EXPLICACION DEL CODIGO DE TRABAJO FINAL DE PYTHON

¿QUÉ UTILIZAMOS?:

FRAMEWORK

Un framework es un conjunto de herramientas y reglas que proporciona una estructura para el desarrollo de software. Está diseñado para facilitar y agilizar el proceso de construcción de aplicaciones al proporcionar una base sobre la cual los desarrolladores pueden construir.

Los frameworks proporcionan un esqueleto organizativo y arquitectónico para la aplicación. Ahorra tiempo y fomenta mejores prácticas. En este proyecto utilizamos Flask y Bootstrap.

FLASK

Flask es un framework ligero de desarrollo web para Python. Está diseñado para ser simple y fácil de usar, permitiendo a los desarrolladores crear aplicaciones web rápidamente con un mínimo esfuerzo y con una curva de aprendizaje baja.

BOOTSTRAP

Bootstrap es un marco de diseño (framework) de código abierto que se utiliza para el desarrollo de sitios web y aplicaciones móviles. Fue creado por Twitter y es mantenido actualmente por un grupo de desarrolladores individuales y la comunidad en general.

Bootstrap proporciona un conjunto de herramientas y estilos CSS predefinidos que facilitan la creación de interfaces de usuario consistentes y atractivas. Incluye una combinación de HTML, CSS y JavaScript para ayudar a los desarrolladores a diseñar rápidamente páginas web responsivas y móviles.

Instalamos la extensión de Bootstrap en VSC, y usamos la documentación para poder crear los archivo html.

BIBLIOTECAS DE PYTHON

Las bibliotecas de Python son conjuntos de módulos y funciones predefinidos que facilitan la realización de tareas específicas. Estas bibliotecas extienden la funcionalidad del lenguaje Python, permitiendo a los desarrolladores utilizar herramientas ya creadas para diversas aplicaciones.

Las bibliotecas de Python son fundamentales: para reutilizar código, hacer código más eficiente, seguir estándares de desarrollo, asegurar compatibilidad, optimizar rendimiento, etc.

PANDA

Pandas es una biblioteca de Python que proporciona estructuras de datos de alto rendimiento y fáciles de usar, así como herramientas de análisis de datos. Su nombre se deriva de "Panel Data" (datos en paneles), que es un término económico para conjuntos de datos multidimensionales.

Pandas es una herramienta fundamental en el ecosistema de Python para el análisis de datos y la manipulación de conjuntos de datos. Su capacidad para manejar datos estructurados y su integración con otras bibliotecas, como NumPy y Matplotlib, la convierten en una elección popular entre los científicos de datos y analistas.

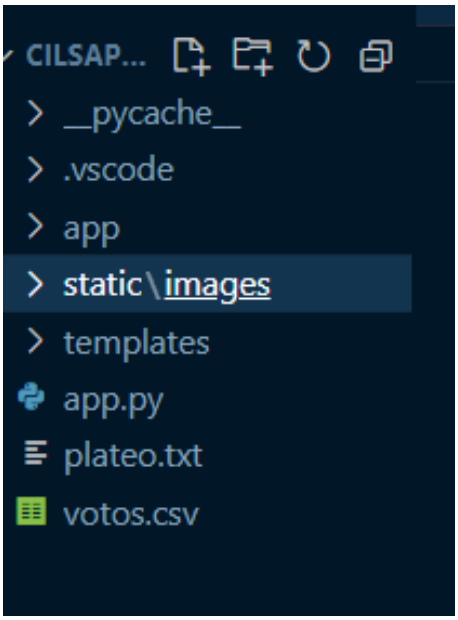
ARCHIVOS CSV

Un archivo CSV (Comma-Separated Values) es un tipo de archivo de texto plano que se utiliza para almacenar datos en forma de tabla, donde cada línea del archivo representa una fila de la tabla y los valores de cada

columna están separados por comas u otro delimitador. Los archivos CSV son ampliamente utilizados para intercambiar datos entre diferentes aplicaciones, ya que son fáciles de leer y escribir, y son compatibles con muchas herramientas y plataformas.

