



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

Joel Dalton Montero Mendoza

06/07/2025

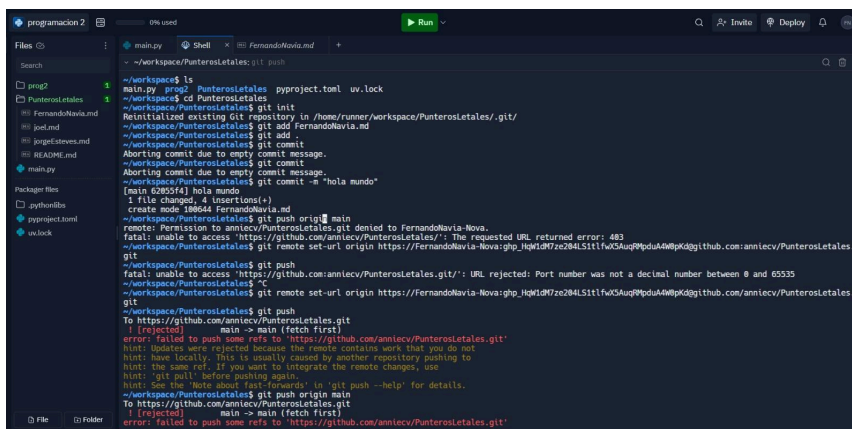
En esta práctica aprendí varios comandos de Git que antes no conocía. Por ejemplo:

git init : entendí que sirve para iniciar un repositorio en la carpeta donde estoy trabajando. Es como decirle a Git: *“aquí voy a empezar a controlar cambios.”*

git add : sirve para agregar todos los archivos modificados o nuevos para que estén listos para guardar en el historial. Me di cuenta que el punto (.) significa *“todos los archivos.”*

git commit -m "mensaje" : entendí que es como tomar una foto de los cambios y dejarle un comentario para recordar qué hice.

git push origin main : aprendí que es el comando para subir mis cambios al repositorio en Github. Aquí fue donde más me costó porque me pedía el token y me daba errores.



```
programacion 2 0m used
Files  Search  main.py  Shell  FernandoNavia.md  Run
~/workspace/PunterosIstales: git push
~/workspace$ ls
main.py  prog2  PunterosIstales  pyproject.toml  uv.lock
~/workspace$ cd PunterosIstales
~/workspace/PunterosIstales$ git init
Reinitialized existing Git repository in /home/runner/workspace/PunterosIstales/.git/
~/workspace/PunterosIstales$ git add FernandoNavia.md
~/workspace/PunterosIstales$ git add .
~/workspace/PunterosIstales$ git commit
Aborting commit due to empty commit message.
~/workspace/PunterosIstales$ git commit
Aborting commit due to empty commit message.
~/workspace/PunterosIstales$ git commit -m "hola mundo"
[main 62855f4] hola mundo
1 file changed, 4 insertions(+)
Create mode 100644 FernandoNavia.md
~/workspace/PunterosIstales$ git push origin main
remote: Permission to anniecv/PunterosIstales.git denied to FernandoNavia-Nova.
fatal: unable to access 'https://github.com/anniecv/PunterosIstales/': The requested URL returned error: 403
~/workspace/PunterosIstales$ git remote set-url origin https://FernandoNavia-Nova:ghp_HqM1Dh7zc2B4LSitfwXSaugPMduA4Mpxd3@github.com:anniecv/PunterosIstales.git
~/workspace/PunterosIstales$ git push
To https://github.com/anniecv/PunterosIstales.git
! [rejected]    main -> main (fetch first)
error: failed to push some refs to 'https://github.com/anniecv/PunterosIstales.git'
hint: Updates were rejected because the remote contains work that you do not
hint: have locally. This is usually caused by another repository pushing to
hint: the same ref. If you want to integrate the remote changes, use
hint: 'git pull'. Before pushing again,
hint: see the 'Note about fast-forwards' in 'git push --help' for details.
~/workspace/PunterosIstales$ git push origin main
To https://github.com/anniecv/PunterosIstales.git
! [rejected]    main -> main (fetch first)
error: failed to push some refs to 'https://github.com/anniecv/PunterosIstales.git'
```

Me costó al principio, sobre todo lo del token de acceso, pero gracias a mis compañeros y al material del profe, entendí cómo generarlo en Github y usarlo para poder subir mis archivos.

