PROGRAMACIÓN II

1º AÑO

Clase Nº 6: Introducción a Flask y Desarrollo Web con Python

Contenido:

En la clase de hoy trabajaremos los siguientes temas:

- El rol de Flask y su arquitectura básica.
- Escritura de nuestra primera aplicación web en Python.
- Integración de formularios HTML con funciones de backend.
- Uso de Jinja2 para mostrar contenido dinámico en nuestras páginas.

1. Presentación:

Hoy comenzamos una nueva etapa en nuestro recorrido por el mundo de la programación. Hasta ahora, hemos aprendido los fundamentos del desarrollo web —HTML, CSS y JavaScript— que nos permiten crear estructuras, estilos e interactividad en el navegador. A partir de esta clase, vamos a integrar esos conocimientos con Python para dar vida a nuestras primeras aplicaciones web dinámicas, funcionales y personalizadas.

Para lograrlo, utilizaremos Flask, un framework liviano pero muy potente de Python, diseñado para crear aplicaciones web de manera sencilla y flexible. Flask nos permitirá construir nuestros propios servidores, definir rutas para

distintas páginas, recibir y procesar datos a través de formularios, y mostrar contenido personalizado utilizando plantillas HTML dinámicas.

Este enfoque marca un antes y un después: pasamos de trabajar con programas que se ejecutan en consola a construir sitios web que pueden ser accedidos desde cualquier navegador. Vamos a ver cómo Python puede interactuar con el mundo web, lo cual es clave para el desarrollo de dashboards, aplicaciones con bases de datos, sistemas de carga de datos y mucho más.

Estamos dando los primeros pasos hacia el desarrollo de aplicaciones completas, y esta base será fundamental para proyectos más avanzados que veremos en lo que resta del año. ¡Vamos a aprovecharla al máximo!

Los invito a ver el siguiente video sobre flask y Python, reflexiona sobre los posibles usos: https://www.youtube.com/watch?v=A7K1Cmwt9KI

2. Desarrollo

¿Qué es Flask?

Flask es un *micro-framework* escrito en Python que nos permite desarrollar aplicaciones web de forma sencilla, rápida y eficiente. Se lo llama “micro” no porque tenga pocas funcionalidades, sino porque ofrece una estructura mínima y no impone una arquitectura rígida al desarrollador. Esto lo convierte en una herramienta ideal para quienes están dando sus primeros pasos en el desarrollo web con **Python**, ya que permite entender claramente cómo

funcionan las partes fundamentales de una aplicación web.

Con **Flask**, podemos:

- Crear un servidor web local para probar nuestras aplicaciones.
- Definir diferentes rutas (URLs) que responden a solicitudes del navegador.
- Recibir datos a través de formularios y procesarlos.
- Mostrar contenido dinámico mediante plantillas HTML.
- Integrar fácilmente tecnologías como CSS, JavaScript, Bootstrap, y más.
- Conectarnos con bases de datos o APIs externas, entre otras muchas posibilidades.

Una de las principales ventajas de Flask es que **no viene con demasiadas cosas "preinstaladas"**, lo que nos obliga a comprender mejor qué herramientas necesitamos según cada proyecto. Esto lo diferencia de otros ***frameworks*** como **Django**, que ya viene con una estructura más completa y robusta, pero también más compleja para quien está empezando.

Flask se basa en dos pilares fundamentales:

- **Werkzeug**, una biblioteca que gestiona el enrutamiento y las peticiones **HTTP**.
- **Jinja2**, un motor de plantillas que permite generar páginas **HTML** dinámicas a partir de datos.

Además, Flask es altamente extensible. Existen muchas *extensiones* que se pueden agregar cuando el proyecto lo necesita, como:

- **Flask-WTF** para validación de formularios,
- **Flask-Login** para gestión de sesiones y usuarios,
- **Flask-SQLAlchemy** para trabajar con bases de datos de forma más amigable.

Gracias a su simplicidad, claridad y flexibilidad, **Flask** se ha convertido en una herramienta muy popular tanto en el mundo educativo como en el desarrollo profesional de **APIs**, paneles de administración, prototipos y dashboards de datos.

Instalación de Flask

Antes de comenzar a programar con Flask, es importante crear un **entorno virtual** para trabajar organizadamente y evitar conflictos con otros proyectos de Python.

