PROGRAMACIÓN II

1º AÑO

Clase N.º 10: Repaso Integrador y Preparación para el Parcial

Contenido:

En la clase de hoy trabajaremos los siguientes temas:

- Programación Orientada a Objetos (POO): clases, objetos, atributos, métodos, herencia y encapsulamiento.
- HTML, CSS y JavaScript: elementos básicos, formularios y comportamiento dinámico en el navegador.
- Flask: estructura de una aplicación, rutas, renderización de plantillas y manejo de formularios.
- Jinja2: uso de plantillas dinámicas para integrar lógica en el HTML.
- SQLAlchemy: definición de modelos, conexión con bases de datos.
- Operaciones CRUD: crear, leer, actualizar y eliminar datos desde la app.
- Integración completa: conexión entre frontend (HTML/CSS) y backend (Flask + base de datos).

1. Presentación

En esta instancia del cuatrimestre, llegamos a una clase muy importante: la antesala de nuestro primer examen parcial práctico. A lo largo de estas semanas, hemos construido paso a paso un conjunto sólido de conocimientos

y habilidades que nos permiten hoy comenzar a pensar como científicos de datos..

El objetivo principal de esta clase es integrar todos los contenidos abordados desde la clase 1 hasta la clase 9, y prepararnos de manera activa para enfrentar el desafío del parcial. Para lograrlo, vamos a trabajar sobre un caso práctico completo, muy similar en estructura y dificultad al que se presentará en el examen.

Vamos a repasar los conceptos fundamentales de la programación web con Python, incluyendo la lógica de la Programación Orientada a Objetos, el uso de bases de datos y la estructura de una aplicación web moderna.

Exploraremos cómo todos estos componentes trabajan juntos dentro de un proyecto con Flask, nuestra herramienta principal para el desarrollo web en este tramo de la cursada.

Realizaremos ejercicios guiados para poner en práctica cada parte del proceso: desde el diseño del modelo de datos, hasta la creación de rutas, formularios y operaciones CRUD.

Propondremos actividades colaborativas y desafíos, para fomentar el trabajo en equipo, la discusión entre pares y la puesta en común de soluciones alternativas.

Habrá espacio para consultas abiertas y revisión de dudas específicas de cara al examen, con recomendaciones y sugerencias sobre cómo organizar el tiempo y estructurar el código.

Esta clase no es solo un repaso: es una oportunidad para consolidar el conocimiento adquirido, identificar posibles dificultades y llegar al examen con mayor confianza y preparación.

Desarrollo

Git y GitHub: herramientas clave para el desarrollo colaborativo

Git es una herramienta fundamental en el trabajo de cualquier desarrollador o desarrolladora. Se trata de un sistema de control de versiones que permite **registrar y gestionar el historial de cambios** en un proyecto de software, facilitando la evolución del código de forma organizada, segura y eficiente. Gracias a Git, es posible trabajar en diferentes versiones del mismo proyecto, corregir errores, desarrollar nuevas funcionalidades y volver a estados anteriores cuando sea necesario.

Complementando esta herramienta, **GitHub** funciona como una plataforma en la nube que permite **alojar repositorios de código** y coordinar el trabajo en equipo. No solo facilita la colaboración entre varios desarrolladores de forma remota, sino que también actúa como un espacio de respaldo y documentación del proyecto.

Durante las clases, aprendimos a utilizar los comandos básicos de Git para llevar adelante el ciclo de trabajo cotidiano:

- Inicializar un nuevo repositorio con git init,
- Agregar archivos al área de preparación con git add .,
- Confirmar los cambios mediante git commit -m "mensaje",
- Sincronizar el proyecto con un repositorio remoto en GitHub usando git push y git pull.

Además, exploramos la utilidad de las **ramas (branches)**, que nos permiten desarrollar nuevas características o realizar pruebas sin modificar la versión principal del proyecto. Esto garantiza un entorno de trabajo más ordenado y seguro, donde cada funcionalidad puede desarrollarse de manera independiente antes de integrarse al código principal.

