DJANGO

Tabla de contenidos

Introducción

Tabla de contenidos

Django es un framework gratuito, es un marco de desarrollo de web de alto nivel y de código abierto que fomenta el desarrollo rápido y limpio en Python.

Las características claves de Django se centran en la eficiencia y simplicidad. Proporciona un conjunto robusto de herramientas y componentes que permiten a los desarrolladores construir aplicaciones web de manera rápida y con un código claro y mantenible.

El marco sigue el patrón de diseño de Modelo-Vista-Controlador(MVC), aunque en Django se le conoce más como Modelo-Vista-Plantilla(MVT). En este patrón, el modelo representa los datos y la lógica de la aplicación, la vista se encarga de la presentación y la plantilla maneja la representación de los datos. La filosofia de Django se basa en la simplicidad, la reutilización de código y el prinicpio de no te repitas.

Django está diseñado para ser rápido, seguro y escalable. Ofrece una serie de características que facilitan el desarrollo de aplicaciones complejas, como por ejemplo:

- Un sistema de plantillas potente y flexible que permite crear interfaces de usuario atractuvas y fáciles de usar.
- Un sistema de gestión de base de datos integrado que facilita el acceso a los datos.
- Una serie de librerías y herramientas que facilitan el desarrollo de tareas comunes, como la autenticacióin de usuarios, la gestión de errores y la depuración.

Django es una buena opción para el desarrollo de aplicaciones web de cualquier tipo, pero es especialmente adeacuado para aplicaciones web controladas por datos, como sitios de comercio electrónico blogs y aplicaciones de gestión.

Instalación Django

Tabla de contenidos

Para instalar Django, necesitarás cumplir con algunos requisitos previos. A continuación, se detallan los pasos básicos para instalar Django en un entorno de desarrollo.

- Requisitos para instalar Django.
    - Actualizar el sistema: para ello debemos abrir la terminal y escribir el siguiente fragmento de codigo en la terminal.

```
sudo apt update
sudo apt upgrade
```

    - Comprobamos la versión de python que tenemos.

```
pythom3 --version
```

- Si no tenemos python instalado en nuestro equipo debemos escribir en la terminal el siguiente código:

```
sudo apt install python3
```

Tenemos que tener en cuenta que la versión de python que tenemos instalada debe ser la 3.6 o superior.

- Ahora empezaremos con la isntalación de django.
  - Primero crearemos una carpeta donde vamos a establecer nuestro entorno gráfico de django. En mi caso la creare en documentos.

```
mkdir django
```

  - Nos moveremos a dicha carpeta:

```
cd django
```

  - Crear un entorno virtual llamado "django":

```
python3 -m venv django
```

Este comando crea un entorno virtual de Python llamado "django". Un entorno virtual es un ambiente aislado donde puedes instalar paquetes de Python sin afectar a otros proyectos o al sistema Python global.

  - Instalar el paquete python3.10-venv:

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

Este comando instala el módulo venv para Python 3.10 en un sistema operativo basado en Debian, como Ubuntu. venv es un módulo que viene con Python 3.3 y versiones posteriores, y se utiliza para crear entornos virtuales.

  - Crear nuevamente un entorno virtual llamado "django"

```
python3 -m venv django
```

Aquí, estás volviendo a crear el entorno virtual. Si ya creaste uno en el primer paso, este comando no es necesario. Puedes omitirlo si ya tienes un entorno virtual creado.

- Instalar Django dentro del entorno virtual:

```
pip install Django
```

El comando pip install Django se utiliza para instalar Django, que es un marco de trabajo de desarrollo web de alto nivel en Python. pip es el gestor de paquetes de Python y se utiliza para instalar y administrar paquetes de software escritos en Python.

- Comprobamos la versión de Django que tenemos

```
python3 -m django --version
```

- Crear un proyecto de Django llamado "hola_mundo":

```
django-admin startproject hola_mundo
```

Este comando crea un proyecto de Django llamado "hola_mundo". Un proyecto de Django es una colección de configuraciones y aplicaciones para un sitio web específico. Django viene con un comando de administración que se utiliza para crear proyectos de Django.

hola_mundo/ manage.py hola_mundo/ **init**.py settings.py urls.py asgi.py wsgi.py

- Explicamos que consiste cada archivo que se ha creado mediante comando:
    - hola_munsdo/: es el directorio raíz del proyecto. Contiene el archivo manage.py y el paquete hola_mundo.
    - manage.py: es un script de utilidad que se utiliza para administrar el proyecto. Con manage.py, puedes iniciar un servidor de desarrollo, crear y aplicar migraciones de base de datos, crear usuarios, etc.
    - hola_mundo/: es un paquete de Python que contiene los archivos de configuración del proyecto.
    - **init**.py: es un archivo vacío que indica a Python que el directorio es un paquete de Python.
    - settings.py: contiene la configuración del proyecto.
    - urls.py: contiene las URL del proyecto.
    - asgi.py: contiene la configuración para el servidor ASGI.
    - wsgi.py: contiene la configuración para el servidor WSGI.

