Esta clase va a ser

- grabada

Clase 18. PYTHON

Portfolio Parte II

Temario

17

Django Portfolio Parte I

- ✓ Django
- ✓ Plantillas Django

18

Django Portfolio Parte II

- ✓ Mejoras en las plantillas
- ✓ Modelo
- ✓ ¿Cómo crear una app?

19

Playground Intermedio Parte I

- ✓ Profundizando en MVT
- ✓ Visitas
- ✓ URLs

CODERHOUSE

Objetivos de la clase

- **Reconocer** un modelo.
- **Identificar** algunos tag de los templates.
- **Crear** nuestro portafolio.

Repositorio Github

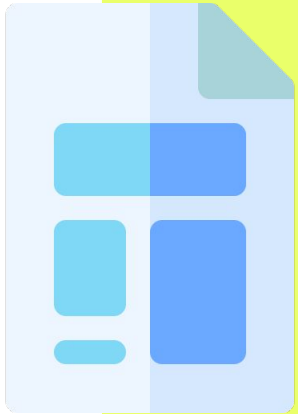
Te dejamos el acceso al Repositorio de Github donde encontrarás todo el material complementario y scripts de la clase.

✓ Repositorio Python

CODERHOUSE

Mejoras en las plantillas

Variables a las plantillas



Desde una plantilla podemos hacer referencia a variables de nuestra vista. Para esto debemos darle esa variable al contexto, lo usaremos por primera vez.

Veámoslo con nuestra función **probandoTemplate** de la clase anterior. El envío de variables al contexto se realizará desde diccionarios de python. 👉

CODERHOUSE

Variables a las plantillas

Por ejemplo, creamos las variables nom y ap, con ellas generamos un diccionario y se lo enviamos al contexto:

Luego en mi plantilla haremos referencia a lo enviado en el diccionario, usando **{{ xxx }}**

```python
def probandoTemplate(self):

    nom = "Nicolas"
    ap = "Perez"

    diccionario = {"nombre":nom, "apellido":ap}  #<-----Para enviar al contexto

    miHtml = open("C:/Users/nico_/Desktop/PythonProtecto1/Proyecto1/Proyecto1/plantillas/template1.html")

    plantilla = Template(miHtml.read()) #Se carga en memoria nuestro documento, template1
    ##OJO importar template y contex, con: from django.template import Template, Context

    miHtml.close() #Cerramos el archivo

    miContexto = Context(diccionario) #le doy al contexto mi nombre y apellido

    documento = plantilla.render(miContexto) #Aca renderizamos la plantilla en documento

    return HttpResponse(documento)
```

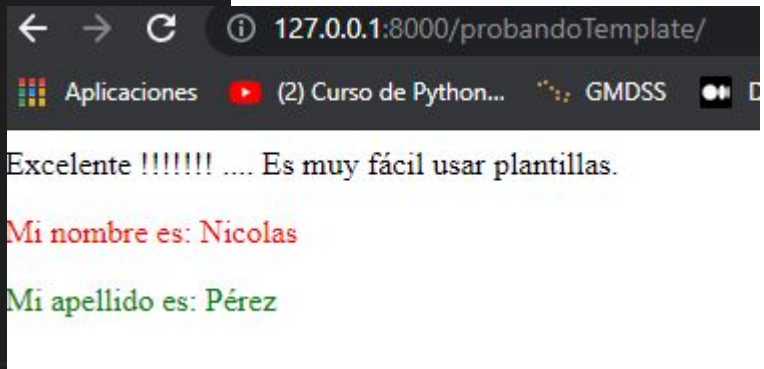
CODERHOUSE

¡A no preocuparse si no sabemos HTML, con lo que veamos, será más que suficiente!

CODERHOUSE

Variables a las plantillas

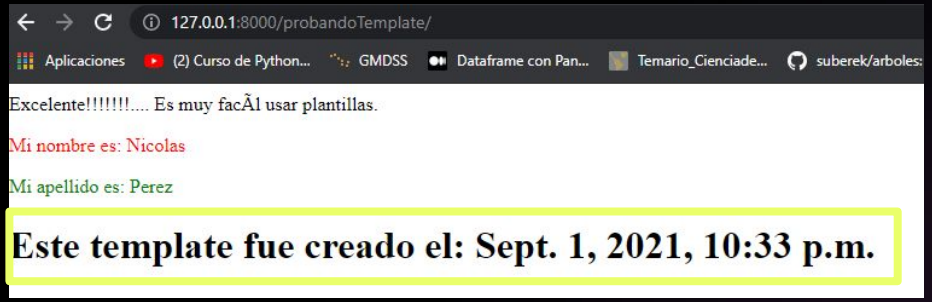
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>


    Excelente!!!!!!!.... Es muy facíl usar plantillas.