Lo que más me gustó fue darme cuenta que con Git se puede trabajar en equipo sin sobrescribir el trabajo de los demás, y que siempre puedo volver atrás si algo sale mal. Siento que aprendí algo muy útil para proyectos grandes.

Fecha y hora actual: PM-07-08 17:21:57

joel Dalton – 09/07/2025 – 20:16

1. Descubrí el poder de las pruebas unitarias (testfunction.py)

python

```
assert clasificacionPelicula(15) == "Puede ver peliculas clasificadas PG-13", "Error para edad 15"
```

Mi aprendizaje:

Los assert son como guardianes del código. Antes solo probaba manualmente, ¡pero ahora sé que puedo automatizar pruebas!

Entendí por qué fallaba mi función con valores negativos: no había validado if edad < 0 inicialmente .

Lección clave: *Programar ≠ adivinar. Verificar con casos extremos (como -5) es esencial.*

2. La magia de la modularización (main.py y funciones.py)

python

```
# main.py
```

```
from funciones import saludar, sumar, clasificacionPelicula, tablaMultiplicar
```

Mi error y solución:

Error inicial: Tenía todo el código en un solo archivo (¡800 líneas de caos!).

Solución: Aprendí a dividir lógica en módulos:

funciones.py = "caja de herramientas"

main.py = "interfaz de usuario"

Ventaja: Si cambio una función (ej: sumar), ¡no afecto todo el programa!

3. Mi mayor desafío: Entender el flujo

Ejemplo en clasificacionPelicula:

python

```
def clasificacionPelicula(edad):
```

```
    if edad < 0:
```

```
        return "Edad no válida" # ¡Este 'return' corta el flujo!
```

```
    elif edad < 13:
```

```
        return "Puede ver peliculas clasificadas G o PG"
```

¡Eureka moment!

Al principio no entendía por qué mi función ignoraba edad < 13 cuando ponía -5.

Descubrí: El orden de los if importa. Si ponía edad < 13 primero, ¡nunca llegaba a validar números negativos!

4. Detalles que mejoraron mi código

a) Función saludar:

```
python
# Antes
print("hola " + saludo + " bienvenido")
```

```
# Ahora sé que puedo usar f-strings:
print(f"Hola {saludo}, bienvenido a tu primer trabajo!")
```

b) Función tablaMultiplicar:

```
python
# ¡Añadí formato para mejor visualización!
print(f"{numero} x {i:>2} = {numero * i:>3}")
# Resultado: "5 x 3 = 15" (alineado a la derecha)
```

Conclusión

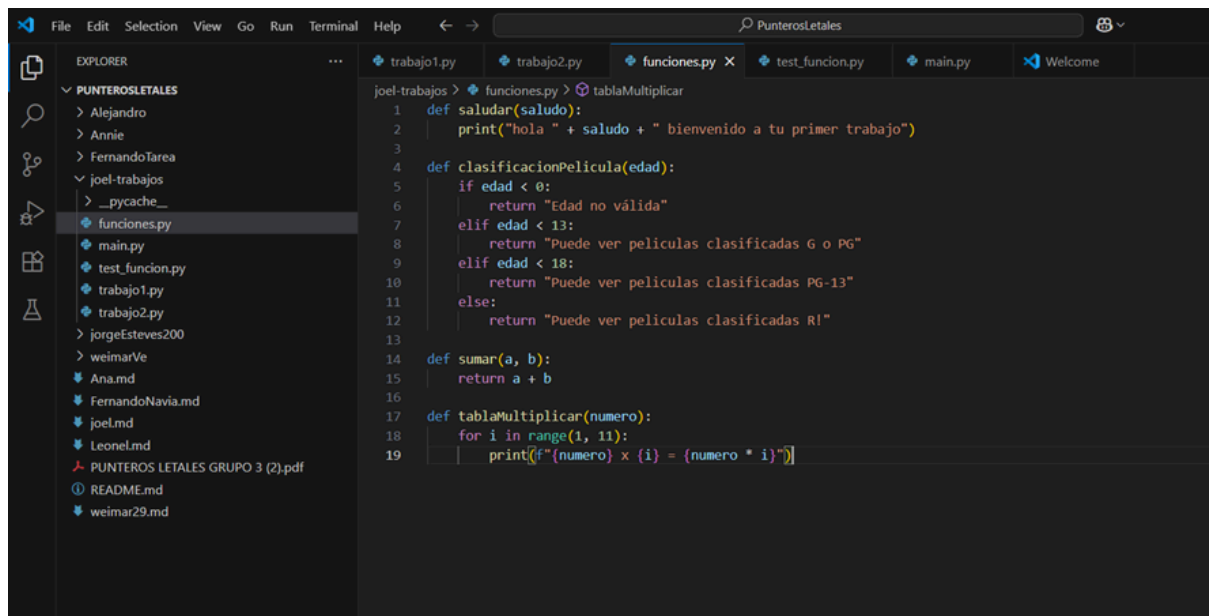
Aprendí que programar bien es como construir LEGO:

Pruebas unitarias = Instrucciones que evitan que el castillo se derrumbe .

Módulos = Separar piezas por color (¡nunca mezclar azules con rojos!).

Flujo de funciones = Saber en qué orden apilar los bloques.

¡Ahora cuando veo un assert o un import, ya no me asusto! Al contrario, sé que son superpoderes para crear código confiable.



The screenshot shows the Visual Studio Code editor interface. The Explorer panel on the left displays the project structure for 'PUNTEROSLETALES', including a 'joel-trabajos' folder with subfolders like 'Alejandro', 'Annie', 'FernandoTarea', and 'joel-trabajos'. The 'joel-trabajos' folder contains files like 'funciones.py', 'main.py', 'test_funcion.py', 'trabajo1.py', and 'trabajo2.py'. The 'main.py' file is selected and open in the editor. The code in 'main.py' imports functions from 'funciones.py' and implements a main function that prompts the user for a name, age, and a number, then calls the corresponding functions to calculate the sum and display a multiplication table.

```
1 from funciones import saludar,sumar,clasificacionPelicula,tablaMultiplicar
2 def main():
3     saludo = input("Por favor, ingresa tu nombre : ")
4     saludar(saludo)
5     print("Vamos a realizar algunas operaciones básicas.")
6
7     # Clasificación de películas
8     edad = int(input("Por favor, ingresa tu edad: "))
9     clasificacionPelicula(edad)
10
11    # Sumar dos números
12    num1 = int(input("Ingresa el primer número: "))
13    num2 = int(input("Ingresa el segundo número: "))
14    print("La suma es:", sumar(num1, num2))
15
16    # Tabla de multiplicar
17    numero = int(input("Ingresa un número para ver su tabla de multiplicar: "))
18    tablaMultiplicar(numero)
19 main()
```

The screenshot shows the Visual Studio Code editor interface with the 'test_funcion.py' file open. The code contains a series of unit tests using the 'assert' statement to verify the behavior of the functions imported from 'funciones.py'. The tests cover the 'saludar' function with various names, the 'clasificacionPelicula' function with different ages, and the 'sumar' function with various pairs of numbers. A final print statement indicates that all tests passed successfully.

```
1 from funciones import saludar,sumar,clasificacionPelicula
2 # Pruebas unitarias para la función clasificacionPelicula
3 assert saludar("Joel") == None, "Error en la función saludar"
4 assert saludar("Marta") == None, "Error en la función saludar con otro nombre"
5 assert saludar("Ana") == None, "Error en la función saludar con nombre Ana"
6 assert clasificacionPelicula(20) == "Puede ver películas clasificadas R1", "Error para edad 20"
7 assert clasificacionPelicula(15) == "Puede ver películas clasificadas PG-13", "Error para edad 15"
8 assert clasificacionPelicula(10) == "Puede ver películas clasificadas G o PG", "Error para edad 10"
9 assert clasificacionPelicula(-5) == "Edad no válida", "Error para edad -5"
10 assert sumar(5, 3) == 8, "Error en la suma de 5 y 3"
11 assert sumar(-1, 1) == 0, "Error en la suma de -1 y 1"
12 assert sumar(0, 0) == 0, "Error en la suma de 0 y 0"
13
14 print("Todas las pruebas pasaron exitosamente.")
15
16 |
```