En conjunto, Git y GitHub se convierten en herramientas esenciales para el desarrollo profesional de software, tanto en proyectos individuales como en equipos colaborativos.

Actividad 1: Git y GitHub

En este ejercicio vas a realizar los pasos básicos para comenzar a trabajar con Git y publicar tu proyecto en GitHub.

Instrucciones:

1. Creá una carpeta para tu proyecto web en tu computadora.
2. Inicializá un repositorio local con git init.
3. Agregá un archivo README.md con una breve descripción del proyecto.
4. Registrá los cambios con Git y realizá el primer commit.

5. Subí el proyecto a GitHub usando Git.

Programación Orientada a Objetos

La **Programación Orientada a Objetos (POO)** es un paradigma de programación que propone estructurar el código basándose en "objetos", es decir, en representaciones de entidades del mundo real o conceptual. Este enfoque busca **modelar sistemas complejos de forma más natural, modular y organizada**, facilitando tanto el diseño como el mantenimiento del software.

En lugar de programar en función de procesos o acciones aisladas, la POO organiza el código en **clases**, que son como moldes o planos a partir de los cuales se crean **objetos**. Cada objeto tiene:

- **Atributos**: representan su estado o características (por ejemplo, el nombre y la edad de una persona).
- **Métodos**: representan su comportamiento o lo que puede hacer (por ejemplo, hablar, caminar, calcular).

Este modelo permite trabajar con componentes más cercanos a cómo pensamos los problemas en la vida real, haciendo que el código sea más intuitivo, reutilizable y escalable.

Los 3 pilares fundamentales de la POO

1. Encapsulamiento

Es el principio que consiste en **ocultar los detalles internos del funcionamiento de un objeto**, exponiendo solo lo necesario para su uso. Esto se logra a través

de mecanismos que permiten **proteger los datos** y **controlar el acceso** a ellos.

El encapsulamiento permite evitar errores, ya que otras partes del programa no pueden modificar directamente el estado interno del objeto sin seguir ciertas reglas. A su vez, favorece la modularidad, ya que los cambios internos no afectan al resto del programa si la interfaz pública se mantiene.

2. Herencia

La herencia permite **crear nuevas clases a partir de clases ya existentes**, heredando sus atributos y métodos. Esto significa que una clase "hija" puede aprovechar todo lo que ya ofrece una clase "padre", y además puede **agregar o modificar funcionalidades** específicas.

Este principio favorece la **reutilización del código**, la **extensión de funcionalidades** sin duplicar lógica, y una organización jerárquica de clases. Por ejemplo, podríamos tener una clase Animal y luego clases hijas como Perro o Gato, que heredan lo común y agregan lo propio.

3. Polimorfismo

El polimorfismo permite que **objetos de distintas clases respondan de manera diferente a un mismo método**. Esto se logra cuando varias clases comparten una **interfaz común**, pero implementan el comportamiento de forma particular. Gracias a este principio, podemos escribir código más flexible y general, ya que podemos tratar distintos tipos de objetos de forma homogénea sin saber exactamente qué clase tienen, confiando en que responderán correctamente.

Actividad 2: POO Aplicado: Clase Producto

En este ejercicio vas a aplicar los conceptos básicos de la Programación Orientada a Objetos (POO) para crear una clase sencilla que represente un producto. El objetivo es poner en práctica la creación de clases, el uso de atributos y la definición de métodos.

Instrucciones:

1. Definí una clase llamada Producto.

   Esta clase debe contener tres atributos que representen las características principales de un producto:

   - nombre: el nombre del producto.
   - precio: el valor numérico del producto.
   - imagen: una URL o texto que represente una imagen (puede ser solo un string por ahora).

2. Agregá un método llamado mostrar_info().

   Este método debe mostrar en pantalla los datos del producto de forma clara y ordenada (nombre, precio e imagen), utilizando un formato legible para el usuario.