    <p style="color: red">Mi nombre es: {{nombre}}</p>
    <p style="color: green">Mi apellido es: {{apellido}}</p>

</body>
</html>
```



CODERHOUSE

Para pensar

¿Cómo lograrían esto?



Contesta mediante el chat de Zoom

CODERHOUSE

Bucles y Condicionales

CODERHOUSE

Para pensar

Supongamos que quiero enviar una variable compleja a mi template, como por ejemplo una clase, una clase padre heredada, una tupla o una lista. ¿Se podrá?

Contesta mediante el chat de Zoom

Las plantillas nos sorprenden

Efectivamente, se puede y es muy sencillo, veamos un ejemplo con una lista:

Claramente es igual 😍

```python
def probandoTemplate(self):

    nom = "Nicolas"
    ap = "Perez"

    listaDeNotas = [2,2,3,7,2,5]

                                                    #GUAU enviamos una variable compleja como una lista
    diccionario = {"nombre":nom, "apellido":ap, "hoy":datetime.datetime.now(), "notas":listaDeNotas} #<-----Para enviar al contexto

    miHtml = open("C:/Users/nico_/Desktop/PythonProtecto1/Proyecto1/Proyecto1/plantillas/template1.html")

    plantilla = Template(miHtml.read()) #Se carga en memoria nuestro documento, template1
    ##OJO importar template y contex, con: from django.template import Template, Context

    miHtml.close() #Cerramos el archivo

    miContexto = Context(diccionario) #le doy al contexto mi nombre y apellido

    documento = plantilla.render(miContexto) #Aca renderizamos la plantilla en documento

    return HttpResponse(documento)
```

CODERHOUSE

Las plantillas nos sorprenden

Podemos ejecutar código Python desde la plantilla gracias a Django, usando **{{}}** para variables, o **{% %}** para bucles y condicionales.

Veamos cómo recorrer esa lista 👀

Abre el for

Recorre

Cierra el for

```
Excelente!!!!!!.... Es muy facíl usar plantillas.


<p style="color:red">Mi nombre es: {{nombre}}</p>
<p style="color:green">Mi apellido es: {{apellido}}</p>


<h1>Este template fue creado el: {{hoy}}</h1>

<h3>Estas son mis notas de la Universidad: </h3>

<p>
{% for  n in notas %}

    {{n}}
    <br>

{% endfor %}
<p>
```

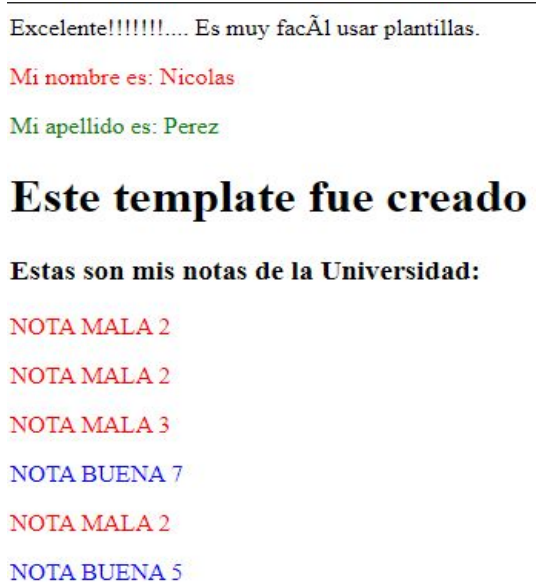
CODERHOUSE

Las plantillas nos sorprenden

Ahora le agregaremos un condicional, si la nota es menor que 4 en rojo, en otro caso en azul:

```
<p>
{% for  n in notas %}

    {% if n < 4 %}
        <p style="color:red"> NOTA MALA {{n}}</p>
    {% else %} <p style="color:blue">NOTA BUENA {{n}}</p>
    {% endif %}


{% endfor %}
<p>
```

Más TAG: Built-in template tags and filters | docs.djangoproject.com



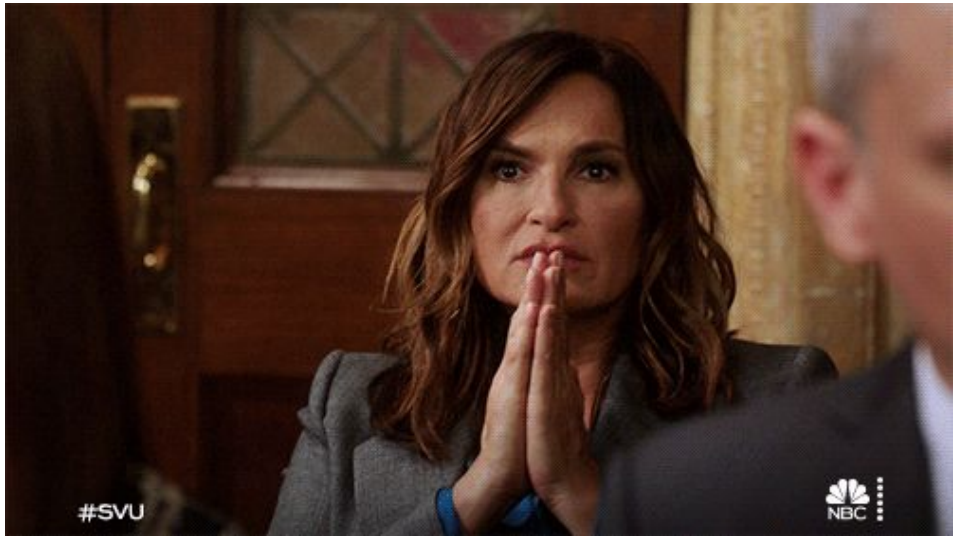
Cargadores de plantillas

Cargadores de plantilla

Si tuvieras que cargar muchas plantillas, hacer esto sería horrible:

```python
def probandoTemplate(self):

    nom = "Nicolas"
    ap = "Perez"

    diccionario = {"nombre":nom, "apellido":ap}  #<-----Para enviar al contexto

    miHtml = open("C:/Users/nico_/Desktop/PythonProtecto1/Proyecto1/Proyecto1/plantillas/template1.html")

    plantilla = Template(miHtml.read()) #Se carga en memoria nuestro documento, template1
    ##OJO importar template y contex, con: from django.template import Template, Context

    miHtml.close() #Cerramos el archivo

    miContexto = Context(diccionario) #le doy al contexto mi nombre y apellido

    documento = plantilla.render(miContexto) #Aca renderizamos la plantilla en documento

    return HttpResponse(documento)
```

Para solucionar ésto, aparece el concepto de **Cargador** con el que podrás manejar de forma más ordenada las plantillas.



CODERHOUSE

Cargamos plantillas desde Django

¿Cómo hacemos esto? Guardamos todas las plantillas en la misma carpeta y se lo avisamos a Django, en nuestro caso la carpeta se llamaba plantillas.

1. Primero en las vistas, importamos el cargador:
   from django.template import loader
2. Cargamos la plantilla **pero sin el open**, para eso abrimos el archivo **settings.py,** buscamos la lista TEMPLATES y en DIRS[] ponemos la ruta, en este caso:

C:/Users/nico_/Desktop/PythonProtecto1/Proyecto1/Proyecto1/plantillas/

Cargamos plantillas desde Django

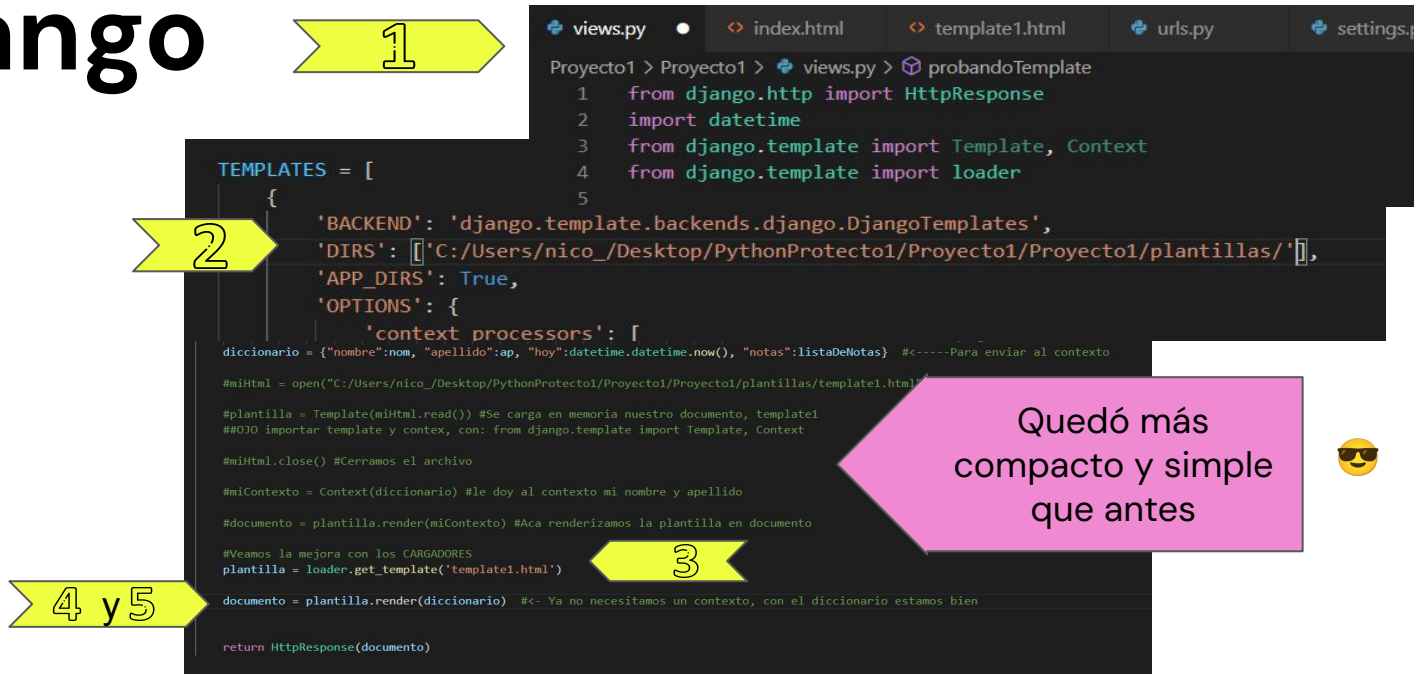
3. Volvemos a la vista y llamamos a nuestra plantilla con:
   loader.get_pemplate('plantilla1.html)

4. Luego renderizamos:
   render(contexto)

5. El contexto debe dejar de serlo, ahora **debe** ser un diccionario.

Veamos entonces qué fue todo esto 👉

Cargamos plantillas desde Django



CODERHOUSE

Agregando cargadores

Modifica alguna de tus vistas para que funcione usando cargadores.

Duración: **15 minutos**

CODERHOUSE

ACTIVIDAD EN CLASE

Agregando cargadores

Modifica alguna de tus vistas para que funcione usando cargadores.

CODERHOUSE

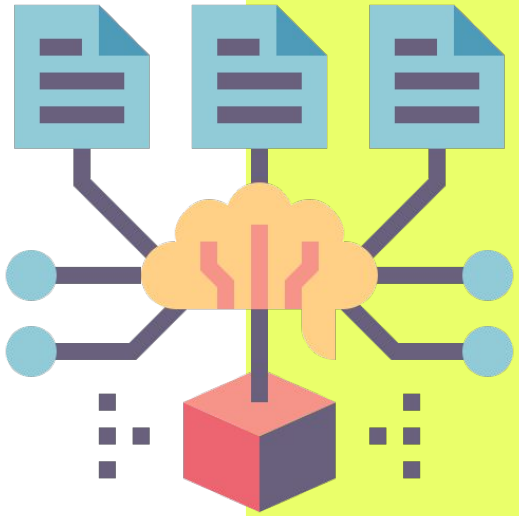
