Documentación Django (Phyton)

Tabla de contenidos

Introducción

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 (Entorno virtual + Django + Python)

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

```
python3 --version
```

En 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 (Entorno virtual)

- **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 he llamado mysite pues es mysite/bin/activate si fuera por ejemplo otroNombre sería 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

Lo primero que vamos a ver es cómo 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 del proyecto. En este fichero se añaden las URLs de las aplicaciones que se van 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 ha 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 además 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 aplicación 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 hacer esto, vamos a **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.

Lo primero que vamos a hacer es 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),
]
```

> [!NOTE] Si no sabes de donde a salido la carpeta HelloWorld es la carpeta que se crea al crear el proyecto Django. Os dejo un enlace a donde explique la estructura de los ficheros al crear un proyecto Django Creación de un proyecto.

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 básica 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"
```

Hay nombre de variables que son "Ilegales" en Python, como por ejemplo:

```
2myvar = "Hola Mundo" # No puede empezar por un número
my-var = "Hola Mundo" # No puede contener guiones
my var = "Hola Mundo" # No puede contener espacios en blanco
myvar = "Hola Mundo" # Esta es la forma correcta de declarar una variable
```

> [!NOTE] Las variables son sensibles a mayúsculas y minúsculas, por lo que las variables **myvar** y **myVar** son diferentes. Además, las variables se pueden declarar globalmente y localmente. Para declarar una variable global dentro de una función, se utiliza la palabra clave global.

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

myfunc()

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

> También podemos declarar una variable fuera de una función y eso se puede utilizar en cualquier lugar del código seria un ejemplo de variable global como la anterior.

```
x = "fantastico"

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

myfunc()
```

Tipos de datos en Python

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

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

```
# Texto (str) -> Se utiliza para representar texto.
texto = "Hola, mundo!"

# Entero (int) -> Se utiliza para representar números enteros.
entero = 10

# Decimal (float) -> Se utiliza para representar números reales.
decimal = 3.14

# Complejo (complex) -> Se utiliza para representar números complejos
(números con una parte real e imaginaria).
complejo = 1+2j
```

```python
# Booleano (bool) -> Se utiliza para representar valores booleanos (True o False).
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 y se clasifican en:

Operadores aritméticos en Python

Operador	Nombre	Ejemplo	Explicacion
+	Suma	x + y	Suma x e y
-	Resta	x - y	Resta x e y
*	Multiplicacion	x * y	Multiplica x por y
/	Division	x / y	Divide x entre y
%	Modulo	x % y	Devuelve el resto de dividir x entre y
**	Potecia	x ** y	Eleva x a la potencia y
//	Division entera	x // y	Divide x entre y y devuelve el resultado sin decimales

A continuación, vamos a ver algunos ejemplos de operadores aritméticos en Python.

```
x = 5
y = 3

print(x + y) # 8
print(x - y) # 2
print(x * y) # 15
print(x / y) # 1.6666666666666667
print(x % y) # 2
print(x ** y) # 125
print(x // y) # 1
```

Operadores de asignación en Python

Operador	Ejemplo	Explicacion
=	x = 5	Asigna el valor 5 a la variable x
+=	x += 5	Suma 5 a la variable x y asigna el resultado a la variable x
-=	x -= 5	Resta 5 a la variable x y asigna el resultado a la variable x
*=	x *= 5	Multiplica la variable x por 5 y asigna el resultado a la variable x
/=	x /= 5	Divide la variable x entre 5 y asigna el resultado a la variable x
%=	x %= 5	Divide la variable x entre 5 y asigna el resto a la variable x

Operador	Ejemplo	Explicacion
**=	x **= 5	Eleva la variable x a la potencia 5 y asigna el resultado a la variable x
//=	x //= 5	Divide la variable x entre 5 y asigna el resultado sin decimales a la x
:=	x := 5	Asigna el valor 5 a la variable x (Python 3.8+)
&=	x &= 5	Realiza una operación AND entre la variable x y 5 y asigna el resultado a x
|=	x |= 5	Realiza una operación OR entre la variable x y 5 y asigna el resultado a x
^=	x ^= 5	Realiza una operación XOR entre la variable x y 5 y asigna el resultado a x
>>=	x >>= 5	Realiza una operación de desplazamiento a la derecha y asigna el resultado x
<<=	x <<= 5	Realiza una operación de desplazamiento a la izquierda y asigna el resultado x

A continuación, vamos a ver algunos ejemplos de operadores de asignación en Python.

```
x = 5

x += 5 # 10
x -= 5 # 0
x *= 5 # 25
x /= 5 # 1.0
x %= 5 # 0.0
x **= 5 # 3125.0
x //= 5 # 0.0
```

Operadores de comparación en Python

Operador	Nombre	Ejemplo	Explicacion
==	Igual	x == y	Devuelve True si x es igual a y
!=	No igual	x != y	Devuelve True si x no es igual a y
>	Mayor que	x > y	Devuelve True si x es mayor que y
<	Menor que	x < y	Devuelve True si x es menor que y
>=	Mayor o igual que	x >= y	Devuelve True si x es mayor o igual que y
<=	Menor o igual que	x <= y	Devuelve True si x es menor o igual que y

A continuación, vamos a ver algunos ejemplos de operadores de comparación en Python.

