

DJANGO ¿PYTHON?

*Ainhoa Gómez Toro
Carmen Tamayo Doña*

django



01

¿QUÉ ES?

Breve descripción de Django



¿Qué es?

Framework web extremadamente popular y completamente funcional, escrito en Python

- *Gratis*
- *De código abierto*
- *Acelerar el proceso de aplicaciones web*

02

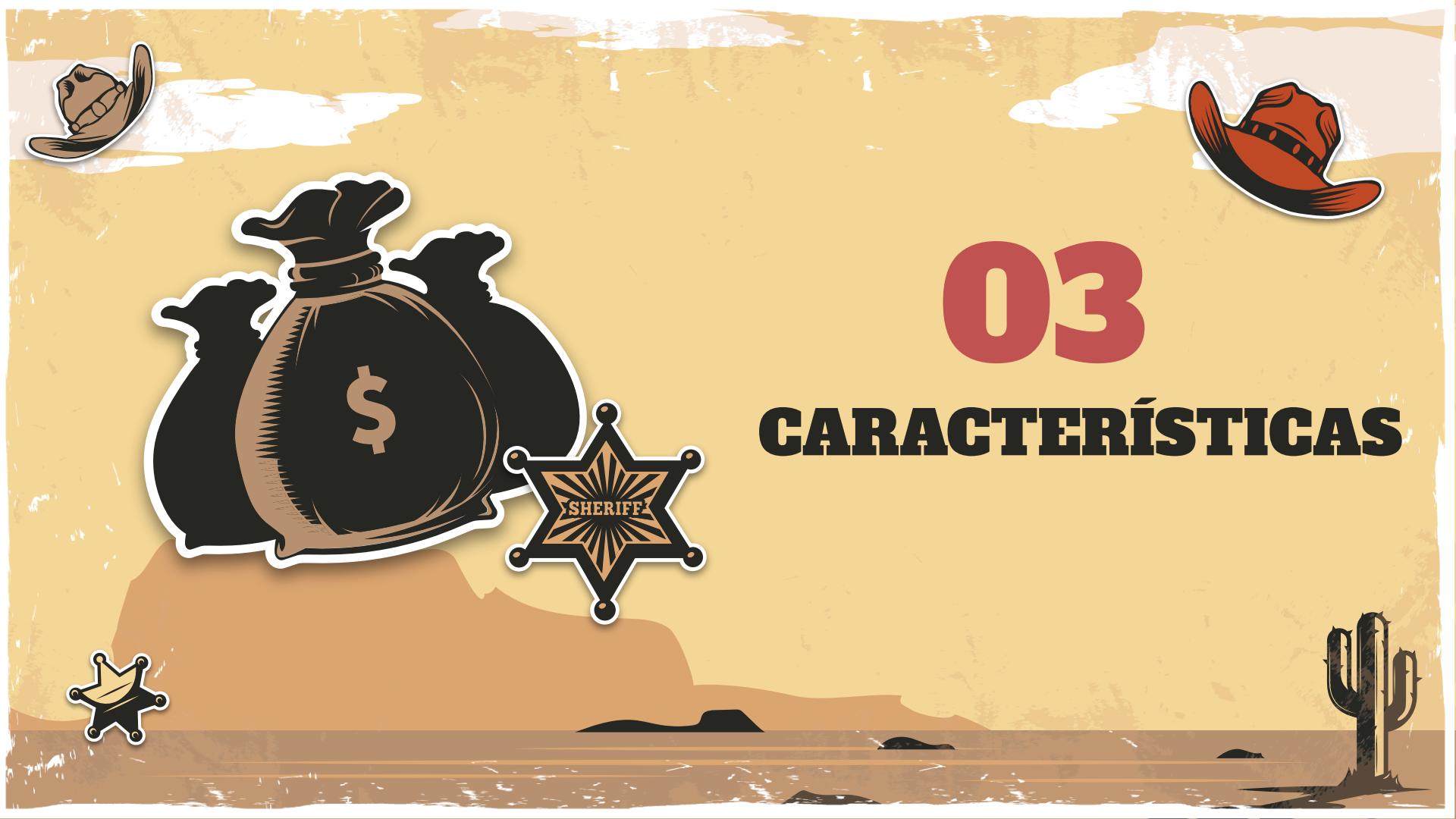
HISTORIA





- World Company de Lawrence, Kansas licencia BSD en julio de 2005.
- Nombrado en alusión al guitarrista de jazz gitano **Django Reinhardt**.