EXPLICACIÓN DEL PROYECTO POR SEGMENTOS

El proyecto está organizado en formato de página web para que la presentación del proyecto sea dinámica y divertida.

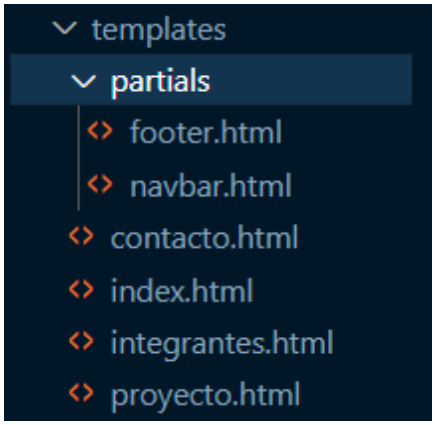


cilsaProyecto se encuentra consta con 5 carpetas dentro. __pycahce__, .vscode y app fueron creadas con extensiones Flask, cache e init py por defecto para el funcionamiento. Como el trabajo inició haciendo otros elementos, puede que queden resto del proyecto original instalado.

La carpeta *templates* fue creada para que flask pueda buscar los archivos html, la carpeta static para tener los elementos estáticos de la página, pero como se terminó realizando todo con bootstrap se borró la carpeta y el archivo style.css.

Se ven dentro del archivo raíz los archivos ***app.py*** (que es donde se encuentra el desarrollo del proyecto y las rutas), el archivo votos.csv (que fue parte del ejercicio y entregado por el docente), y finalmente el archivo planteo.txt donde está el planteo del trabajo final.

CONTENIDO DE LA CARPETA TEMPLATE



En la raíz de *templates* se el archivo *index.html* (que es la página principal), las subpáginas que son proyecto.html, integrantes.html, contacto.html.

HTML

HTML (Hypertext Markup Language) es un lenguaje de marcado utilizado para la creación y estructuración de contenido en páginas web. Consiste en una serie de etiquetas que definen diferentes elementos dentro de una página, como encabezados, párrafos, enlaces, imágenes y más. Estas etiquetas permiten al navegador interpretar y mostrar el contenido de manera coherente y estructurada. HTML es esencial para la creación de páginas web y proporciona la base para su diseño y funcionalidad.

La carpeta *partials* contiene el *navbar* y *footer* del proyecto.

APP.PY

El archivo *app.py* es el entry point del programa, donde se desarrollan las rutas y las funciones del trabajo final del grupo 1.

Para crear el proyecto tuvimos que instalar Panda y Flask desde la terminal con los comandos:

```
pip install pandas

pip install Flask
```

Luego se los importa:

```
# app.py

#1era parte importar bibliotecas y frameworks, luego ser instalados

#se importa csv para poder leer este tipo de archivo
import csv

#se importa panda para mejor gestion de la información
import pandas as pd

#de flask se importa flask para manejo de archivo, y render_template para poder mostrar en la pagina web

from flask import Flask, render_template

#cache es para el almacenamiento temporal de información, por ejemplo como cuando se cuenta la cantidad total de votantes
from flask_caching import Cache

#crea una instancia de la clase Flask que representa una aplicación web. Esta instancia, usualmente llamada app, se utiliza para configurar y ejecutar tu aplicación Flask. Puedes definir rutas, vistas, configuraciones y otros aspectos de tu aplicación utilizando esta instancia.
app = Flask(__name__)

#en esta linea usando cache se usa para que la información manipulada no realice un almacenamiento real, sino uno transito. El real en este proyecto se guarda en el archivo gonzalez.txt
cache = Cache(app, config={'CACHE_TYPE': 'null'})
```

Se utiliza cache para poder manipular el archive votos.csv:

```
#utilizar la biblioteca Pandas para leer un archivo CSV (Comma-Separated Values) llamado 'votos.csv' y cargar sus datos en un objeto DataFrame.
df = pd.read_csv('votos.csv')
```