```
x = 5
y = 3

print(x == y) # False
print(x != y) # True
print(x > y) # True
print(x < y) # False
print(x >= y) # True
print(x <= y) # False
```

Operadores lógicos en Python

Operador	Nombre	Ejemplo	Explicacion
and	Y	x and y	Devuelve True si x y y son True
or	O	x or y	Devuelve True si x o y son True
not	No	not x	Devuelve True si x no es True

A continuación, vamos a ver algunos ejemplos de operadores lógicos en Python.

```
x = 5

print(x > 3 and x < 10) # True
print(x > 3 or x < 4) # True
print(not(x > 3 and x < 10)) # False
```

Operadores de identidad en Python

Operador	Nombre	Ejemplo	Explicacion
is	Es	x is y	Devuelve True si x e y son el mismo objeto
is not	No es	x is y	Devuelve True si x e y no son el mismo objeto

A continuación, vamos a ver algunos ejemplos de operadores de identidad en Python.

```
x = ["manzana", "platano"]
y = ["manzana", "platano"]
z = x

print(x is z) # True
print(x is y) # False
print(x == y) # True
```

Operadores de pertenencia en Python

Operador	Nombre	Ejemplo	Explicacion
in	En	x in y	Devuelve True si x esta en y
not in	No en	x in y	Devuelve True si x no esta en y

A continuación, vamos a ver algunos ejemplos de operadores de pertenencia en Python.

```
x = ["manzana", "platano"]

print("manzana" in x) # True
print("cereza" not in x) # True
```

Operadores de bits en Python

Operador	Nombre	Ejemplo	Explicacion
&	AND	x & y	Devuelve 1 si ambos bits son 1
|	OR	x | y	Devuelve 1 si uno de los bits es 1
^	XOR	x ^ y	Devuelve 1 si uno de los bits es 1 pero no ambos
~	NOT	~x	Invierte todos los bits
<<	Desplazamiento a	x << y	Desplaza los bits de x, y posiciones a la izquierda, los bits vacios se llenan con 0
>>	Desplazamiento a	x >> y	Desplaza los bits de x, y posiciones a la derecha, los bits vacios se llenan con 0

A continuación, vamos a ver algunos ejemplos de operadores de bits en Python.

```
x = 0b1100
y = 0b1010

print(bin(x & y)) # 0b1000
print(bin(x | y)) # 0b1110
print(bin(x ^ y)) # 0b0110
print(bin(~x)) # -0b1101
print(bin(x << 2)) # 0b110000
print(bin(x >> 2)) # 0b11
```

Phyton Casting

Tabla de contenidos

En Python, el tipo de datos de una variable se puede cambiar a otro tipo de datos. Esto se llama conversión de tipos de datos y se hace con funciones de conversión de tipos de datos incorporadas.

Las funciones de conversión de tipos de datos comunes son:

- **int()**: Convierte un objeto a un entero.
- **float()**: Convierte un objeto a un número decimal.
- **str()**: Convierte un objeto a una cadena.

Y así con todos los demás tipos de datos, y ahora vamos a ver algunos ejemplos de casting en Python

```
x = int(1) # x sera 1
y = int(2.8) # y sera 2
z = int("3") # z sera 3

x = float(1) # x sera 1.0
y = float(2.8) # y sera 2.8
z = float("3") # z sera 3.0
w = float("4.2") # w sera 4.2

x = str("s1") # x sera 's1'
y = str(2) # y sera '2'
z = str(3.0) # z sera '3.0'
```

Estructuras de control en Python

Tabla de contenidos

En Python, las estructuras de control se utilizan para tomar decisiones y ejecutar acciones en función de esas decisiones. Las estructuras de control más comunes son:

Ahora vamos a ver algunos ejemplos de estructuras de control en Python.

```
# if -> Se utiliza para ejecutar un bloque de código si se cumple una
condición.

 x = 5
 y = 3

 if x > y:
     print("x es mayor que y")

# if...else -> Se utiliza para ejecutar un bloque de código si se cumple
una condición y otro bloque de código si no se cumple la condición.

 x = 5
 y = 3

 if x > y:
     print("x es mayor que y")
 else:
     print("x no es mayor que y")

# if...elif...else -> Se utiliza para ejecutar un bloque de código si se
cumple una condición, otro bloque de código si se cumple otra condición y
otro bloque de código si no se cumple ninguna de las condiciones.

  x = 5
  y = 3

  if x > y:
      print("x es mayor que y")
  elif x == y:
      print("x es igual a y")
  else:
      print("x es menor que y")
```

Bucles en Python

Tabla de contenidos

Luego tenemos los bucles que son estructuras de control que se utilizan para repetir una acción varias veces. Los bucles más comunes son:

```
# while -> Se utiliza para ejecutar un bloque de código mientras se cumple una condición.

  i = 1

  while i < 6:
      print(i)
      i += 1

# for -> Se utiliza para ejecutar con un bloque de código un número determinado de veces.

  frutas = ["manzana", "platano", "cereza"]

  for x in frutas:
      print(x)
```

Ahora dentro de los bucles tenemos las sentencias break y continue que se utilizan para cambiar el comportamiento de los bucles.