03

CARACTERÍSTICAS

CARACTERÍSTICAS



FLUJO DE DATOS



SISTEMA ORM

Permite interactuar con la base de datos mediante objetos.



MARCO DE PRUEBAS



PATRÓN MTV

Separa lógica, vista y modelo.



PROTECCIÓN

Inyecciones SQL,
Cross-Site Scripting,
Cross-Site Request
Forgery...





¡EMPECEMOS!



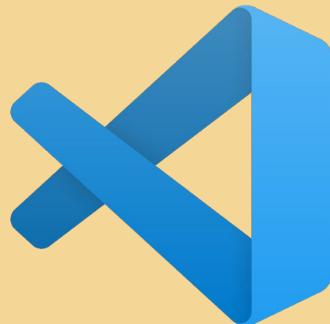
A continuación veremos cómo se crean y desarrollan proyectos en django.

04

CREACIÓN DE UN PROYECTO

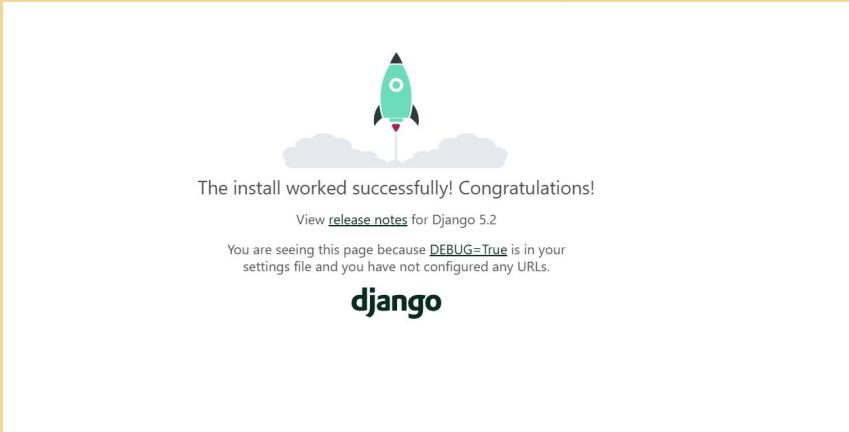


django-admin startproject nombre



Abrimos Visual Studio Code y desde la consola
escribimos el comando.

EJECUCIÓN



***python manage.py
runserver***

ESTRUCTURA DEL PROYECTO



CREACIÓN DE LA APP

Django trabaja con aplicaciones, es decir que tendremos que ir creando y estructurando nuestro proyectos en aplicaciones. Piensa en estas aplicaciones como componentes reutilizables al estilo de Angular.

Estas aplicaciones se crean con el siguiente comando:

***python manage.py
startapp miapp***

Una vez creadas debemos añadirla al settings para que el proyecto la reconozca:

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



```
holamundo
├── holamundo
│   ├── __pycache__
│   ├── __init__.py
│   ├── asgi.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
└── miApp
    ├── migrations
    │   └── __init__.py
    ├── __init__.py
    ├── admin.py
    ├── apps.py
    ├── models.py
    ├── tests.py
    ├── views.py
    └── manage.py
```

admin: Sirve para registrar todos los modelos que vayamos a usar.

apps: La configuración de nuestra aplicación.

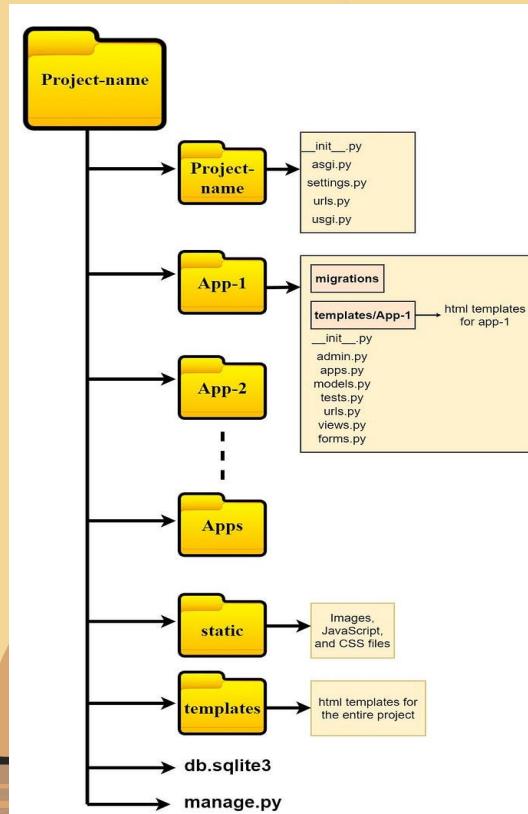
models: Almacena todos los modelos que vayamos a usar a modo de clases.