Con el entorno virtual activo, ejecuta:

Esto descargará e instalará Flask en tu entorno. Podés comprobar que se instaló correctamente con:

```
pip list
```

Crear el archivo app.py

Ya podés comenzar a escribir tu primera aplicación Flask. Por ejemplo:

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def inicio():
    return '¡Hola, Flask!'

if __name__ == '__main__':
    app.run(debug=True)
```

Guarda este archivo como app.py y ejecútalo con:

```
python app.py
```

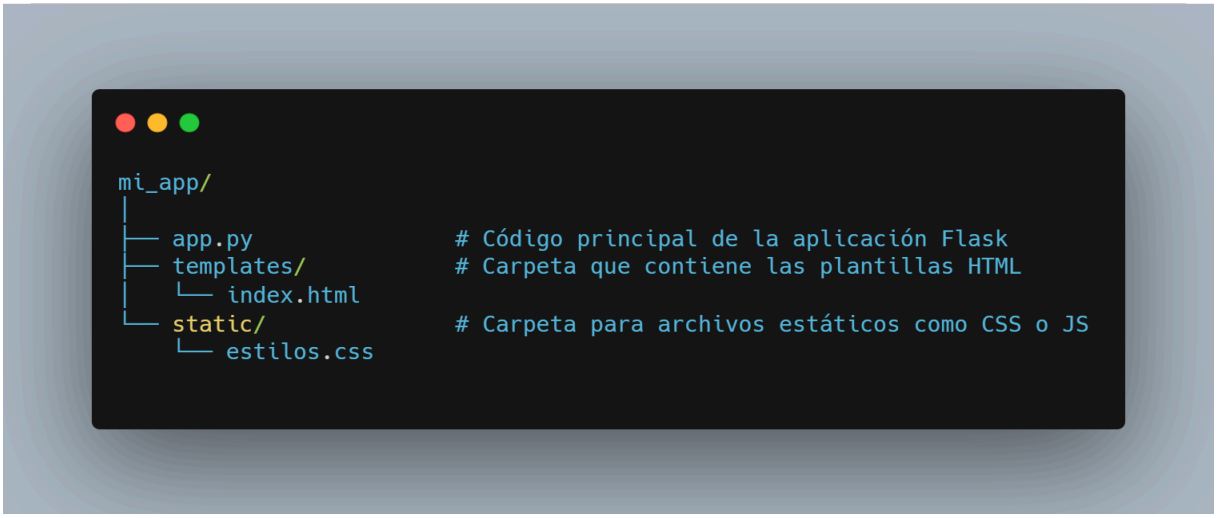
Después abrí tu navegador y visitá http://localhost:5000 para ver tu app en acción.

Arquitectura básica de una aplicación Flask

Una aplicación creada con Flask puede comenzar siendo extremadamente simple, incluso con unas pocas líneas de código. Sin embargo, a medida que crece y se vuelve más compleja, es importante organizarla de forma ordenada. Para eso, Flask permite escalar desde una estructura mínima hasta una aplicación modular y profesional.

📁 Estructura mínima

Una aplicación básica puede tener la siguiente estructura:



app.py – Ejemplo de código básico

```python
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/')
def inicio():
    return render_template('index.html')

if __name__ == '__main__':
    app.run(debug=True)
```

¿Qué hace cada parte?

- from flask import Flask: importa la clase Flask desde el framework.
- app = Flask(__name__): crea una instancia de la aplicación.
- @app.route('/'): define la ruta raíz del sitio. Es decir, qué sucede cuando accedemos al sitio web.
- render_template('index.html'): carga una plantilla HTML ubicada en la carpeta templates.
- app.run(debug=True): inicia el servidor local y activa el modo *debug* (muy útil para desarrollo, ya que recarga la app ante cada cambio y muestra errores detallados).

Carpeta templates/

Contiene los archivos HTML que serán renderizados dinámicamente con la ayuda de **Jinja2**. Por convención, debe llamarse templates.

Ejemplo: templates/index.html

```html
<!DOCTYPE html>
<html>
<head>
    <title>Mi primera app Flask</title>
</head>
<body>
    <h1>¡Hola desde Flask!</h1>
</body>
</html>
```

Carpeta static/

Aquí guardamos archivos como hojas de estilo (CSS), imágenes, scripts JavaScript y cualquier otro recurso estático. Para usarlos en un template se emplea la función url_for.

