PROGRAMACIÓN II

1º AÑO

Clase N.º 7: Repaso y Sistematización (Clases 1 a 6)

Contenido:

En la clase de hoy realizaremos un repaso integral de los siguientes temas trabajados hasta ahora:

- Tipos de datos complejos en Python: listas, tuplas, diccionarios.
- Uso de Git y GitHub para el control de versiones.
- Integración de Git con Visual Studio Code.
- Introducción a la Programación Orientada a Objetos (POO) en Python.
- Fundamentos de HTML, CSS y JavaScript.
- Desarrollo de aplicaciones web con Flask y Jinja2.

1. Presentación:

Bienvenidos y bienvenidas a la clase N.º 7 de Programación II 👋. Llegamos al punto medio del cuatrimestre, y es un excelente momento para detenernos y repasar todo lo que hemos aprendido hasta ahora.

El objetivo de esta clase es consolidar los conocimientos, clarificar dudas y conectar los distintos conceptos que venimos trabajando.

La programación es una disciplina que se construye paso a paso, y revisar los fundamentos es clave para avanzar con seguridad en los próximos contenidos.
Vamos a recorrer los temas desde una mirada integradora, viendo cómo se relacionan entre sí y de qué manera forman la base sobre la cual construiremos nuestras aplicaciones.

2. Desarrollo

1. Tipos de datos complejos en Python

Python nos ofrece estructuras versátiles para almacenar múltiples datos. Las listas, tuplas y diccionarios son herramientas clave en cualquier programa:

- **Listas**: Son secuencias ordenadas, mutables y heterogéneas. Podemos recorrerlas con bucles, modificar sus elementos y aplicar métodos como append(), insert(), remove(), pop(), sort() y reverse(). Esto las hace ideales para manejar colecciones dinámicas de datos como registros de usuarios, tareas o resultados.

```python
frutas = ["manzana", "banana", "naranja"]
frutas.append("pera")
print(frutas[2])  # naranja
```

- **Tuplas**: Muy similares a las listas, pero inmutables. Una vez creadas, no se pueden modificar, lo cual es útil para proteger datos que no deben alterarse. Son más eficientes en memoria y se usan también como claves en diccionarios.

```
coordenadas = (10, 20)
print(coordenadas[0])  # 10
```

- **Diccionarios**: Almacenan información en pares clave:valor. Nos permiten representar entidades (por ejemplo, un estudiante con nombre, edad y materias). Se accede a los valores por sus claves y permiten una estructura más expresiva y cercana al formato JSON.

```
alumno = {"nombre": "Ana", "edad": 20, "carrera": "Datos"}
print(alumno["nombre"])
alumno["edad"] = 21
```

Estas estructuras son imprescindibles para organizar, procesar y manipular datos de forma profesional.

Actividad 1 – Manipulación de Listas, Tuplas y Diccionarios

Consigna: Crear un pequeño sistema que almacene datos de

estudiantes. Cada estudiante debe tener: nombre, edad y una lista de materias.

- Crear una tupla con 3 estudiantes (nombre, edad, materias).
- Convertir esa tupla en una lista para agregar un nuevo estudiante.
- Transformar la estructura en un diccionario donde cada clave sea el nombre del estudiante y el valor sus datos.
- Imprimir los datos de cada estudiante.

Git y GitHub

Aprender a usar Git y GitHub es fundamental para el trabajo colaborativo y profesional:

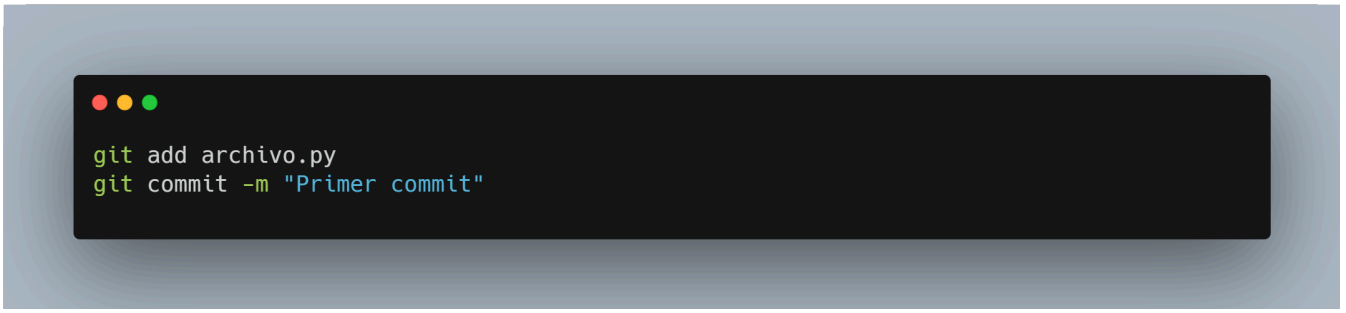
- **Git** nos permite llevar un registro detallado del desarrollo de un proyecto. Podemos crear ramas para experimentar, hacer commits para guardar cambios y retroceder en caso de errores. Además, podemos revisar el historial, colaborar y fusionar trabajo de otros integrantes.