Break

¡10 minutos y volvemos!

MODELO

¿Qué sabemos de Django?

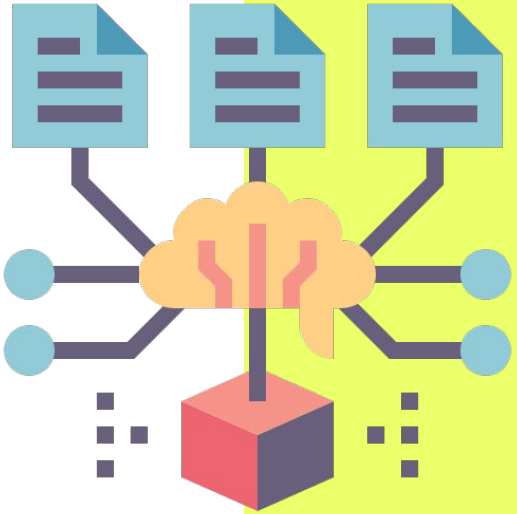
Hemos visto que Django se basa en el **patrón MTV**, ya conocemos los **Templates (lo que se ve)**, **las View (la información que se pasa al template)**, ahora nos falta saber qué es el **modelo.**



CODERHOUSE

¿Qué es el MODELO?

El modelo es la parte de nuestro proyecto que almacena, borra, modifica y manipula el caudal principal de los datos, apoyándose en alguna bb.dd.
De base de datos no debemos saber mucho gracias a Django 😉
En nuestro caso usaremos la base de datos integrada **SQlite** que viene integrada y simplificada.



CODERHOUSE

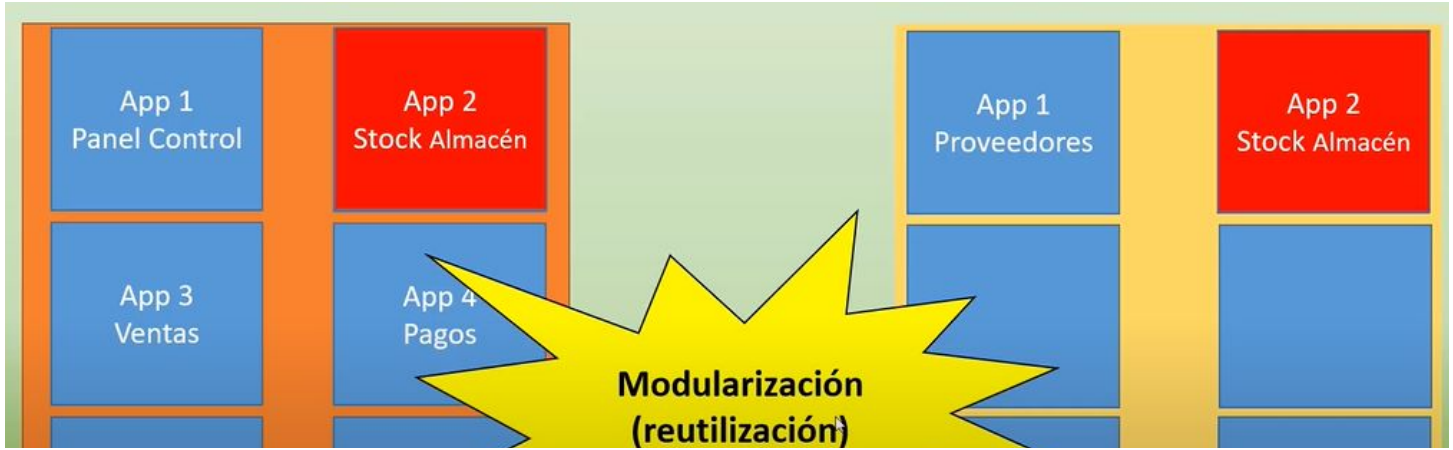
Proyecto vs Aplicación

PARA RECORDAR

Ya sabemos realizar un proyecto de Django, pero Django hace una distinción entre proyecto y aplicación. Puntualmente, un proyecto es todo, dentro del proyecto habrá varias aplicaciones, donde cada aplicación tendrá su función.

Proyecto vs Aplicación

Es fundamental tratar de crear aplicaciones dentro del proyecto para que el programa sea entendible y fácilmente manipulable.



CODERHOUSE

¿Cómo crear una aplicación?

CODERHOUSE

¿Cómo crear una aplicación?

Crearemos la aplicación AppCoder 😎

Dentro de esa aplicación, tendremos a los **estudiantes**, a los **Profesores** y por ejemplo los **Entregables**.

- ✓ **Estudiantes** (nombre, apellido, email)
- ✓ **Profesor** (nombre, apellido, email, profesión)
- ✓ **Entregable** (nombre, fechaDeEntrega,entregado)
- ✓ **Curso** (nombre, camada (“ o comisión”))

Esto último es el MODELO de nuestra app, es la información que manejará nuestro proyecto por medio de nuestra APP.

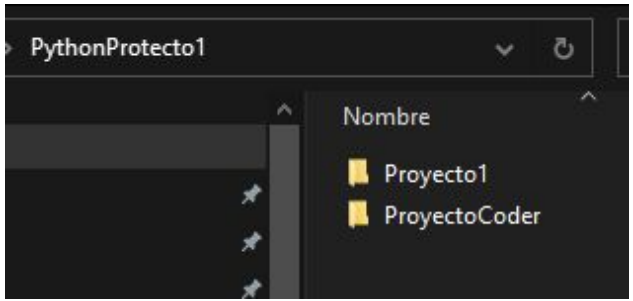
¿Cómo crear una aplicación?

Empecemos con un proyecto nuevo que se llamará **ProyectoCoder**, ese proyecto tendrá una **AppCoder** y dentro de esa app tendremos los modelos para **Estudiantes**, **Profesor**, **Entregable** y **Curso**.

👉

¿Cómo crear una aplicación?

