Hola Mundo)

Tabla de contenidos

Llegados a este punto, tenemos todo lo necesario para empezar a desarrollar nuestra aplicación web con Django. Pero antes de empezar, vamos a ver algunos conceptos básicos de Python.

En primer lugar vamos a crear una aplicación Django. Para ello, nos situamos en el directorio donde se encuentra el fichero manage.py y escribimos:

```
$ python manage.py startapp prueba
```

Esto creará un directorio llamado prueba con la siguiente estructura:

```
prueba/
    __init__.py
    admin.py
    apps.py
    migrations/
        __init__.py
    models.py
    tests.py
    views.py
```

Ahora vamos a crear una vista. Para ello, abrimos el fichero views.py y escribimos:

```
from django.http import HttpResponse

def index(request):
    return HttpResponse("Hola Mundo")
```

Ahora vamos a crear un archivo llamado urls.py en la carpeta prueba y luego abrimos el fichero urls.py y escribimos:

```
from django.urls import path

from . import views

urlpatterns = [
    path('', views.index, name='index'),
]
```

Ahora vamos abrir en la carpeta HelloWorld el fichero urls.py y escribimos:

```
from django.contrib import admin
```

```
from django.urls import include, path

urlpatterns = [
    path('prueba/', include('prueba.urls')),
    path('admin/', admin.site.urls),
]
```

Si todo va bien y hemos seguido los pasos hasta ahora la estructura de nuestro proyecto debería ser la siguiente:

```
HelloWorld/
    manage.py
    HelloWorld/
        __init__.py
        settings.py
        urls.py
        asgi.py
        wsgi.py
    prueba/
        __init__.py
        admin.py
        apps.py
        migrations/
            __init__.py
        models.py
        tests.py
        views.py
```

Ahora vamos a ejecutar el servidor de desarrollo de Django. Y si todo ha ido bien, podemos acceder a nuestro servidor de desarrollo desde un navegador web en la dirección http://127.0.0.1:8000/prueba/ en mi caso esa es la dirección, pero puede variar en función de la configuración de nuestro equipo. Si todo ha ido bien, veremos el mensaje Hola Mundo.

## Sintaxis basica de Python

Tabla de contenidos

La sintaxis de Python es muy sencilla y fácil de aprender. En Python, el código se agrupa en bloques indentados, no se utilizan llaves para delimitar los bloques de código. Los comentarios comienzan con el carácter # y se extienden hasta el final de la línea.

```
# Esto es un comentario
```

A continuación, vamos a ver algunos conceptos básicos de Python.

## Variables en Python

En Python, las variables se crean cuando se les asigna un valor. No es necesario declararlas antes de usarlas o declarar su tipo. El tipo de la variable se determina cuando se le asigna un valor.

```
x = 5
y = "Hola Mundo"
```

## Tipos de datos en Python

En Python, los datos se pueden almacenar en diferentes tipos de variables. Los tipos de datos más comunes son:

- **Texto**: str
- **Entero**: int
- **Decimal**: float
- **Complejo**: complex
- **Booleano**: bool
- **Lista**: list
- **Ninguno**: NoneType
- **Tupla**: tuple
- **Rango**: range
- **Diccionario**: dict
- **Conjunto**: set
- **Frozenset**: frozenset
- **Bytes**: bytes
- **Bytearray**: bytearray
- **Memoria de vista**: memoryview

Ahora vamos a ver algunos ejemplos de tipos de datos en Python.

```
 x = "Hola Mundo" # str
# Texto (str)
texto = "Hola, mundo!"

# Entero (int)
entero = 10

# Decimal (float)
decimal = 3.14

# Complejo (complex) -> Se utiliza para representar números complejos
(números con una parte real e imaginaria).
```

```
complejo = 1+2j

# Booleano (bool)
booleano = True

# Lista (list) -> Se utiliza para almacenar varios elementos en una sola
variable.
lista = [1, 2, 3, 4, 5]

# Ninguno (NoneType) -> Se utiliza para representar un valor nulo o que no
existe.
ninguno = None

# Tupla (tuple) -> Es similar a una lista pero no se puede modificar
tupla = (1, 2, 3)

# Rango (range) -> Se utiliza para generar una secuencia de números que no
se puede modificar.
rango = range(10)

# Diccionario (dict) -> Se utiliza para almacenar pares clave: valor.
diccionario = {"nombre": "Juan", "edad": 30}

# Conjunto (set) -> Se utiliza para representar una colección no ordenada
de elementos unicos.
conjunto = {1, 2, 3, 4, 5}

# Frozenset (frozenset) -> Es similar a un conjunto pero no se puede
modificar.
frozenset_ = frozenset([1, 2, 3, 4, 5])

# Bytes (bytes) -> Se utiliza para almacenar una secuencia inmutable de
números en el rango 0 <= x < 256.
bytes_ = b"Hola mundo"

# Bytearray (bytearray) -> Es similar a bytes pero se puede modificar.
bytearray_ = bytearray([119, 51, 114, 100])

# Memoria de vista (memoryview) -> Se utiliza para acceder al subconjunto
de los datos de un objeto mutable.
memoria_de_vista = memoryview(bytes(5))
```

## Operadores en Python

Los operadores se utilizan para realizar operaciones en variables y valores.

- **Aritméticos**: +, -, *, /, %, **, //
- **Asignación**: =, +=, -=, *=, /=, %=, //=, **=
- **Comparación**: ==, !=, >, <, >=, <=
- **Lógicos**: and, or, not
- **Operadores de identidad**: &, |, ^, ~, <<, >>