

# PRACTICOS DE PROGRAMACION



## LOS TARANTINOS

### INTEGRANTES:

Alexander Yanarico Quispe

Jose Andres Salame Lijeron

Jeremy Said Pardo Camacho

Erwin David Pardo Hernandez

Pablo Jesus Fernandez Jaime

Cristhian David Puña Salto

Hugo Clementelli Castedo

### ASIGNATURA

Programacion II

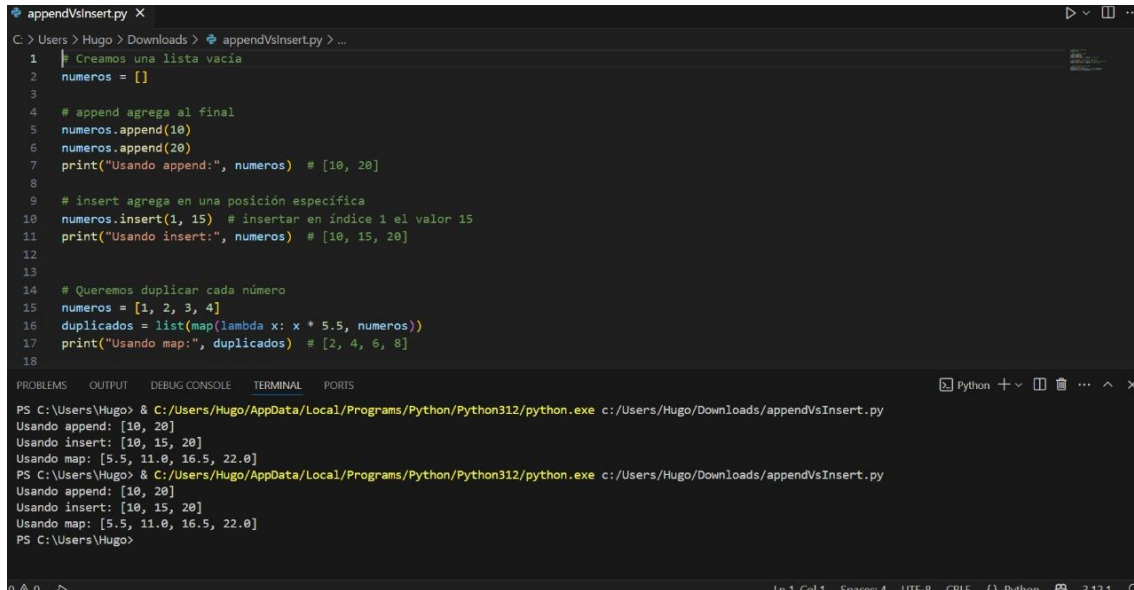
### DOCENTE

Ing. Jimmy Nataniel Requena Llorentty

# PRACTICOS DE PROGRAMACION

## EJERCICIO 1

### (appendVsinsert)



```
appendVsInsert.py X
C:\Users\Hugo> Downloads > appendVsInsert.py > ...
1  Creamos una lista vacía
2  numeros = []
3
4  # append agrega al final
5  numeros.append(10)
6  numeros.append(20)
7  print("Usando append:", numeros) # [10, 20]
8
9  # insert agrega en una posición específica
10 numeros.insert(1, 15) # insertar en índice 1 el valor 15
11 print("Usando insert:", numeros) # [10, 15, 20]
12
13
14 # Queremos duplicar cada número
15 numeros = [1, 2, 3, 4]
16 duplicados = list(map(lambda x: x * 5.5, numeros))
17 print("Usando map:", duplicados) # [2, 4, 6, 8]
18

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python + - [ ] ... ^ x

PS C:\Users\Hugo> & C:/Users/Hugo/AppData/Local/Programs/Python/Python312/python.exe c:/Users/Hugo/Downloads/appendVsInsert.py
Usando append: [10, 20]
Usando insert: [10, 15, 20]
Usando map: [5.5, 11.0, 16.5, 22.0]
PS C:\Users\Hugo> & C:/Users/Hugo/AppData/Local/Programs/Python/Python312/python.exe c:/Users/Hugo/Downloads/appendVsInsert.py
Usando append: [10, 20]
Usando insert: [10, 15, 20]
Usando map: [5.5, 11.0, 16.5, 22.0]
PS C:\Users\Hugo>
```

El programa muestra cómo agregar elementos al final o en una posición específica de una lista, y cómo transformar una lista aplicando una operación a cada elemento. Luego imprime los resultados.