```
PS C:\Users\nico_\Desktop\PythonProtecto1> django-admin startproject ProyectoCoder



PS C:\Users\nico_\Desktop\PythonProtecto1> python manage.py startapp AppCoder
C:\Users\nico_\AppData\Local\Microsoft\WindowsApps\python.exe: can't open file 'C:\Users\ni
o 2] No such file or directory
PS C:\Users\nico_\Desktop\PythonProtecto1> cd ProyectoCoder
PS C:\Users\nico_\Desktop\PythonProtecto1\ProyectoCoder> python manage.py startapp AppCoder
PS C:\Users\nico_\Desktop\PythonProtecto1\ProyectoCoder>
PS C:\Users\nico_\Desktop\PythonProtecto1\ProyectoCoder>
```



1. Crear proyecto: **django-admin startproject ProyectoCoder**
2. Entramos al proyecto con cd ProyectoCoder
3. Creamos la app: **python manage.py startapp AppCoder**

Crear el modelo

Vamos a **models.py** que se encuentra en nuestra app, **AppCoder** y creamos la estructura de nuestros modelos, por medio de clases, empezaremos por **Curso**:

ProyectoCoder > AppCoder > models.py > Curso

```python
from django.db import models

# Create your models here.
class Curso(models.Model):

    nombre=models.CharField(max_length=40)
    camada = models.IntegerField()

```

CODERHOUSE

Crear el modelo

Sigamos con las otras:

Ya tenemos la estructura de nuestra app. Entonces debemos informar a Django.

- ✓ Vamos a **settings,**
- ✓ **INSTALLED_APPS** y
- ✓ **Agregamos nuestra APP.**

ProyectoCoder > AppCoder > models.py > Entregable

```python
from django.db import models

# Create your models here.
class Curso(models.Model):

    nombre=models.CharField(max_length=40)
    camada = models.IntegerField()


class Estudiante(models.Model):
    nombre= models.CharField(max_length=30)
    apellido= models.CharField(max_length=30)
    email= models.EmailField()

class Profesor(models.Model):
    nombre= models.CharField(max_length=30)
    apellido= models.CharField(max_length=30)
    email= models.EmailField()
    profesion= models.CharField(max_length=30)

class Entregable(models.Model):
    nombre= models.CharField(max_length=30)
    fechaDeEntrega = models.DateField()
    entregado = models.BooleanField()

```

Crear el modelo

Es buen momento para ver si hemos realizado todo bien hasta ahora:

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

python manage.py check AppCoder

```
PS C:\Users\nico_\Desktop\PythonProtecto1\ProyectoCoder> python manage.py check AppCoder
System check identified no issues (0 silenced).
PS C:\Users\nico_\Desktop\PythonProtecto1\ProyectoCoder>
```

Crear el modelo – BB.DD

Llegó la hora de transformar nuestros modelos en base de datos:

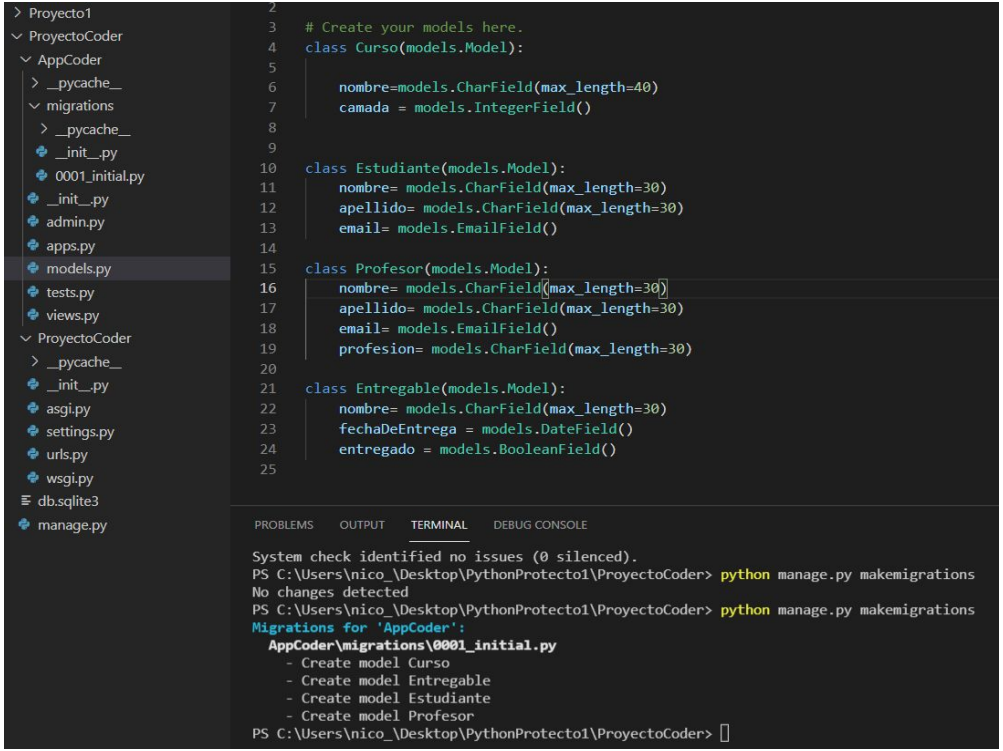
python manage.py makemigrations

