



Basado en Curso Django de Píldoras Informáticas:

<https://www.youtube.com/watch?v=7XO1AzwkPPE&list=PLU8oAIHdN5BmfwxFO7HdPciOCmmYneAB&index=2>



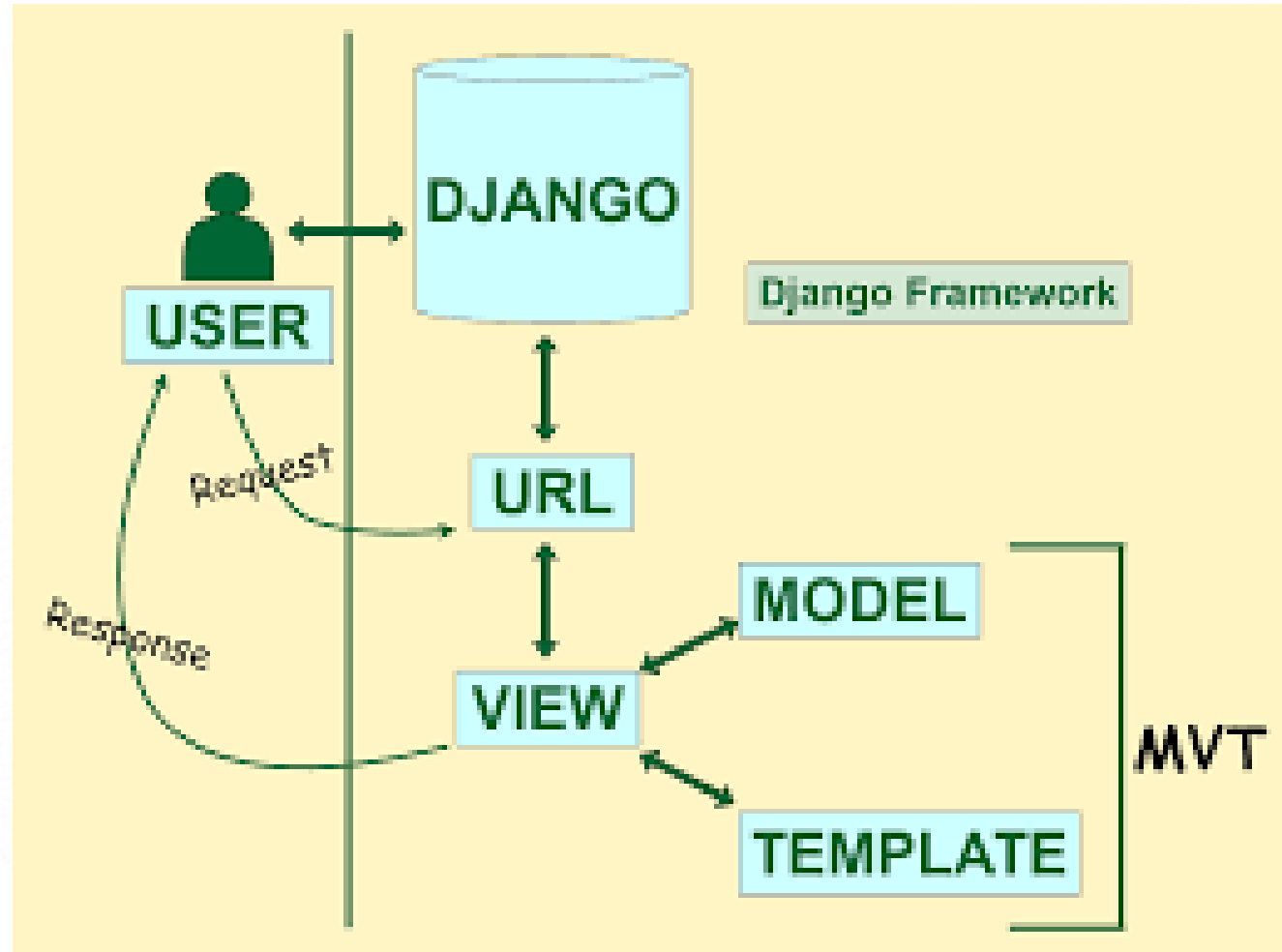
[www.sena.edu.co](http://www.sena.edu.co)



**Píldoras  
Informáticas**

Cursos de informática

# Django Base de Datos (Model)



# Base de Datos Soportadas por Django

Django incorpora compatibilidad directa con las siguientes bases de datos relacionales:

SQLite : La que viene configurada por defecto

PostgreSQL,

MySQL,

Oracle,

MariaDB.

En caso de querer usar otras bases de datos relacionales (como SQL Server) o no relacionales (como MongoDB) se debe hacer uso de librerías específicas para dicho propósito

# Proyectos vs Aplicación en Django

Proyecto Django: Es el conjunto global de configuración y aplicaciones que forman un sitio web. Incluye configuraciones generales, URLs principales, configuraciones de bases de datos, etc. Un proyecto puede contener múltiples aplicaciones.

Aplicación Django: Es un conjunto de funcionalidades relacionadas que se pueden reutilizar en diferentes proyectos. Una aplicación puede incluir modelos, vistas, URLconf, plantillas y archivos estáticos que trabajan juntos para realizar una función específica dentro del sitio web.

En resumen, el proyecto es el sitio web completo, mientras que una aplicación es una parte modular del mismo, diseñada para realizar una función específica.

# Proyectos vs Aplicación en Django

Proyecto Django: Es el conjunto global de configuración y aplicaciones que forman un sitio web. Incluye configuraciones generales, URLs principales, configuraciones de bases de datos, etc. Un proyecto puede contener múltiples aplicaciones.

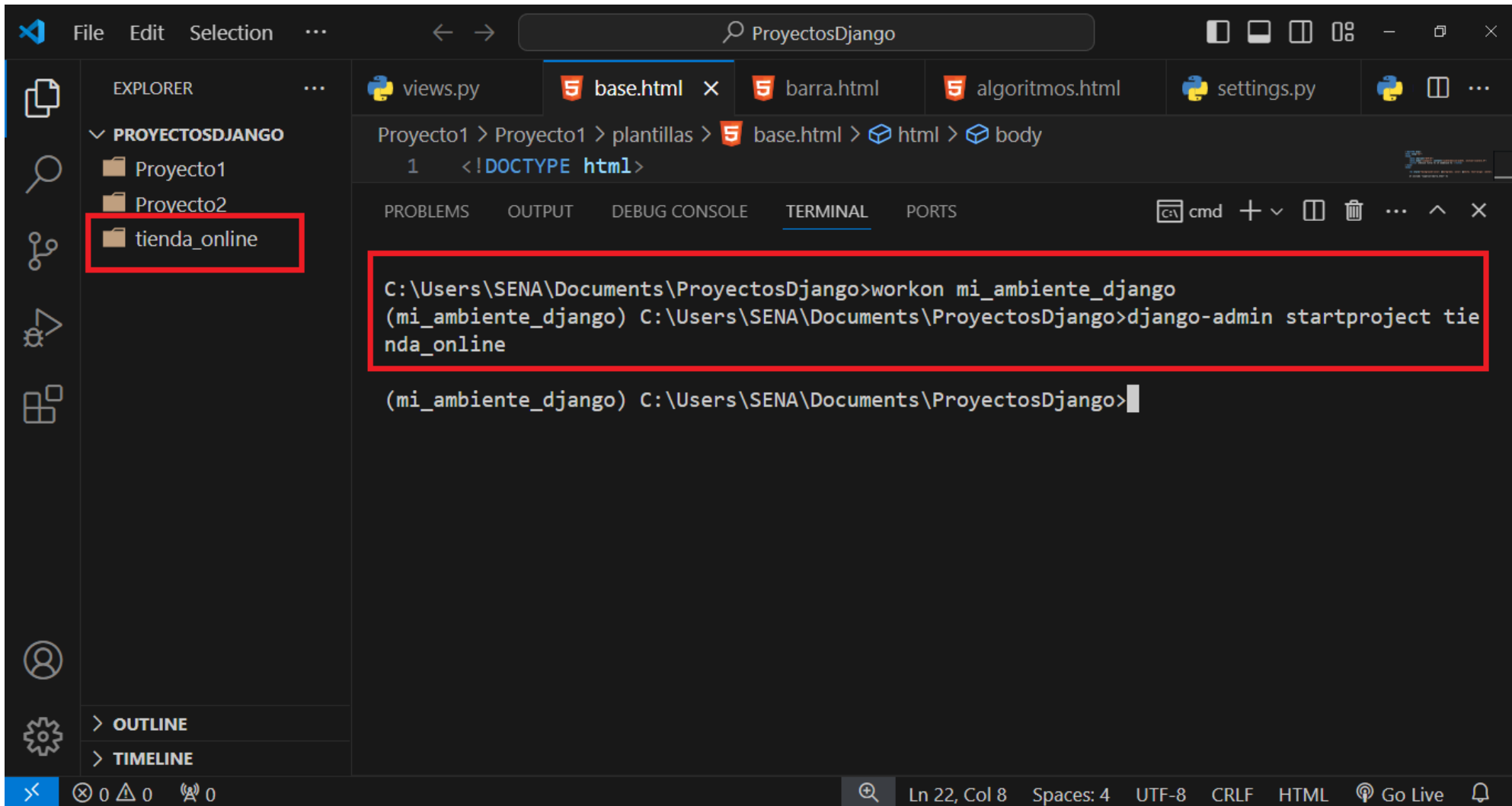
Aplicación Django: Es un conjunto de funcionalidades relacionadas que se pueden reutilizar en diferentes proyectos. Una aplicación puede incluir modelos, vistas, URLconf, plantillas y archivos estáticos que trabajan juntos para realizar una función específica dentro del sitio web.

En resumen, el proyecto es el sitio web completo, mientras que una aplicación es una parte modular del mismo, diseñada para realizar una función específica.

# Proyecto a Realizar: Base de Datos SQLite3



# Crear un nuevo proyecto



# Crear una aplicación

Para crear una aplicación seguiremos los siguientes pasos:

1. Utilizar el comando **django-admin startapp <nombre\_app>** en la terminal para crear una nueva aplicación Django.

Nota: Django generará una estructura de directorios y archivos básica para tu aplicación, incluyendo `models.py`, `views.py`, `urls.py`, y más.