- **GitHub** amplía las capacidades de Git permitiendo el almacenamiento remoto y el trabajo en equipo a través de Internet. Podemos subir nuestro código, colaborar mediante pull requests, documentar con archivos README, y organizar el proyecto con issues y proyectos.

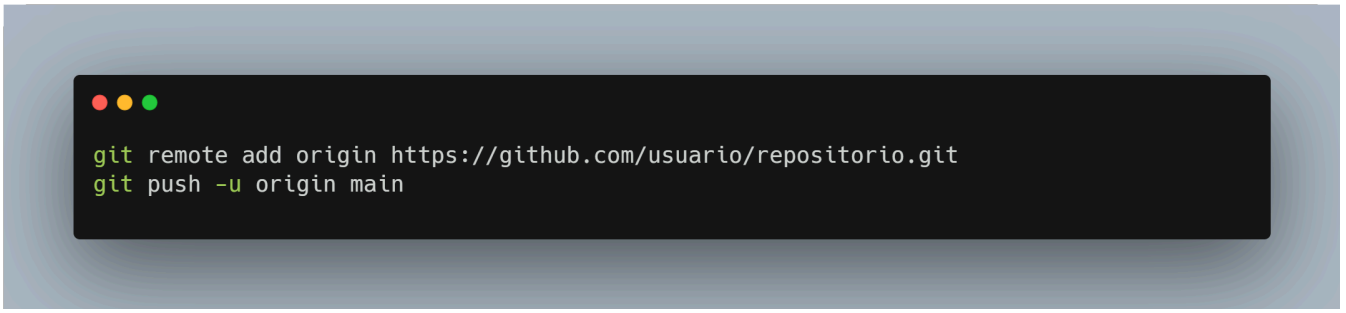
En conjunto, Git y GitHub promueven buenas prácticas de trabajo, transparencia y eficiencia en el desarrollo de software.

- Inicializar un repositorio:

- Crear un commit:



- Subir a GitHub:



Programación Orientada a Objetos (POO)

La POO nos brinda una forma poderosa de organizar y modelar programas complejos:

- A través de **clases** y **objetos**, podemos representar conceptos del mundo

real: una clase Auto, por ejemplo, permite instanciar objetos como mi_auto o tu_auto, cada uno con sus atributos (marca, modelo) y métodos (acelerar, frenar).

```python
class Persona:
    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad

    def saludar(self):
        print(f"Hola, soy {self.nombre} y tengo {self.edad} años")

ana = Persona("Ana", 30)
ana.saludar()
```

- El **encapsulamiento** nos permite proteger atributos sensibles mediante métodos de acceso (getters/setters), mejorando la seguridad y claridad del código.
- La **herencia** facilita la reutilización de código al permitir que una clase derive de otra: por ejemplo, una clase Vehículo puede ser la base de Auto, Moto o Camión.

```python
class Estudiante(Persona):
    def estudiar(self):
        print("Estoy estudiando...")

juan = Estudiante("Juan", 22)
juan.estudiar()
```

- El **polimorfismo** nos permite tratar objetos distintos con una misma interfaz. Por ejemplo, distintos tipos de animales pueden tener el método hablar(), pero cada uno lo implementa de forma distinta.

La POO es esencial para construir software escalable, reutilizable y mantenible.

Actividad 2 – POO: Modelando Vehículos

Crear una clase Vehículo con atributos marca, modelo, año. Agregar un método mostrar_info().

Luego crear una subclase Auto que agregue el atributo puertas y sobreescriba el método para mostrar también ese dato.

HTML, CSS y JavaScript

Estas tres tecnologías permiten construir la interfaz visual e interactiva de las aplicaciones web:

- **HTML**: Proporciona la estructura del sitio web. Usamos etiquetas para definir encabezados, párrafos, enlaces, imágenes, formularios, listas, etc. Todo contenido visible en el navegador parte de un documento HTML.

```html
<!DOCTYPE html>
<html>
<head><title>Mi página</title></head>
<body>
  <h1>Bienvenidos</h1>
  <p>Esto es una página web.</p>
</body>
</html>
```

- **CSS**: Nos permite aplicar estilos. Controlamos el color, fuente, posición, espaciado y estética general. Podemos definir estilos internos, en línea o externos, y aplicar clases e identificadores para segmentar estilos según el contenido.

```css
body {
  background-color: #f2f2f2;
  font-family: Arial;
}
```

- **JavaScript**: Agrega lógica al navegador. Detecta eventos del usuario (clic, teclado), permite validaciones en formularios, genera efectos dinámicos, y manipula el DOM para cambiar el contenido en tiempo real.

```html
<button onclick="alert('¡Hola!')">Haz clic</button>
```

Juntas, estas tecnologías conforman la base del desarrollo frontend moderno.

