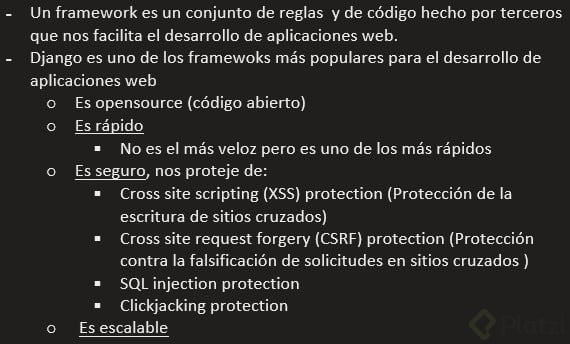
**Bienvenido al desarrollo backend con Python**

# Curso Básico de Django

# ¿Qué es Django?

Es un framework:



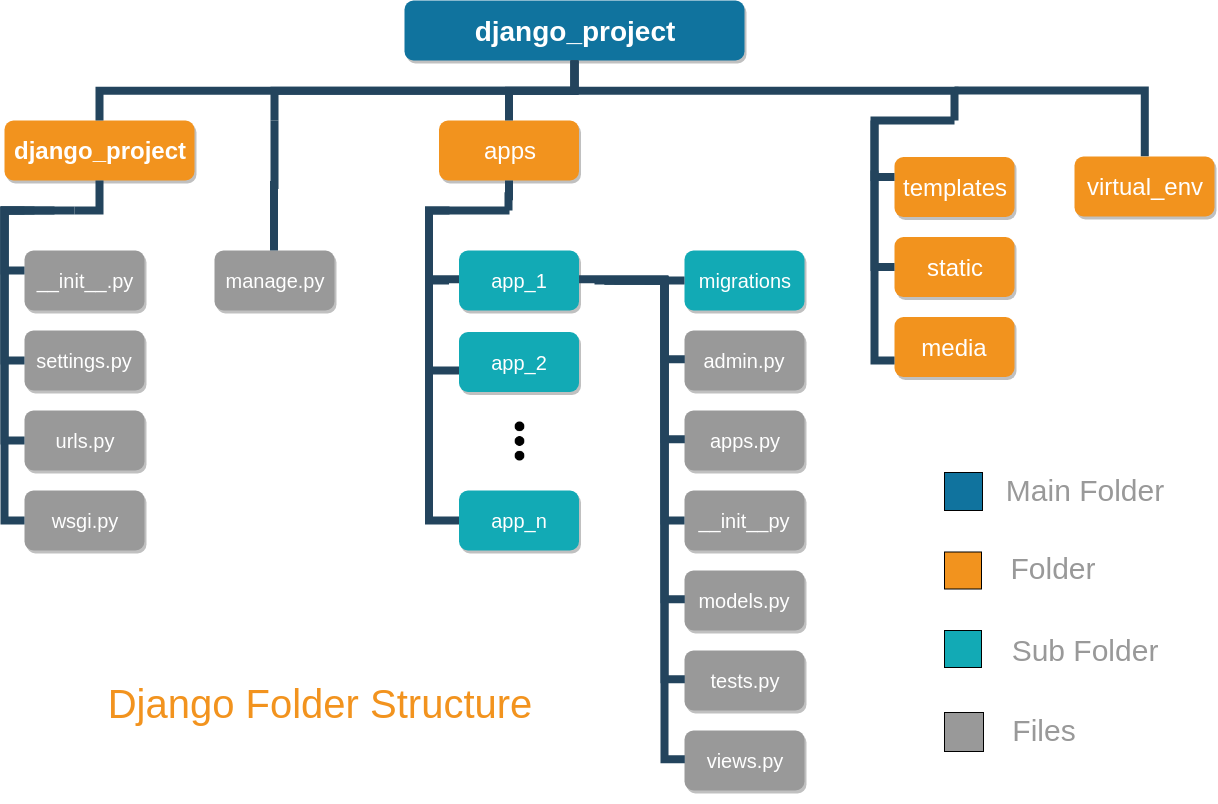
Django es uno de los framework gratis y open source más populares para crear aplicaciones web. Es muy veloz, seguro y escalable.  
Es el segundo framework de desarrollo web más usado, siendo Flask el más usado solo por un 1% y el tercero siendo FastAPI.  
Algunos proyectos que usan Django son:

* Instagram
* Pinterest
* National Geographic
* Platzi

# Instalación de Django

1. Debemos crear la carpeta donde trabajaremos:mkdir django\_premiosplatzi
2. Crear el entorno virtual con venv o pyenv.
3. Instalar django: pip install django
4. Iniciar el repositorio de git: git init
5. Iniciar el proyecto de django: django-admin startproject premiosplatziapp
6. Para evitar llevar al repositorio remoto archivos innecesarios es importante crear el archivo: .gitignore (Donde se suele agregar la carpeta venv)

# Explorando los archivos que creó Django



# El servidor de desarrollo

Siempre que crees un proyecto web (Siendo uno mismo el creador) siempre va “vivir” en dos lugares:

* En local: Entorno de trabajo que creamos y editamos para desarrollar.
* En producción, es el servidor (Jamas tocamos el código directamente).  
  Django nos permite utilizar un servidor (local) de desarrollo.  
  Para ver el servidor local debemos ejecutar el comando: py manage.py runserver

Detalle de mensaje:

* **Watching for file … :** A cada cambio que hagas en los archivo, Django lo notará y lo reflejará en el servidor.
* **System check identified no issues … :** No hay problemas y se silenciaron 0.
* **You have 18 unapplied migration(s) … :** No se ha creado una base de datos efectiva.
* **Date, Version Django**
* **Using settings ‘premiosplatziapp.settings’ :** Tomar el archivo settings-py, ver la variables de configuración y aplicarlas para tenerlas disponibles al crear el código.
* **Starting development … :** Servidor desplegado (iniciado) de manera local.

# Nuestro primer proyecto: Premios Platzi App

Dentro de Django hay 2 cosas importantes para diferenciar:

* Proyecto : Un proyecto es una colección de configuraciones y aplicaciones para un sitio web en particular. Un proyecto puede contener varias aplicaciones. *Una aplicación puede estar en varios proyectos.*
* Apps : Una aplicación es una aplicación web que hace algo, por ejemplo, un sistema de blogs, una base de datos de registros públicos o una pequeña aplicación de encuestas.

Cada aplicación que escribe en Django consta de un paquete de Python que sigue una determinada convención. Django viene con una utilidad que genera automáticamente la estructura básica de directorios de una aplicación, por lo que puede concentrarse en escribir código en lugar de crear directorios.

# Nuestro primer proyecto: Premios Platzi App 2

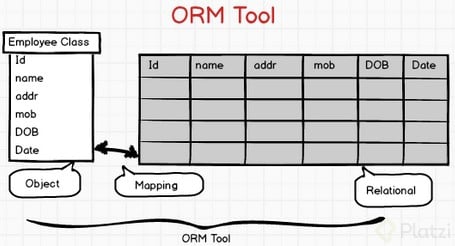
# Ajustando el archivo settings.py

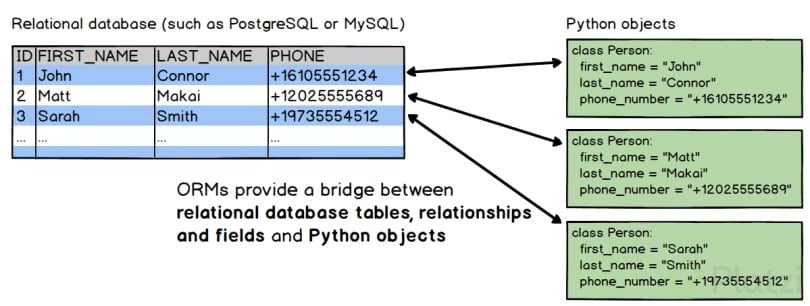
# ¿Qué es ORM? ¿Qué es un modelo?

Un **ORM (Object-Relational Mapping)** es una técnica que nos permite crear una **Base de Datos Orientada a Objectos (virtual)**, que opera sobre la **Base de Datos Relacional (real)**.

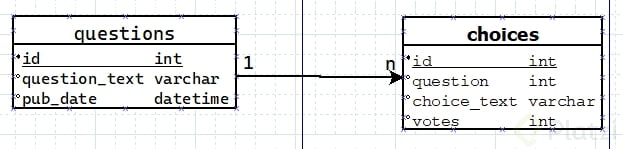
Utilizando un ORM podemos operar sobre la base de datos aprovechando las características propias de la orientación a objetos, como **herencia y polimorfismo**.