### Funcionalidades

**append()** :Añade elementos al final de una lista.

**insert()** :Inserta elementos en una posición específica.

**map() + lambda** :Aplica una función a cada elemento de la lista.

**print()**:Muestra los resultados para seguimiento visual.

**list()** : Convierte el resultado del map en una lista normal.

**Lambda** : Define funciones pequeñas, ideales para operaciones rápidas y simples.

### Características

**Estructura dinámica:** Uso de listas que crecen y se modifican.

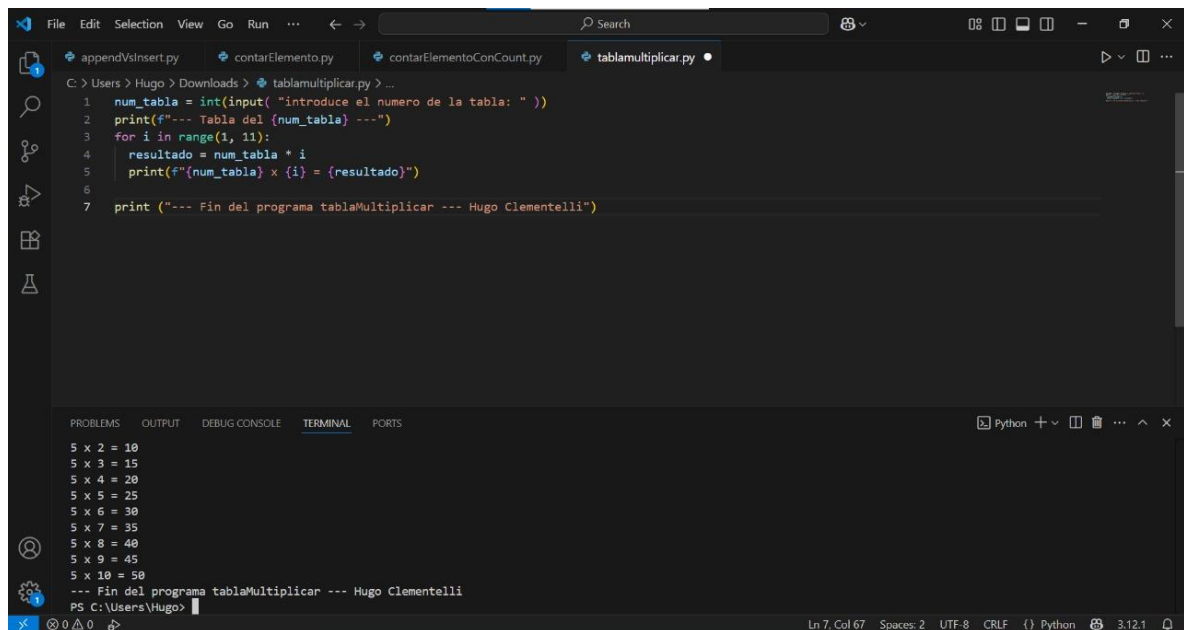
**Programación funcional:** Uso de map con funciones lambda.

**Código modular:** Cada bloque tiene una tarea específica, fácil de seguir.

**Flexibilidad:** Se puede adaptar fácilmente a otras operaciones (sumar, dividir, etc.).

## EJERCICIO 2

### tablamultiplicar



```
File Edit Selection View Go Run ... Search
appendVsInsert.py contarElemento.py contarElementoConCount.py tablaMultiplicar.py
C:\Users\Hugo\Downloads> tablaMultiplicar.py
1 num_tabla = int(input("introduce el numero de la tabla: "))
2 print(f"--- Tabla del {num_tabla} ---")
3 for i in range(1, 11):
4     resultado = num_tabla * i
5     print(f"{num_tabla} x {i} = {resultado}")
6
7 print("--- Fin del programa tablaMultiplicar --- Hugo Clementelli")

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50
--- Fin del programa tablaMultiplicar --- Hugo Clementelli
PS C:\Users\Hugo>
```

Es un programa que imprime la tabla de multiplicar de un número que el usuario ingresa.