3. Instanciá al menos tres objetos de la clase Producto, cada uno con valores diferentes para sus atributos.

   Luego, llamá al método mostrar_info() en cada uno de ellos para verificar que los datos se muestren correctamente.

HTML, CSS y Bootstrap

Para desarrollar aplicaciones web completas, no solo necesitamos que

funcionen correctamente desde lo técnico (como con Flask o Python), sino también que tengan una estructura clara y una apariencia visual agradable para el usuario. En este sentido, HTML, CSS y Bootstrap son herramientas esenciales.

🧱 HTML (HyperText Markup Language)

HTML es el **lenguaje base de la web**. Se utiliza para definir la **estructura y contenido** de una página. A través de etiquetas, podemos organizar los elementos que el usuario ve: títulos, párrafos, formularios, imágenes, enlaces, listas, tablas, etc.

HTML no se encarga del diseño o la estética, sino de establecer el "esqueleto" de la página. Por ejemplo, define que algo es un botón, un campo de texto, una sección o un formulario.

🎨 CSS (Cascading Style Sheets)

CSS se usa para **aplicar estilos visuales** a los elementos definidos en HTML. Permite cambiar colores, fuentes, márgenes, tamaños, posiciones y mucho más. Con CSS se controla cómo se ve y se comporta cada parte de una página en diferentes dispositivos o tamaños de pantalla.

En resumen: si HTML define **qué hay**, CSS define **cómo se ve**.

⚙️ Bootstrap

Bootstrap es un **framework** de código abierto que combina **CSS y JavaScript** para facilitar el diseño web. Ofrece una colección de componentes predefinidos (botones, tarjetas, formularios, menús, alertas, etc.) y un sistema de **grillas responsivas**, lo que significa que los sitios se adaptan automáticamente a distintos tamaños de pantalla (computadoras, tablets, celulares).

Usar Bootstrap permite lograr diseños atractivos y funcionales sin tener que escribir todo el CSS desde cero. Solo hace falta aplicar ciertas clases predefinidas a los elementos HTML.

Actividad 3: Creá tu formulario con ayuda de ChatGPT

Instrucciones

1. **Planteá tu necesidad a ChatGPT de forma clara.**

   En vez de escribir el código desde cero, vas a pedirle a ChatGPT que lo genere por vos. Usá una consigna como esta:

✉️ *"Generá un formulario en HTML usando Bootstrap para ingresar un producto con nombre, precio e imagen. Que esté dentro de un container y sea responsivo."*

2. **Copiá el código que te dé y pegalo en tu archivo HTML.**

   Guardá el archivo con un nombre como formulario.html.

3. **Abrí el archivo en tu navegador y observá cómo se ve.**

   Probalo en modo escritorio y modo celular (podés usar el inspector del navegador para simular distintos tamaños de pantalla).

4. **Revisá y mejorá el diseño con ayuda de ChatGPT.**

   Por ejemplo, podés preguntarle:

✉️ *"¿Cómo puedo centrar el formulario en la página?"*

✉️ *"¿Qué clase puedo agregar para que los campos se vean más separados?"*

✉️ *"¿Cómo hago que el botón esté alineado a la derecha?"*

5. **Personalizá el formulario.**

Cambiá colores, textos o tamaños usando clases de Bootstrap o consultando:

Construyendo aplicaciones web dinámicas con Python

Flask es un **microframework** escrito en Python que permite desarrollar aplicaciones web de forma simple, rápida y flexible. A pesar de su ligereza, ofrece todo lo necesario para crear servidores web funcionales, manejar rutas, procesar formularios, conectarse a bases de datos y mucho más. Es una herramienta ampliamente utilizada tanto para proyectos pequeños como para prototipos y aplicaciones profesionales.

Una de las características más importantes de Flask es que nos permite definir **rutas**: estas son las direcciones que el usuario escribe en el navegador y que se asocian a funciones específicas en nuestro código. Por ejemplo, podemos definir qué debe suceder cuando alguien accede a la página principal (/), a una sección de contacto (/contacto) o a un formulario de registro (/registro).