También podemos acceder a los atributos de una Entidad de la misma forma que accedemos a los atributos de una Clase, realizar operaciones para **obtener, crear, modificar y eliminar datos, todo desde el código de programación sin tener que escribir SQL**. Esto además nos permite escribir el código una sola vez y garantizarnos que va a seguir funcionando **incluso si en el futuro se cambia el motor de Base de Datos** (por ejemplo, de MySQL a Microsoft SQL Server).





# Creando un diagrama entidad-relación para nuestro proyecto



# Creando los modelos Question y Choice

py manage-py makemigrations polls Crea un archivo “migration/001\_initial-py” en el que django automaticamente describe toda la creación de las tablas en las BD, uso del concepto **ORM**

py manage-py migrate Tomar el archivo creado y ejecutarlo en la BD. “Applying polls.001\_initial”

# La consola interactiva de Django

Comando para acceder a la consola interactiva de Django: python3 manage.py shell

# Importacion de los modelos

**from** polls.models **import** Question, Choice

# Llamado de todos los registros de un modelo

Question.objects.all()

# Creacion de un nuevo registro

q = Question(question\_text="¿Cual es el mejor curso de Platzi?", pub\_date=timezone.now())

# Guardado del nuevo registro

q.save()

# El método \_\_str\_\_

# Filtrando los objetos creados desde la consola interactiva

# El método filter

Para resaltar:  
el método filter es para traer una lista de objetos que cumplan con los parámetros de consulta que se proporcionan. Si no hay objetos, igual te entrega una lista vacía.

A diferencia, el método get, es para obtener un sólo objeto y podrás modificar sus atributos porque no estás trabajando sobre una lista sino sobre la instancia de un modelo. Si no hay algun objeto que cumpla con los parámetros de consulta, te saltará un error, es por esto que siempre se meneja dentro de un try-except.

Tip: Si sólo necesitas confirmar que existe un objeto y estás seguro que sólo habrá no más de uno y no necesitas modificar sus atributos, puedes usar el método filter() y el método exists(). Ejemplo:  
Question.objects.filter(pk=1).exists()  
Esto te dará un True o False.  
Pero si necesitas modificar algún parámetro, obligatoriamente debes usar el get con un try-except.

# Accediendo al conjunto de respuestas

Una herramienta bastante útil es “django-extensions”, para el manejo del shell:

Instalar:

pip install django-extensions

Luego en el archivo “settings” del proyecto incluir:

INSTALLED\_APPS = (

...

'django\_extensions',

...

)

y luego ejecutar el comando:

python3 manage.py shell\_plus

Ahora la consola permite autocompletar y también importa los modelos de forma automática.

# El administrador de Django

Para registrar uno o mas modelos puedes pasarlos dentro de una lista:

**from** django.contrib **import** **admin**

**from** .models **import** Question, Choice

**admin**.site.register([Question, Choice])

Cosas a tener en cuenta aquí:

1. El formulario se genera automáticamente a partir del **Questionmodelo**.
2. Los diferentes tipos de campo de modelo ( **DateTimeField, CharField**) corresponden al widget de entrada HTML adecuado. Cada tipo de campo sabe cómo mostrarse en el administrador de Django.
3. Cada uno **DateTimeField** obtiene atajos de JavaScript gratuitos. Las fechas obtienen un acceso directo “Hoy” y una ventana emergente de calendario, y las horas obtienen un acceso directo “Ahora” y una ventana emergente conveniente que enumera las horas ingresadas comúnmente.

# ¿Qué son las views o vistas?

**Diferencia entre MVC y MTV 😮**

Muy similar a MVC (Modelo, vista, controlador) donde:  
.  
Modelo - Hace referencia a todo lo que tiene que ver con bases de datos.

Vista - Con la parte visual.

Controlador - Con toda la parte lógica.

.  
En Django siendo MTV 😃

Modelo - Hace referencia a todo lo que tiene que ver con bases de datos (En este caso Django hace demasiada alusión a su nombre donde las bases de datos quedan implícitas y manejamos todo con el ORM)

Template - En este caso no debemos confundir el View del MTV con el del MVC puesto en que en Django no hace referencia a lo visual, **template** si hace alusión con la parte visual de las Web Apps con Django.