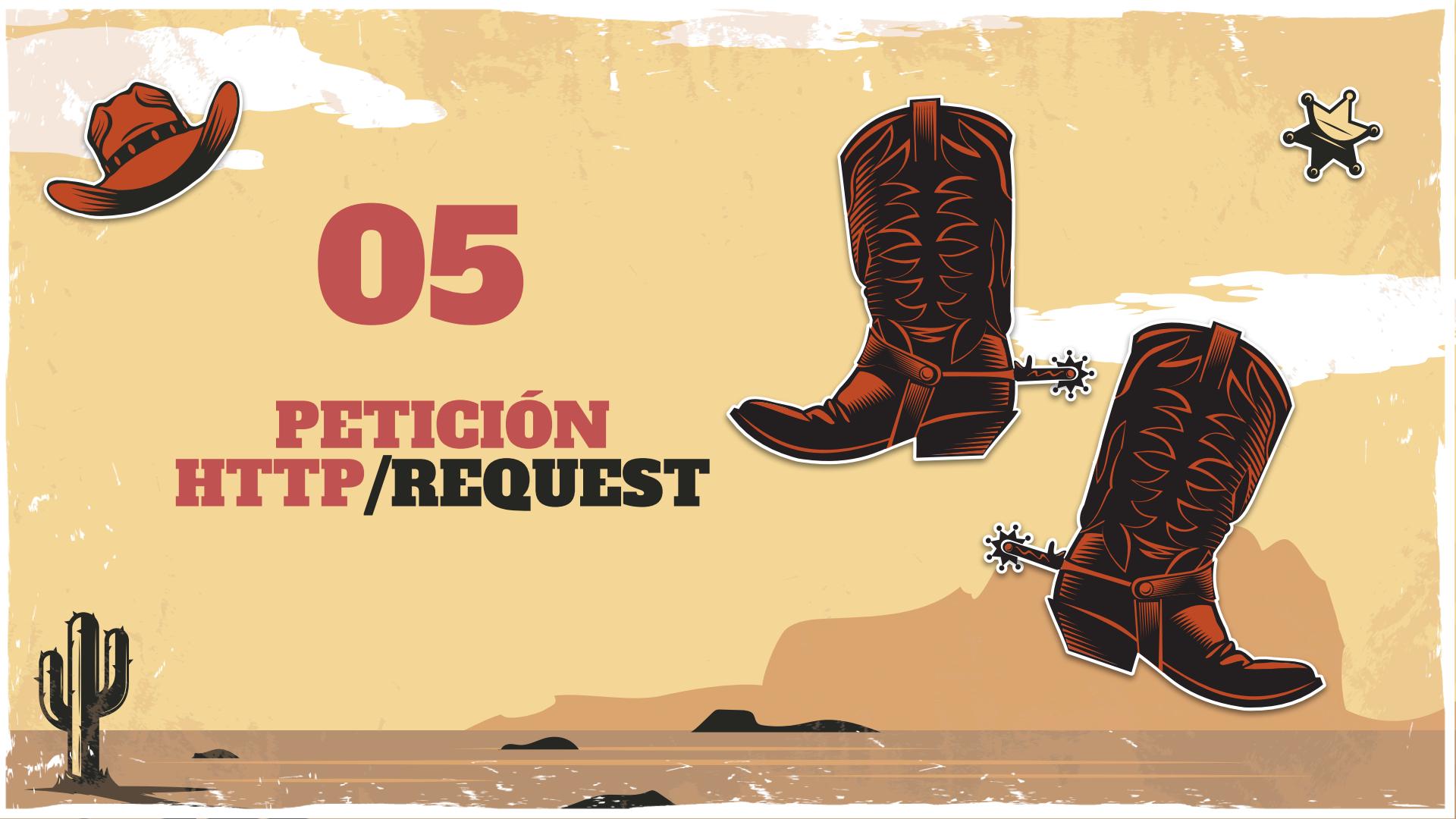
test: Para el desarrollo de pruebas.

views: Contiene todas las funciones de la aplicación , es decir, la parte lógica que une plantilla y modelo.

