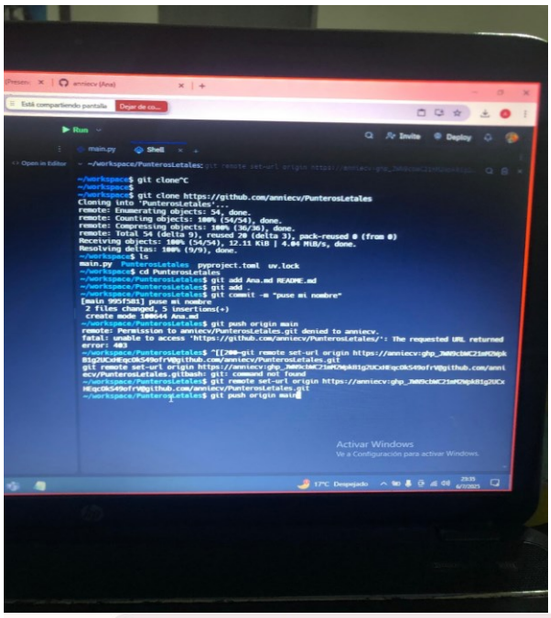
PRÁCTICO 1

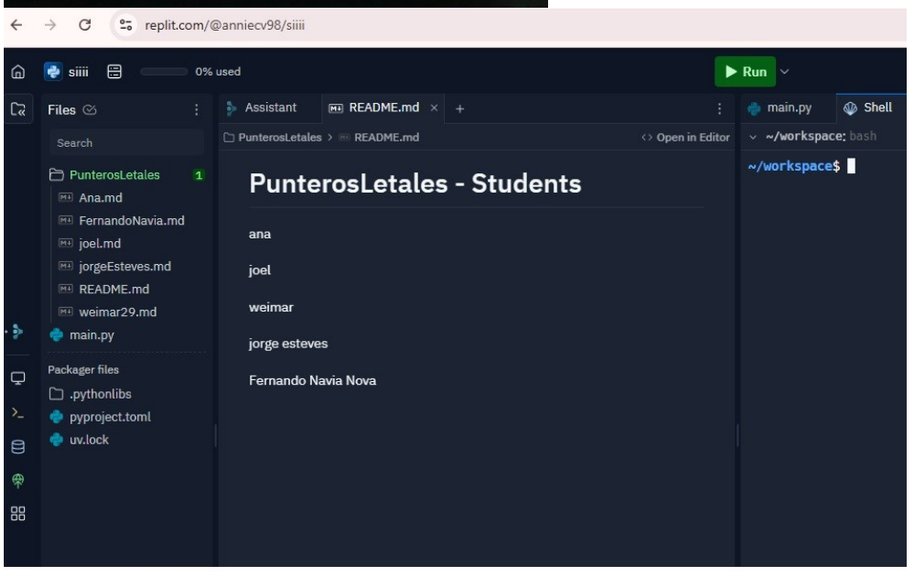
Jorge Luis Esteves Salas
Fernando Navia
Joel Dalton Montero
Ana Laura Cuellar
Weimar Valda
Leonel Eguez Camargo
ALEJANDRO HURTADO