View - Todo lo relacionado con la lógica es aquí donde entra el tema de las vistas genéricas y demás (Que si solo nos especializamos en el back con Django es lo que mas utilizaremos)

# Creando vistas para la aplicación

1- Se manda un request por la barra de URL en el navegador.  
2 - Se recibe en el archivo **urls py** y según la URL envidada se determina cual es el view correspondiente en el archivo **views py**.  
3 - Ser recibe el request en **views py** en la función o clase correspondiente y se retorna una respuesta.

# Templates de Django

El render es una función importada de Django. Necesita tres parámetros

* El *request*
* La ubicación de la template correlacionada. En la ruta se omite la carpeta */template/* ya que Django junta todas las carpetas templates de todas las aplicaciones de nuestro proyecto y las hace una sola
* Un objeto con la o las variables que le estaremos pasando a nuestra **template**

# Creando el template del home

# Elevando el error 404

Este es un tema muy interesante acerca de los shortcut functions de Django, en este documento cortico encontrarán toda la información valiosa para entender el funcionamiento más a profundidad, espero sea de ayuda.

Allí se tratan 4 puntos importantes:

* render()
* redirect()
* get\_object\_or\_404()
* get\_list\_or\_404()

# Utilizando la etiqueta url para evitar el hard coding

# Formularios: lo básico

**pro tip**: SIEMPRE usa {% csrf\_token %} en los formularios POST para evitar ataques de hacking.

# Creando la vista vote

# Creando la vista results

**Creando la vista results**

Hasta ahora tenemos la interfaz de voto y el conteo de votos programado, ahora vamos a programar la vista resultados para ver la cantidad de votos por opción cuando votamos, para esto hacemos cambios en los archivos views.py y results.html, este último lo vamos a crear, recordemos que todo lo trabajamos dentro de la carpeta de la aplicación.  
**En el archivo views.py**

**def** **results**(request, question\_id):

# siempre se llega después de ejecutar vote

question= get\_object\_or\_404(Question, pk=question\_id)

**return** render(request, "polls/results.html", {

"question": question

})

Recordemos que al llamar los templates debemos enviar los argumentos si son necesarios, en este caso “question”

**En el archivo results.html**

<h1>{{ question.question\_text }}</h1>

<ul>

{% for choice in question.choice\_set.all %}

<li>

{{ choice.choice\_text }} -- {{ choice.votes }}

vote{{ choice.votes|pluralize }}

</li>

{% endfor %}

</ul>

<a href="{% url 'polls:detail' question.id %}">

¿Te gustaría votar de nuevo?

</a>

El indentado de la programación cambia un poco, considero que así se ve más organizado

# Generic Views

# Implementando generic views en la aplicación

# Curso de Django Intermedio: Testing, Static Files, Django Admin

# TESTING

# ¿Qué son los tests?

Los test son funciones que verifican que tu código opere correctamente.