ESTRUCTURA DE CARPETAS

La estructura de carpetas un proyecto django
sería esta:





05

PETICIÓN HTTP/REQUEST

PETICIÓN HTTP/REQUEST

Es un protocolo que define un conjunto de métodos de petición para indicar la acción que se desea realizar para un cliente determinado.

GET

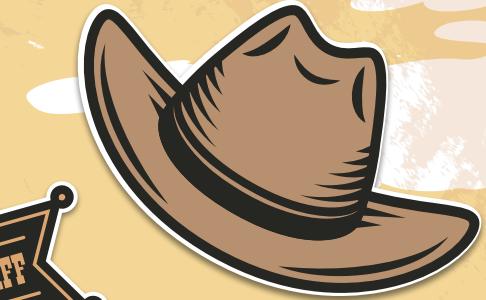
POST

PUT

DELETE

```
def hola_mundo(request):  
    return HttpResponse("Hola mundo!")
```

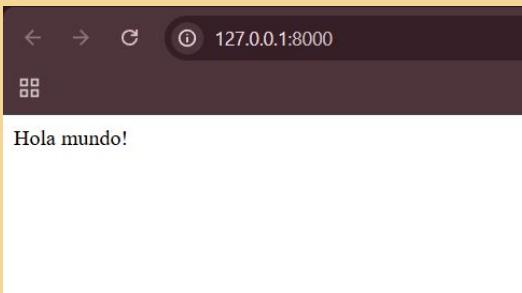
Recibe una petición (request) hacia el servidor al iniciar la página (la url) y devuelve una respuesta http de texto. Este request es un diccionario del que podemos obtener información.



HOLA MUNDO

```
return  
HttpResponse("Hola  
mundo!")
```

```
return  
render(request,"hola_mundo.  
html")
```



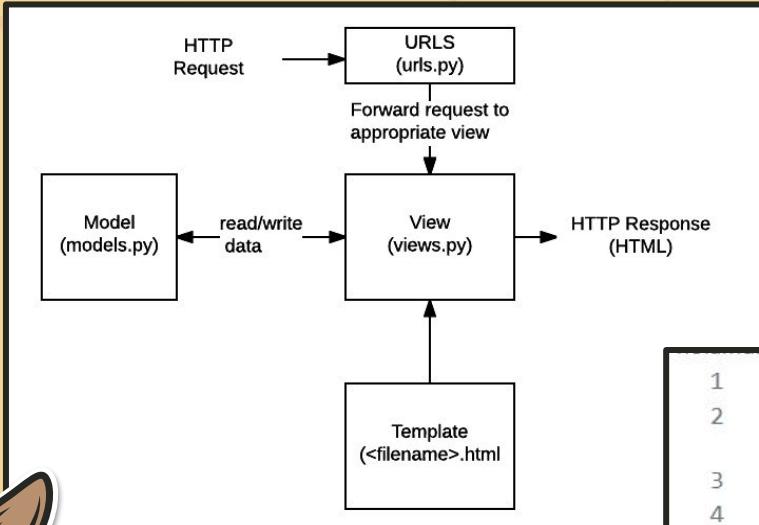


06

RENDER Y LA LÓGICA DE LA VISTA



RENDER



- Cargar un template

- Rellenar el template con datos con el diccionario que le pasas como contexto.

- Devuelve un `HttpResponse` que django envía al navegador.

```
1 from django.shortcuts import render
2
3 Windsurf: Refactor | Explain | Generate Docstring | X
4 def hola_mundo(request):
5     return render(request, 'hola.html')
```

```
6 def recorrer_datos(request):
7     alumnos=['Laura','Sara','Manuel','Olga','Esteban','Jose','Miriam']
8     return render(request,"lista_alumnos.html",{'alumnos':alumnos})
9
10
```



07

PLANTILLAS





El lenguaje de plantillas

Django Template Language



Variables

`{{ variable }}`



El motor de plantillas

Procesa el archivo de plantilla y crea la salida HTML



Filtros

`{{ variable|upper }}`



Etiquetas

`{% if %} ... {% endif %}`

FILTROS Y ETIQUETAS

upper	Convierte a mayúsculas
lower	Convierte a minúsculas
length	Devuelve la longitud
date:"d/m/Y"	Formatear fechas
truncatechars:20	Corta texto a X caracteres