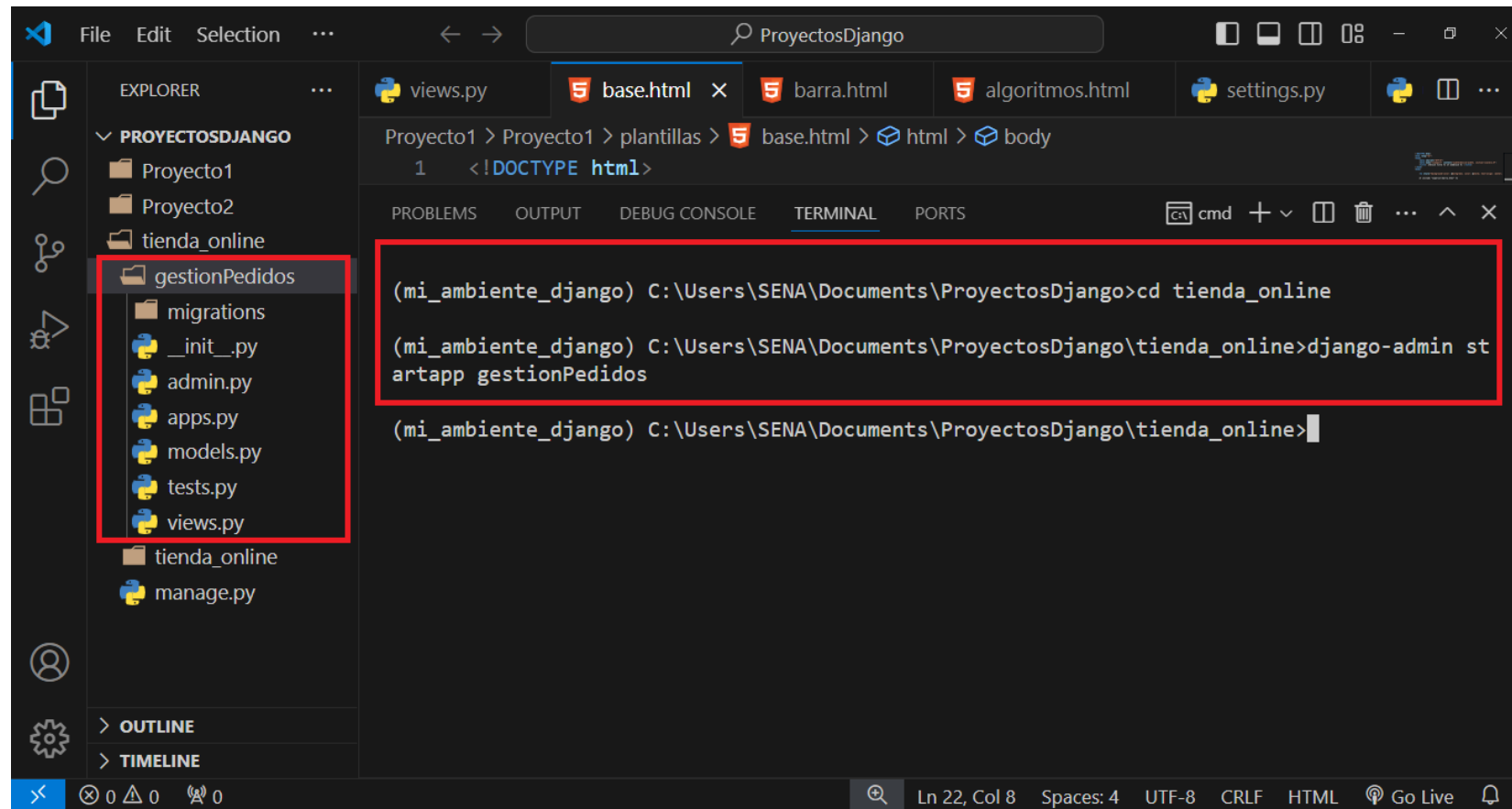
Define tus modelos en `models.py`, tus vistas en `views.py`, y las URL en `urls.py` para gestionar las solicitudes HTTP.

2. Agregar la aplicación al archivo `INSTALLED_APPS` en `settings.py` para que Django la reconozca.



# Crear una aplicación

Estando dentro del proyecto **tienda\_online** ejecutamos el comando:



The screenshot shows the Visual Studio Code interface with the following components:

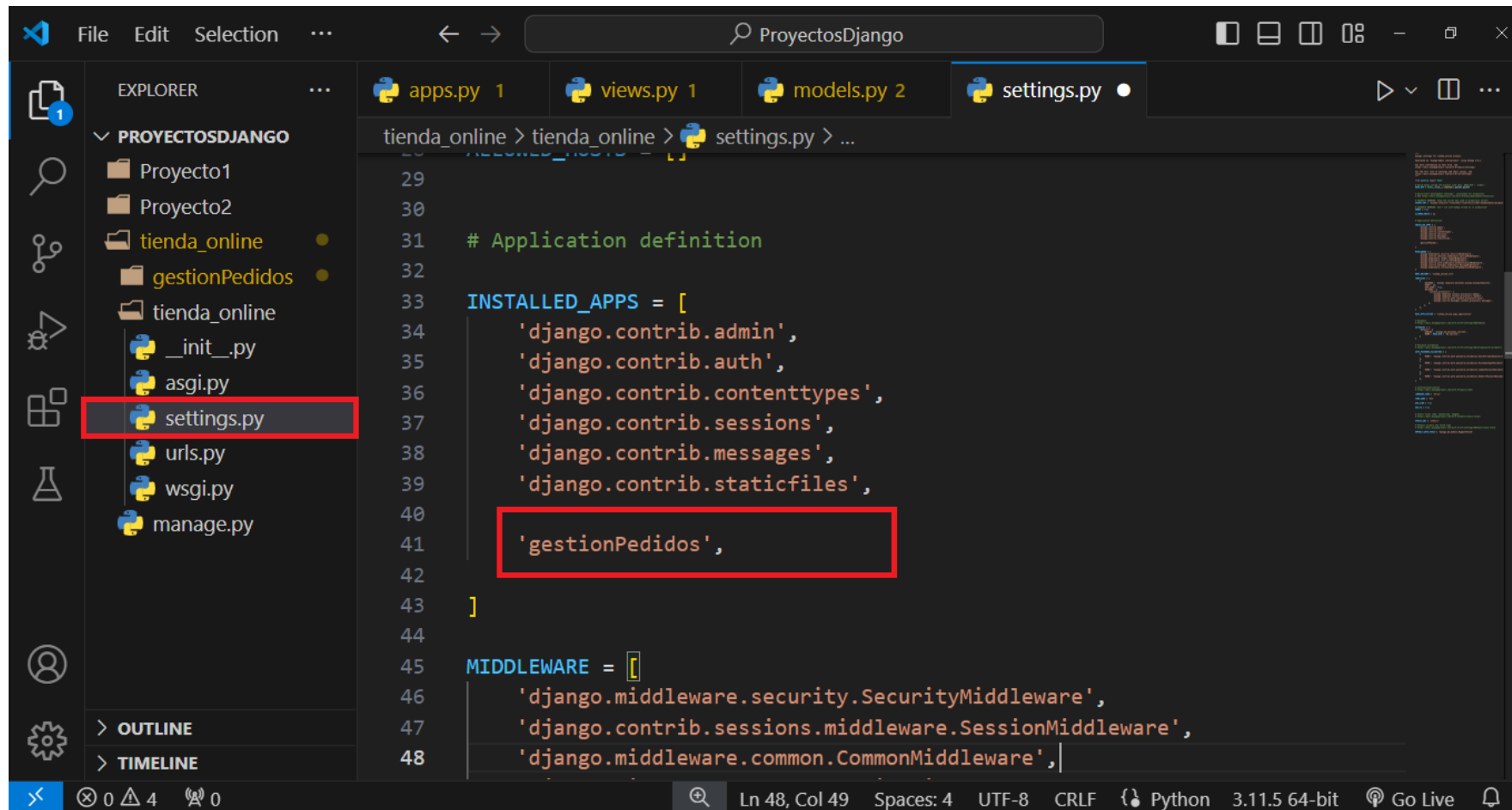
- EXPLORER:** Displays the project structure. The 'tienda\_online' folder is expanded, showing subfolders 'migrations' and 'tienda\_online', and files '\_init\_.py', 'admin.py', 'apps.py', 'models.py', 'tests.py', 'views.py', and 'manage.py'. The 'gestionPedidos' folder is highlighted with a red box.
- EDITOR:** Shows the 'base.html' file with the following content:

```
1 <!DOCTYPE html>
```
- TERMINAL:** Shows the execution of the following commands:

```
(mi_ambiente_django) C:\Users\SENA\Documents\ProyectosDjango>cd tienda_online  
(mi_ambiente_django) C:\Users\SENA\Documents\ProyectosDjango\tienda_online>django-admin st  
artapp gestionPedidos  
(mi_ambiente_django) C:\Users\SENA\Documents\ProyectosDjango\tienda_online>
```

# Registrar la aplicación

Vamos a **settings.py** en **INSTALLED\_APPS**



```

29
30
31 # Application definition
32
33 INSTALLED_APPS = [
34     'django.contrib.admin',
35     'django.contrib.auth',
36     'django.contrib.contenttypes',
37     'django.contrib.sessions',
38     'django.contrib.messages',
39     'django.contrib.staticfiles',
40
41     'gestionPedidos',
42 ]
43
44
45 MIDDLEWARE = [
46     'django.middleware.security.SecurityMiddleware',
47     'django.contrib.sessions.middleware.SessionMiddleware',
48     'django.middleware.common.CommonMiddleware',

```

# Registrar la aplicación



Para verificar que todo marche correctamente utilizamos el comando:

**python manage.py check gestionPedidos**

A screenshot of the Visual Studio Code editor interface. The Explorer panel on the left shows a project structure with folders 'Proyecto1', 'Proyecto2', 'tienda\_online', and 'gestionPedidos'. The 'tienda\_online' folder is expanded, showing files like '\_\_pycache\_\_', '\_\_init\_\_.py', 'asgi.py', 'settings.py', 'urls.py', 'wsgi.py', and 'manage.py'. The 'settings.py' file is open in the editor, showing the 'INSTALLED\_APPS' list with 'gestionPedidos' added at line 40. The Terminal panel at the bottom shows a command prompt where the command 'python manage.py check gestionPedidos' has been executed, resulting in the output 'System check identified no issues (0 silenced)'. The command is highlighted with a red box.

```
(mi_ambiente_django) C:\Users\SENA\Documents\ProyectosDjango\tienda_online>python manage.py check gestionPedidos
System check identified no issues (0 silenced).

(mi_ambiente_django) C:\Users\SENA\Documents\ProyectosDjango\tienda_online>
```

# Creación de Modelos (tablas) en Django

Los modelos en Django son **clases** de Python que representan las tablas de la base de datos. Cada modelo define los campos de la tabla y sus tipos de datos correspondientes. Los modelos también pueden contener métodos que permiten realizar operaciones sobre los datos. En resumen, los modelos en Django proporcionan una forma de interactuar con la base de datos utilizando código Python en lugar de consultas SQL directas. Los pasos para trabajar con modelos en Django son:

1. Definir modelos: Se crean las clases de Python en el archivo **models.py** de la aplicación Django para representar las tablas que se crearan en la base de datos.
2. Generar migraciones ó Crear Base de Datos: Se utiliza el comando **python manage.py makemigrations** para generar archivos de migración basados en los modelos definidos. Es decir, se crea la estructura de la base de datos
3. Aplicar las migraciones: Se emplea el comando **python manage.py migrate** para aplicar las migraciones y crear las tablas en la base de datos seleccionada.

# Tipos de datos que se pueden definir en los modelos de Django

**CharField():** Un campo de texto corto, que generalmente se utiliza para cadenas de caracteres de longitud limitada.

**TextField():** Un campo de texto largo, que puede contener una cantidad considerable de texto.

**IntegerField():** Un campo para almacenar números enteros.

**FloatField():** Un campo para almacenar números de punto flotante.

**DecimalField():** Similar a FloatField, pero utilizado para precisión decimal exacta.

**BooleanField():** Un campo que almacena valores booleanos (True o False).

# Tipos de datos que se pueden definir en los modelos de Django

**DateField():** Un campo para almacenar fechas.

**DateTimeField():** Un campo para almacenar fechas y horas.

**TimeField():** Un campo para almacenar horas.

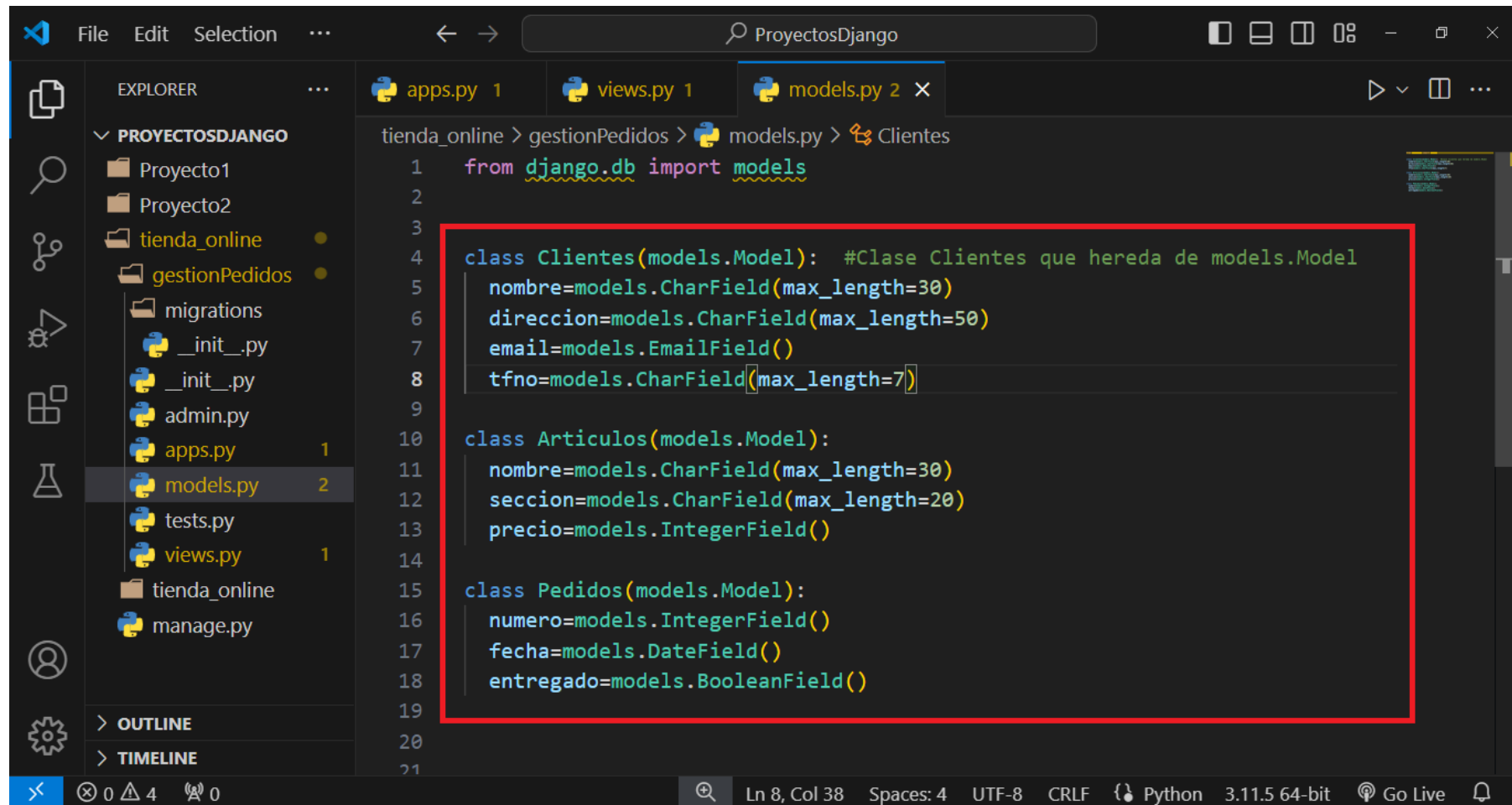
**EmailField():** Un campo para almacenar direcciones de correo electrónico.

**FileField():** Un campo para subir archivos.

**ImageField():** Similar a FileField, pero específicamente diseñado para manejar imágenes.

# 1. Creación de modelos

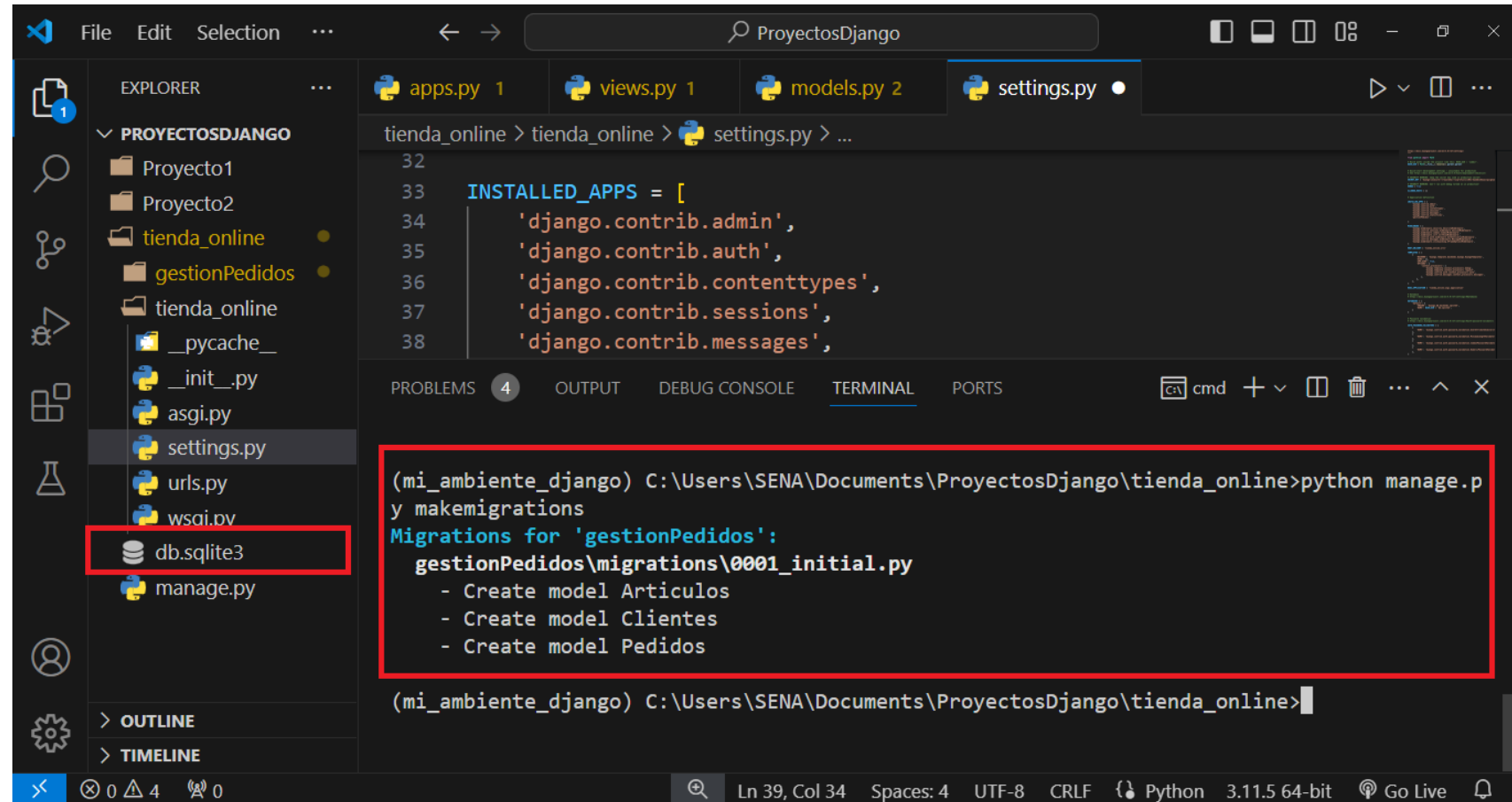
Estando en el archivo **models.py** procedemos a crear las clases que representan las tablas:



```
tienda_online > gestionPedidos > models.py > Clientes
1  from django.db import models
2
3
4  class Clientes(models.Model): #Clase Clientes que hereda de models.Model
5      nombre=models.CharField(max_length=30)
6      direccion=models.CharField(max_length=50)
7      email=models.EmailField()
8      tfno=models.CharField(max_length=7)
9
10 class Articulos(models.Model):
11     nombre=models.CharField(max_length=30)
12     seccion=models.CharField(max_length=20)
13     precio=models.IntegerField()
14
15 class Pedidos(models.Model):
16     numero=models.IntegerField()
17     fecha=models.DateField()
18     entregado=models.BooleanField()
19
20
21
```

## 2. Generar migraciones:

Se utiliza el comando **python manage.py makemigrations** para generar archivos de migración basados en los modelos definidos. Es decir, se crea la base de datos.