```
# break -> Se utiliza para terminar el bucle antes de que se cumpla la condición.
  i = 1

  while i < 6:
      print(i)
      if i == 3:
          break
      i += 1

# continue -> Se utiliza para saltar una iteración del bucle.
  i = 0

  while i < 6:
      i += 1
      if i == 3:
          continue
      print(i)
```

Exceptions en Python

Tabla de contenidos

Las excepciones se utilizan para manejar errores que pueden ocurrir durante la ejecución de un programa.
Las excepciones más comunes son:

```
# try...except -> Se utiliza para manejar excepciones.
  try:
      print(x)
  except:
      print("Ha ocurrido un error")

# try...except...else -> Se utiliza para manejar excepciones y ejecutar un
bloque de código si no se produce ninguna excepción.
  try:
      print("Hola")
  except:
      print("Ha ocurrido un error")
  else:
      print("Ningun error ha ocurrido")

# try...except...finally -> Se utiliza para manejar excepciones y ejecutar
un bloque de código, independientemente de si se produce una excepción o
no.
  try:
      print(x)
  except:
      print("Ha ocurrido un error")
  finally:
      print("El bloque try...except ha terminado")

# raise -> Se utiliza para generar una excepción.
  x = -1

  if x < 0:
      raise Exception("El numero no puede ser negativo")

# pass -> Se utiliza cuando no se necesita código.
  try:
      print(x)
  except:
      pass

# assert -> Se utiliza para comprobar si una condición es verdadera.

  x = 1

  assert x > 0, "El numero debe ser positivo"
```

Arrays en Python

Tabla de contenidos

Los arrays se utilizan para almacenar varios valores en una sola variable.

Las arrays o listas en Python se pueden crear de varias formas:

```
lista = ["manzana", "platano", "cereza"]

tupla = ("manzana", "platano", "cereza")

rango = range(6)
```

Estas son todas las formas de declarar una array o lista en Python y para acceder a los elementos de un array, se utiliza el número de índice. El primer elemento tiene el índice 0.

```
         # 0         1         2
lista = ["manzana", "platano", "cereza"]

print(lista[1]) # platano
```

También podemos modificar el valor de un elemento de un array utilizando el número de índice.

```
lista = ["manzana", "platano", "cereza"]

lista[1] = "naranja"

print(lista) # ['manzana', 'naranja', 'cereza']
```

Para recorrer los elementos de un array, se puede utilizar el bucle for.

```
lista = ["manzana", "platano", "cereza"]

for x in lista:
    print(x) # manzana platano cereza -> Imprime cada elemento de la
lista
```

Métodos de arrays en Python

Tabla de contenidos

Para devolver la longitud de un array, se utiliza la función len().

```
lista = ["manzana", "platano", "cereza"]

print(len(lista)) # 3
```

Si quieres añadir un elemento al final de un array, se utiliza el método append().

```
lista = ["manzana", "platano", "cereza"]

lista.append("naranja")

print(lista) # ['manzana', 'platano', 'cereza', 'naranja']
```

Si quieres añadir un elemento en una posición específica de un array, se utiliza el método insert().

```
lista = ["manzana", "platano", "cereza"]

lista.insert(1, "naranja")

print(lista) # ['manzana', 'naranja', 'platano', 'cereza']
```

Si quieres eliminar un elemento de un array, se utiliza el método remove().

```
lista = ["manzana", "platano", "cereza"]

lista.remove("platano")

print(lista) # ['manzana', 'cereza']
```

Si quieres eliminar el último elemento de un array, se utiliza el método pop().

```
lista = ["manzana", "platano", "cereza"]
lista.pop()
print(lista) # ['manzana', 'platano']
```

Si quieres eliminar un elemento de un array en una posición específica, tambien se puede utilizar con el método pop().

```
lista = ["manzana", "platano", "cereza"]
lista.pop(1)
print(lista) # ['manzana', 'cereza']
```

Si quieres vaciar un array, se utiliza el método clear().

```
lista = ["manzana", "platano", "cereza"]
lista.clear()
print(lista) # []
```

Si quieres copiar un array, se utiliza el método copy().

```
lista = ["manzana", "platano", "cereza"]
lista2 = lista.copy()
print(lista2) # ['manzana', 'platano', 'cereza']
```

Si quieres unir dos arrays, se utiliza el método extend().