Actividad 3 – Crear una Página HTML con Estilo y Script

Crear un archivo HTML con un botón. Estilizarlo con CSS y al hacer clic, mostrar un mensaje con JavaScript.

Flask y Jinja2

```python
from flask import Flask
app = Flask(__name__)

@app.route("/")
def inicio():
    return "Hola desde Flask"

app.run(debug=True)
```

Flask es una herramienta fundamental para unir Python con la web. Es un micro-framework que nos permite desarrollar aplicaciones web funcionales desde cero:

```python
from flask import render_template

@app.route("/saludo")
def saludo():
    return render_template("saludo.html", nombre="Carlos")
```

- Creamos **rutas** que definen qué función ejecutar ante cada URL.
- Podemos **procesar formularios**, validar datos y responder al usuario.
- Con **Jinja2**, incrustamos variables de Python directamente en nuestros HTML. También nos permite condicionales, bucles, inclusión de plantillas, y más.
- Flask permite integrar bases de datos, gestionar sesiones.

```html
<h1>Hola {{ nombre }}</h1>
```

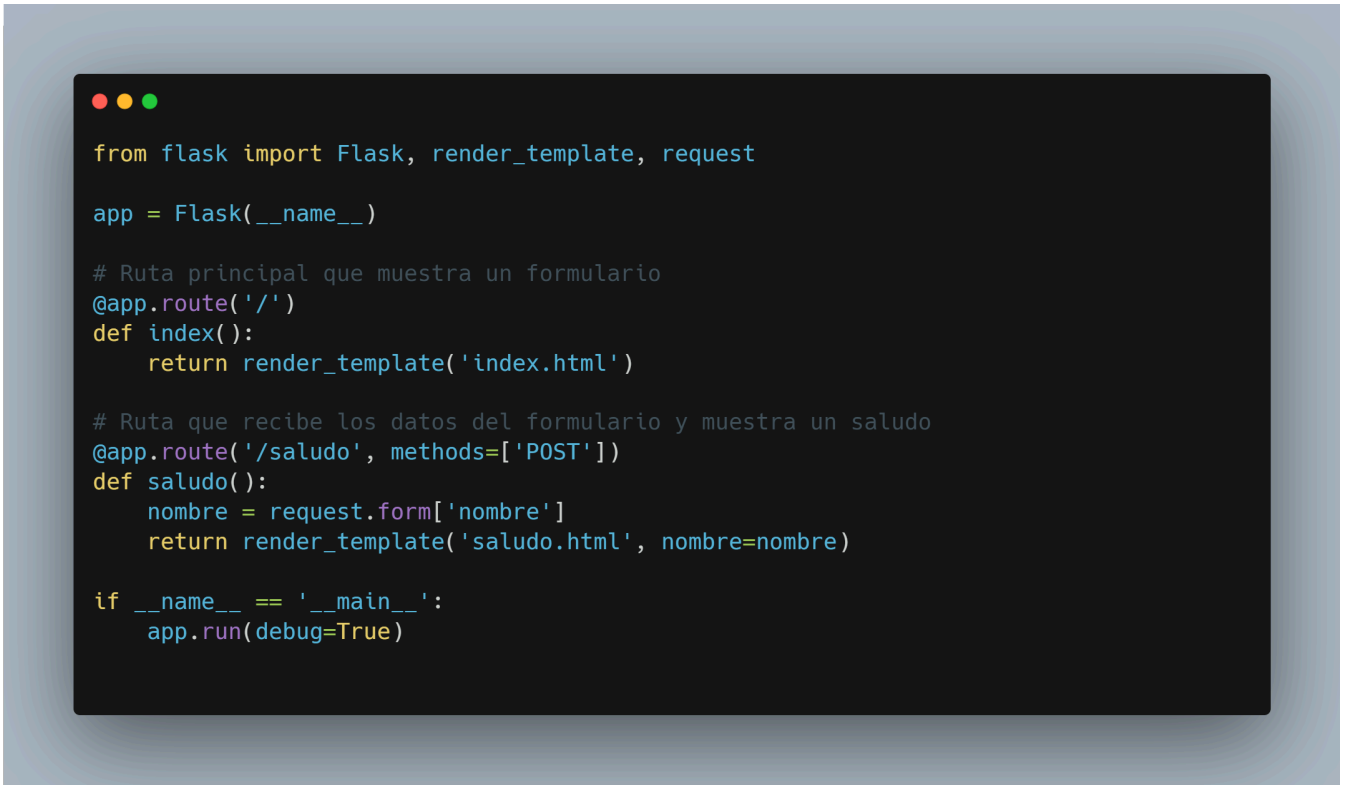
Gracias a su flexibilidad, es ideal para aprender cómo funciona una aplicación web en el lado del servidor.