🧩 Jinja2: el motor de plantillas de Flask

Flask incorpora de forma predeterminada a **Jinja2**, un **motor de plantillas** que nos permite escribir código HTML dinámico. Es decir, nos da la posibilidad de **incrustar lógica de Python dentro de archivos HTML**, de forma segura y controlada.

Gracias a Jinja2 podemos:

- Mostrar valores de variables pasadas desde Flask.
- Recorrer listas o diccionarios con bucles.

- Evaluar condiciones (if, else).
- Utilizar herencia de plantillas para reutilizar estructuras comunes (como encabezados o pies de página).

Esto nos permite separar claramente la **lógica de la aplicación** (que se escribe en Python) de la **presentación visual** (que se construye en HTML), lo cual es una buena práctica en el desarrollo web.

Comunicación entre Flask y Jinja2

Cuando queremos mostrar contenido dinámico en nuestras páginas HTML, usamos la función render_template() desde Flask. Esta función busca el archivo HTML correspondiente dentro de la carpeta templates/ y le envía los datos que queremos mostrar.

En ese caso, estamos enviando la variable lista_productos al archivo productos.html, donde podremos acceder a ella como productos y usarla dentro del HTML con la sintaxis de Jinja2:

```
<ul>
  {% for producto in productos %}
    <li>{{ producto.nombre }} - ${{ producto.precio }}</li>
  {% endfor %}
</ul>
```

Esto nos permite generar contenido de forma automática en función de los datos disponibles, y actualizarlo fácilmente desde el backend sin modificar manualmente el HTML.

Actividad 4: Aplicación con Flask, Jinja2 y Bootstrap

En este ejercicio vas a construir una pequeña aplicación web utilizando Flask. El objetivo es que practiques la creación de rutas, el uso de plantillas HTML dinámicas con Jinja2 y la integración de Bootstrap para mostrar los datos de forma visualmente atractiva.

Instrucciones:

1. **Creá un archivo llamado app.py.**

   Este archivo será el punto de entrada de tu aplicación Flask. Asegurate de importar las librerías necesarias y configurar la aplicación correctamente.

2. **Definí dos rutas principales:**
   - Una ruta /nuevo que muestre un formulario en HTML para cargar nuevos productos (nombre, precio, imagen).
   - Una ruta /productos que muestre una lista de productos

previamente cargados.

3. **Utilizá render_template()** para vincular cada ruta con su respectiva plantilla HTML. El formulario debe estar en una plantilla (por ejemplo, nuevo.html) y la lista de productos en otra (por ejemplo, productos.html).
4. **Pasá una lista de productos desde Flask al HTML** utilizando variables en render_template(). Esta lista puede estar predefinida en el código o cargarse desde un formulario (según el enfoque que estés practicando).
5. **Mostrá los productos en la plantilla utilizando Jinja2**, recorriendo la lista con un bucle {% for %}. Cada producto debe mostrarse dentro de una **tarjeta de Bootstrap**, incluyendo su nombre, precio e imagen.
6. **Aplicá clases de Bootstrap** para dar estilo al formulario y a las tarjetas, asegurándote de que la visualización sea **responsiva** y se vea bien tanto en computadoras como en dispositivos móviles.

SQLAlchemy y operaciones CRUD

SQLAlchemy es una poderosa biblioteca de Python que nos permite interactuar con **bases de datos relacionales** (como SQLite, PostgreSQL o MySQL) de una manera más simple, clara y estructurada. En lugar de escribir directamente sentencias SQL, SQLAlchemy nos permite trabajar con **clases y objetos**, utilizando el paradigma de la **Programación Orientada a Objetos** para modelar nuestras tablas y filas.

Cada **clase** que definimos representa una **tabla** en la base de datos, y cada **instancia** de esa clase representa una **fila** o registro. Esto facilita la lectura, escritura y mantenimiento del código, además de permitirnos integrar más

fácilmente la lógica del backend con los datos que manipulamos.