## Funcionalidades del Programa

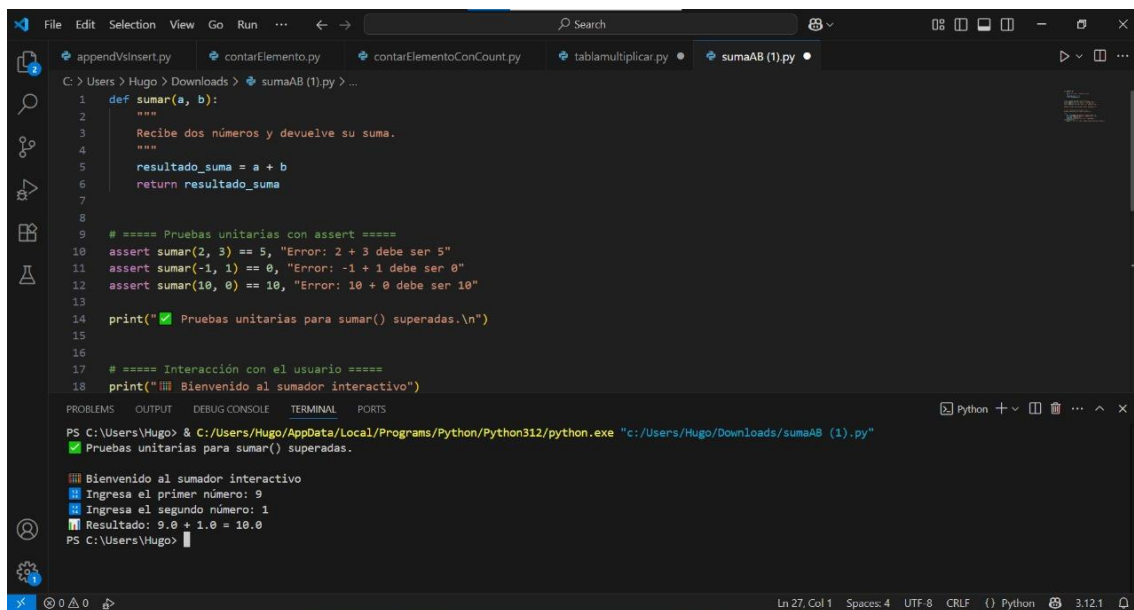
1. **Solicita entrada del usuario** Interactividad con el usuario.
2. **Muestra una cabecera personalizada** Visualización clara y organizada.
3. **Calcula la tabla de multiplicar** Cálculo automático mediante repetición.
4. **Presenta los resultados de forma ordenada** Salida clara, precisa y formateada.
5. **Finaliza con un mensaje de cierre** Señala el fin del programa y da crédito al creador.

## Características

- 1.**Entrada de datos:** Usa `input()` para pedir al usuario un número.
- 2.**Conversión de tipo:** `int()` convierte la entrada (cadena de texto) a un número entero.
- 3.**Variables:** Usa `num_tabla`, `i`, y `resultado` para almacenar datos temporalmente.
- 4.**Estructura de control:** El bucle `for` permite repetir acciones sin escribirlas 10 veces manualmente.
- 5.**Modularidad:** Aunque es un solo bloque, se puede separar en funciones para mejorar diseño.
- 6.**Interacción:** Involucra al usuario al pedirle un número, lo que le da un aspecto dinámico.

## EJERCICIO 3

## suma AB



```
1 def sumar(a, b):
2     """
3     Recibe dos números y devuelve su suma.
4     """
5     resultado_suma = a + b
6     return resultado_suma
7
8
9 # ===== Pruebas unitarias con assert =====
10 assert sumar(2, 3) == 5, "Error: 2 + 3 debe ser 5"
11 assert sumar(-1, 1) == 0, "Error: -1 + 1 debe ser 0"
12 assert sumar(10, 0) == 10, "Error: 10 + 0 debe ser 10"
13
14 print("✅ Pruebas unitarias para sumar() superadas.\n")
15
16
17 # ===== Interacción con el usuario =====
18 print("👋 Bienvenido al sumador interactivo")
19
20 while True:
21     print("\nIngresa el primer número: ")
22     a = int(input())
23     print("\nIngresa el segundo número: ")
24     b = int(input())
25     resultado = sumar(a, b)
26     print(f"Resultado: {a} + {b} = {resultado}")
27     print("\n")
```

Este es un programa de suma interactiva que define una función para sumar dos números, incluye pruebas unitarias automatizadas, y permite al usuario realizar sumas de manera interactiva.

### Funcionalidades del programa

**1. Función de suma:** Recibe dos números como parámetros, calcula la suma aritmética, retorna el resultado

**2. Sistema de pruebas automatizadas:**

- **Prueba básica:** Números positivos ( $2 + 3 = 5$ )
- **Prueba con negativos:** Número negativo y positivo ( $-1 + 1 = 0$ )
- **Prueba con cero:** Suma con elemento neutro ( $10 + 0 = 10$ )

**3. Interfaz interactiva:** Solicita dos números al usuario. Muestra el resultado formateado, usa emojis para mejor experiencia visual

**4. Validación automática:** Las pruebas assert verifican que la función trabaje correctamente, si alguna prueba falla, el programa se detiene con mensaje de error