Ejemplo integrado: Aplicación simple en Flask en Python

A continuación, presentamos una aplicación web sencilla desarrollada con Flask, que sintetiza varios de los conocimientos adquiridos hasta ahora:

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

- Se define una **ruta principal** (/) que muestra un formulario HTML para ingresar un nombre.



```python
from flask import Flask, render_template, request

app = Flask(__name__)

# Ruta principal que muestra un formulario
@app.route('/')
def index():
    return render_template('index.html')

# Ruta que recibe los datos del formulario y muestra un saludo
@app.route('/saludo', methods=['POST'])
def saludo():
    nombre = request.form['nombre']
    return render_template('saludo.html', nombre=nombre)

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

- Se utiliza el método **POST** para enviar los datos del formulario a una segunda ruta (/saludo).

```html
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Formulario de saludo</title>
</head>
<body>
    <h1>Bienvenido/a</h1>
    <form action="/saludo" method="post">
        <label for="nombre">Ingresá tu nombre:</label>
        <input type="text" id="nombre" name="nombre" required>
        <button type="submit">Enviar</button>
    </form>
</body>
</html>
```

- En esta ruta, Flask recibe los datos del formulario y los **procesa desde el backend** usando request.form.

```html
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Saludo</title>
</head>
<body>
    <h1>¡Hola {{ nombre }}!</h1>
    <a href="/">Volver al inicio</a>
</body>
</html>
```

- Luego, se utiliza render_template junto con **Jinja2** para mostrar una página personalizada con el nombre ingresado por el usuario.

Este ejemplo ilustra cómo conectar HTML con Python, cómo gestionar formularios, y cómo generar contenido dinámico en una página web.

Actividad 4 – Aplicación web simple con Flask y Jinja2

Crear una aplicación con Flask que reciba un nombre por formulario y devuelva un saludo personalizado usando una plantilla HTML.

Cierre

En esta séptima clase hemos realizado un recorrido integrador por los principales conceptos que sustentan el desarrollo moderno de software, articulando conocimientos adquiridos desde distintos frentes. Empezamos repasando las estructuras de datos complejas en Python (listas, tuplas, diccionarios), que nos permiten almacenar y manipular grandes volúmenes de información con eficiencia.

Avanzamos luego sobre el uso profesional de herramientas de control de versiones con Git y GitHub, fundamentales para el trabajo en equipo y la trazabilidad de proyectos.

Nos adentramos en la Programación Orientada a Objetos (POO), un paradigma clave que nos permite organizar nuestro código de forma escalable, reutilizable y alineada con problemas del mundo real. Complementamos este enfoque incorporando habilidades para construir interfaces web interactivas usando HTML, CSS y JavaScript, que constituyen la base del desarrollo del lado del cliente.

Finalmente, dimos los primeros pasos con Flask, el micro-framework de Python, y su integración con Jinja2, para construir aplicaciones web dinámicas. Esta etapa marca la conexión entre el backend (donde vive la lógica del sistema) y el frontend (lo que el usuario ve e interactúa), consolidando una visión completa del desarrollo de software.

Este repaso no sólo busca reforzar los conocimientos adquiridos, sino también prepararnos para una nueva etapa más avanzada. En la próxima clase, nos enfocaremos en el trabajo con bases de datos relacionales, explorando cómo implementar persistencia de datos, manejar modelos de información y realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) desde nuestras aplicaciones web.

Estamos cada vez más cerca de poder desarrollar sistemas web funcionales, robustos y conectados a datos reales, que pueden ser desplegados en internet y utilizados por personas y organizaciones.

Material complementario sugerido

Beazley, D. M. (2021). *Python. Guía de referencia rápida* (2.ª ed.). O'Reilly Media.

Pilgrim, M. (2010). *Dive into Python 3*. Apress.

https://histo.ucsf.edu/BMS270/diveintopython3-r802.pdf

Chacon, S., & Straub, B. (2014). *Pro Git* (2.ª ed.). Apress.

https://git-scm.com/book/en/v2

Freeman, E., & Robson, E. (2014). *Head First HTML and CSS*. O'Reilly Media.

Grinberg, M. (2018). *Flask Web Development* (2.ª ed.). O'Reilly Media.