- Moverse al directorio "hola_mundo":

```
cd hola_mundo
```

Este comando te mueve al directorio "hola_mundo" que se creó en el paso anterior.

- Ejecutar el servidor de desarrollo:

```
python3 manage.py runserver
```

Este comando ejecuta el servidor de desarrollo de Django. El servidor de desarrollo es un servidor web ligero que se utiliza para probar tu aplicación web durante el desarrollo. Por defecto, el servidor de desarrollo se ejecuta en el puerto 8000.

- Abrir el navegador web y navegar a http://localhost:8000:

```
http://localhost:8000
```

Este comando abre el navegador web y navega a http://localhost:8000. El servidor de desarrollo de Django se ejecuta en el puerto 8000 de forma predeterminada. Si el puerto 8000 está ocupado, puedes especificar un puerto diferente con el siguiente comando:

```
python3 manage.py runserver 8080
```

Este comando ejecuta el servidor de desarrollo en el puerto 8080.

- Detener el servidor de desarrollo:

```
Ctrl + C
```

Este comando detiene el servidor de desarrollo de Django.

- Podemos cambiar los puertos de escucha de nuestro servidor de desarrollo, para ello debemos escribir el siguiente comando:

```
python3 manage.py runserver 8080
```

Este comando ejecuta el servidor de desarrollo en el puerto 8080.

Creando la aplicación hola mundo:

- Crear una aplicación de Django llamada "hola_mundo":

```
python3 manage.py startapp hola_mundo
```

Este comando crea una aplicación de Django llamada "hola_mundo". Una aplicación de Django es un conjunto de código que se utiliza para realizar una tarea específica. Por ejemplo, una aplicación de Django puede ser un blog, un sitio web de comercio electrónico, un sitio web de noticias, etc.

- Los archivos que se han creado son los siguientes:

```
hola_mundo/
    manage.py
    hola_mundo/
        __init__.py
        settings.py
        urls.py
        asgi.py
        wsgi.py
    hola_mundo/
        __init__.py
        admin.py
        apps.py
        models.py
        tests.py
        views.py
```

- Explicamos que consiste cada archivo que se ha creado mediante comando:
    - hola_mundo/: es el directorio raíz de la aplicación. Contiene el archivo models.py y el paquete hola_mundo.
    - **init**.py: es un archivo vacío que indica a Python que el directorio es un paquete de Python.
    - admin.py: contiene la configuración para la interfaz de administración.
    - apps.py: contiene la configuración de la aplicación.
    - models.py: contiene los modelos de la aplicación.
    - tests.py: contiene las pruebas de la aplicación.
    - views.py: contiene las vistas de la aplicación.
- Abrir el archivo hola_mundo/settings.py:
- Añadir 'hola_mundo' a la lista INSTALLED_APPS:

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

```
    'hola_mundo',
]
```

Este comando añade la aplicación "hola_mundo" a la lista INSTALLED_APPS en el archivo hola_mundo/settings.py. INSTALLED_APPS es una lista de aplicaciones de Django que se utilizan en el proyecto.

- Abrir el archivo hola_mundo/urls.py:
- Añadir la siguiente línea al archivo hola_mundo/urls.py:

```
path('', include('hola_mundo.urls')),
```

Este comando añade la ruta de la aplicación "hola_mundo" al archivo hola_mundo/urls.py. La ruta de la aplicación "hola_mundo" se añade a la lista urlpatterns en el archivo hola_mundo/urls.py. urlpatterns es una lista de rutas de URL que se utilizan en el proyecto.

- Crear un archivo llamado urls.py en el directorio hola_mundo:

```
touch hola_mundo/urls.py
```

Este comando crea un archivo llamado urls.py en el directorio hola_mundo.

- Abrir el archivo hola_mundo/urls.py:
- Añadir el siguiente código al archivo hola_mundo/urls.py:

```
from django.urls import path
from . import views

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

Este comando añade la ruta de la aplicación "hola_mundo" al archivo hola_mundo/urls.py. La ruta de la aplicación "hola_mundo" se añade a la lista urlpatterns en el archivo hola_mundo/urls.py. urlpatterns es una lista de rutas de URL que se utilizan en el proyecto.

- Crear un archivo llamado views.py en el directorio hola_mundo:

```
touch hola_mundo/views.py
```

Este comando crea un archivo llamado views.py en el directorio hola_mundo.

- Abrir el archivo hola_mundo/views.py:

- Añadir el siguiente código al archivo hola_mundo/views.py:

```
from django.http import HttpResponse

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

Este comando añade una vista llamada index al archivo hola_mundo/views.py. Una vista es una función de Python que procesa una solicitud web y devuelve una respuesta web. En este caso, la vista index devuelve una respuesta web que dice "Hola Mundo!".