### Características

**1. Estructura modular:**

```
def sumar(a, b):    # Función principal
    # Lógica de suma
    return resultado
```

**2. Pruebas unitarias:**

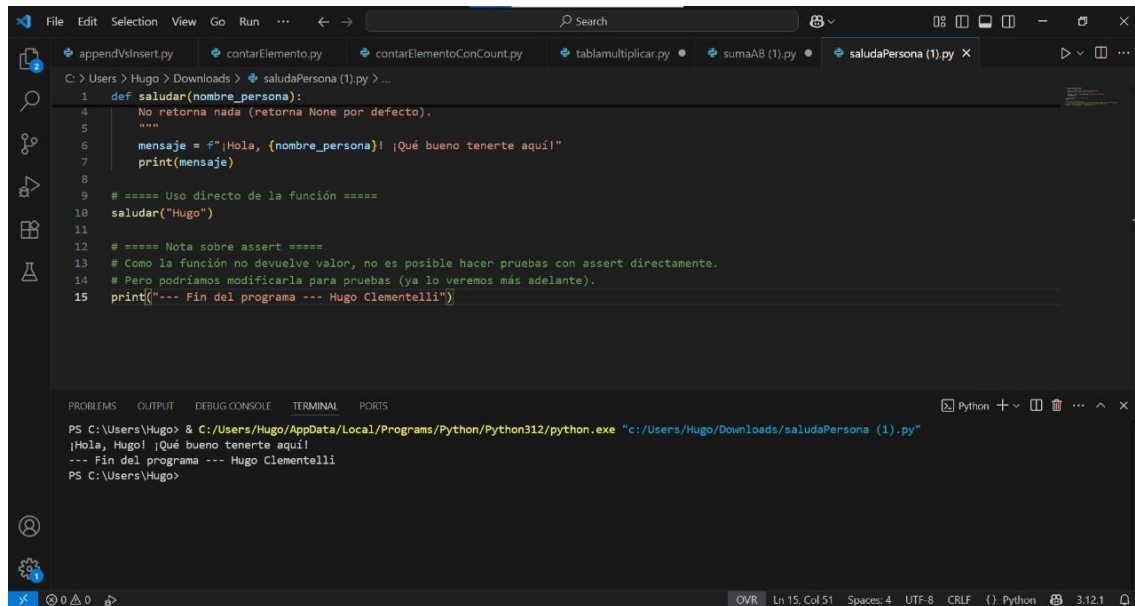
```
assert sumar(2, 3) == 5, "Error: 2 + 3 debe ser 5"
```

Usa assert para verificar funcionamiento ,cada prueba incluye mensaje de error personalizado

**3. Interacción con usuario:** Solicita entrada de datos , procesa y muestra resultados y formato visual atractivo con emojis

## EJERCICIO 4

### saludaPersona



```
File Edit Selection View Go Run ... Search
appendVsInsert.py contarElemento.py contarElementoConCount.py tablamultiplicar.py sumaAB (1).py saludaPersona (1).py x
C:\Users\Hugo> Downloads\saludaPersona (1).py
1 def saluda(nombre_persona):
2     No retorna nada (retorna None por defecto).
3     """
4     mensaje = f"Hola, {nombre_persona}! ¡Qué bueno tenerte aquí!"
5     print(mensaje)
6
7 # ===== Uso directo de la función =====
8 saluda("Hugo")
9
10 # ===== Nota sobre assert =====
11 # Como la función no devuelve valor, no es posible hacer pruebas con assert directamente.
12 # Pero podríamos modificarla para pruebas (ya lo veremos más adelante).
13 print("--- Fin del programa --- Hugo Clementelli")
14
15
PS C:\Users\Hugo> & C:/Users/Hugo/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/Hugo/Downloads/saludaPersona (1).py"
¡Hola, Hugo! ¡Qué bueno tenerte aquí!
--- Fin del programa --- Hugo Clementelli
PS C:\Users\Hugo>
```

### Función de saluda(nombre\_persona)

Define una función que recibe un parámetro nombre\_persona

Si no se pasa ningún nombre, la función retorna None (sale sin hacer nada)

Crea un mensaje de saludo combinando texto fijo con el nombre recibido y lo imprime