Control de flujo	{% if %} ... {% endif %} {% if ... else %} {% if ... elif ... else %} {% for item in lista %} ... {% endfor %} {% empty %} {% comment %} ... {% endcomment %} #{ comentario #}
Archivos estáticos y URLs	{% load static %} {% static 'ruta/archivo.css' %} {% url 'nombre_de_vista' %}
Variables y contexto	{% with %} ... {% endwith %} {% firstof var1 var2 %} {% now "Y-m-d" %} {% debug %}



Incrustación

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Document</title>
7  </head>
8  <body>
9      {% include "barra.html" %}
10     <h1>Bienvenidos al blog!</h1>
11     {% include "pie.html" %}
12 </body>
13 </html>
```

Herencia

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <title>{% block title %} {% endblock %}</title>
5  </head>
6  <body>
7      <ul id="barra">
8          <li><a href="#">Home</a></li>
9          <li><a href="#">Servicios</a></li>
10         <li><a href="#">Contacto</a></li>
11         <li><a href="#">Foro</a></li>
12     </ul>
13     {% block content %} {% endblock %}
```



```
1  {% extends "basico.html" %}
2  {% block title %} Mi blog {% endblock %}
3  {% block content %}
4      <h1>Bienvenido a nuestra página!</h1>
5      <h1>Estamos a dia: {{dia}}</h1>
6  {% endblock %}
```

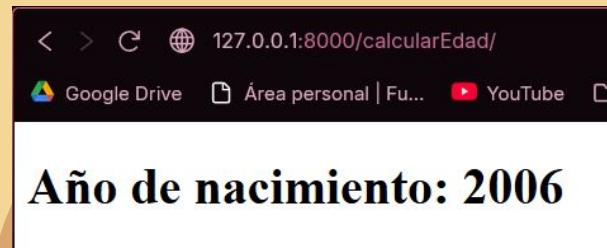
EJERCICIO 1: Comunicación entre Htmls

En este primer ejercicio aprenderemos a transmitir información entre htmls distintos a través de formulario.

Enunciado: Crear un formulario que te pida la edad y en otra página mostrar de qué año eres. Utilizar render, formularios, POST y urls.



A screenshot of a web browser window. The address bar shows the URL `127.0.0.1:8000/formulario`. The page content is titled "Formulario para pedir la edad". It contains a text input field with the value "19" and a button labeled "Enviar". The browser's navigation bar includes icons for back, forward, search, and refresh, along with links to Google Drive, Área personal, YouTube, and HTML.



A screenshot of a web browser window. The address bar shows the URL `127.0.0.1:8000/calcularEdad/`. The page content displays the text "Año de nacimiento: 2006". The browser's navigation bar includes icons for back, forward, search, and refresh, along with links to Google Drive, Área personal, YouTube, and HTML.

PROPUESTO 1: Suma de dos números

Enunciado: Desarrollar una página ahora que nos solicite dos números, los envíe mediante post y nos muestre en otra página el resultado de la suma.

Introduce número 1: Introduce número 2:

Resultados de la suma:

$$1 + 2 = 3$$



EJERCICIO 2: Recorrer datos



En este ejercicio aprenderemos a recorrer y transmitir listas.

Enunciado: Crea una lista de alumnos y recorrerla en una lista desordenada.



¡Listado de alumnos!

- Alumno Laura
- Alumno Sara
- Alumno Manuel
- Alumno Olga
- Alumno Esteban
- Alumno Jose
- Alumno Miriam

EJERCICIO 3: Control de flujo



En este ejercicio aprenderemos a evaluar los datos enviados por el contexto.

Enunciado: Modifica el primer ejercicio de forma que al enviar la edad diga si eres mayor o no de edad.



Formulario para pedir la edad

Edad: Enviar

Eres mayor de edad: 22



PROPUESTO 2: Mostrar calificaciones

Enunciado: Html que pide una calificación entre 1 y 10 y dependiendo de esta escribe excelente, muy bien, aprobado o suspenso.

Introduce una calificación:

Introduce una calificación:

Resultados:

Muy bien!

EJERCICIO 4 : Ver si existe algo en una lista

En este ejercicio aprenderemos a evaluar los datos enviados por el contexto.

Enunciado: Comprobamos si existe o no una asignatura en una lista.

Comprueba si existe la asignatura

Nombre: Comprobar

La asignatura Lengua se encuentra en la lista

EJERCICIO 5: Insertar un alumno en una lista

Enunciado: Crearemos una lista de alumnos, después un formulario para agregar un nuevo alumno, este se añadirá a la lista y se volverá a mostrar con el formulario.