```html
<link rel="stylesheet" href="{{ url_for('static', filename='estilos.css') }}">
```

Estructura mínima de una aplicación Flask

Una aplicación básica de Flask puede tener solo unas pocas líneas de código. Ejemplo:

```python
from flask import Flask

app = Flask(__name__)

@app.route('/')
def home():
    return 'Hola, mundo desde Flask!'

if __name__ == '__main__':
    app.run(debug=True)
```

Rutas y métodos en Flask

En una aplicación web, las **rutas** definen qué contenido o función se debe ejecutar cuando un usuario accede a una determinada URL. En Flask, estas rutas se crean usando decoradores (@app.route) que están asociados a funciones de Python.

¿Qué es una ruta?

Una ruta es una URL que responde a una solicitud del navegador. Por ejemplo:

- / → página de inicio
- /contacto → formulario de contacto
- /productos → listado de productos

Ejemplo básico de ruta:

```python
@app.route('/')
def inicio():
    return 'Bienvenido a mi sitio web'
```

Cuando el usuario visita http://localhost:5000/ Flask ejecuta la función inicio y muestra el texto "Bienvenido a mi sitio web".

Rutas con parámetros

También podemos capturar valores desde la URL usando los signos < >. Estos valores se convierten en parámetros que se pasan a la función.

```python
@app.route('/saludo/<nombre>')
def saludo(nombre):
    return f'¡Hola, {nombre}!'
```

Si el usuario accede a http://localhost:5000/saludo/Ana, verá:

¡Hola, Ana!

Actividad 1: Tu primera app Flask

Instrucciones:

Abrí ChatGPT y escribí este prompt: *"Generar un ejemplo mínimo de Flask que muestre un saludo al ingresar a la página."*

1. Copiá el código que te responde en un archivo app.py y ejecutalo.
2. Respondé: ¿Funcionó? ¿Qué línea te pareció más clara?

Objetivo: Identificar la estructura mínima de una app Flask y familiarizarte con el uso de la IA como asistente para programar.

Métodos HTTP: GET y POST

Cuando un navegador accede a una ruta, utiliza un **método HTTP**. Los dos más comunes son:

- **GET**: solicita datos desde el servidor (por ejemplo, acceder a una página).
- **POST**: envía datos al servidor (por ejemplo, enviar un formulario).

Por defecto, todas las rutas en Flask aceptan solo **GET**, pero podemos especificar otros métodos:

```python
@app.route('/formulario', methods=['GET', 'POST'])
def formulario():
    if request.method == 'POST':
        nombre = request.form['nombre']
        return f'Tu nombre es {nombre}'
    return render_template('formulario.html')
```

¿Qué es request.form?

Es una forma de acceder a los datos enviados desde un formulario HTML. Permite capturar los valores de los campos usando el nombre del campo como

clave.

📌 Resumen

Concepto	Descripción	Ejemplo
@app.route()	Define la URL que responde una función	@app.route('/')
Ruta con parámetro	Captura valor desde la URL	/saludo/<nombre>
methods=['POST']	Habilita recibir datos desde formularios	@app.route(..., methods=['POST'])
request.form	Accede a datos enviados por formulario	request.form['campo']
render_template()	Muestra una página HTML dinámica desde carpeta templates/	render_template('inicio.html')

Renderizar HTML con Flask

Para mostrar páginas HTML usamos el método render_template. Los archivos

HTML deben guardarse en una carpeta llamada templates.

```python
from flask import render_template

@app.route('/')
def inicio():
    return render_template('index.html')
```

Paso de variables a templates

Podemos pasar datos desde Python al HTML usando variables.