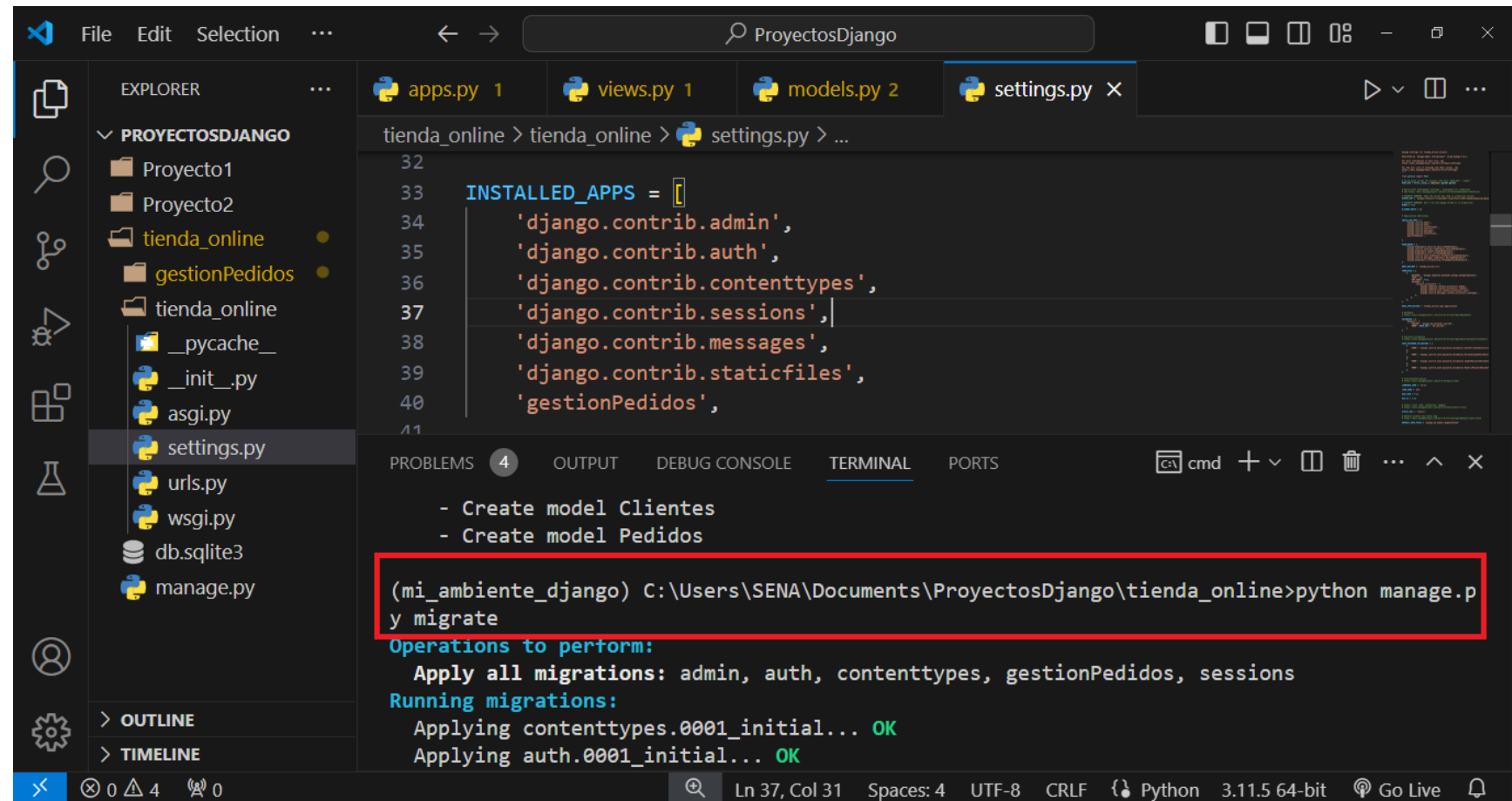
The screenshot shows a Visual Studio Code editor with a Django project named 'ProyectosDjango'. The Explorer sidebar on the left shows the project structure, including a folder named 'tienda\_online' which contains a subfolder 'gestionPedidos'. The 'db.sqlite3' file is highlighted in the Explorer. The main editor window shows the 'settings.py' file with the 'INSTALLED\_APPS' list. The Terminal panel at the bottom shows the command `python manage.py makemigrations` being executed, resulting in the creation of a migration file `gestionPedidos\migrations\0001_initial.py`. The migration details are as follows:

```
(mi_ambiente_django) C:\Users\SENA\Documents\ProyectosDjango\tienda_online>python manage.py makemigrations
Migrations for 'gestionPedidos':
  gestionPedidos\migrations\0001_initial.py
    - Create model Articulos
    - Create model Clientes
    - Create model Pedidos
```



## 3. Aplicar las migraciones:

Se emplea el comando **python manage.py migrate** para aplicar las migraciones y crear las tablas en la base de datos SQLite3.



The screenshot shows the Visual Studio Code interface with a Django project named 'ProyectosDjango'. The Explorer panel on the left shows the project structure, including 'tienda\_online' and 'gestionPedidos' folders. The main editor displays the 'settings.py' file, where the 'INSTALLED\_APPS' list includes 'django.contrib.admin', 'django.contrib.auth', 'django.contrib.contenttypes', 'django.contrib.sessions', 'django.contrib.messages', 'django.contrib.staticfiles', and 'gestionPedidos'. The TERMINAL panel at the bottom shows the command `(mi_ambiente_django) C:\Users\SENA\Documents\ProyectosDjango\tienda_online>python manage.py migrate` being executed. The output indicates that all migrations will be applied: 'admin', 'auth', 'contenttypes', 'gestionPedidos', and 'sessions'. The terminal also shows the progress of applying migrations, with 'contenttypes.0001\_initial...' and 'auth.0001\_initial...' both marked as 'OK'.

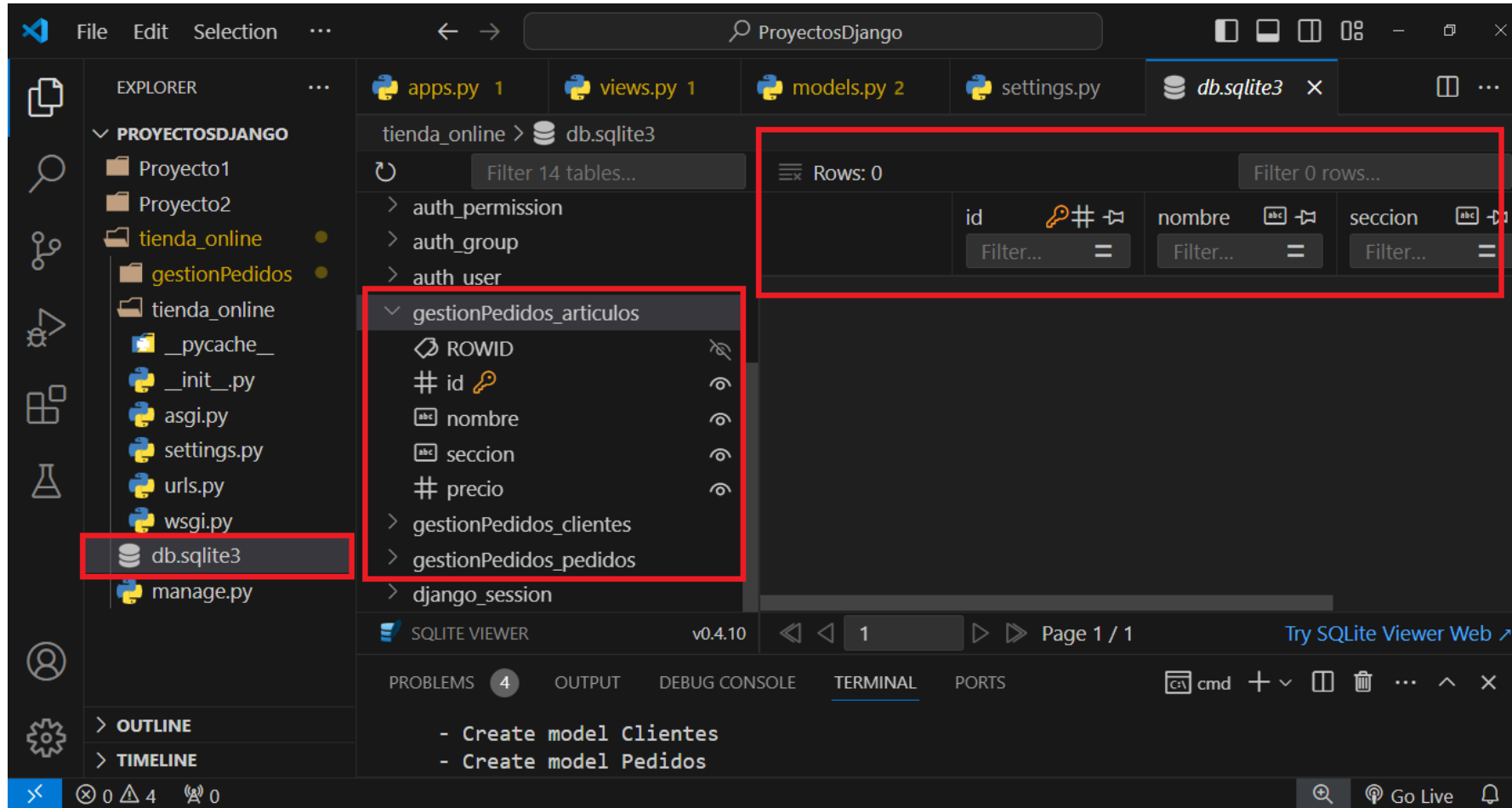
```
tienda_online > tienda_online > settings.py > ...
32
33 INSTALLED_APPS = [
34     'django.contrib.admin',
35     'django.contrib.auth',
36     'django.contrib.contenttypes',
37     'django.contrib.sessions',
38     'django.contrib.messages',
39     'django.contrib.staticfiles',
40     'gestionPedidos',
41 ]

PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL PORTS
- Create model Clientes
- Create model Pedidos

(mi_ambiente_django) C:\Users\SENA\Documents\ProyectosDjango\tienda_online>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, gestionPedidos, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
```