### Ejecución del programa

Llama a la función pasando "Hugo" como argumento

Imprime un mensaje indicando que el programa terminó

### Resultado

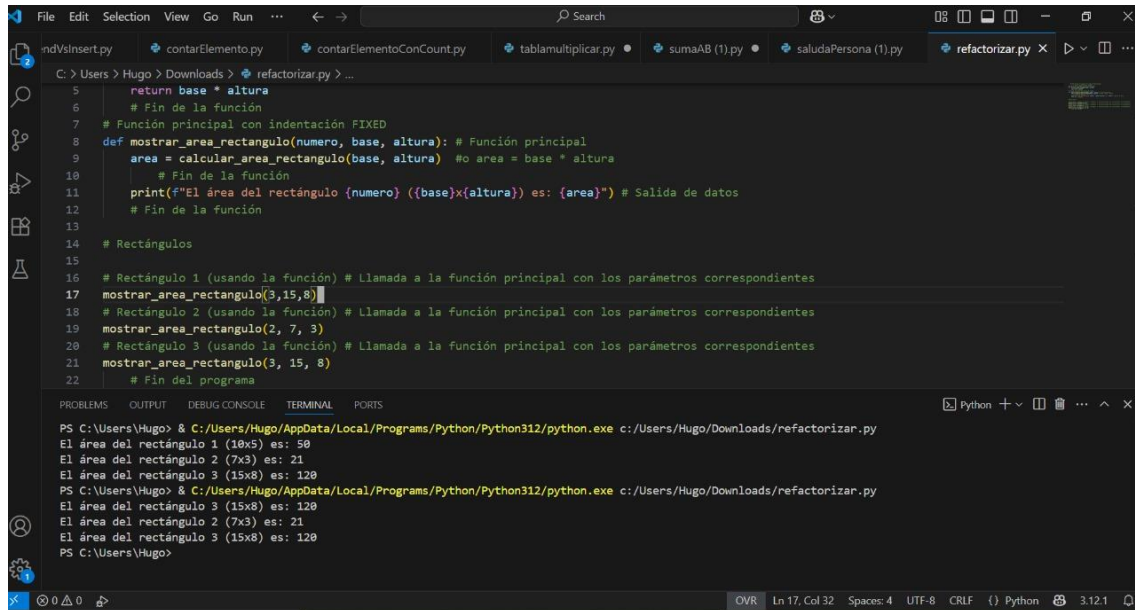
Cuando ejecutas el programa, verás:

- "¡Hola, Hugo! ¡Qué bueno tenerte aquí!"
- "--- Fin del programa --- Hugo Clementelli"

El programa básicamente saluda a una persona por su nombre y luego indica que terminó. La función tiene una validación simple para evitar errores si no se proporciona un nombre.