***Dirigido por el docente**:*
JIMMY REQUENA LLORENTTY
Materia:
Programación II

ACTIVIDAD 1

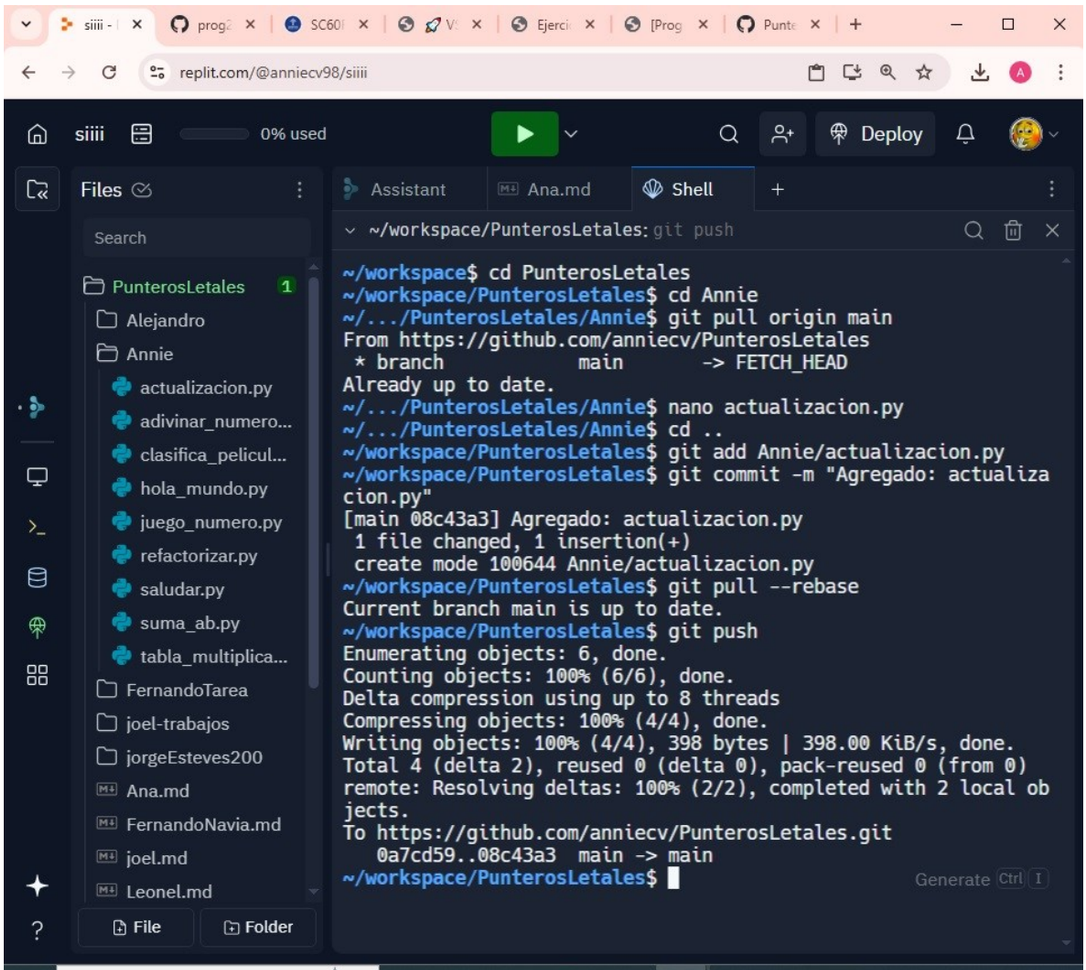
ANA CUÉLLAR
06/07/2025
PRIMER EJERCICIO REPLY

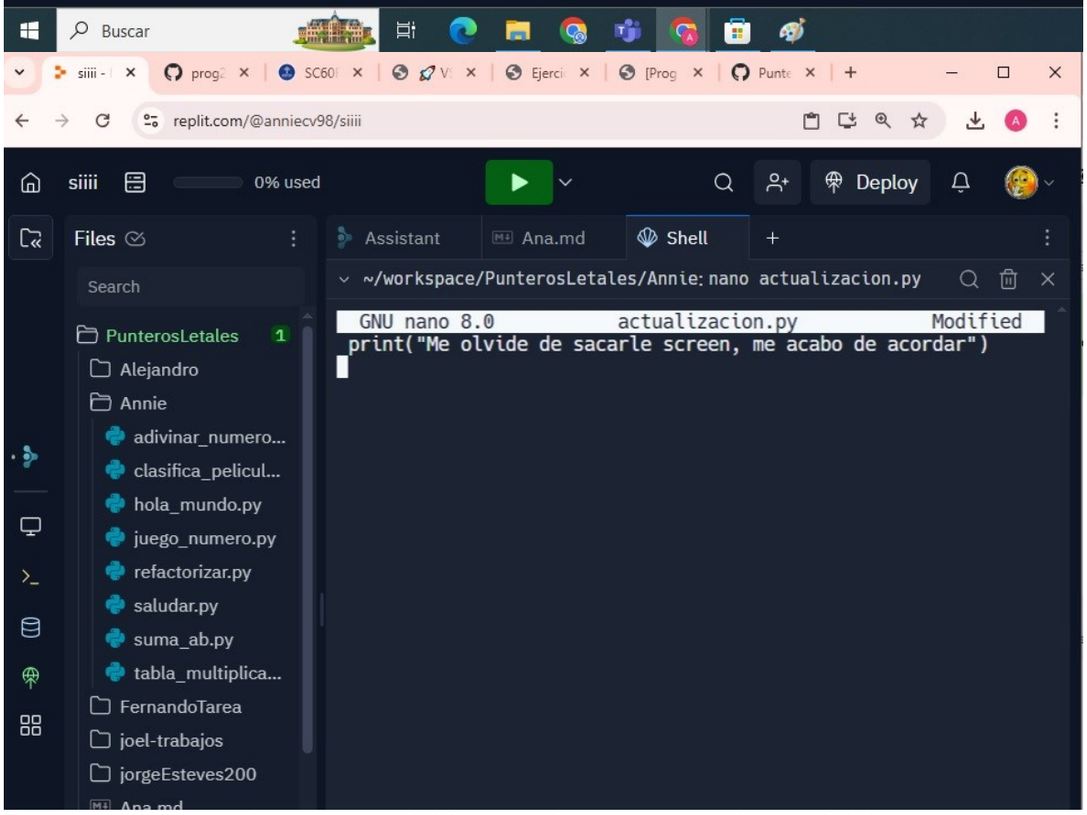




Hice un archivo con mi nombre, fue la primera vez que use comandos como git commit, git push, git add. Un compañero me guió gracias a Dios y así pude subir mi carpeta. Me pareció interesante ver como se puede colaborar de forma organizada y simultánea en un mismo proyecto, ya que cada uno puede subir sus avances sin pisar el trabajo del otro.