```
  lista = ["manzana", "platano", "cereza"]
  lista2 = ["naranja", "limon", "sandia"]

  lista.extend(lista2)

  print(lista) # ['manzana', 'platano', 'cereza', 'naranja', 'limon',
'sandia']
```

Si queremos ordenar los elementos de un array, se utiliza el método sort().

```
  lista = ["manzana", "platano", "cereza"]

  lista.sort()

  print(lista) # ['cereza', 'manzana', 'platano']
```

El metodo count() devuelve el número de elementos con el valor especificado.

```
  points = [1, 4, 2, 9, 7, 8, 9, 3, 1]

x = points.count(9)

print(x) # 2 por que hay dos 9
```

El metodo index() devuelve el índice del primer elemento con el valor especificado.

```
  points = [1, 4, 2, 9, 7, 8, 9, 3, 1]

x = points.index(8)

print(x) # 5 por que el 8 esta en la posicion 5
```

POO en Python (Programación Orientada a Objetos)

Tabla de contenidos

La programación orientada a objetos (POO) es un paradigma de programación que utiliza objetos y sus interacciones para diseñar aplicaciones y programas informáticos. Está basado en varias técnicas, incluyendo herencia, modularidad, polimorfismo , encapsulamiento y abstracción. Su uso se popularizó a principios de la década de 1990. Actualmente, es la forma más popular de programar.

Clases y objetos en Python

Tabla de contenidos

En Python, una clase es una plantilla para crear objetos. Para crear una clase, utilizamos la palabra clave class seguida del nombre de la clase y dos puntos. A continuación, vamos a ver un ejemplo de clase en Python.

```
class Coche:
    pass
```

Ahora vamos a crear un objeto de la clase Coche. Para ello, escribimos:

```
coche = Coche()
```

Atributos en Python

Tabla de contenidos

Los atributos son variables que pertenecen a una clase y se utilizan para almacenar datos sobre el objeto. Como ejemplo puede ser:

```
class Coche:
    marca = "Ford"
    modelo = "Mustang"
    color = "Rojo"
```

Para acceder a los atributos de un objeto primero creamos un objeto de la clase coche como hemos hecho anteriormente y utilizamos la sintaxis objeto.atributo. A continuación, vamos a ver algunos ejemplos de atributos en Python.

```
coche = Coche()

print(coche.marca) # Ford
print(coche.modelo) # Mustang
print(coche.color) # Rojo
```

Métodos en Python (Modularidad)

Tabla de contenidos

Los métodos son funciones que pertenecen a una clase y se utilizan para realizar operaciones con los atributos de la clase. Para crear un método, utilizamos la palabra clave def seguida del nombre del método y dos puntos. A continuación, vamos a ver un ejemplo de método en Python.

> [!WARNING]
>
> En Python, los métodos de instancia de una clase necesitan tener al menos un argumento, que es una referencia a la instancia de la clase que está llamando al método. Por >convención, este argumento se llama self.
>
> Si defines el método acelerar() sin el argumento self, Python lo tratará como un método estático. Los métodos estáticos no tienen acceso a las instancias de la clase, por lo que no pueden modificar ni acceder a los atributos de instancia.
>
> Para entender mejor esto podemos compararlo con el this de Java o el this de JavaScript que en este caso seria el self de Python.

```
class Coche:
    marca = "Ford"
    modelo = "Mustang"
    color = "Rojo"

    def acelerar(self):
        print("El coche está acelerando")
```

Para llamar a un método de un objeto primero creamos un objeto de la clase coche como hemos hecho anteriormente y utilizamos la sintaxis objeto.metodo(). A continuación, vamos a ver algunos ejemplos de métodos en Python.

```
coche = Coche()

coche.acelerar() # El coche está acelerando
```

Constructores en Python

Tabla de contenidos

Los constructores se utilizan para inicializar los atributos de un objeto. Para crear un constructor, utilizamos el método **init**(). A continuación, vamos a ver un ejemplo de constructor en Python.

```
class Coche:
    def __init__(self, marca, modelo, color):
        self.marca = marca
        self.modelo = modelo
        self.color = color
```

Ahora vamos a crear un objeto de la clase Coche. Para ello, escribimos:

```
coche = Coche("Ford", "Mustang", "Rojo")
```

Para llamar a un atributo de un objeto, utilizamos la sintaxis objeto.atributo. A continuación, vamos a ver algunos ejemplos de constructores en Python.

```
print(coche.marca) # Ford
print(coche.modelo) # Mustang
print(coche.color) # Rojo
```

Getters y setters en Python

Y ahora vamos a ver los getters y setters que son métodos que se utilizan para obtener y establecer los valores de los atributos de un objeto. Para crear un getter, utilizamos el método **get**() y para crear un setter, utilizamos el método **set**(). A continuación, vamos a ver un ejemplo de getters y setters en Python.