## EJERCICIO 5

### Refactorizar



```
5     return base * altura
6     # Fin de la función
7     # Función principal con indentación FIXED
8     def mostrar_area_rectangulo(numero, base, altura): # Función principal
9         area = calcular_area_rectangulo(base, altura) #o area = base * altura
10        # Fin de la función
11        print(f"El área del rectángulo {numero} ((base)x{altura}) es: {area}") # Salida de datos
12    # Fin de la función
13
14    # Rectángulos
15
16    # Rectángulo 1 (usando la función) # Llamada a la función principal con los parámetros correspondientes
17    mostrar_area_rectangulo(1, 10, 5)
18    # Rectángulo 2 (usando la función) # Llamada a la función principal con los parámetros correspondientes
19    mostrar_area_rectangulo(2, 7, 3)
20    # Rectángulo 3 (usando la función) # Llamada a la función principal con los parámetros correspondientes
21    mostrar_area_rectangulo(3, 15, 8)
22    # Fin del programa
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\Hugo> & C:/Users/Hugo/AppData/Local/Programs/Python/Python312/python.exe c:/Users/Hugo/Downloads/refactorizar.py  
El área del rectángulo 1 (10x5) es: 50  
El área del rectángulo 2 (7x3) es: 21  
El área del rectángulo 3 (15x8) es: 120  
PS C:\Users\Hugo> & C:/Users/Hugo/AppData/Local/Programs/Python/Python312/python.exe c:/Users/Hugo/Downloads/refactorizar.py  
El área del rectángulo 3 (15x8) es: 120  
El área del rectángulo 2 (7x3) es: 21  
El área del rectángulo 3 (15x8) es: 120  
PS C:\Users\Hugo>

Este programa calcula el área de rectángulos usando funciones. Está dividido en dos partes principales: definición de funciones y ejecución de ejemplos.

### Flujo de ejecución

1. **Definición:** Se definen las dos funciones pero no se ejecutan
2. **Ejecución:** Se ejecutan las llamadas a `mostrar_area_rectangulo()` en orden
3. **Cálculo:** Cada llamada internamente usa `calcular_area_rectangulo()` para obtener el área
4. **Salida:** Se imprime el resultado formateado en la consola

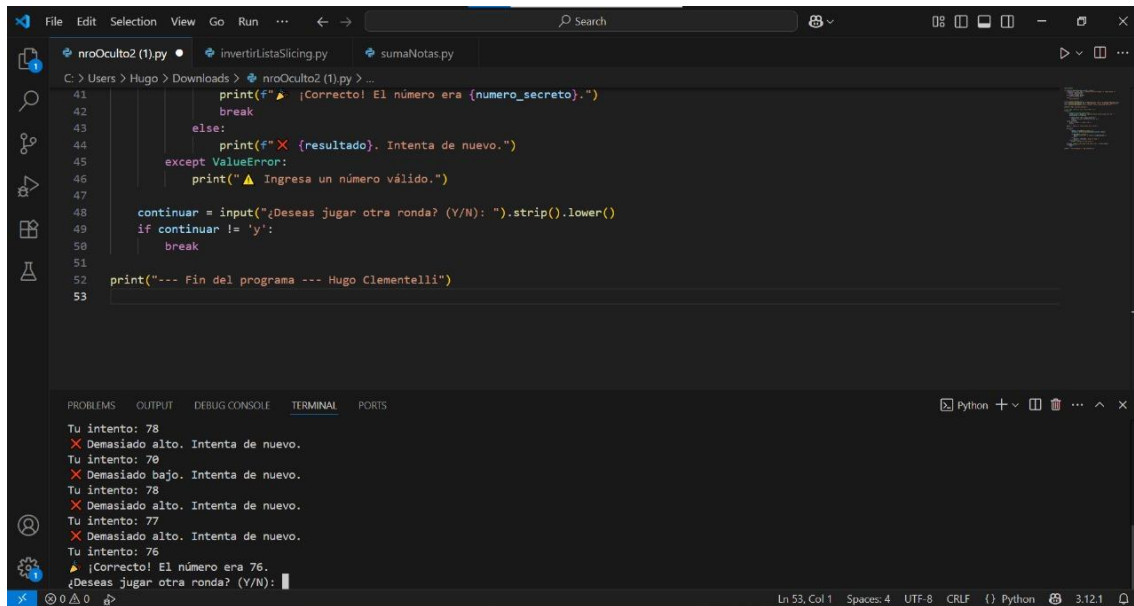
### Características

- **Modularidad:** Separa el cálculo (función simple) de la presentación (función con formato)
- **Reutilización:** La función se puede usar múltiples veces con diferentes valores
- **Legibilidad:** Usa nombres descriptivos y comentarios
- **Formato:** Utiliza f-strings para una salida clara y profesional

El programa demuestra buenas prácticas de programación al separar la lógica de cálculo de la presentación de resultados.

## EJERCICIO 6

### nroOculto