- Ejecutar el servidor de desarrollo:

```
python3 manage.py runserver
```

Este comando ejecuta el servidor de desarrollo de Django. El servidor de desarrollo es un servidor web ligero que se utiliza para probar tu aplicación web durante el desarrollo. Por defecto, el servidor de desarrollo se ejecuta en el puerto 8000.

- Abrir el navegador web y navegar a http://localhost:8000:

```
http://localhost:8000
```

Este comando abre el navegador web y navega a http://localhost:8000. El servidor de desarrollo de Django se ejecuta en el puerto 8000 de forma predeterminada. Si el puerto 8000 está ocupado, puedes especificar un puerto diferente con el siguiente comando:

```
python3 manage.py runserver 8080
```

Este comando ejecuta el servidor de desarrollo en el puerto 8080.

- Detener el servidor de desarrollo:

```
Ctrl + C
```

Este comando detiene el servidor de desarrollo de Django.

Instalación PyCharm

Tabla de contenidos

- Introducción sobre PyCharm:

  Es un entorno de desarrollo integrado (IDE) creado por JetBrains específicamente para el desarrollo de proyectos en Python.

  - Características:

    - Editor de código avanzado: ofrece un editor de código robusto con resaltado de sintaxis, completado automático, refactorización de código y navegación inteligente.
    - Gestión de Proyectos Eficiente: Facilita la creación y gestión de proyectos Python. Ofrece herramientas para la instalación y gestión de paquetes, y permite trabajar con entornos virtuales para aislar las dependencias de cada proyecto.
    - Depuración Integrada: permite establecer puntos de interrupción, inspeccionar variables.
    - Integración con Herramientas Externas: en este caso podríamos incluir un sistema de versión de controles como es git y base de datos como mysql.

- Instalación de PycCharm:

  - Para la instalación necesitamos acceder al siguiente sitio web https://www.jetbrains.com/es-es/pycharm/ y ahi descargar pycharm, una vez descargado escribiremos en consola la siguiente línea de comando:

  Descomprimir el Archivo de PyCharm:

```
tar -zxf pyCharm-professional-2023.3.1.tar.gz
```

  Ahora crearemos un alias para poder acceder a pyCharm más rapido

```
nano source ~/.bashrc
```

  Una vez dentro al final del documento debemos escribir la siguiente línea para crear un alias:

```
alias pyCharm="la ruta donde se encuentra este archivo"pyCharm-
2023.3.1/bin/pycharm.sh
```

  Actualizar la Configuración del Shell:

```
source ~/.bashrc
```

Sintaxis de Django

Tabla de contenidos

Python utiliza la sangría para indicar bloques de código. Django utiliza llaves, corchetes y paréntesis para indicar bloques de código. Por ejemplo, las llaves se utilizan para indicar bloques de código en las plantillas de Django. Los corchetes se utilizan para indicar bloques de código en los archivos de configuración de Django. Los paréntesis se utilizan para indicar bloques de código en las vistas de Django.

Un ejemplo de de sangría en python:

```
if 5 > 2:
  print("Cinco es mayor que dos!")
```

- Comentarios:
    - Comentarios de una línea:

```
{# Este es un comentario de una línea #}
```

    - Comentarios de varias líneas:

```
{% comment %}
  Este es un comentario de varias líneas.
{% endcomment %}
```

- Variables: Una variable se crea en el momento en que se le asigna un valor. Las variables se utilizan para almacenar datos, como cadenas de texto, números enteros, números de punto flotante, etc. En Django, las variables se utilizan para almacenar datos que se utilizan en las plantillas. Las variables de Django se crean utilizando la sintaxis {{variable}}. Por ejemplo, la variable {{nombre}} se utiliza para almacenar el nombre de un usuario.

```
x = 5
y = "Hola Mundo!"
print(x)
print(y)
```

  print se utiliza para imprimir en pantalla el valor de una variable en la consola.