```
class Coche:
    def __init__(self, marca, modelo, color):
        self.marca = marca
        self.modelo = modelo
        self.color = color

    def getMarca(self):
        return self.marca

    def setMarca(self, marca):
        self.marca = marca

    def getModelo(self):
        return self.modelo

    def setModelo(self, modelo):
        self.modelo = modelo

    def getColor(self):
        return self.color

    def setColor(self, color):
        self.color = color
```

Ahora voy a poner un ejemplo de como utilizar los getters y setters en Python.

```
coche = Coche("Ford", "Mustang", "Rojo")

print(coche.getMarca()) # Ford
print(coche.getModelo()) # Mustang
print(coche.getColor()) # Rojo

coche.setMarca("Ferrari")
coche.setModelo("F40")
coche.setColor("Amarillo")

print(coche.getMarca()) # Ferrari
print(coche.getModelo()) # F40
print(coche.getColor()) # Amarillo
```

Vamos a crear un ejemplo con todo lo visto para que todo este mas claro.

```
class Coche:
    def __init__(self, marca, modelo, color):
        self.marca = marca
        self.modelo = modelo
        self.color = color

    def getMarca(self):
        return self.marca
    def setMarca(self, marca):
        self.marca = marca

    def getModelo(self):
        return self.modelo

    def setModelo(self, modelo):
        self.modelo = modelo

    def getColor(self):
        return self.color

    def setColor(self, color):
        self.color = color

    def __str__(self): # Este metodo se utiliza para representar un objeto como una cadena de texto se puede comparar con el toString() de Java por ejemplo.
        return f"Marca: {self.marca}, Modelo: {self.modelo}, Color: {self.color}"

    def __len__(self): # Este metodo se utiliza para devolver la longitud de un objeto se puede comparar con el length() de Java por ejemplo.
        return 3

coche = Coche("Ford", "Mustang", "Rojo")

print(coche.getMarca()) # Ford
print(coche.getModelo()) # Mustang
print(coche.getColor()) # Rojo

coche.setMarca("Ferrari")
coche.setModelo("F40")
coche.setColor("Amarillo")

print(coche.getMarca()) # Ferrari
print(coche.getModelo()) # F40
print(coche.getColor()) # Amarillo

print(coche) # Marca: Ferrari, Modelo: F40, Color: Amarillo
print(len(coche)) # 3
```

Herencia en Python

Tabla de contenidos

La herencia es una forma de crear una clase a partir de otra clase. La clase que hereda se llama clase hija y la clase que hereda se llama clase padre. Para crear una clase hija, pasamos la clase padre como parámetro en la definición de la clase hija. A continuación, vamos a ver un ejemplo de herencia en Python.

```
class Coche:
    def __init__(self, marca, modelo, color):
        self.marca = marca
        self.modelo = modelo
        self.color = color

    def acelerar(self):
        print("El coche está acelerando")

class CocheDeportivo(Coche): # CocheDeportivo hereda de Coche
    pass
```

Si queremos añadir atributos adicionales a la clase hija al heredar de la clase padre, podemos añadirlos en el método **init**() de la clase hija. A continuación, vamos a ver un ejemplo de herencia en Python.

```
class Coche:
    def __init__(self, marca, modelo, color):
        self.marca = marca
        self.modelo = modelo
        self.color = color

    def acelerar(self):
        print("El coche está acelerando")

class CocheDeportivo(Coche): # CocheDeportivo hereda de Coche
    def __init__(self, marca, modelo, color, velocidadMaxima):
        self.marca = marca
        self.modelo = modelo
        self.color = color
        self.velocidadMaxima = velocidadMaxima
```

Ahora vamos a crear un objeto de la clase CocheDeportivo. Para ello, escribimos:

```
coche = CocheDeportivo("Ford", "Mustang", "Rojo" , 300)
```

Para llamar a un atributo de un objeto, utilizamos la sintaxis objeto.atributo. A continuación, vamos a ver algunos ejemplos de herencia en Python.

```
print(coche.marca) # Ford
print(coche.modelo) # Mustang
print(coche.color) # Rojo
print(coche.velocidadMaxima) # 300
```

Herencia múltiple en Python

Tabla de contenidos

Python sí permite la herencia múltiple, que es una característica que permite a una clase heredar comportamientos y características de más de una clase base.