Personas de la lista:

- Laura
- Sara
- Manuel
- Olga
- Esteban
- Jose
- Miriam
- Ainhoa

Introduce un nombre:

API FORMS



Django tiene un sistema de formularios que permite manejar los datos de forma segura y validarlos de manera automática. Además, con forms puedes generar etiquetas HTML automáticas:

A screenshot of a web browser displaying a "New post" form from a "Django Girls Blog". The form has two fields: "Title" and "Text", both of which have validation errors indicating they are required fields.

.as_p

.as_table

.as_ul

Creamos un archivo forms.py con las clases de formulario que vayamos a utilizar y lo inyectamos dentro del html.



EJERCICIO 6: Crear con API forms un formulario de contacto

Enunciado: Crearemos con API forms un formulario de contacto que incluya los campos: nombre, email y mensaje. Nos llevará a otro HTML que nos dará las gracias y el nombre del cliente.

Registrar Nuevo Cliente

Nombre:

Email:



Incluye un signo "@" en la dirección de correo electrónico. La dirección "ail.com" no incluye el signo "@".

PROPUESTO 3: Lista de tareas

Enunciado: Realizar una página que permita registrar tareas a una lista y que se muestre esta lista en una tabla con herencia de plantillas, API forms y utilizando el contexto de las vistas para enviar la información. Las tareas dispondrán de título y si están completadas o no.

- [Home](#)
- [Servicios](#)
- [Contacto](#)
- [Foro](#)

Título:

Completado:

© 2025 Mi blog

- [Home](#)
- [Servicios](#)
- [Contacto](#)
- [Foro](#)

Título:

Completado:

Tarea	Completado
Escribir	Completada
Programar	Completada
Dormir	Incompleta

© 2025 Mi blog



08

MIGRACIONES



Creación del modelo

Como hemos mencionado, django utiliza un ORM (Object-Relational Mapping) para manejar bases de datos, es decir que cada tabla de la base de datos corresponde a un modelo o clase de nuestra aplicación, cada registro es un objeto de la clase y cada campo es un atributo de ese objeto.



Para comenzar a crear modelos, primero tendremos que crear la aplicación y desde la aplicación ir al archivo models e importar: `from django.db import models`

Django.db es el módulo que utiliza django para las bases de datos ORM y models es el submódulo que contiene todos los tipos de campos.

Los tipos de campos que existen en el submódulo models son:

CharField(max_length)	Sirve para textos cortos y es obligatorio indicar la longitud máxima.
TextField()	Texto largo sin límite de caracteres.
EmailField()	Correo electrónico.
SlugField()	Texto corto válido para urls.
URLField()	Url válida.
FilePathField()	Ruta a un archivo del sistema.

IntegerField()	Números enteros.
PositiveIntegerField()	Números enteros positivos.
SmallIntegerField()	Entero más pequeño.
PositiveSmallIntegerField()	Entero positivo pequeño.
BigIntegerField()	Entero grande.
FloatField()	Números con decimales.

DecimalField(max_digits=N, decimal_places=M)	Número decimal fijo.
BooleanField()	Booleanos.
DateField()	Campo de fecha.
TimeField()	Campo de hora.
DateTimeField()	Campo de fecha y hora.

Otra peculiaridad de django es que ya genera una clave primaria autoincremental por defecto. Las restricciones o campos especiales son los siguientes:

Restricción	Ejemplo
ForeignKey(Model)	autor=models.ForeignKey(Usuario,on_delete=models.CASCADE)
primary_key=True	nombre=models.CharField(max_length=100,primary_key=True)
unique=True	nombre_usuario=models.CharField(max_length=100,unique=True)

Cómo funcionan las claves foráneas



Las claves foráneas son el objeto correspondiente al que hace referencia, por ejemplo, para una base de datos de un foro, donde tenemos por un lado Comentario y Usuario, en la tabla Comentario, el atributo autor del comentario apuntaría directamente al objeto de Usuario que lo ha escrito. Gracias a esto podemos acceder a los atributos de ese objeto para hacer por ejemplo : **comentario.autor.nombre**.



Además supone otra ventaja, ya que establece automáticamente para cada clave foránea un acceso de relación inversa, es decir puedo acceder a los comentarios desde la clase usuario también y sacar todos los comentarios de ese usuario con: `usuario.comentario_set.all()`.