    - Casting: Si queremos especificar el tipo de dato de una variable, podemos usar la función de casting:

```
```

```
x = str(3)    # x será '3'
y = int(3)    # y será 3
z = float(3)  # z será 3.0
```

- Obtener el tipo de dato de una variable: Python tiene varios tipos de datos estándar, pero también podemos definir nuestros propios tipos de datos personalizados. Para obtener el tipo de dato de una variable, podemos usar la función type():

```
x = 5
y = "Hola Mundo!"
print(type(x))
print(type(y))
```

- Las variables de cadena de texto se pueden declarar de varias formas:

```
x = "Hola Mundo!"
# es lo mismo que
x = 'Hola Mundo!'
```

- Las variables numéricas se declaran de la siguiente forma:

```
x = 1    # int
y = 2.8  # float
z = 1j   # complex
```

- Obtener el tipo de dato de una variable: Python tiene varios tipos de datos estándar, pero también podemos definir nuestros propios tipos de datos personalizados. Para obtener el tipo de dato de una variable, podemos usar la función type():

```
x = 5
y = "Hola Mundo!"
print(type(x))
print(type(y))
```

- Asignar múltiples valores a múltiples variables: Python le permite asignar valores a múltiples variables en una línea:

```
x, y, z = "Naranja", "Plátano", "Cereza"
print(x)
print(y)
print(z)
```

- Asignar el mismo valor a múltiples variables: Python le permite asignar el mismo valor a múltiples variables en una línea:

```
x = y = z = "Naranja"
print(x)
print(y)
print(z)
```

- Python diferencia sus variables entre mayúsculas y minúsculas, por lo que las variables x y X son diferentes. Por ejemplo:

```
a = 4
A = "Sally"
#A es una variable diferente a a
```

- Podemos nombrar las variables de diferente forma:

```
myvar = "John"
my_var = "John"
_my_var = "John"
myVar = "John"
MYVAR = "John"
myvar2 = "John"
```

- Las variables no pueden comenzar con un número:

```
2myvar = "John"
```

- Las variables no pueden tener espacios:

```
my var = "John"
```

- Podemos desempaquetar una colección de valores en variables:

```
frutas = ["manzana", "plátano", "cereza"]
x, y, z = frutas
print(x)
print(y)
print(z)
```

- Podemos utilizar la función + para combinar variables de cadena de texto:

```
x = "Python es "
y = "increíble"
z =  x + y
print(z)
```

También podemos utilizar la funcion + en el print para concatenar variables de cadena de texto:

```
x = "Python es "
y = "increíble"
print(x + y)
```

- Para los números, el operador + funciona como una operación matemática:

```
x = 5
y = 10
print(x + y)
```

- Si intentamos combinar una cadena de texto y un número, Python nos dará un error:

```
x = 5
y = "John"
print(x + y)
```

- Podemos utilizar la función global() para crear una variable global, incluso si se crea dentro de una función:

```
x = "awesome"

def myfunc():
print("Python is " + x)

myfunc()
```

- Si se crea una variable con el mismo nombre dentro de una función, esta será local y solo se podrá utilizar dentro de la función. La variable global con el mismo nombre seguirá existiendo como variable global.

```
x = "awesome"
```

```
def myfunc():
x = "fantastic"
print("Python is " + x)

myfunc()

print("Python is " + x)
```

- Para crear una variable global podemos utilizar la palabra clave global:

```
def myfunc():
global x
x = "fantastic"

myfunc()

print("Python is " + x)
```

- También podemos cambiar el valor de una variable global dentro de una función:

```
x = "awesome"

def myfunc():
global x
x = "fantastic"

myfunc()

print("Python is " + x)
```

- Tipos de datos en python:
  - Texto: str
  - Numérico: int, float, complex
  - Secuencia: list, tuple, range
  - Mapeo: dict
  - Set: set, frozenset
  - Booleano: bool
  - Binario: bytes, bytearray, memoryview | Tipo de Dato | Ejemplo | Ejemplo con Constructor | | --- | --- | --- | | str | `x = "Hello World"` | `x = str("Hello World")` | | int | `x = 20` | `x = int(20)` | | float | `x = 20.5` | `x = float(20.5)` | | complex | `x = 1j` | `x = complex(1j)` | | list | `x = ["apple", "banana", "cherry"]` | `x = list(("apple", "banana", "cherry"))` | | tuple | `x = ("apple", "banana", "cherry")` | `x = tuple(("apple", "banana", "cherry"))` | | range | `x = range(6)` | `x = range(6)` | | dict | `x = {"name" : "John", "age" : 36}` | `x = dict(name="John", age=36)` | | set | `x = {"apple", "banana", "cherry"}` | `x = set(("apple", "banana", "cherry"))` | | frozenset | `x = frozenset({"apple",`

`"banana", "cherry"})` | `x = frozenset(("apple", "banana", "cherry"))` | | bool | `x = True` | `x = bool(5)` | | bytes | `x = b"Hello"` | `x = bytes(5)` | | bytearray | `x = bytearray(5)` | `x = bytearray(5)` | | memoryview | `x = memoryview(bytes(5))` | `x = memoryview(bytes(5))` |