INICIAR MANIPULACIÓN DE DATOS

Función de Cargar Archivos: es para poder acceder a la información del archivo votos.csv, y desde allí iniciar la funcionalidad del proyecto. Para ello creamos la función cargar_datos.

```
# Función para cargar los datos desde el archivo CSV
def cargar_datos(archivo):
    try:
        # Intenta abrir el archivo CSV
        with open(archivo, newline='', encoding='utf-8') as csvfile:
            # Utiliza DictReader de csv para leer el archivo CSV y crear un diccionario para cada fila
            reader = csv.DictReader(csvfile)
            # Convierte el lector a una lista de diccionarios y devuelve esa lista
            return list(reader)
    except FileNotFoundError:
        # Captura una excepción si el archivo no se encuentra
        print(f"Error: El archivo {archivo} no se encuentra.")
        #Y retorna una lista vacia
        return []
    except Exception as e:
        # Captura cualquier otra excepción y muestra un mensaje de error
        print(f"Error al cargar los datos: {e}")
        #Y retorna una lista vacia
        return []
```

Explicación detallada:

1. **with open(archivo, newline='', encoding='utf-8') as csvfile:**: Utiliza el contexto **with** para abrir el archivo CSV llamado **archivo**. **newline=''** se utiliza para manejar correctamente las líneas nuevas en

diferentes sistemas operativos, y **encoding='utf-8'** especifica la codificación del archivo.
2. **reader = csv.DictReader(csvfile)**: Crea un objeto **DictReader** que lee las filas del archivo CSV y crea un diccionario para cada fila, utilizando la primera fila como encabezados.
3. **return list(reader)**: Convierte el objeto **DictReader** a una lista de diccionarios y devuelve esa lista. Cada elemento de la lista es un diccionario que representa una fila del archivo CSV.
4. **except FileNotFoundError:**: Captura la excepción **FileNotFoundError** si el archivo no se encuentra. Imprime un mensaje de error y devuelve una lista vacía.
5. **except Exception as e:**: Captura cualquier otra excepción que pueda ocurrir durante la lectura del archivo. Imprime un mensaje de error que incluye la descripción de la excepción y devuelve una lista vacía.

Ahora sí, con acceso a la información del archivo con extensión csv podemos comenzar a resolver los puntos del poryecto.

PUNTO 1: Calcular y mostrar la cantidad de votos totales por partido

Función Calcular Votos Totales:

```
#función para calcular los votos totales. Primera parte del punto 1 del planteo
def calcular_votos_totales(datos):
    # Crear un diccionario para almacenar el recuento total de votos por partido
    votos_totales = {}

    # Iterar sobre cada persona en los datos
    for persona in datos:

        # Obtener el partido de la persona o establecer 'Sin Partido' si no hay información
        partido = persona.get('Partido', 'Sin Partido')

        # Actualizar el recuento de votos para el partido en el diccionario
        votos_totales[partido] = votos_totales.get(partido, 0) + 1

     # Imprimir el recuento total de votos por partido
```

```
    print("Votos totales por partido:", votos_totales)

    # Devolver el diccionario con el recuento total de votos
    return votos_totales
```

Explicación detallada:

1. `votos_totales = {}`: Inicializa un diccionario vacío llamado `votos_totales` que se utilizará para almacenar el recuento total de votos por partido.
2. `for persona in datos:`: Inicia un bucle `for` que itera sobre cada diccionario `persona` en la lista `datos`.
3. `partido = persona.get('Partido', 'Sin Partido')`: Obtiene el valor asociado con la clave 'Partido' en el diccionario `persona`. Si no hay información sobre el partido, establece el valor predeterminado como 'Sin Partido'.
4. `votos_totales[partido] = votos_totales.get(partido, 0) + 1`: Actualiza el recuento de votos para el partido en el diccionario `votos_totales`. Utiliza `get` para obtener el valor actual del partido y agregar 1 al recuento. Si el partido no existe en el diccionario, `get` devuelve 0 (el valor predeterminado) y se suma 1.
5. `print("Votos totales por partido:", votos_totales)`: Imprime en la consola el recuento total de votos por partido. Esto podría ser útil para propósitos de depuración, pero en una aplicación real podría eliminarse.
6. `return votos_totales`: Devuelve el diccionario `votos_totales` que contiene el recuento total de votos por partido.