```
41     print(f"🎯 ¡Correcto! El número era {numero_secreto}.")
42     break
43     else:
44         print(f"❌ {resultado}. Intenta de nuevo.")
45     except ValueError:
46         print("⚠️ Ingresa un número válido.")
47
48     continuar = input("¿Deseas jugar otra ronda? (Y/N): ").strip().lower()
49     if continuar != 'y':
50         break
51
52 print("--- Fin del programa --- Hugo Clementelli")
53
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
Tu intento: 78
❌ Demasiado alto. Intenta de nuevo.
Tu intento: 78
❌ Demasiado bajo. Intenta de nuevo.
Tu intento: 78
❌ Demasiado alto. Intenta de nuevo.
Tu intento: 77
❌ Demasiado alto. Intenta de nuevo.
Tu intento: 76
✅ ¡Correcto! El número era 76.
¿Deseas jugar otra ronda? (Y/N):
```

Este es un **juego de adivinanza de números** (también conocido como "Adivina el número secreto"). Es un juego interactivo donde el usuario debe adivinar un número que la computadora ha elegido aleatoriamente.

## Funcionalidades

1. **Generación aleatoria** de números secretos
2. **Interacción continua** con el usuario
3. **Retroalimentación inteligente** (pistas direccionales)
4. **Validación robusta** de entradas
5. **Sistema de rondas múltiples**
6. **Manejo de errores** sin interrupciones

## Características

**Manejo de errores:** Usa try-except para capturar entradas inválidas

**Control de flujo:** Usa break para terminar bucles cuando sea necesario

**Interfaz amigable:** Usa emojis y mensajes claros

**Validación de entrada:** Procesa la respuesta del usuario con .strip() y .lower()

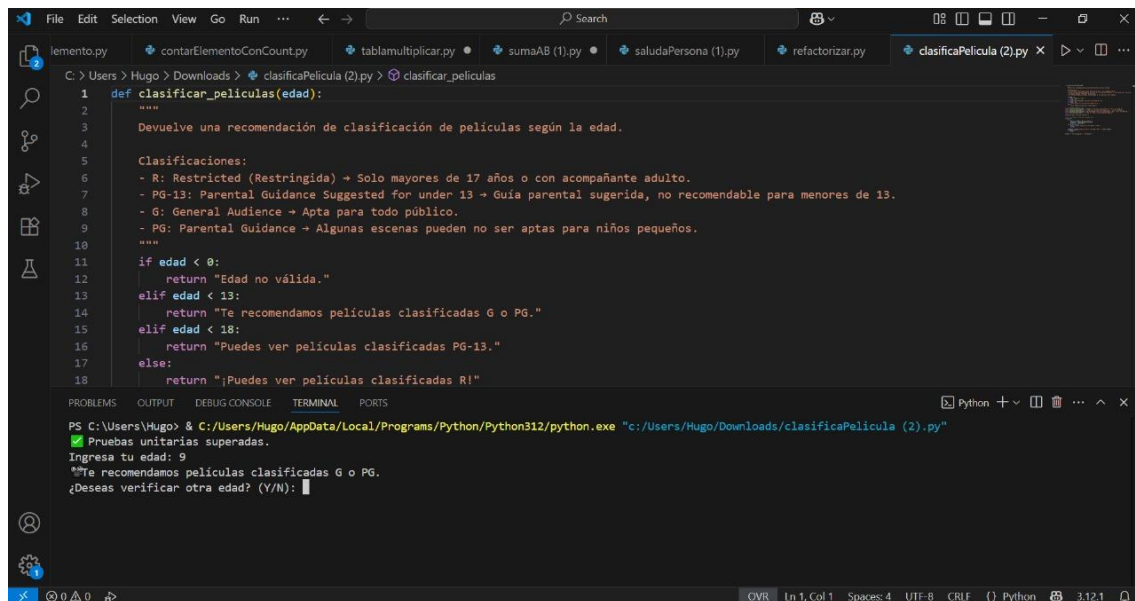
**Estructura modular:** Separa la lógica de juego de la lógica de continuación

Este fragmento muestra la parte final del juego donde se manejan los resultados y se da la opción de jugar nuevamente.

## EJERCICIO 7

**clasificaPelicula**





```
1 def clasificar_peliculas(edad):
2     """
3     Devuelve una recomendación de clasificación de películas según la edad.
4
5     Clasificaciones:
6     - R: Restricted (Restringida) + Solo mayores de 17 años o con acompañante adulto.
7     - PG-13: Parental Guidance Suggested for under 13 + Guía parental sugerida, no recomendable para menores de 13.
8     - G: General Audience + Apta para todo público.
9     - PG: Parental Guidance + Algunas escenas pueden no ser aptas para niños pequeños.
10    """
11    if edad < 0:
12        return "Edad no válida."
13    elif edad < 13:
14        return "Te recomendamos películas clasificadas G o PG."
15    elif edad < 18:
16        return "Puedes ver películas clasificadas PG-13."
17    else:
18        return "¡Puedes ver películas clasificadas R!"
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\Hugo> & C:\Users\Hugo\AppData\Local\Programs\Python\Python312\python.exe "c:/Users/Hugo/Downloads/clasificaPelicula (2).py"

✓ Pruebas unitarias superadas.

Ingresa tu edad: 9

""Te recomendamos películas clasificadas G o PG.

¿Deseas verificar otra edad? (Y/N):

Este es un sistema de clasificación de películas por edad que recomienda qué tipo de contenido puede ver una persona según su edad, basándose en el sistema de clasificación cinematográfica.

## Funcionalidades

1. **Clasificación automática** por rangos de edad
2. **Validación robusta** de entrada de datos
3. **Sistema de recomendaciones** personalizado
4. **Pruebas unitarias** para verificar funcionamiento
5. **Lógica de rangos** bien definida

## Características

### 1. Estructura condicional:

```
if edad < 0:           # Validación
elif edad < 13:        # Niños
elif edad < 18:        # Adolescentes
else:                  # Adultos
```