Operaciones CRUD con SQLAlchemy

Una de las grandes ventajas de SQLAlchemy es que nos permite realizar de forma sencilla las operaciones CRUD (Crear, Leer, Actualizar y Eliminar), que son fundamentales en cualquier aplicación que trabaje con datos.

Crear (Create)

Para agregar un nuevo registro, creamos una instancia de la clase (tabla) y la añadimos a la sesión de la base de datos:

```
nuevo_producto = Producto(nombre="Café", precio=1200)
db.session.add(nuevo_producto)
db.session.commit()
```

Leer (Read)

Podemos consultar la base de datos de distintas maneras:

Obtener todos los registros:

```
Producto.query.all()
```

Obtener un registro por su ID:

```
Producto.query.get(1)
```

Filtrar por un atributo específico:

```
Producto.query.filter_by(nombre="Café").first()
```

Actualizar (Update)

Para modificar un registro existente, primero lo consultamos, luego cambiamos sus atributos y finalmente guardamos los cambios:

```
producto = Producto.query.get(1)
producto.precio = 1400
db.session.commit()
```

Eliminar (Delete)

Si queremos borrar un registro de la base de datos, lo consultamos y lo eliminamos de la sesión:

```
producto = Producto.query.get(1)
db.session.delete(producto)
db.session.commit()
```

Actividad 5: Manejo de datos con SQLAlchemy y Flask

En este ejercicio vas a aplicar los conocimientos adquiridos sobre **modelado de datos con SQLAlchemy** y **rutas dinámicas en Flask**. Vas a crear una clase que represente productos, insertar registros, mostrarlos desde el backend y permitir su eliminación mediante una ruta.

🧭 Instrucciones:

1. **Definí una clase Producto que represente una tabla de la base de datos.**

   La clase debe incluir los siguientes campos:

   - id (clave primaria, entero, autoincremental)
   - nombre (cadena de texto)
   - precio (número)
   - imagen (cadena que contenga la URL o nombre de la imagen)

Esta clase actuará como modelo en SQLAlchemy para mapear la tabla de productos.

2. **Insertá al menos tres productos desde el código Python.**

   Podés hacerlo dentro de una ruta especial o directamente al inicializar la aplicación. Recordá utilizar db.session.add() para agregar los objetos y db.session.commit() para guardar los cambios en la base de datos.

3. **Mostrá los productos en pantalla.**

   Podés visualizarlos en consola con print() (para pruebas simples), o preferentemente en una **vista HTML** utilizando render_template() y Jinja2 para recorrer la lista de productos y mostrarlos en tarjetas Bootstrap o en una tabla.

4. **Agregá una ruta dinámica /eliminar/<id> que permita borrar un producto específico.**

   Esta ruta debe recibir el ID del producto como parámetro, buscarlo en la base de datos, eliminarlo con db.session.delete() y guardar los cambios con db.session.commit(). Luego, podés redirigir al usuario nuevamente a la página que muestra la lista de productos actualizada.

Seguridad y distribución en el desarrollo web con Flask

En el desarrollo web moderno, **proteger los datos del usuario** no es opcional: es una responsabilidad central de cualquier aplicación. Uno de los aspectos más importantes es el manejo seguro de las **contraseñas**, ya que almacenar contraseñas en texto plano representa un riesgo grave.

Durante las clases, aprendimos a **encriptar contraseñas** utilizando la función generate_password_hash() de la biblioteca werkzeug.security. Esta función convierte una contraseña en una cadena encriptada (hash), que no puede revertirse a su forma original. De esta forma, incluso si alguien accede a la base de datos, no podrá conocer las contraseñas reales de los usuarios.

Cuando un usuario inicia sesión, no comparamos la contraseña directamente, sino que utilizamos la función check_password_hash() para verificar si la contraseña ingresada coincide con el hash almacenado. Este proceso garantiza que las contraseñas nunca se manipulen en texto plano, ni se guarden de forma insegura.