```
PS C:\Users\nico_\Desktop\PythonProtecto1\ProyectoCoder> python manage.py makemigrations
Migrations for 'AppCoder':
  AppCoder\migrations\0001_initial.py
    - Create model Curso
    - Create model Entregable
    - Create model Estudiante
    - Create model Profesor
PS C:\Users\nico_\Desktop\PythonProtecto1\ProyectoCoder>
```

CODERHOUSE

Crear el modelo – BB.DD

¡Ya se ha creado la BB.DD!



CODERHOUSE

Crear el modelo - BB.DD

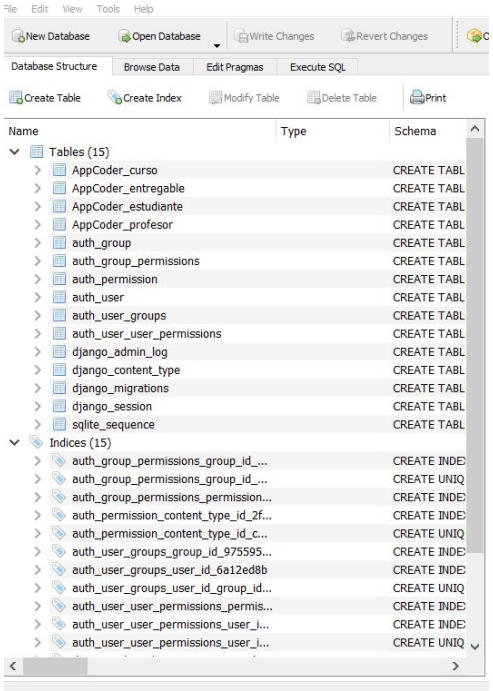
Ya tenemos la BD, pero aún está vacía, debemos generar su estructura, para eso:

1. **python manage.py sqlmigrate AppCoder 0001**
   (Eso nos dará muchas líneas en código sql)

2. **python manage.py migrate**
   (Con esto, esas líneas de sql impactan en nuestra base de datos)

DB Browser

DB Browser

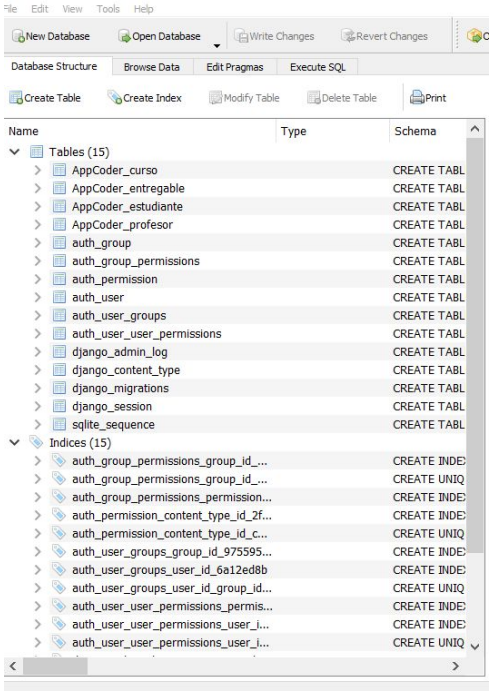


Hasta ahora deberían confiar en nosotros, porque no tienen manera de saber si de verdad se creó la base de datos.

Para saberlo y verlo de una forma más tangible, instalaremos **DB Browser**, que nos permite acceder a la información de una base de datos **SQLite**.

CODERHOUSE

DB Browser



Instalarlo es muy sencillo, solo siguiente, siguiente, siguiente, etc.

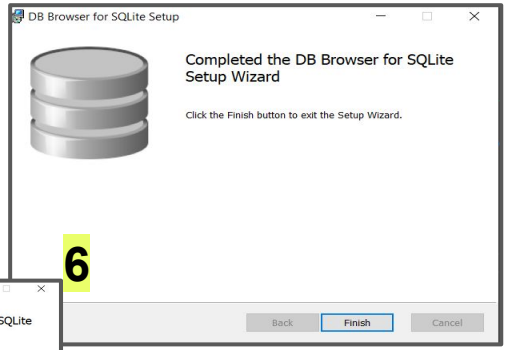
Tutorial de instalación DB Browser:
DB Browser for SQLite | sqlitebrowser.org

Lo abrimos y vemos nuestra base de datos

👉

CODERHOUSE

DB Browser – Instalación



CODERHOUSE

Agregar información a mi BD

CODERHOUSE

Ejemplo en vivo

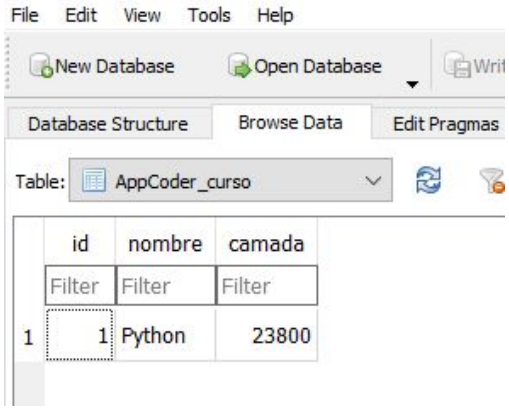
Agregaremos info a nuestra BD desde consola
y desde DB Browser.

Agregar información a mi BB.DD

- ✓ Agregaremos algún registro, por ejemplo agregaremos algún curso.
- ✓ Vamos a la consola y escribimos **python manage.py shell** para pararnos en la consola.
- ✓ Luego vamos a importar el modelo, en este caso la clase Curso.
  - **from AppCoder.models import Curso**
- ✓ Pasamos a crear un Curso:

curso = Curso(nombre="Python", camada=23800)

- ✓ Luego enviamos ese curso a la BD: **curso.save()**



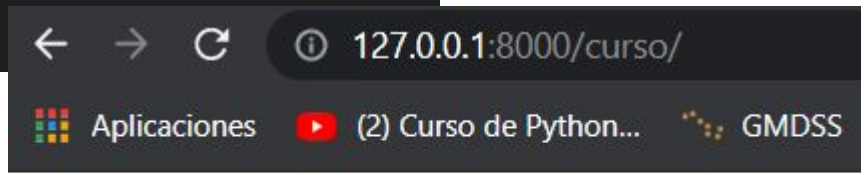
CODERHOUSE

Agregar información a mi BD

Ya probamos desde la consola que todo funciona, ahora miremos cómo quedaría una vista que guarda datos y luego los muestra en la web. Sería lo mismo, pero solo haciendo una vista, y modificando el archivo urls.py.