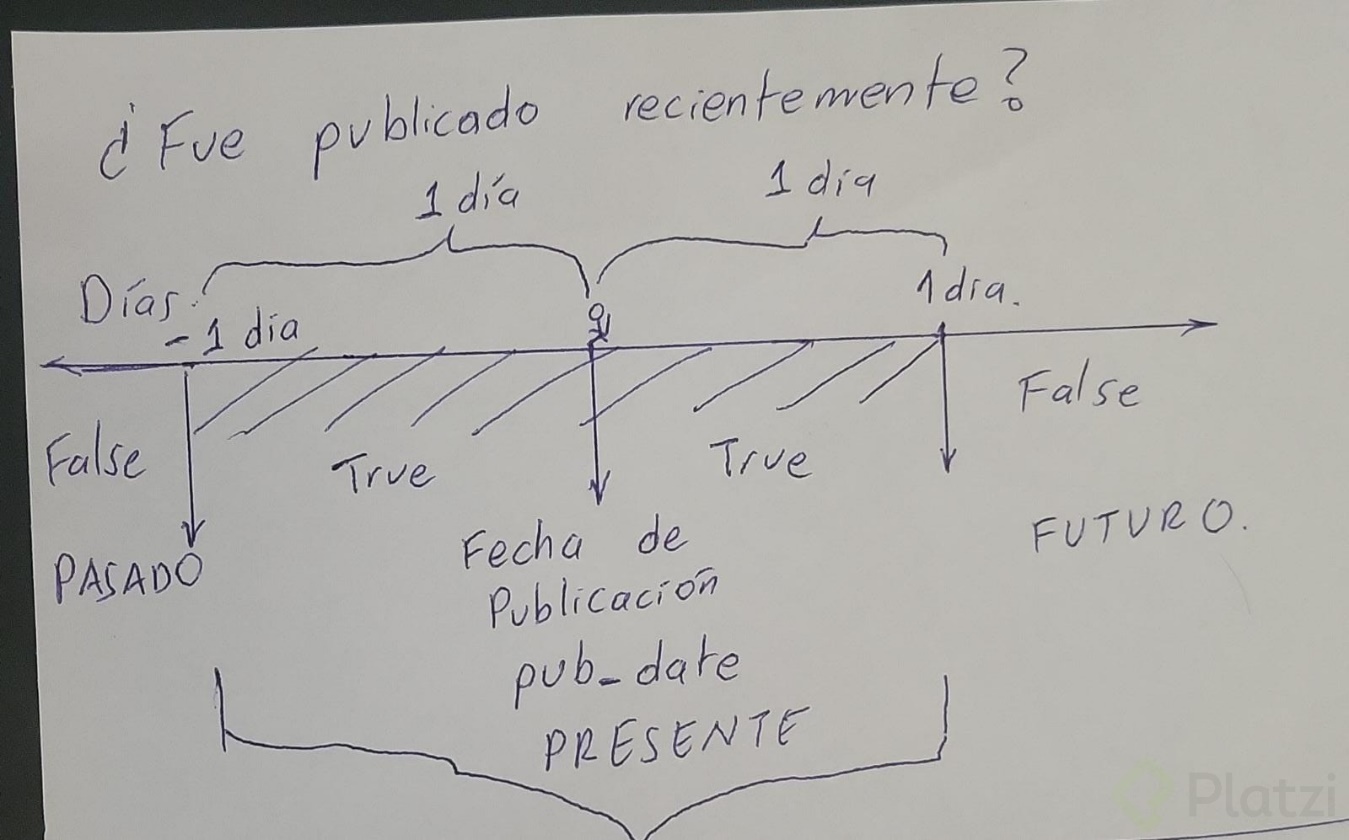
TDD o Test-Driven Development (desarrollo dirigido por tests) es una práctica de programación que consiste en escribir primero las pruebas (generalmente unitarias), después escribir el código fuente que pase la prueba satisfactoriamente y, por último, refactorizar el código escrito.

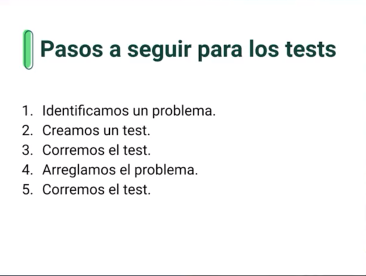
# Escribiendo nuestro primer test

El comando python manage.py test por defecto va a hacer:

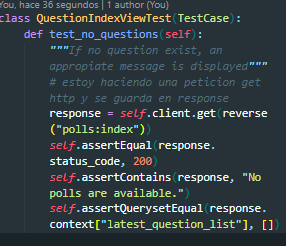
1. Busca todos los archivos cuyo nombre tengan la forma *test\*.py*
2. En esos archivos, busca todos los TestCase, que son las clases que heredan de *django.test.TestCase*
3. De esos TestCase, busca todos los métodos cuyo nombre comience con *test*

# Solucionando el error encontrado





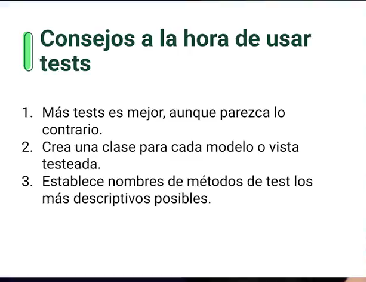
# Testing de Views



# Creando más tests para IndexView

# Ajustando detalles en los tests para IndexView

# Creando tests para DetailView



# Agregando estilos a nuestro proyecto

# Añadiendo una imagen de fondo