Las relaciones inversas se llaman con un `_set`.



Motores de base de datos

Django soporta múltiples motores de bases de datos como SQLite, PostgreSQL, MySQL/MariaDB... La configuración del motor que vamos a utilizar se encuentra en settings:

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.sqlite3',  
        'NAME': BASE_DIR / 'db.sqlite3',  
    },  
}
```

Por defecto, la base de datos será en Sqlite y se almacenará todo en nuestro archivo d.sqlite3.

Migraciones



Una migración es el proceso mediante el cual se trasladan datos de una o más bases de datos de origen a una o más bases de datos de destino mediante un servicio de migración de bases de datos. Cada vez que migras en django, estás guardando los modelos actualizados en la base de datos, transformando los modelos orientados a objetos a lenguaje sql. Los comandos son los siguientes:

python manage.py makemigrations: Django crea un archivo dentro de la carpeta migrations con las instrucciones para crear o modificar las tablas.

python manage.py makemigrations: nombre_app: hace las migraciones de una aplicación en específico.

python manage.py showmigrations: muestra todas las migraciones y si están aplicadas.

python manage.py sqlmigrate nombre_app numero_migracion: muestra el sql que se ejecutará en esta migración.

python manage.py migrate: lee los archivos de migración y crea las tablas en el sistema.

python manage.py flush: limpia la base de datos.

Ejemplo clase Persona

Para entender cómo funcionan los modelos crearemos una aplicación nueva y dentro de models una clase Persona:

```
migraciones / mapp / models.py
from django.db import models

Windsurf: Refactor | Explain
class Persona(models.Model):
    nombre = models.CharField(max_length=100)
    edad = models.IntegerField()

Windsurf: Refactor | Explain | Generate Docstring | X
def __str__(self):
    return self.nombre
```

Para migrar esta clase que acabamos de crear a la base de datos escribiremos el siguiente comando:

***python manage.py
makemigrations***

python manage.py migrate



09

SHELL Y COMANDOS BDD





```
python manage.py shell
exit()
```



El shell es la consola interactiva de django. Para acceder a una tabla desde el shell, importamos la clase que contiene los modelos y escribimos: NombreTabla.objects



exact: igualdad exacta.

iexact: igualdad sin mayúsculas o minúsculas.

contains: contiene texto.

icontains: contiene texto sin tomar en cuenta mayúsculas o minúsculas.

in: dentro de una lista.

gt: mayor que.

gte: mayor o igual que.

lt: menor que.

lte: menor o igual que.

startswith: empieza por.

istartswith: empieza por (insensible a may/min).

endswith: termina por.

iendswith: termina por (insensible a may/min).

range: establece un rango entre números.

year/month/day/week_day: filtra por fecha.

hour/minute/second: filtra por hora.

isnull: comprueba si es nulo

Agregar datos a la clase Persona

Abriremos el shell con el comando: python manage.py shell e insertaremos nuevos registros a la tabla Persona de nuestra base de datos sqlite.

```
>>> from miapp.models import Persona  
>>> persona1 = Persona(nombre='Pedro',edad=23)  
>>> persona1.save()
```

```
>>> from miapp.models import Persona  
>>> per1 = Persona.objects.create(nombre='Ana', edad=28)
```

Editar:

```
per1 = Persona.objects.get(id=1)  
per1.edad = 15  
per1.save()
```

	<u>id</u>	nombre	edad
	Fil...	Filtro	Filtro
1	1	Pedro	23
2	2	Ana	15

Buscar datos en el Shell

Obtener todos:

```
>>> personas=Persona.objects.all()  
[<Persona: Pedro>, <Persona: Pedro>, <Persona: Pedro>, <Persona: Pedro>, <Persona: Pedro>, <Persona: Pedro>]
```

Consultas varias:

```
>>> Persona.objects.filter(nombre='Pedro')  
<QuerySet [<Persona: Pedro>]>  
>>> Persona.objects.filter(edad__gte = 20)  
<QuerySet [<Persona: Pedro>]>  
>>> Persona.objects.filter(edad__lte = 20)  
<QuerySet []>  
>>> Persona.objects.filter(edad__range = [20,30])  
<QuerySet [<Persona: Pedro>]>  
>>> Persona.objects.filter(edad=23,id=1)  
<QuerySet [<Persona: Pedro>]>
```

Eliminar registro:

```
>>> per2 = Persona.objects.get(id=2)  
>>> per2.delete()  
(1, {'miapp.Persona': 1})
```