FUNCIÓN CALCULAR VOTOS POR PARTIDO:

```
# Funcion para calcular votos por partido
def calcular_votos_por_partido():
     # Inicializa un diccionario para almacenar el recuento de votos por
partido
    votos_por_partido = {}

    # Itera sobre cada fila en el DataFrame df
    for _, fila in df.iterrows():
        # Obtiene el valor de la columna 'Voto' para la fila actual
        partido = fila['Voto']

        # Verifica si el partido ya está en el diccionario votos_por_partido
```

```
        if partido in votos_por_partido:
            # Si se incrementa el recuento de votos para ese partido
            votos_por_partido[partido] += 1
        else:
             # Si no, agrega el partido al diccionario con un recuento
inicial de 1
            votos_por_partido[partido] = 1
    # Devuelve el diccionario con el recuento total de votos por partido
    return votos_por_partido
```

Explicación detallada:

1. **votos_por_partido = {}**: Inicializa un diccionario vacío llamado **votos_por_partido** que se utilizará para almacenar el recuento de votos por partido.
2. **for _, fila in df.iterrows():**: Itera sobre cada fila en el DataFrame **df**. **iterrows()** es un método de Pandas que devuelve pares (índice, serie) para cada fila en el DataFrame.
3. **partido = fila['Voto']**: Obtiene el valor de la columna 'Voto' para la fila actual. Suponemos que 'Voto' es la columna que contiene la información sobre el partido votado.
4. **if partido in votos_por_partido:**: Verifica si el partido ya está en el diccionario **votos_por_partido**.
    - **Si está presente (True)**: Incrementa el recuento de votos para ese partido en 1.
    - **Si no está presente (False)**: Agrega el partido al diccionario con un recuento inicial de 1.
5. **return votos_por_partido**: Devuelve el diccionario **votos_por_partido** que contiene el recuento total de votos por partido.

Esta función es útil para obtener una visión general de cuántos votos ha recibido cada partido en los datos representados por el DataFrame **df**.

PUNTO 2: Mostrar las personas con DNI mayor a 40,000,000 y con nombre "Juan"

Para realizar el punto 2, creamos la función filtrar_personas, para poder separar las personas con DNI mayor a 40.000.000 y que se llamen "Juan".

```
# Función para filtrar personas por DNI y nombre
def filtrar_personas(datos, dni_limite, nombre):
     # Utiliza una lista  para filtrar las personas que cumplen con las condiciones datos, dni y nombre
    return [persona for persona in datos if int(persona['DNI']) > dni_limite and persona['Nombre'] == nombre]
```

Explicación detallada:

1. `datos`: Es la lista de personas que se va a filtrar. Cada persona está representada como un diccionario con claves como 'DNI' y 'Nombre'.
2. `dni_limite`: Es un límite dado para el DNI. Solo se incluirán en la lista resultante aquellas personas cuyo DNI (convertido a entero) sea mayor que este límite.
3. `nombre`: Es el nombre específico que se busca. Solo se incluirán en la lista resultante aquellas personas cuyo nombre coincida exactamente con este valor.
4. `[persona for persona in datos if int(persona['DNI']) > dni_limite and persona['Nombre'] == nombre]`:
    - `for persona in datos`: Itera sobre cada persona en la lista de datos.
    - `int(persona['DNI']) > dni_limite`: Verifica si el DNI de la persona (convertido a entero) es mayor que el límite especificado.
    - `and persona['Nombre'] == nombre`: Verifica si el nombre de la persona coincide exactamente con el nombre especificado.
    - `[persona ...]`: Construye una nueva lista que incluye solo aquellas personas que cumplen con ambas condiciones.
5. `return ...`: Devuelve la nueva lista de personas que cumplen con los criterios de filtrado.