### 3. Aplicar las migraciones:

Se pueden revisar la base de datos y las tablas que hemos creado, se observa que Django agrega automáticamente un campo **id** para la llave primaria :



The screenshot displays the Django database interface within VS Code. The Explorer panel on the left shows the project structure, with 'db.sqlite3' highlighted. The middle panel shows the 'gestionPedidos\_articulos' table with fields: ROWID, id (primary key), nombre, seccion, and precio. The right panel shows the table structure with columns: id, nombre, and seccion. The bottom panel shows the terminal output: '- Create model Clientes' and '- Create model Pedidos'.

# Shell de Django

Es una interfaz de línea de comandos que permite interactuar directamente con un proyecto Django utilizando código Python. Se puede acceder a este shell ejecutando el comando **python manage.py shell** desde la raíz del proyecto Django.

Una vez dentro del shell interactivo de Django, se tiene acceso a todo el entorno del proyecto, incluyendo modelos, vistas, funciones y cualquier otra configuración definida. Esto permite explorar y probar rápidamente el código, realizar consultas a la base de datos, crear o modificar objetos de la aplicación y realizar otras tareas de desarrollo de manera interactiva.

```
(mi_ambiente_django) C:\Users\SENA\Documents\ProyectosDjango\tienda_online>python manage.p
y shell
Python 3.11.5 (tags/v3.11.5:cce6ba9, Aug 24 2023, 14:38:34) [MSC v.1936 64 bit (AMD64)] on
win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> █
```

# CRUD de registros usando el Shell de Django: Create

- Importar el modelo correspondiente que se desea utilizar para insertar el registro. Esto se puede hacer utilizando una declaración **import**, por ejemplo:

```
from <mi_aplicacion.models> import <MiModelo>.
```

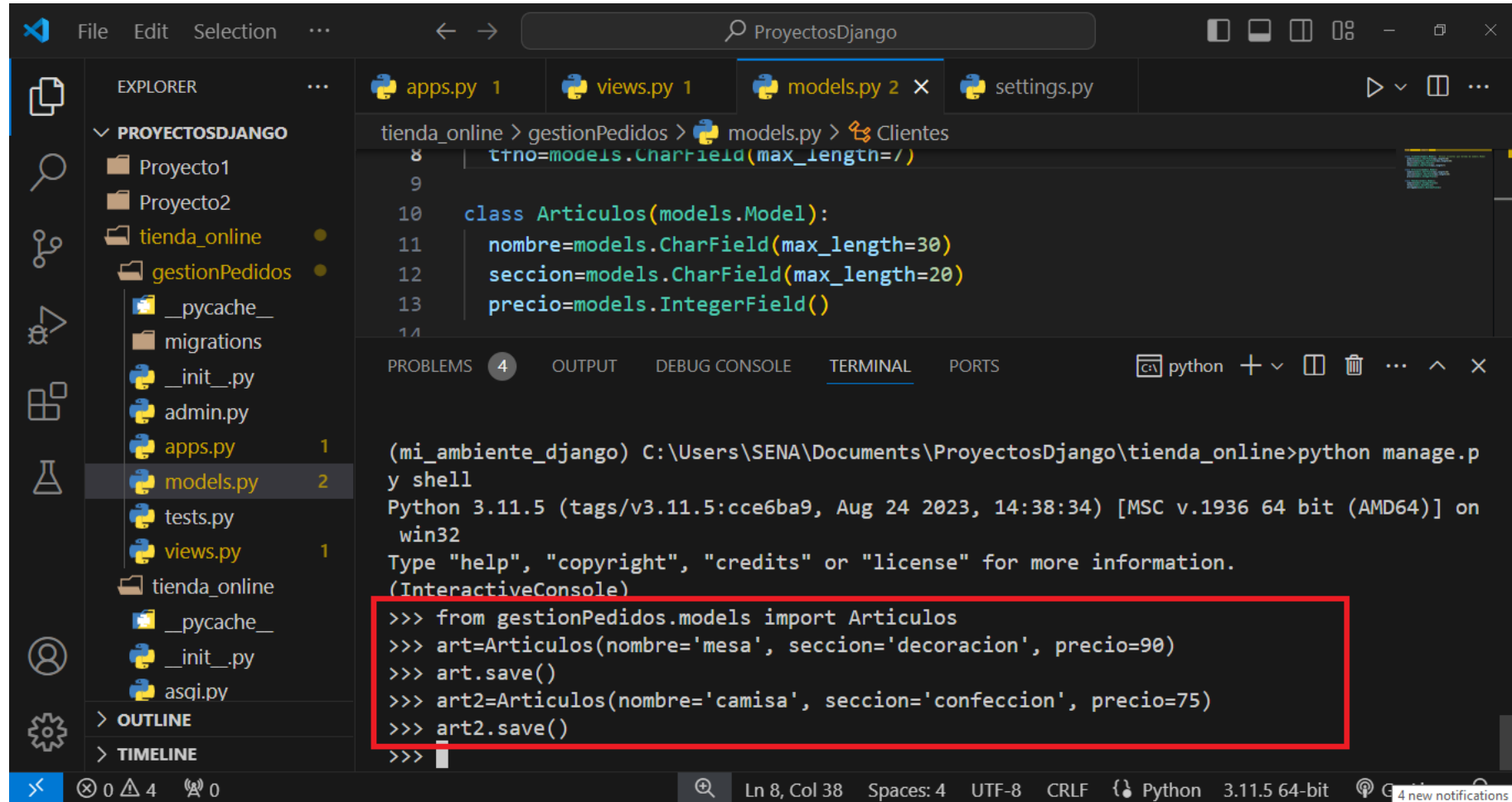
- Crear una instancia del modelo con los valores deseados para cada campo. Esto se hace instanciando la clase del modelo y asignando valores a los atributos correspondientes. Por ejemplo:

```
nuevo_registro = MiModelo(campo1='valor1', campo2='valor2').
```

- Guardar el registro en la base de datos utilizando el método `save()`. Por ejemplo:

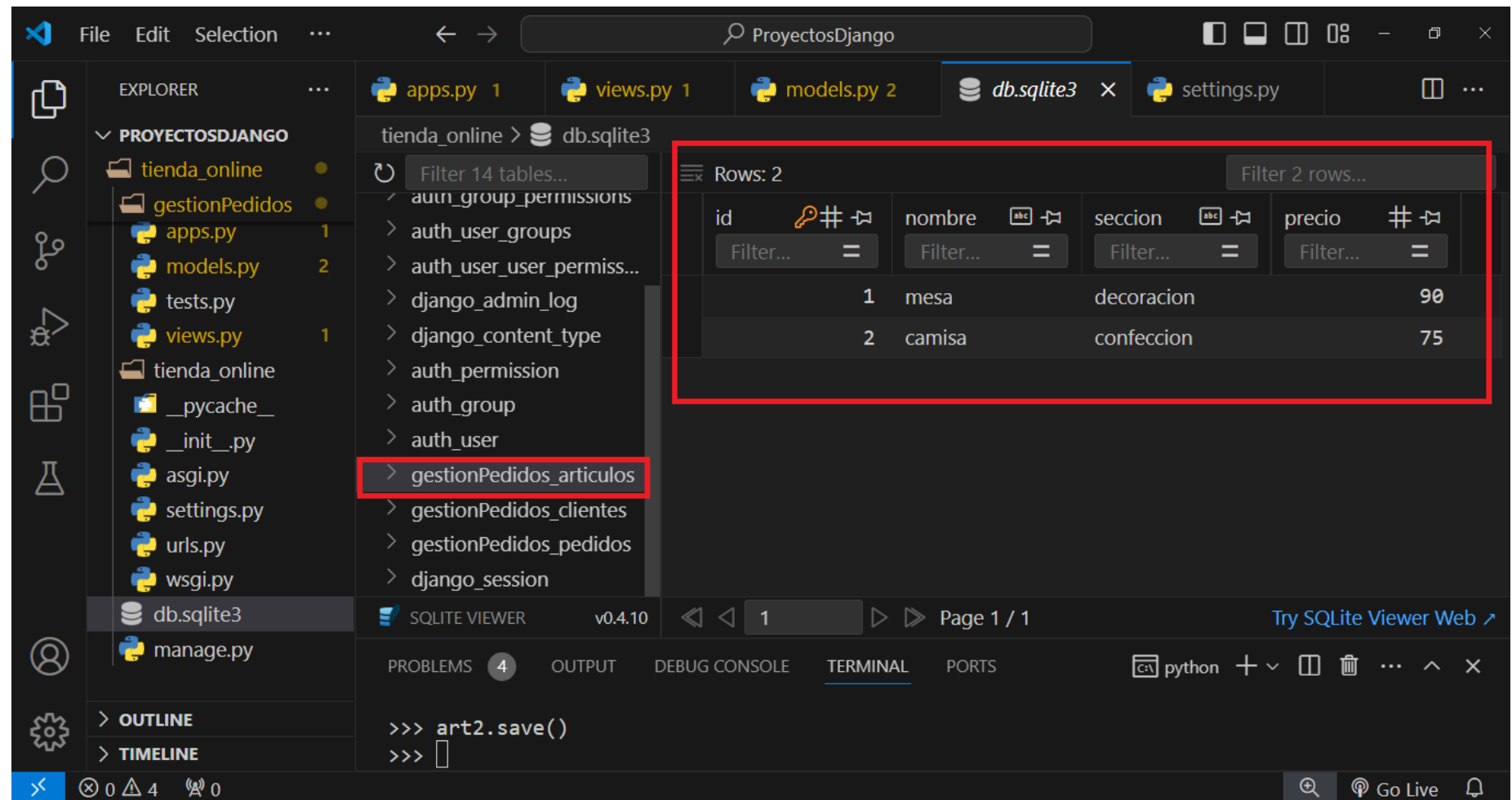
```
nuevo_registro.save()
```

## Insertamos dos registros en el modelo o la tabla Articulos



# CRUD de registros usando el Shell de Django

Verificamos la inserción



The screenshot shows a Visual Studio Code editor with a Django project named 'ProyectosDjango'. The Explorer panel on the left shows the project structure, including the 'tienda\_online' app. The 'tienda\_online' app folder is expanded, showing files like 'apps.py', 'models.py', 'views.py', and 'urls.py'. The 'db.sqlite3' file is selected, and the SQLite Viewer extension is open, displaying the database schema and data.

The SQLite Viewer shows the following tables:

- auth\_group\_permissions
- auth\_user\_groups
- auth\_user\_user\_permissions
- django\_admin\_log
- django\_content\_type
- auth\_permission
- auth\_group
- auth\_user
- gestionPedidos\_articulos
- gestionPedidos\_clientes
- gestionPedidos\_pedidos
- django\_session

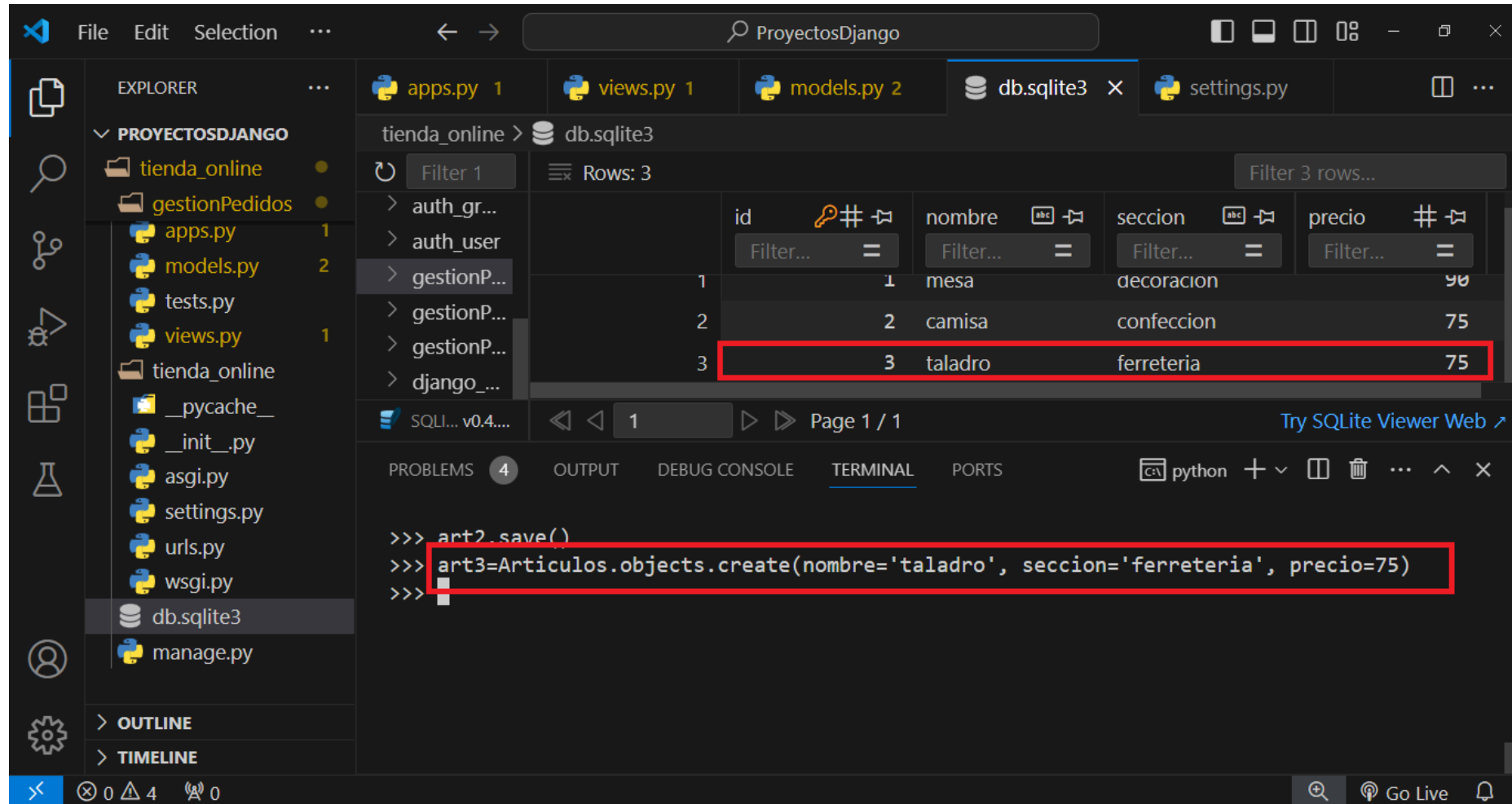
The 'gestionPedidos\_articulos' table is highlighted in red. The table data is as follows:

id	nombre	seccion	precio
1	mesa	decoracion	90
2	camisa	confeccion	75

The terminal at the bottom shows the command `>>> art2.save()` being executed.

# CRUD de registros usando el Shell de Django

Insertamos otro artículo en la tabla Artículos usando un solo paso con el método `Artículos.objects.create()`



The screenshot shows the Django project structure in VS Code. The `tienda_online` app is selected, and the `db.sqlite3` database is open. The `Articulos` table is displayed with the following data:

id	nombre	seccion	precio
1	mesa	decoracion	90
2	camisa	confeccion	75
3	taladro	ferreteria	75

The terminal shows the following Django shell commands:

```
>>> art2.save()
>>> art3=Articulos.objects.create(nombre='taladro', seccion='ferreteria', precio=75)
>>>
```

# CRUD de registros usando el Shell de Django: Update

- Importar el modelo correspondiente que se desea utilizar para actualizar el registro. Esto se hace mediante una declaración de importación, por ejemplo:

**`from mi_aplicacion.models import MiModelo.`**

- Obtener el registro que se desea actualizar utilizando consultas o métodos de filtrado proporcionados por Django. Por ejemplo:

**`registro = MiModelo.objects.get(id=1).`**

- Actualizar los campos del registro obtenido según sea necesario. Esto se hace directamente modificando los atributos del objeto obtenido. Por ejemplo:

**`registro.campo1 = 'nuevo_valor'.`**

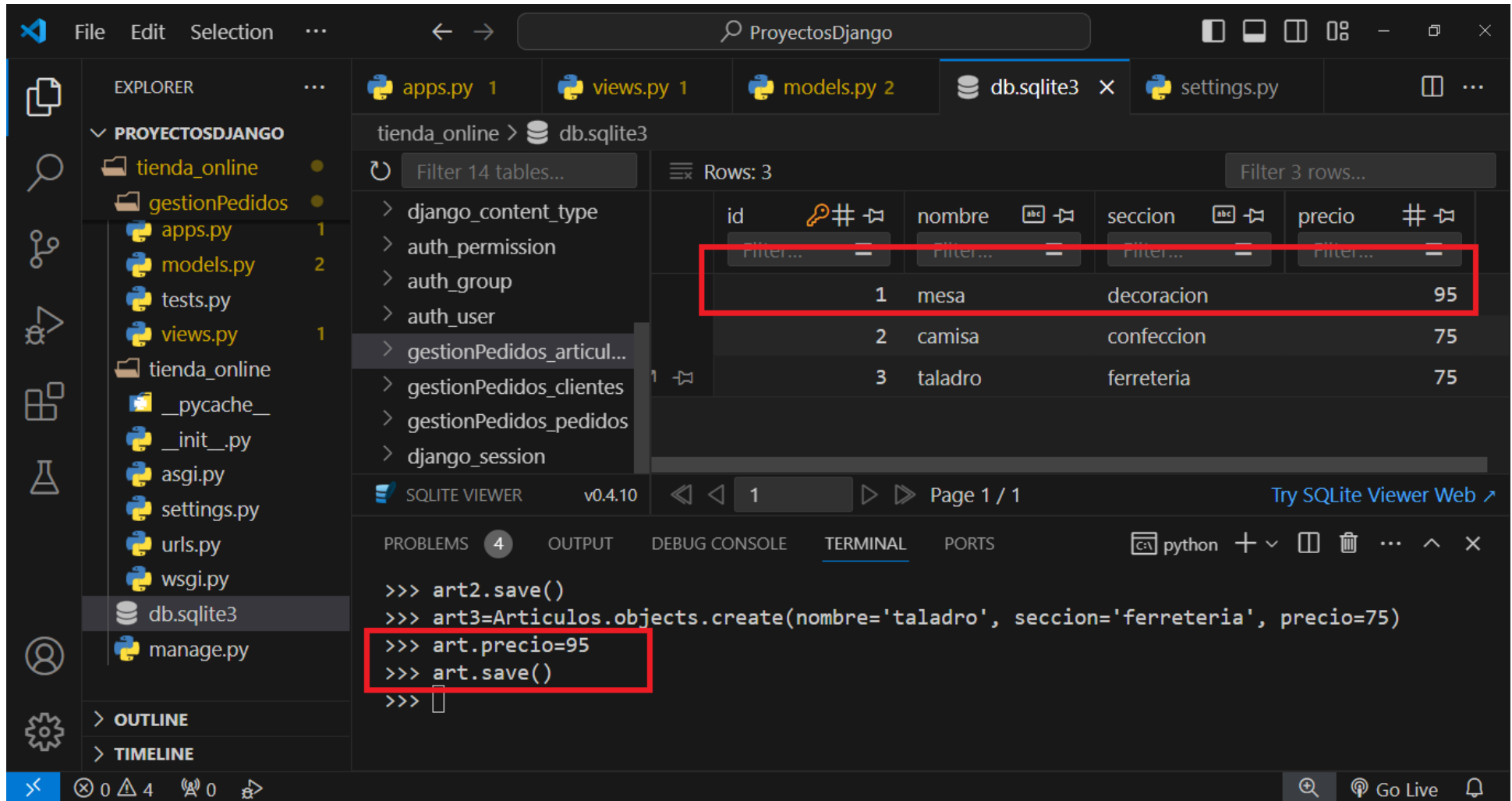
- Guardar los cambios en la base de datos utilizando el método `save()` en el objeto del registro. Por ejemplo:

**`registro.save().`**



# CRUD de registros usando el Shell de Django

Actualizar el primer artículo insertado en la tabla **Articulos**



The screenshot shows a Django project named 'ProyectosDjango' in VS Code. The Explorer panel on the left shows the project structure, including the 'tienda\_online' app. The SQLite Viewer panel in the center displays the 'Articulos' table with 3 rows. The first row is highlighted with a red box, showing an article with id 1, name 'mesa', section 'decoracion', and price 95. The Terminal panel at the bottom shows the following commands:

```
>>> art2.save()
>>> art3=Articulos.objects.create(nombre='taladro', seccion='ferreteria', precio=75)
>>> art.precio=95
>>> art.save()
>>>
```

The last two lines of the terminal output are highlighted with a red box.

# CRUD de registros usando el Shell de Django: Delete

- Importar el modelo correspondiente que se desea utilizar para eliminar el registro. Esto se realiza mediante una declaración de importación, por ejemplo:

```
from mi_aplicacion.models import MiModelo.
```

- Obtener el registro que se desea eliminar utilizando consultas o métodos de filtrado proporcionados por Django. Por ejemplo:

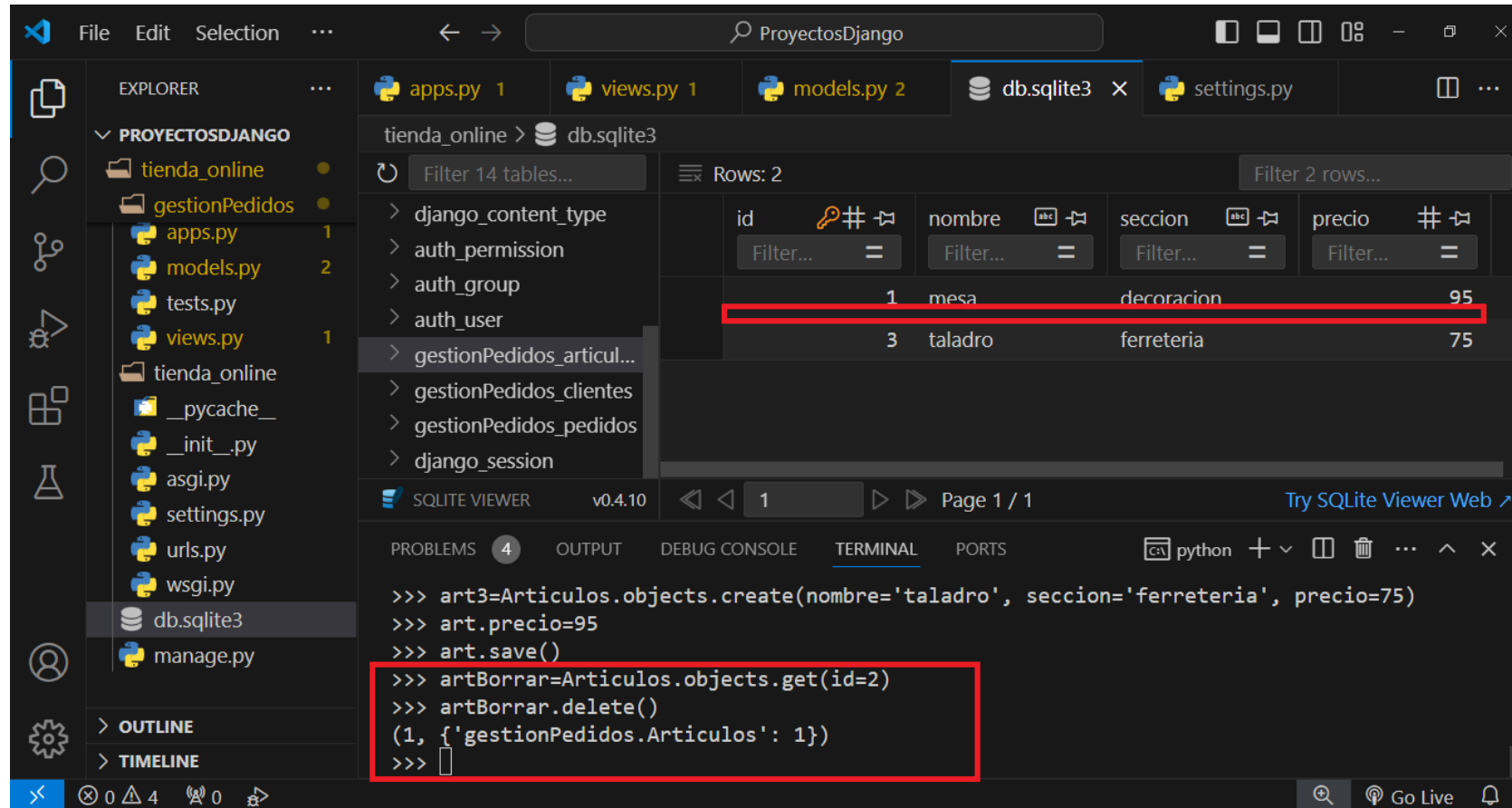
```
registro = MiModelo.objects.get(id=1).
```

- Eliminar el registro utilizando el método delete() en el objeto del registro obtenido. Por ejemplo:

```
registro.delete().
```

# CRUD de registros usando el Shell de Django

Borrar el segundo artículo insertado en la tabla **Articulos**



The screenshot shows a Visual Studio Code editor with a Django project named 'ProyectosDjango'. The Explorer sidebar shows the project structure, including 'tienda\_online' and 'gestionPedidos'. The 'db.sqlite3' database is open in the SQLite Viewer, showing a table with 2 rows. The first row has id=1, nombre='mesa', seccion='decoración', and precio=95. The second row has id=3, nombre='taladro', seccion='ferreteria', and precio=75. The terminal window shows the following Python code being executed:

```
>>> art3=Articulos.objects.create(nombre='taladro', seccion='ferreteria', precio=75)
>>> art.precio=95
>>> art.save()
>>> artBorrar=Articulos.objects.get(id=2)
>>> artBorrar.delete()
(1, {'gestionPedidos.Articulos': 1})
>>>
```

# CRUD de registros usando el Shell de Django: Read

- Para consultar registros utilizando el shell de Django, se pueden seguir estos pasos :
- Importar el modelo correspondiente que contiene los registros que se desean consultar. Esto se realiza mediante una declaración de importación, por ejemplo:

```
from mi_aplicacion.models import MiModelo.
```

- Realizar consultas utilizando métodos proporcionados por Django, como `all()`, `filter()`, `get()`, etc. Por ejemplo:

```
registros = MiModelo.objects.all()
```

para obtener todos los registros de la tabla.

- Iterar sobre los resultados para procesar o mostrar la información según sea necesario. Por ejemplo:

## CRUD de registros usando el Shell de Django: Read

- Iterar sobre los resultados para procesar o mostrar la información según sea necesario.  
Por ejemplo:

```
for registro in registros:  
    print(registro.campo1)
```

