1. **¿Qué es Django y por qué lo usaríamos?**
   https://aws.amazon.com/es/what-is/django/#:~:text=Django%20es%20un%20software%20que,y%20la%20administraci%C3%B3n%20de%20cookies.
   Django es un conjunto de herramientas, bibliotecas, componentes que facilitan el desarrollo de aplicaciones web.
   Se escribió en el lenguaje Python, es uno de sus muchos marcos web.
   Está bien organizado y es fácil de instalar y aprender. Se creó para implementar rápidamente cualquier arquitectura web en el código.
   Tiene tareas ya integradas que se pueden usar sin tener que programarlas desde cero como:
   - **autenticación de usuarios** (inicio de sesión, registro..)
   - **administración de contenido** (parte del sitio web donde como administrador, podés agregar o eliminar información fácilmente, sin tener que escribir código)
   - **mapas del sitio** (archivo que lista todas las páginas importantes del sitio web)
   - **fuentes RSS** (forma de compartir contenido actualizado automáticamente)

2. **¿Qué es el patrón MTV (Model-Template-View) en Django? (simplificado de MVC). Compará MTV con MVC.**
   https://www.freecodecamp.org/news/how-django-mvt-architecture-works/#:~:text=b%C3%A1sicos%20de%20Python.-,%C2%BFQu%C3%A9%20es%20la%20arquitectura%20MVT?,en%20la%20interfaz%20de%20usuario.
   https://urianviera.com/django/arquitectura-mvc-en-django
   El patrón MTV es una arquitectura que organiza el código de una aplicación web en tres componentes principales:
   1) MODELO: gestiona los datos e interactúa con la base de datos
      Los modelos gestionan la estructura y la interacción de los datos dentro de una aplicación Django, que terminan siendo la base de la aplicación por su papel fundamental.
      Siguen una forma particular de declaración. La estructura básica es:
      ```
      class <model_name>(models.Model):
          <field_name> = models.<field_type>(<optional_field_characteristics>)
      ```
      class: palabra clave para definir un modelo
      model_name: nombre del modelo
      models.Model: clase base de la que hereda la clase modelo
      field_name: nombre de la columna de la base de datos
      field_type: tipo de datos que contiene el campo (charField, boolean...)
      optional_field_characteristics: para definir con más detalle cómo se comporta el campo.

   2) VISTA: actúa como intermediario, maneja la lógica y administra el flujo de datos
      Las vistas se encargan de procesar las solicitudes de los usuarios y devolver las respuestas. Actúan como puente entre el Modelo y Plantilla, recopilando datos de los objetos del Modelo, realizando operaciones lógicas con ellos y después pasándolos a la Plantilla para su visualización
      Estructura básica:
      ```
      def <view_name>(request):
          # View Logic goes in here....
          return render(request, <template>, <context>)
      ```
      view_name: nombre de la vista
      request:
      return render: se utiliza para generar la respuesta HTML. Requiere:
      - request:
      - template: el archivo de plantilla
      - context: variables disponibles en la plantilla, casi siempre en forma de diccionario

   3) PLANTILLA: la capa de presentación, representa el contenido HTML en la interfaz del usuario
      Las plantillas son responsables de generar una imagen o video a partir de un