```
class Clase1:
    pass

class Clase2:
    pass

class MultipleHerencia(Base1, Base2):
    pass
```

Sin embargo, aunque la herencia múltiple puede ser útil en algunos casos, también puede llevar a una serie de complicaciones, como el problema del diamante (cuando una clase hereda de dos clases que tienen una clase base común) y la complejidad adicional en la estructura de la clase.

Por lo tanto, la herencia múltiple debe usarse con cuidado. Esto es opinión personal, pero creo que la herencia múltiple no es una buena práctica de programación y que debería evitarse siempre que sea posible.

Encapsulación en Python

Tabla de contenidos

La encapsulación es una forma de restringir el acceso a los atributos y métodos de una clase. En Python, podemos restringir el acceso a los atributos y métodos de una clase utilizando el guion bajo como prefijo. A continuación, vamos a ver un ejemplo de encapsulación en Python.

> [!NOTE] Para poder acceder a los atributos y métodos de una clase encapsulada, tendriamos que crear getter y setters para cada atributo como lo e explicado antes para asi poder >acceder a los atributos y métodos de una clase encapsulada.

```
class Coche:
    def __init__(self, marca, modelo, color):
        self._marca = marca # soy un atributo que solo se puede acceder
desde la clase Coche
        self._modelo = modelo
        self._color = color

    def getMarca(self):
        return self._marca

    def setMarca(self, marca):
        self._marca = marca

coche = Coche("Ford", "Mustang", "Rojo")

print(coche.getMarca()) # Ford

coche.setMarca("Ferrari")

print(coche.getMarca()) # Ferrari
```

Abstracción en Python

Tabla de contenidos

La abstracción es una forma de ocultar los detalles de implementación y mostrar solo la funcionalidad al usuario. En Python, podemos utilizar la clase abstractmethod del módulo abc para crear una clase abstracta. A continuación, vamos a ver un ejemplo de abstracción en Python.

```
from abc import ABC, abstractmethod

class Coche(ABC): # Coche es una clase abstracta porque hereda de ABC
    def __init__(self, marca, modelo, color):
        self.marca = marca
        self.modelo = modelo
        self.color = color

    def acelerar(self):
        print("El coche está acelerando")

    @abstractmethod
    def frenar(self):
        pass

class CocheDeportivo(Coche):
    def __init__(self, marca, modelo, color, velocidadMaxima):
        self.marca = marca
        self.modelo = modelo
        self.color = color
        self.velocidadMaxima = velocidadMaxima

    def frenar(self):
        print("El coche está frenando")
```

Las clases abstractas suelen ser clases que no se pueden instanciar y que se utilizan como clases padres. Si intentamos instanciar una clase abstracta, se producirá un error. A continuación, vamos a ver un ejemplo de abstracción en Python.

```
coche = Coche("Ford", "Mustang", "Rojo" , 300) # TypeError: Can't
instantiate abstract class Coche with abstract methods frenar
```

Polimorfismo en Python

Tabla de contenidos

El polimorfismo es una forma de utilizar una clase de manera diferente. Para entender esto fácilmente podríamos usar la funcion len(). La función len() se puede utilizar con muchos tipos de datos diferentes, como cadenas, listas, tuplas, etc. Pero se interpreta de manera diferente por ejemplo para una lista, devolverá el total de elementos presentes y para una cadena, y el total de caracteres presentes.

```
print(len("Hola")) # 4
print(len(["manzana", "platano", "cereza"])) # 3
```

Otro ejemplo seria el operador +. El operador + se puede utilizar para sumar dos números o para concatenar dos cadenas.

```
print(1 + 2) # 3
print("Hola" + "Mundo") # HolaMundo
```

Incluso podemos crear nuestra clase y lograr esto. Dos clases pueden tener funciones con el mismo nombre pero con diferentes propósitos y diferentes definiciones.

```
class A:
    def fun(self):
        print("Class A")


class B:
    def fun(self):
        print("Class B")


ob1 = A()
ob2 = B()
for i in (ob1, ob2):
    i.fun() # Imprime el resultado de la funcion fun() de cada clase ->
Class A Class B
```

Métodos especiales en Python

Tabla de contenidos

Los métodos especiales son métodos que se utilizan para realizar operaciones especiales. Los métodos especiales más comunes son:

```python
# __init__() -> Se utiliza para inicializar los atributos de un objeto.

  class Coche:
      def __init__(self, marca, modelo, color):
          self.marca = marca
          self.modelo = modelo
          self.color = color

      def acelerar(self):
          print("El coche está acelerando")

# __str__() -> Se utiliza para representar un objeto como una cadena.

  class Coche:
      def __init__(self, marca, modelo, color):
          self.marca = marca
          self.modelo = modelo
          self.color = color

      def acelerar(self):
          print("El coche está acelerando")

      def __str__(self):
          return f"Marca: {self.marca}, Modelo: {self.modelo}, Color:
{self.color}"

# __len__() -> Se utiliza para obtener la longitud de un objeto.

  class Coche:
      def __init__(self, marca, modelo, color):
          self.marca = marca
          self.modelo = modelo
          self.color = color

      def acelerar(self):
          print("El coche está acelerando")

      def __str__(self):
          return f"Marca: {self.marca}, Modelo: {self.modelo}, Color:
{self.color}"

      def __len__(self):
          return 3
```

Ahora vamos a crear un objeto de la clase Coche. Para ello, escribimos:

```
coche = Coche("Ford", "Mustang", "Rojo")
```

Para llamar a un atributo de un objeto, utilizamos la sintaxis objeto.atributo. A continuación, vamos a ver algunos ejemplos de métodos especiales en Python.