```python
def curso(self):

    curso =  Curso(nombre="Desarrollo web", camada="19881")
    curso.save()
    documentoDeTexto = f"--->Curso: {curso.nombre}    Camada: {curso.camada}"


    return HttpResponse(documentoDeTexto)
```

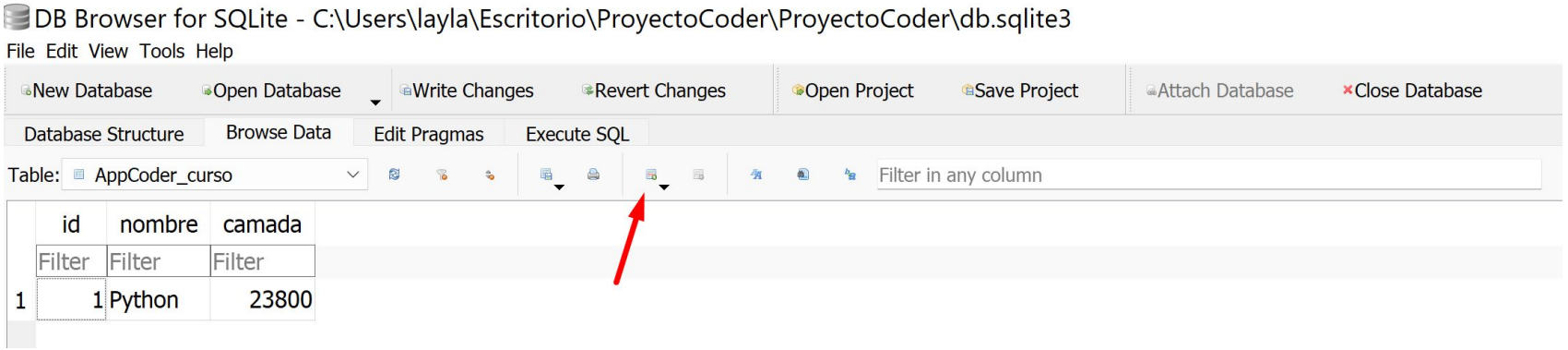
```python
from AppCoder.views import curso