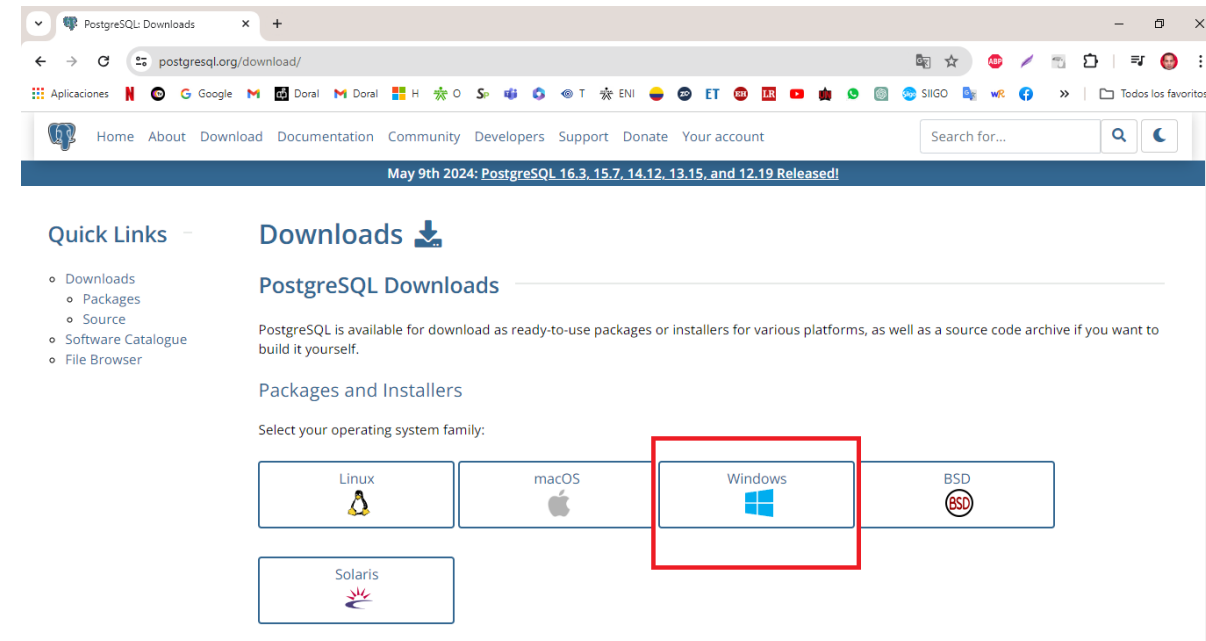
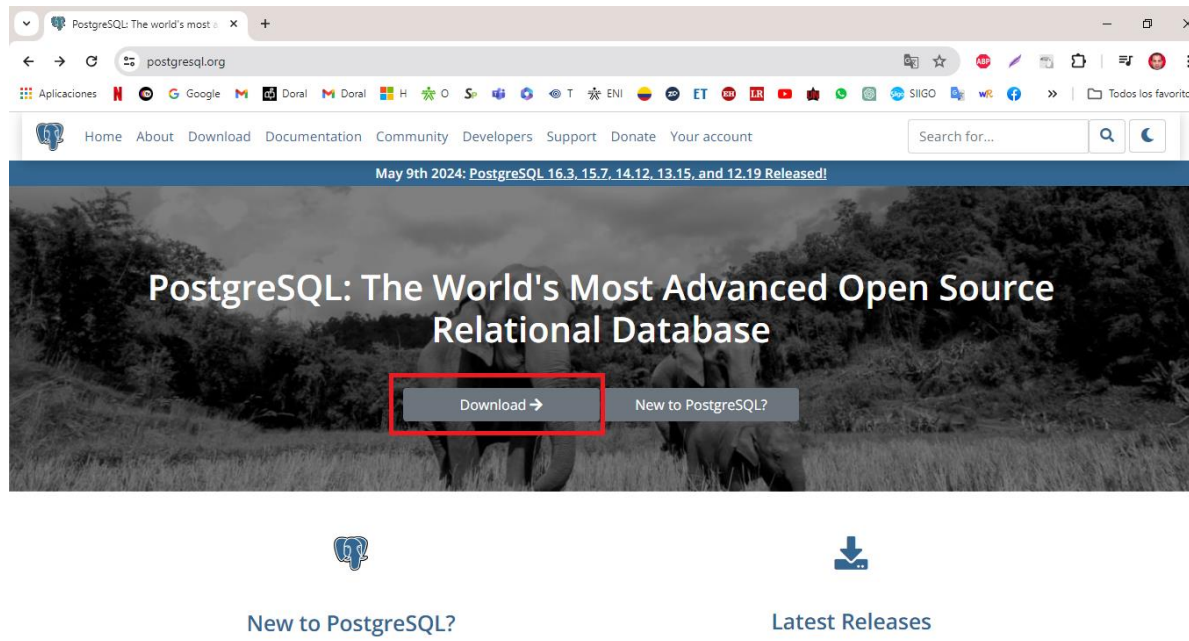
Nota: Para ejecutar el for exterior en el Shell recuerda dar doble enter al finalizar el bucle

```
>>> Lista=Articulos.objects.all()  
>>> for registro in Lista:  
...     print(registro.nombre)  
...  
mesa  
taladro
```

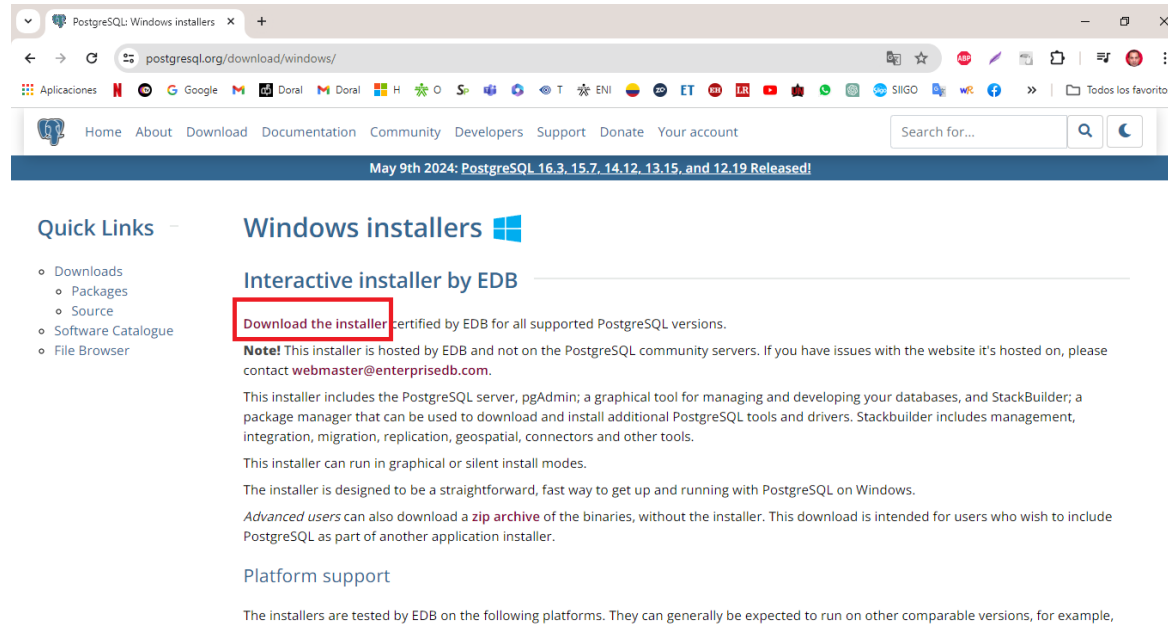
- Para salir del Shell de Django utilizar el comando **exit()**

# Utilizar PostgreSQL con Django: Instalando PostgreSQL

1. Descargar PostgreSQL desde: <https://www.postgresql.org/>



# Utilizar PostgreSQL con Django: Instalando PostgreSQL



PostgreSQL: Windows installers

May 9th 2024: PostgreSQL 16.3, 15.7, 14.12, 13.15, and 12.19 Released!

**Quick Links**

- Downloads
- Packages
- Source
- Software Catalogue
- File Browser

**Windows installers**

**Interactive installer by EDB**

**Download the installer** certified by EDB for all supported PostgreSQL versions.

**Note!** This installer is hosted by EDB and not on the PostgreSQL community servers. If you have issues with the website it's hosted on, please contact [webmaster@enterprisedb.com](mailto:webmaster@enterprisedb.com).

This installer includes the PostgreSQL server, pgAdmin; a graphical tool for managing and developing your databases, and StackBuilder; a package manager that can be used to download and install additional PostgreSQL tools and drivers. Stackbuilder includes management, integration, migration, replication, geospatial, connectors and other tools.

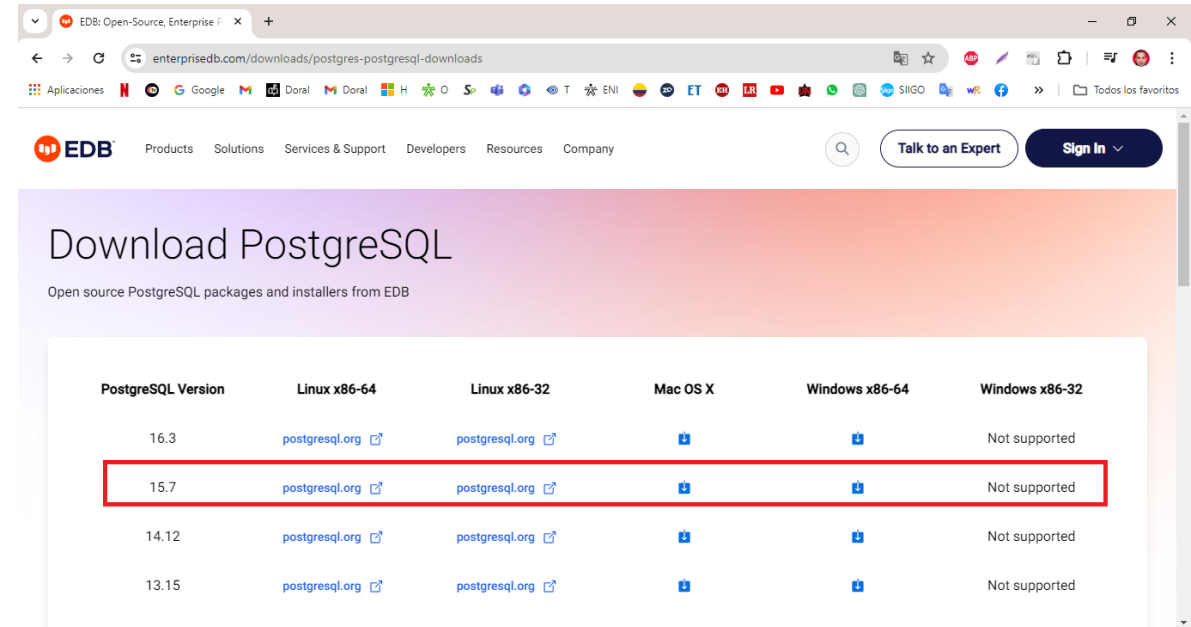
This installer can run in graphical or silent install modes.

The installer is designed to be a straightforward, fast way to get up and running with PostgreSQL on Windows.

Advanced users can also download a **zip archive** of the binaries, without the installer. This download is intended for users who wish to include PostgreSQL as part of another application installer.

**Platform support**

The installers are tested by EDB on the following platforms. They can generally be expected to run on other comparable versions, for example,



EDB: Open-Source, Enterprise PostgreSQL

Products Solutions Services & Support Developers Resources Company

**Download PostgreSQL**

Open source PostgreSQL packages and installers from EDB

PostgreSQL Version	Linux x86-64	Linux x86-32	Mac OS X	Windows x86-64	Windows x86-32
16.3	<a href="#">postgresql.org</a>	<a href="#">postgresql.org</a>			Not supported
15.7	<a href="#">postgresql.org</a>	<a href="#">postgresql.org</a>			Not supported
14.12	<a href="#">postgresql.org</a>	<a href="#">postgresql.org</a>			Not supported
13.15	<a href="#">postgresql.org</a>	<a href="#">postgresql.org</a>			Not supported