```
print(len(coche)) # 3
```

Métodos estáticos en Python

Tabla de contenidos

Los métodos estáticos son métodos que se utilizan sin crear un objeto. Para crear un método estático, utilizamos el decorador @staticmethod. A continuación, vamos a ver un ejemplo de método estático en Python.

```
class Coche:
    def __init__(self, marca, modelo, color):
        self.marca = marca
        self.modelo = modelo
        self.color = color

    def acelerar(self):
        print("El coche está acelerando")

    @staticmethod
    def frenar():
        print("El coche está frenando")
```

Por ejemplo el metodo frenar() como es un metodo estatico siempre va a devolver el mismo resultado por lo que no es necesario crear un objeto para llamar a este metodo.

```
Coche.frenar() # El coche está frenando
```

Métodos de clase en Python

Tabla de contenidos

Los métodos de clase son métodos que se utilizan sin crear un objeto. Para crear un método de clase, utilizamos el decorador @classmethod. A continuación, vamos a ver un ejemplo de método de clase en Python.

```
class Coche:
    def __init__(self, marca, modelo, color):
        self.marca = marca
        self.modelo = modelo
        self.color = color

    def acelerar(self):
        print("El coche está acelerando")

    @classmethod
    def frenar(cls):
        print("El coche está frenando")
```

Un método de clase puede ser llamado tanto en la clase como en las instancias de la clase. A diferencia de los métodos de instancia, que tienen acceso a la instancia específica y a sus atributos, los métodos de clase tienen acceso a la clase y a sus atributos. No pueden modificar el estado de una instancia específica de la clase, pero pueden modificar el estado de la clase.

CRUD en Python

Tabla de contenidos

CRUD es el acrónimo de Create, Read, Update y Delete. CRUD son las cuatro operaciones básicas que se pueden hacer en una base de datos. En este caso vamos a ver como hacer un CRUD en Python utilizando una base de datos MySQL.

Preparando el entorno (Instalación de paquetes)

Tabla de contenidos

Lo primero que tenemos que hacer es activar el entorno virtual que creamos al principio de este documento te lo dejo por aquí por si no te acuerdas de como se hace y asegurate de desactivar el entorno virtual si esque lo tienes activado escribiendo deactivate en la terminal.

```
source venv/bin/activate
```

Lucgo tenemos que instalar el paquete mysql-connector-python. Para ello, escribimos:

```
pip install mysql-connector-python
```

Hay algunos paquetes que pueden ser interesantes de instalar para poder trabajar con bases de datos. Para ello, escribimos:

```
pip install Pillow
pip install openpyxl
pip install pandas
```

- **Pillow**: Se utiliza para trabajar con imágenes.
- **openpyxl**: Se utiliza para trabajar con archivos de Excel.
- **pandas**: Se utiliza para trabajar con archivos CSV. (Comma Separated Values)

Creando el directorio del proyecto (Crud)

Tabla de contenidos

Lo siguiente que tenemos que tener en cuenta es crear el proyecto con el comando django-admin startproject y luego de hacer eso lo siguiente que tenemos que crear es una aplicación con el comando python manage.py startapp. Puedes echar un vistazo a Creación de un proyecto para recordar como hacerlo si lo necesitas.

En mi caso e creado el Proyecto ProyectoCrud y luego e creado dos vistas una para el curso y otra para el estudiante. Deberiamos tener algo como esto en el directorio de nuestro proyecto.

```
ProyectoCrud
├── ProyectoCrud
│   ├── __init__.py
│   ├── asgi.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
├── Curso
│   ├── __init__.py
│   ├── admin.py
│   ├── apps.py
│   ├── migrations
│   │   └── __init__.py
│   ├── models.py
│   ├── tests.py
│   └── views.py
```

```
    ├── Estudiante
    │   └── ... (lo mismo que en Curso)
    ├── EstudianteCurso
    │   └── ... (lo mismo que en Curso)
    └── manage.py
```

Creando el modelo en Python

Tabla de contenidos

Lo primero que tenemos que añadir es en el archivo settings.py del directorio raiz de nuestro proyecto es añadir la aplicación que hemos creado en INSTALLED_APPS.

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

El siguiente paso seria crear el modelo en el archivo models.py de cada aplicación que hemos creado. En mi caso e creado tres modelos uno para el estudiante y otro para el curso y otro para la relacion entre las dos entidades que es EstudianteCurso. Deberiamos tener algo como esto en el archivo models.py en mi caso esta dentro del directorio de la aplicación Estudiante, Curso o EstudianteCurso. En mi caso tengo que añadir la clase Estudiante, Curso y EstudianteCurso en el archivo models.py de cada aplicación.