```
>>> Persona.objects.count()  
6  
>>> Persona.objects.first()  
<Persona: Pedro>
```

```
>>> personas.query.__str__()  
'SELECT "miapp_persona"."id", "miapp_persona"."nombre", "miapp_persona"."edad" FROM "miapp_persona"'
```

PROPUESTO 4: Ejercicio base de datos en shell

Enunciado: Tendremos un modelo Cliente que tendrá los campos nombre, apellido, email, edad, comentario. El objetivo es hacer los modelos y las migraciones, insertar al menos 10 datos y hacer 2 consultas: Que tengan más de 30 años ordenados descendente y los clientes que su apellido contenga 'a' y su email no contenga la 'x'.

Se enseñará con un documento con capturas.

Métodos que retornan
nuevos conjuntos de
consultas

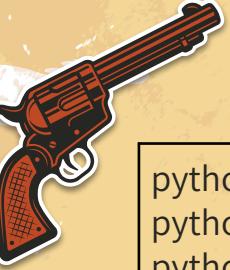
filter(): filtrar y obtener varios registros.
exclude(): excluye los registros que cumplan la condición.
annotate(): añade campos calculados.
order_by(): muestra los campos ordenados.
distinct(): registros distintos.
reverse(): invierte el orden.
dates(): devuelve las fechas de un campo tipo date.
datetimes(): devuelve las fechas de un campo tipo datetime.
none(): devuelve el queryset vacío.
all(): devuelve todos los registros.



10

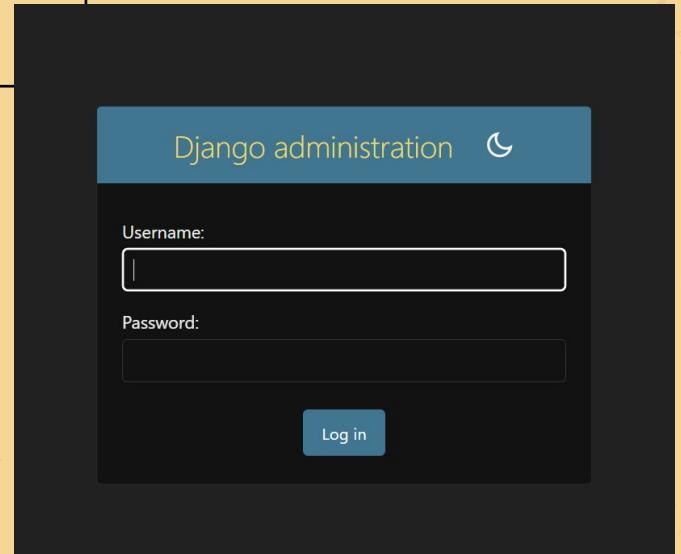
**PANEL DE
ADMINISTRACIÓN**





```
python manage.py createsuperuser
python manage.py changepassword nombre_usuario
python manage.py createsuperuser --username=admin
--email=admin@example.com
```

```
miApp > admin.py
1   from django.contrib import admin
2   from miApp.models import Persona,Cliente
3
4   admin.site.register(Persona)
5   admin.site.register(Cliente)
6
```



EJERCICIO 7: formulario que permita añadir libros a una biblioteca

Enunciado: Crearemos un formulario que permita añadir libros a una biblioteca indicando el autor mediante un SELECT. Se hará con base de datos

Ingrses el título del libro: Selecciona el autor:

<input type="button" value="Gabriel García Márquez"/>
J.K. Rowling
George Orwell
Isabel Allende
Mario Vargas Llosa

Titulo Autor
Ainhoa Gabriel García Márquez
Harry Potter J.K. Rowling



PRÁCTICA: BLOG DE RESEÑA DE PELÍCULAS



El objetivo de esta práctica es crear nuestro propio blog.
Este se basará en un blog de reseñas de películas.

Incluirá:



- Inicio de sesión/Registro
- Navbar con secciones como: inicio, nuestras reseñas, nueva reseña y películas mejor valoradas.



Las películas ya estarán creadas en el blog (mínimo diez), no se podrán crear ni modificar ni borrar. Estás tendrán su título, fecha de lanzamiento y director.

Los modelos anteriores son orientativos/mínimos, un usuario podría tener también también email, apellidos, etc. Se valorará más la funcionalidad de la aplicación.





¡GRACIAS!

¿Alguna pregunta?

¿Os ha gustado Django?