Fecha y hora actual: PM-07-08 17:20:58
Actualización.
Copiamos y pegamos los ejercicios del Ingeniero. Subiendo a la Carpeta que tiene mi nombre cada uno de ellos, usando Replit.





Pero resulta y resalta que me olvide sacarle screen a todo eso jajaja, entonces hice uno con print para mostrarle evidencia al Ingeniero.
Ana Laura Cuéllar – 09/07/2025 – 17:22

Investigación

1. Argumentos por Posición vs. Argumentos por Nombre (Keyword Arguments)

Cuando creamos funciones en Python, a veces queremos que esas funciones reciban datos desde afuera para hacer algo con ellos. Esos datos que le damos a

una función se llaman argumentos.

Los parámetros son los nombres que usamos dentro de la función para referirnos a esos datos. Por ejemplo, si escribimos `def saludar(nombre):`, 'nombre' es un parámetro. Cuando llamamos a la función con un valor, como `saludar("Ana")`, el valor "Ana" es un argumento.

En Python existen dos formas de pasar argumentos: por posición y por nombre.

• Argumentos por posición: Significa que los valores se pasan en el mismo orden que los parámetros fueron definidos. Por ejemplo, si una función espera (nombre, edad), entonces `funcion("Juan", 30)` asignará 'Juan' al parámetro 'nombre' y 30 a 'edad'.
• Argumentos por nombre (keyword arguments): Aquí indicamos explícitamente qué valor corresponde a cada parámetro, sin importar el orden. Por ejemplo: `funcion(edad=30, nombre="Juan")` también funcionará.

¿Cuándo usar uno u otro? Usamos argumentos por nombre cuando queremos mayor claridad o si hay muchos argumentos y queremos evitar errores con el orden. Los argumentos por posición son más comunes cuando la función tiene pocos parámetros y el orden es fácil de recordar.

2. ¿Qué es una función recursiva?

Una función recursiva es una función que se llama a sí misma. Se usa cuando un problema puede resolverse repitiendo la misma acción en partes más pequeñas.

Por ejemplo, imagina que querés contar hacia atrás desde un número hasta llegar a cero. En lugar de escribir muchas líneas, podés hacer que la función se llame a sí misma hasta llegar al final.

Lo importante en una función recursiva es que tenga una condición de salida o de corte, para evitar que se repita infinitamente. Este tipo de funciones es útil para resolver problemas como: calcular factoriales, recorrer estructuras como carpetas, o dividir un problema en partes similares más pequeñas.

3. Métodos importantes de las listas en Python

Las listas en Python son estructuras que nos permiten almacenar varios elementos dentro de una sola variable. Python ofrece métodos especiales (funciones propias de las listas) para trabajar con ellas de manera eficiente.

• append(elemento): Agrega un elemento al final de la lista. Es útil cuando no sabemos cuántos datos vamos a ingresar y queremos ir agregando elementos uno a uno.
Ejemplo:
```
mi_lista = [1, 2, 3]
mi_lista.append(4)  # Resultado: [1, 2, 3, 4]
```

• pop(): Elimina el último elemento de la lista y lo devuelve. Esto es útil si queremos quitar el último dato ingresado o manejar datos como una pila (último en entrar, primero en salir).
Ejemplo:
```
mi_lista = [10, 20, 30]
mi_lista.pop()  # Elimina 30, la lista queda [10, 20]
```

• sort(): Ordena la lista. Si son números, los ordena de menor a mayor; si son cadenas de texto, las ordena alfabéticamente.
Ejemplo:

```
mi_lista = [3, 1, 2]
mi_lista.sort()  # Resultado: [1, 2, 3]
```

4. ¿Qué son los índices negativos en listas?

Cuando accedemos a una lista con índices normales (0, 1, 2...), empezamos desde el primer elemento. Pero en Python también podemos usar índices negativos para acceder a los elementos desde el final de la lista.

- -1 es el último elemento
- -2 es el penúltimo
- -3 es el antepenúltimo

Esto es muy útil cuando no sabemos cuántos elementos tiene la lista, pero queremos acceder a los últimos sin hacer cálculos.
Ejemplo:

```
mi_lista = ['a', 'b', 'c', 'd']
print(mi_lista[-1])  # Resultado: 'd'
```

Ana Laura Cuellar – 10/07/2025 – 18:38