```
from django.db import models

class Estudiante(models.Model):
    nif = models.CharField(max_length=9, primary_key=True)
    nombre_apellido = models.CharField(max_length=50)
    edad = models.IntegerField()
    carrera = models.CharField(max_length=50)
    universidad = models.CharField(max_length=50)

    def __str__(self):
        return self.nombre_apellido
```

```
from django.db import models

 class Curso(models.Model):
     cine = models.IntegerField(primary_key=True)
     nombre = models.CharField(max_length=50)
     creditos = models.IntegerField()
     profesor = models.CharField(max_length=50)
     universidad = models.CharField(max_length=50)

     def __str__(self):
         return self.nombre
```

> [!WARNING] En el caso de la clase EstudianteCurso tenemos que tener en cuenta que tenemos que importar las clases Estudiante y Curso para poder utilizarlas en la clase EstudianteCurso. Para ello, escribimos:

```
from ProyectoCrud import Estudiante, Curso

 class EstudianteCurso(models.Model):
        nif = models.ForeignKey(Estudiante, on_delete=models.CASCADE)
        cine = models.ForeignKey(Curso, on_delete=models.CASCADE)


     def __str__(self):
            return self.nif.nombre_apellido + " - " + self.cine.nombre
```

> [!NOTE] Tenemos que tener en cuenta que no hace falta nada mas ya que Django se encarga de los getters y setters y de los constructores y de todo lo que necesitemos.

Ahora lo que tenemos que hacer es editar el archivo settings.py del proyecto y añadir la configuración de la base de datos. Para ello, escribimos:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql', # Motor de la base de datos
        'NAME': 'DAW2', # Nombre de la base de datos
        'USER': 'root', # Usuario de la base de datos
        'PASSWORD': 'root', # Contraseña de la base de datos
        'HOST': 'localhost', # Host de la base de datos
        'PORT': '3306', # Puerto de la base de datos
    }
}
```

Y por último lo que tenemos que hacer es crear las migraciones y aplicarlas. Para ello, escribimos:

```
python manage.py makemigrations
python manage.py migrate
```

Despliega la aplicación en el servidor local para comprobar que todo funciona correctamente. Para ello, escribimos:

```
python manage.py runserver
```

Crear la base de datos en MySQL

Tabla de contenidos

Lo primero que tenemos que hacer es crear la base de datos en MySQL. Voy a dejar por aquí el script para crear la base de datos y la tabla que vamos a utilizar lo voy a dejar en el repositorio por si te interesa descargarlo.

```
SET @old_autocommit=@@autocommit;

CREATE DATABASE `DAW2` DEFAULT CHARACTER SET utf8mb4;

USE `DAW2`;

DROP TABLE IF EXISTS `Estudiante`;
```

```
CREATE TABLE `Estudiante` (
  `nif` varchar(9) UNIQUE NOT NULL,
  `nombre_apellido` varchar(50) NOT NULL,
  `edad` tinyint NOT NULL,
  `carrera` varchar(50) NOT NULL,
  `universidad` varchar(50) NOT NULL,
  PRIMARY KEY PK_nif (`NIF`),
  INDEX IDX_Estudiante_Nombre (nombre_apellido)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

INSERT INTO `Estudiante` VALUES (1,'Juan
Perez',20,'Informatica','Universidad de Sevilla');
INSERT INTO `Estudiante` VALUES (2,'Maria Lopez',21,'Medicina','Universidad
de Sevilla');

DROP TABLE IF EXISTS `Curso`;

CREATE TABLE `Curso` (
  `cine` MEDIUMINT NOT NULL,
  `nombre` varchar(50) NOT NULL,
  `creditos` INT(11) NOT NULL,
  `profesor` varchar(50) NOT NULL,
  `universidad` varchar(50) NOT NULL,
  PRIMARY KEY PK_cine (`cine`),
  INDEX IDX_Curso_Nombre (nombre)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

INSERT INTO `Curso` VALUES (1,'Programacion',6,'Juan Perez','Universidad de
Sevilla');
INSERT INTO `Curso` VALUES (2,'Matematicas',6,'Maria Lopez','Universidad de
Sevilla');

DROP TABLE IF EXISTS `EstudianteCurso`;

CREATE TABLE `EstudianteCurso` (
  `nif` varchar(9) UNIQUE NOT NULL,
  `cine`MEDIUMINT NOT NULL,
  PRIMARY KEY PK_nif (`NIF`),
  FOREIGN KEY FK_nif (`nif`) REFERENCES `Estudiante` (`nif`) ON UPDATE
CASCADE,
  FOREIGN KEY FK_cine (`cine`) REFERENCES `Curso` (`cine`) ON UPDATE
CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Conexión a la base de datos en Python

Tabla de contenidos

Lo primero que tenemos que hacer es importar el paquete mysql.connector. Para ello, escribimos:

```
import mysql.connector
```