En la línea 127, dentro de @app.route("/proyecto) se ingresan los datos de DNI > 40.000.000 y el nombre Juan:

```
 # Tarea 2: Mostrar personas con DNI mayor a 40,000,000 y nombre "Juan", Se están filtrando personas con DNI mayor a 40,000,000 y nombre "Juan" utilizando la función filtrar_personas(). El resultado se almacena en personas_juan.
    personas_juan = filtrar_personas(datos_votos, 40000000, 'Juan')
```

PUNTO 3: Guardar en un archivo de texto las personas que se apellidan "Gonzalez".

Para este ejercicio creamos la función guardar_gonzales, en ella leyendo "datos" se abre un archivo llamado gonzalez.txt (open) y se escribe (w) utilizando el esquema de codificación utf-8., las personas llamadas Gonzales.

```
# Función para guardar en un archivo de texto personas con apellido "Gonzalez". Punto 3
def guardar_gonzalez(datos):
    # Abre el archivo 'gonzalez.txt' en modo de escritura ('w') con codificación utf-8
    with open('gonzalez.txt', 'w', encoding='utf-8') as file:
        # Itera sobre cada persona en los datos
        for persona in datos:
             # Verifica si el apellido de la persona es 'Gonzalez'
            if persona['Apellido'] == 'Gonzalez':
                 # Escribe el nombre y apellido en el archivo, seguido de un salto de línea
                file.write(f"{persona['Nombre']} {persona['Apellido']}\n")
```

Explicación detallada:

1. **with open('gonzalez.txt', 'w', encoding='utf-8') as file:**: Abre el archivo 'gonzalez.txt' en modo de escritura (**'w'**). El archivo se abrirá en modo de escritura desde cero, sobrescribiendo el archivo si ya existe. Se utiliza **encoding='utf-8'** para asegurar que los caracteres especiales se manejen correctamente.
2. **for persona in datos:**: Itera sobre cada persona en la lista de datos.
3. **if persona['Apellido'] == 'Gonzalez':**: Verifica si el apellido de la persona es igual a 'Gonzalez'.

- **Si es verdadero**: Continúa con la siguiente línea.
- **Si es falso**: Salta al siguiente ciclo de iteración (ignora esta persona).

4. **file.write(f"{persona['Nombre']} {persona['Apellido']}\n")**: Escribe el nombre y apellido de la persona en el archivo 'gonzalez.txt', seguido de un salto de línea. Utiliza una f-string para formatear los nombres antes de escribirlos.
5. **return**: Como la función no tiene un **return** explícito, devuelve **None** por defecto, lo cual es común en funciones que realizan acciones de escritura o modificación directa en lugar de producir un resultado que necesite ser capturado.

En @app.route("/proyecto) en las líneas 130 y 131 se guarda el archivo:

```
# Tarea 3: Guardar en un archivo de texto las personas con apellido
"Gonzalez". Se están filtrando personas con apellido "Gonzalez" utilizando
una lista de comprensión y la condición persona['Apellido'] == 'Gonzalez'.
Luego, se llama a la función guardar_gonzalez() para guardar estas personas
en un archivo de texto.
    personas_gonzalez = [persona for persona in datos_votos if
persona['Apellido'] == 'Gonzalez']
    guardar_gonzalez(personas_gonzalez)
```

```
#se usa para almacenar temporalmente los resultados de una función y evitar
tener que calcular esos resultados nuevamente si se llaman con los mismos
parámetros.
@cache.cached(timeout=0, key_prefix='index')
```

Se renderizan al nevagador todas las funciones creadas:

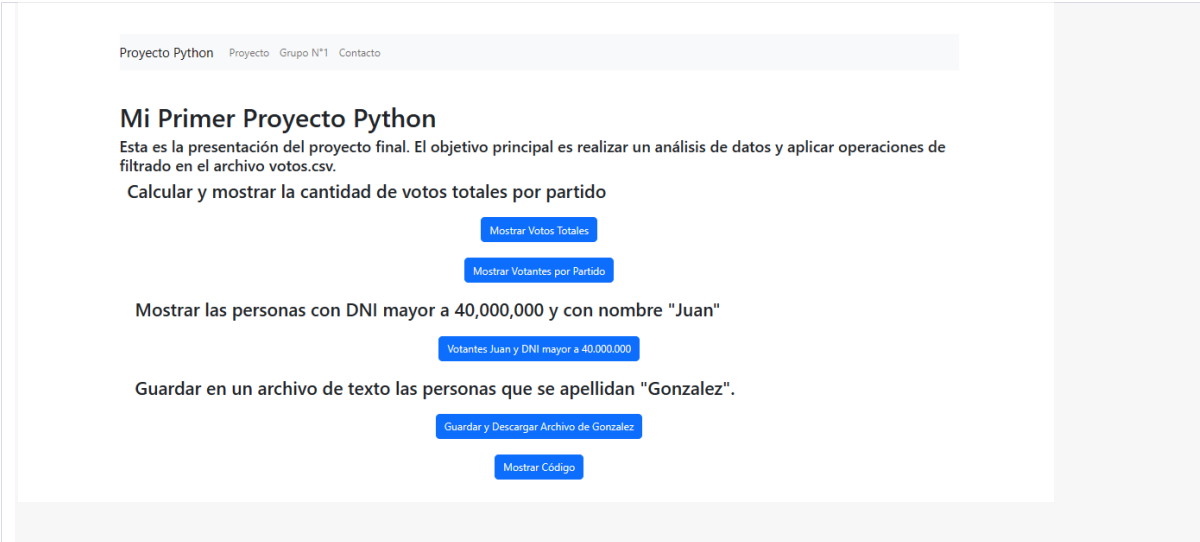
```
 # Devolver la respuesta utilizando render_template. Devuelve el resultado
renderizado utilizando la plantilla 'proyecto.html'. Los resultados de las
tareas se pasan como variables a la plantilla para que puedan ser mostrados
dinámicamente en la interfaz de usuario.
    return render_template('proyecto.html',
                           total_votos=len(datos_votos),
                           votos_por_partido=votos_por_partido,
                           personas_juan=personas_juan,
                           personas_gonzalez=personas_gonzalez)
```

Y finalmente en app.py: contamos con otras líneas para manejo de Flask y Cache.

```
#el script se ejecuta directamente y no es importado
if __name__ == "__main__":

    # Cargar los datos desde el archivo CSV al iniciar la aplicación
    archivo_csv = 'votos.csv'

    #llama cargar_datos y para eso usa archivo csv
    datos_votos = cargar_datos(archivo_csv)

    #cache es para el almacenamiento temporal de información, por ejemplo como cuando se cuenta la cantidad total de votantes
    from flask_caching import Cache

#asocia Flask con app
    cache = Cache(app, config={'CACHE_TYPE': 'null'})

# inicia el servidor de desarrollo de Flask con la función run. Con debug que es para depurar, y configura el host y el puerto
    app.run(debug=True, host='0.0.0.0', port=5050)
```

PÁGINA PROYECTO.HTML

La página proyecto.html utiliza Bootstrap como framework para estilizar la página, Flask y su componente Jinja2 para generar dinamismo a HTML.



Presenta varios botones que al hacer click se muestra el resulta de la función que se crea en app.py, incluido crear el archivo de guardar un archivo de los votantes de apellido “Gonzalez” y se descarga automáticamente.

También se puede acceder al código de app.py.

Esta web cuenta con una página principal index.html como portada, una página con los datos de los integreantes del Grupo N°1, y una con formulario de contacto.

También, desde estás páginas se ven los navbar y footer en cada página, creados en la carpeta partials, que se encuentra dentro de templates.

Esta última parte, de diseño web excede el contenido del curso, por lo que no se detallan todos sus componentes, pero que sí se dejan a disposición.

Muchas Gracias por su tiempo. Esperamos retroalimentación y corrección de nuestro primer proyecto con Python.

Atentamente, Grupo N°1

Liria Olivera

Brenda Yohena Tanoni

Daryl Sean Magro

Repositorio: `https://github.com/tanonibrenda/cilsaProyecto.git`