```python
@app.route('/usuario/<nombre>')
def usuario(nombre):
    return render_template('usuario.html', nombre=nombre)
```

Uso de formularios en Flask

Para capturar datos del usuario usamos formularios HTML y Flask los recibe con

```python
request.form:
from flask import request

@app.route('/procesar', methods=['POST'])
def procesar():
    nombre = request.form['nombre']
    return f'Hola, {nombre}!'
```

Uso de plantillas HTML con Jinja2

Ninja2 es el **motor de plantillas** que utiliza Flask para generar páginas HTML dinámicas. Su función principal es permitir que nuestro código Python "converse" con el HTML, **incrustando datos y lógica directamente dentro de las plantillas** que se mostrarán en el navegador.

En otras palabras, gracias a Jinja2 podemos enviar variables, listas, mensajes o cualquier tipo de dato desde Flask hacia los archivos .html, y personalizar el contenido que ve el usuario en función de esa información.

¿Por qué necesitamos Jinja2?

Cuando trabajamos con Flask, no queremos mostrar solo texto plano o HTML estático. Por ejemplo, si un usuario se registra con su nombre, queremos saludarlo por su nombre, mostrar sus datos, o generar una tabla con resultados dinámicos. Para eso necesitamos una herramienta que permita **mezclar código Python con HTML de forma segura y ordenada**, y ese es el rol de Jinja2.

📁 Estructura

Las plantillas deben guardarse en una carpeta llamada **templates** dentro del proyecto. Ejemplo:

```
mi_app/
│
├── app.py
├── templates/
│   ├── index.html
│   └── saludo.html
```

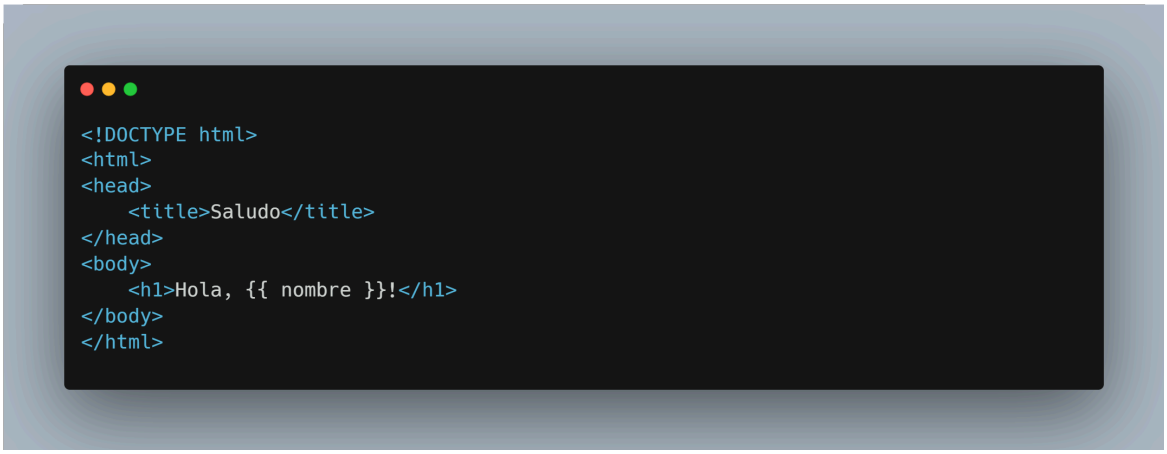
Ejemplo: pasar variables al HTML

```python
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/usuario/<nombre>')
def usuario(nombre):
    return render_template('saludo.html', nombre=nombre)
```

templates/saludo.html:



```html
<!DOCTYPE html>
<html>
<head>
    <title>Saludo</title>
</head>
<body>
    <h1>Hola, {{ nombre }}!</h1>
</body>
</html>
```

Sintaxis básica de Jinja2

Elemento	Sintaxis	Ejemplo en HTML
Insertar variable	{{ variable }}	{{ nombre }}
Estructura condicional	{% if ... %} ... {% endif %}	{% if edad >= 18 %} Mayor de edad {% endif %}
Bucle (for)	{% for x in lista %}	{% for p in productos %} {{ p }} {% endfor %}
Comentario en plantilla	{# Comentario #}	{# Esto no se muestra #}
Incluir otra plantilla	{% include 'menu.html' %}	Reutilización de componentes

Ventajas de usar Jinja2

- Permite separar la lógica (Python) de la presentación (HTML).
- Hace que las páginas sean reutilizables y más limpias.
- Facilita mostrar listas, tablas, formularios y resultados.
- Es compatible con Bootstrap y cualquier otra librería visual.

Actividad 2:

📌 Crear una aplicación Flask que tenga:

- Una ruta principal con un formulario para ingresar un nombre.

- Al enviar el formulario, debe saludar al usuario en otra ruta usando templates.

Opcional: agregar un contador de visitas o registrar los nombres en una lista.

1. ¿Qué es una ruta en Flask?

2. ¿Qué función cumple render_template?

3. ¿Cómo se procesan datos de un formulario?

4. ¿Qué ventajas tiene usar Jinja2?

5. Explica brevemente cómo crear un entorno virtual.

Foro – Obligatorio

Objetivo: Reflexionar sobre el proceso de creación de una aplicación web utilizando Flask y compartir experiencias con compañeros.

Consigna:

Publicá en el foro una reflexión personal respondiendo a las siguientes preguntas:

¿Qué fue lo más desafiante de crear tu primera aplicación Flask?

¿Qué aprendiste sobre la estructura de una app web (carpetas, rutas, plantillas)?

¿Qué ventajas encontrás al usar Flask con HTML y Jinja2 en comparación con el desarrollo tradicional en consola?

¿Cómo imaginás que podrías aplicar Flask en un proyecto futuro o en tu entorno laboral?

Además:

💬 Leé al menos una participación de un/a compañero/a y realizá un comentario constructivo. Podés: Aportar un consejo o compartir una experiencia similar.

Criterios de Evaluación

1. **Claridad y Coherencia en las Respuestas :**
    - Las respuestas deben ser claras, bien estructuradas y fáciles de seguir. Se evaluará la capacidad del estudiante para expresar sus ideas de manera coherente y lógica.
2. **Profundidad del Análisis:**
    - Se valorará la profundidad del análisis crítico en relación con el artículo. Los estudiantes deben demostrar que han comprendido los puntos clave y que pueden relacionarlos con sus propias experiencias y conocimientos.
3. **Argumentación y Justificación:**
    - Los estudiantes deben respaldar sus opiniones con argumentos sólidos y ejemplos concretos. Se evaluará la capacidad para justificar sus afirmaciones y reflexiones de manera efectiva.
4. **Participación y Colaboración en el Foro:**
    - Se tendrá en cuenta la participación activa en el foro, incluyendo la capacidad de responder a las intervenciones de otros compañeros de manera constructiva y respetuosa.

Cierre:

Hoy dimos un paso clave en nuestro camino como desarrolladores: comenzamos a crear nuestras propias aplicaciones web utilizando **Flask**, un framework potente y accesible que nos permite combinar lo mejor de HTML,

CSS y Python en un mismo proyecto.

Aprendimos a levantar un servidor local, definir rutas, renderizar contenido dinámico usando plantillas, y procesar formularios con datos enviados por los usuarios. Estos conceptos son fundamentales para construir sitios web interactivos, sistemas de carga de datos, formularios personalizados y dashboards.

Este enfoque trasciende el desarrollo web tradicional: también es útil en áreas como la automatización de reportes, análisis de datos, inteligencia artificial aplicada a servicios web, y el desarrollo de herramientas internas para empresas.

🔁 En la **próxima clase (N.° 7)** haremos un **repaso general** de todo lo aprendido hasta ahora. Será una oportunidad para reforzar conceptos, aclarar dudas y practicar con ejercicios integradores.

🗃️ Luego, en la **clase N.° 8**, daremos un nuevo salto incorporando **bases de datos** a nuestras aplicaciones Flask. Esto nos permitirá almacenar, consultar y gestionar información de forma persistente, lo cual es esencial para crear sistemas más completos y funcionales.

🚀 ¡Excelente trabajo hasta acá! Seguimos avanzando paso a paso hacia el desarrollo de nuestras propias aplicaciones inteligentes.

Bibliografía Obligatoria:

- Grinberg, M. (2018). *Flask web development: Developing web applications with Python*. O'Reilly Media.
- Ramalho, L. (2022). *Python fluido* (2.ª ed.). Anaya Multimedia. (Original: *Fluent Python*, O'Reilly Media)
- Lutz, M. (2013). *Learning Python* (5th ed.). O'Reilly Media.

Recursos adicionales:

- **Documentación oficial de Flask**

  https://flask.palletsprojects.com/

  Ideal para aprender con ejemplos claros y documentación siempre actualizada.

- **Curso gratuito en español de Flask en YouTube**

  Curso Flask Python desde cero

  Muy recomendable para estudiantes visuales. Explica paso a paso y en español.

- **Real Python – Flask Series**

  https://realpython.com/tutorials/flask/