Además de la seguridad, también vimos cómo **distribuir nuestras aplicaciones Flask** para que puedan ejecutarse fuera del entorno de desarrollo. Una

herramienta útil para esto es **PyInstaller**, que nos permite convertir nuestra aplicación en un archivo ejecutable. Con el comando:

Podemos generar un ejecutable que empaqueta el código de la aplicación y sus dependencias, facilitando su ejecución en otras computadoras, incluso si no tienen Python instalado.

Este tipo de distribución puede ser especialmente útil para entregar prototipos, herramientas internas o pequeñas utilidades de escritorio basadas en una interfaz web local.

Cierre:

Durante la materia, recorrimos los fundamentos esenciales del desarrollo web con Python. Vimos cómo crear aplicaciones dinámicas utilizando **Flask** como base para el backend, y cómo integrar tecnologías del frontend como **HTML, CSS y Bootstrap** para lograr interfaces visuales limpias, ordenadas y adaptables. Aprendimos a modelar datos utilizando **SQLAlchemy**, a crear rutas, a renderizar plantillas con **Jinja2** y a organizar proyectos siguiendo los principios de la **Programación Orientada a Objetos**. Además, incorporamos herramientas como **Git** para el control de versiones, lo que nos permitió dar un paso más hacia un enfoque de trabajo profesional.

Con este conjunto de conocimientos y habilidades, estás en condiciones de enfrentar el examen parcial con confianza. Pero recordá que el examen no es solo una instancia de evaluación técnica, sino una oportunidad para demostrar tu capacidad. En este sentido, no alcanza con que tu aplicación simplemente

"funcione". Es fundamental que esté bien organizada, que el código sea comprensible, mantenible y que respete las buenas prácticas que fuimos incorporando a lo largo de las clases.

Cuando te sientes a desarrollar, asegurate de **probar tu aplicación varias veces antes de darla por terminada**. Ejecutá cada ruta, completá cada formulario, verificá cómo se comporta el sistema frente a distintos tipos de datos. No des nada por sentado: lo que parece obvio puede fallar si no se testea. Al trabajar con entradas del usuario, no olvides **validar cuidadosamente todos los datos** que se reciben. Nunca hay que confiar ciegamente en lo que llega desde un formulario. Pensá qué pasaría si alguien deja un campo vacío, si ingresa texto donde van números o si intenta cargar información maliciosa. Anticiparte a esos casos es parte del desarrollo responsable.

En cuanto al código, es clave que lo **organices de manera clara y modular**. Dividí tu lógica en funciones o archivos separados según su responsabilidad. Esto no solo hace que tu código sea más fácil de entender para vos y para otros, sino que también facilita su mantenimiento y reutilización en el futuro. Evitá escribir todo junto en un mismo archivo o función extensa. Cada bloque de código debe tener un propósito definido y limitado. Comentá lo necesario: explicá aquellas secciones que podrían resultar confusas para alguien que no escribió el código, pero sin sobrecargarlo de explicaciones innecesarias. El código bien escrito debería, en gran parte, explicarse por sí mismo.

Finalmente, te invito a **pensar tu código como una expresión de tu forma de razonar**. Cada línea que escribís refleja tu manera de analizar un problema, de

anticipar dificultades y de construir soluciones. La prolijidad, la claridad y el respeto por las buenas prácticas no son detalles estéticos: son parte de lo que define a un buen programador. Aplicá lo que aprendiste no solo para cumplir con los requisitos del examen, sino como una forma de trabajo que te acompañe en tu crecimiento profesional.

Bibliografía obligatoria

- Grinberg, M. (2018). *Flask Web Development*. O'Reilly Media.
- Chacon, S., & Straub, B. (2014). *Pro Git*. https://git-scm.com/book/en/v2
- Freeman, E., & Robson, E. (2014). *Head First HTML and CSS*.
- Beazley, D. M. (2021). *Python: Guía de referencia rápida*. O'Reilly Media.