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



--->Curso: Desarrollo web Camada: 19881

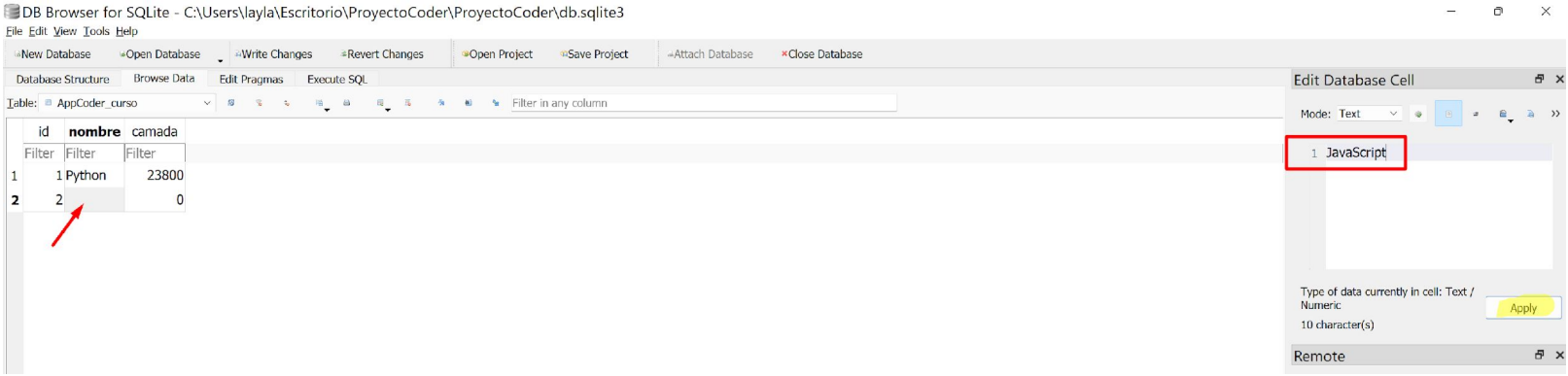
CODERHOUSE

Otra manera de agregar información a mi BD



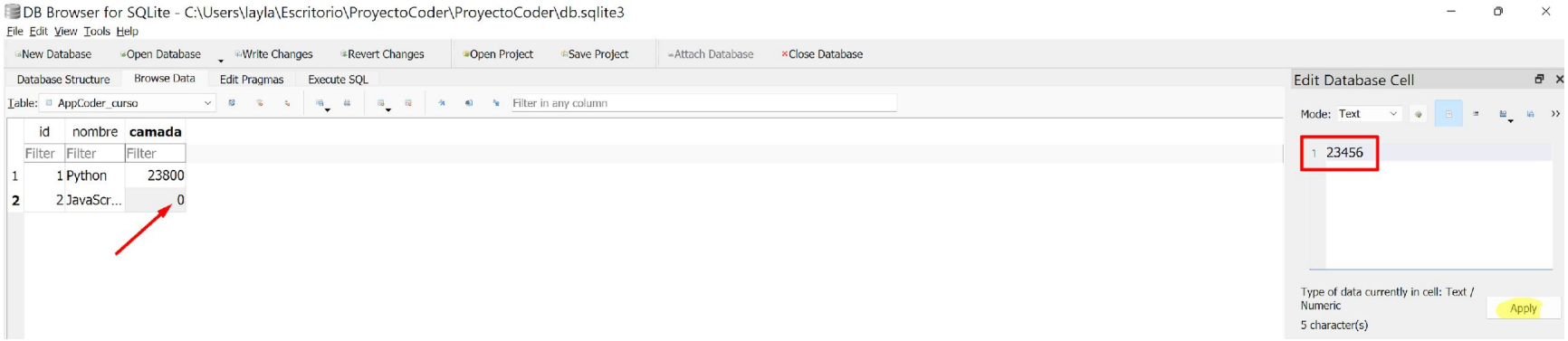
Otra manera de agregar información a mi BD

Agreguemos ahora la camada del curso de JavaScript. De la siguiente manera:



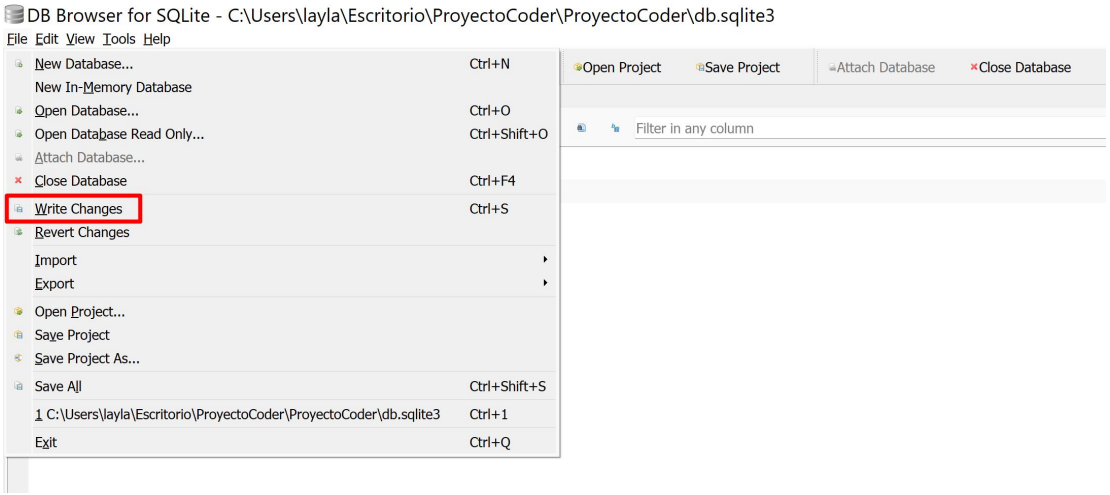
Otra manera de agregar información a mi BD

Hacemos el mismo procedimiento pero ahora con el nro de la Camada:



Otra manera de agregar información a mi BD

Hacemos persistir los cambios ☺️



#CoderAlert

Encontrarás en la **Guía de Actividades** del curso una actividad para aplicar todo lo aprendido hoy sobre DB a tu Proyecto. ¡Será fundamental al momento de realizar tu primera pre entrega en clase Nº21!

Actividad Nº 3

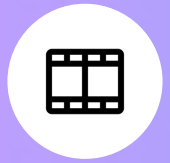
Nuestro primer MVT

Consigna

✓ Crear una web que permite ver los datos de algunos de tus familiares, guardados en un BD.

Aspectos a incluir

✓ Deberá tener un template, una vista y un modelo (como mínimo, pueden usar más)
✓ La clase del modelo, deberá guardar mínimo un número, una cadena y una fecha (puede guardar más cosas)
✓ Se deberán crear como mínimo 3 familiares
✓ Los familiares se deben ver desde la web.



¿Quieres saber más?
Te dejamos material ampliado de la clase

CODERHOUSE

MATERIAL AMPLIADO

Recursos multimedia

- ✓ Agregar información a la BD desde consola y desde DB Browser | Nicolás Pérez

Disponible en nuestro repositorio.

CODERHOUSE

¿Preguntas?

Opina y valora esta clase

Muchas gracias.

#DemocratizandoLaEducación

Resumen
de la clase hoy

- ✓ Aprendimos a usar Python en Html
- ✓ Definimos y creamos nuestro primer Modelo
- ✓ Diferenciamos entre proyecto y app.