modelo o descripción (renderizar).
Implica el uso de etiquetas {% %} y variables de plantilla {{ }}, que permiten acceder al modo Django en la plantilla HTML.
Se pueden diseñar también usando CSS o cualquiera de sus marcos para hacer que la interfaz de usuario sea más presentable
La plantilla es un archivo HTML normal con el lenguaje de plantillas de Django:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Task List</title>
</head>
<body>
    <h1>Task List</h1>
    <ul>
        {% for task in tasks %}
            <li>{{ task.title }} - {{ task.completed|yesno:"Done, not" }}</li>
        {% empty %}
            <p>No tasks available.</p>
        {% endfor %}
    </ul>
</body>
</html>
```

for: el bucle itera a través de cada tarea en la lista **tasks**
title: para cada tarea, muestra esa tarea y
completed: su estado terminado o no
{% empty %}: muestra un mensaje de respaldo diciendo “no tasks available” si la lista tasks está vacía

En cambio, el patrón MVC, separa la lógica de una aplicación en estos tres componentes:
1) **Modelo**
2) **Vista**: Representación visual de los datos
3) **CONTROLADOR**: Gestiona la lógica de interacción entre el Modelo y la Vista. Responsable de recibir las acciones del usuario, actualizando el modelo y seleccionando la vista adecuada para presentar los datos

DIFERENCIAS:

	MCV	MTV
Gestiona lógica de interacción, intermediario	Controlador	Vistas
VISTA	Interfaz de usuario	Lógica de control, interacción

3. ¿Qué entendemos por app en Django?

https://www.codementor.io/@chirilovadrian360/apps-in-django-concept-free-samples-294vudyim5
En Django, se entiende como app a un módulo o componente autónomo dentro de un proyecto web que agrupa una funcionalidad concreta o un conjunto de características relacionadas. Por ejemplo, una app puede encargarse del sistema de usuarios, la gestión de un blog, o el carrito de compras. Cada app es modular, diseñada para funcionar con independencia de otras, lo que facilita su reutilización en diferentes proyectos. Además, una app típica en Django incluye todos los elementos necesarios para realizar esa función: modelos (para definir la estructura de datos), vistas (para procesar peticiones y devolver respuestas), plantillas (templates), archivos URL,

recursos estáticos como CSS o JavaScript, y posiblemente comandos de gestión o pruebas. Esta organización promueve una arquitectura ordenada, escalable y mantenible

4. **¿Qué es el flujo request-response en Django?**
   https://clouddevs.com/django/request-response-cycle/#:~:text=El%20ciclo%20de%20solicitud%2Drespuesta%20de%20Django%20es%20el%20proceso,comprender%20al%20trabajar%20con%20Django.
   El flujo request-response en Django es un concepto fundamental que describe cómo el framework web Django (conjunto de herramientas, bibliotecas, componentes…) procesa las solicitudes HTTP entrantes y genera las respuestas HTTP correspondientes. Es el mecanismo principal mediante el cual Django gestiona las interacciones de los clientes y sirve aplicaciones web, y lo hace usando el patrón MTV.
   Primero, el usuario hace una petición (un "request"), en el que entra a un sitio web, esto genera una petición HTTP (GET, POST…)
   Después, Django recibe la request a través del servidor web, va al archivo "urls.py" y busca una coincidencia con la URL que el usuario escribió, y si se encuentra, llama a la vista (función o clase) asociada.
   Esta procesa la información, interactúa con modelos y bases de datos y prepara los datos para la respuesta. Construye un objeto de respuesta HTTP, que incluye el contenido que se enviará al cliente (contenido HTML, datos JSON, o archivos binarios).
   Antes de enviar esta respuesta, se ejecuta el middleware de Django (funciones o clases que pueden realizar acciones como autenticación, modificación de solicitudes/respuestas, registro, etc. Puede procesar la solicitud antes de que llegue a la función de vista o la respuesta antes de que salga del servidor.
   Finalmente, la respuesta HTTP se envía al cliente a través de la red. El navegador web del cliente procesa la respuesta, renderizando el contenido HTML, ejecutando JavaScript, y mostrando la página web.
   Si la respuesta tiene datos incluídos, el cliente también puede usarlos para interactuar o renderizar.
   El ciclo request-response se repite cada vez que el cliente interactúa con la aplicación web mediante el envío de solicitudes adicionales.

5. **¿Qué es el concepto de ORM (Object-Relational Mapping)?**
   https://inspector.dev/setting-up-django-orm/#:~:text=%C2%BFQu%C3%A9%20es%20un%20ORM%20en,de%20aplicaciones%20basadas%20en%20ellas.
   El ORM en Django es un componente del framework que permite interactuar con una base de datos relacional mediante objetos de Python en lugar de escribir consultas SQL directamente.
   Django proporciona uno potente e intuitivo, que simplifica la manipulación de bases de datos y facilita la creación de aplicaciones basadas en ellas.
   El ORM funciona mapeando clases de Python a tablas de bases de datos y objetos de Python a registros de bases de datos. Proporciona una Interfaz de Programación de Aplicaciones de alto nivel, que oculta la complejidad del SQL y permite realizar operaciones comunes sobre la base de datos (crear, leer, actualizar y eliminar registros) utilizando métodos y atributos en Python
   Este sistema se basa en modelos (clases de Python que definen la estructura y el comportamiento de las tablas de la base de datos). Cada clase modelo representa una tabla, y sus atributos corresponden a las columnas de esta.
   Al definir modelos, se puede crear, consultar y modificar registros de la base de datos sin tener que escribir sentencias o consultas SQL de forma explícita

6. **¿Qué son los templates en Django?**
   https://blog.jetbrains.com/es/pycharm/2025/06/la-guia-definitiva-de-las-plantillas-de-django/#
   Los templates en Django son una parte fundamental del marco de trabajo de este. Permiten separar la presentación visual del sitio web del código oculto. Un template contiene las partes estáticas del resultado HTML que se desea, y una sintaxis especial que describe cómo se insertará el contenido dinámico.

Además, de que los templates o las plantillas de Django también podrían generar páginas web completas mientras que las consultas a la base de datos y otras tareas de procesamiento de datos son gestionadas por vistas y modelos.
Esta separación asegura un código limpio y fácil de mantener. Sin plantillas, se debería insertar HTML directamente en el código Python, lo que dificultaría la lectura