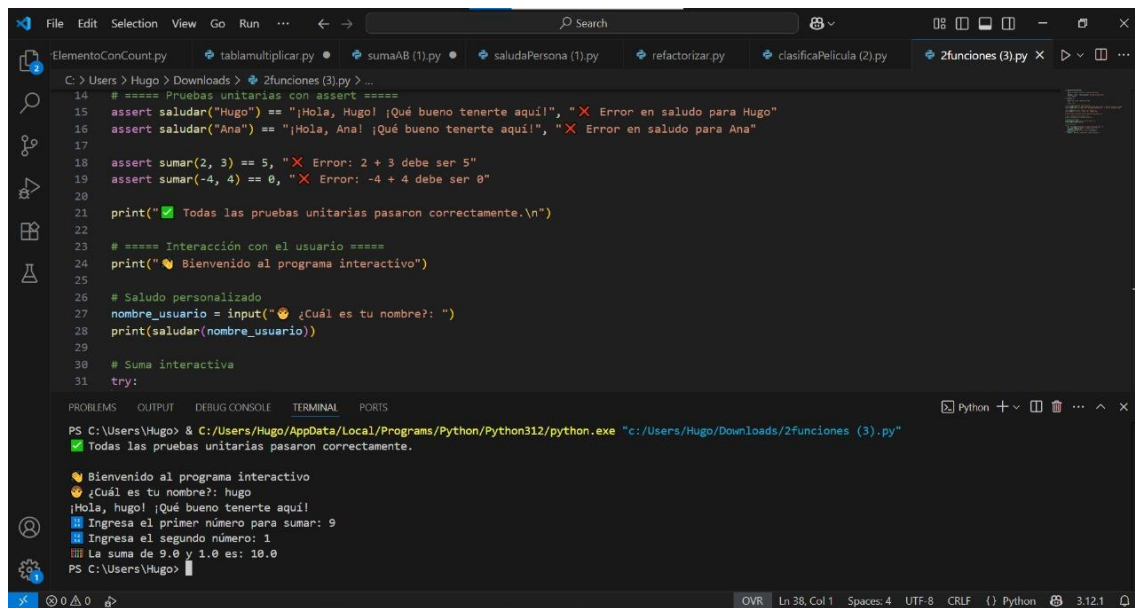
**2. Lógica de rangos de edad:** **0-12:** Contenido familiar, **13-17:** Contenido adolescente y **18+:** Contenido adulto

**3. Mensajes personalizados:** Cada rango de edad tiene una recomendación específica, usa emojis para mejor experiencia visual

## EJERCICIO 8

### 2funciones





```
14 # ===== Pruebas unitarias con assert =====
15 assert saludar("Hugo") == "¡Hola, Hugo! ¡Qué bueno tenerte aquí!", "❌ Error en saludo para Hugo"
16 assert saludar("Ana") == "¡Hola, Ana! ¡Qué bueno tenerte aquí!", "❌ Error en saludo para Ana"
17
18 assert sumar(2, 3) == 5, "❌ Error: 2 + 3 debe ser 5"
19 assert sumar(-4, 4) == 0, "❌ Error: -4 + 4 debe ser 0"
20
21 print("✅ Todas las pruebas unitarias pasaron correctamente.\n")
22
23 # ===== Interacción con el usuario =====
24 print("👋 Bienvenido al programa interactivo")
25
26 # Saludo personalizado
27 nombre_usuario = input("👤 ¿Cuál es tu nombre?: ")
28 print(saludar(nombre_usuario))
29
30 # Suma interactiva
31 try:
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\Hugo> & C:\Users\Hugo\AppData\Local\Programs\Python\Python312\python.exe "c:/Users/Hugo/Downloads/2funciones (3).py"
✅ Todas las pruebas unitarias pasaron correctamente.

👋 Bienvenido al programa interactivo
👤 ¿Cuál es tu nombre?: hugo
¡Hola, hugo! ¡Qué bueno tenerte aquí!
🔢 Ingresa el primer número para sumar: 9
🔢 Ingresa el segundo número: 1
📊 La suma de 9.0 y 1.0 es: 10.0
PS C:\Users\Hugo>
```

Este es un programa multifuncional que incluye dos funciones principales (saludo personalizado y suma), con un sistema completo de pruebas unitarias y una interfaz interactiva para el usuario.

## Funcionalidades

1. **Dos funciones principales** (saludo y suma)
2. **Sistema completo de pruebas unitarias**
3. **Validación automática de funcionamiento**
4. **Manejo de errores robusto**

## Características técnicas

### 1. Estructura modular:

*# Definición de funciones (no mostrada pero implícita)*

*# Pruebas unitarias*

*# Interacción con usuario*

### 2. Tipos de pruebas implementadas:

#### Pruebas de la función saludar():

- **Caso 1:** saludar("Hugo") → "¡Hola, Hugo! ¡Qué bueno tenerte aquí!"
- **Caso 2:** saludar("Ana") → "¡Hola, Ana! ¡Qué bueno tenerte aquí!"

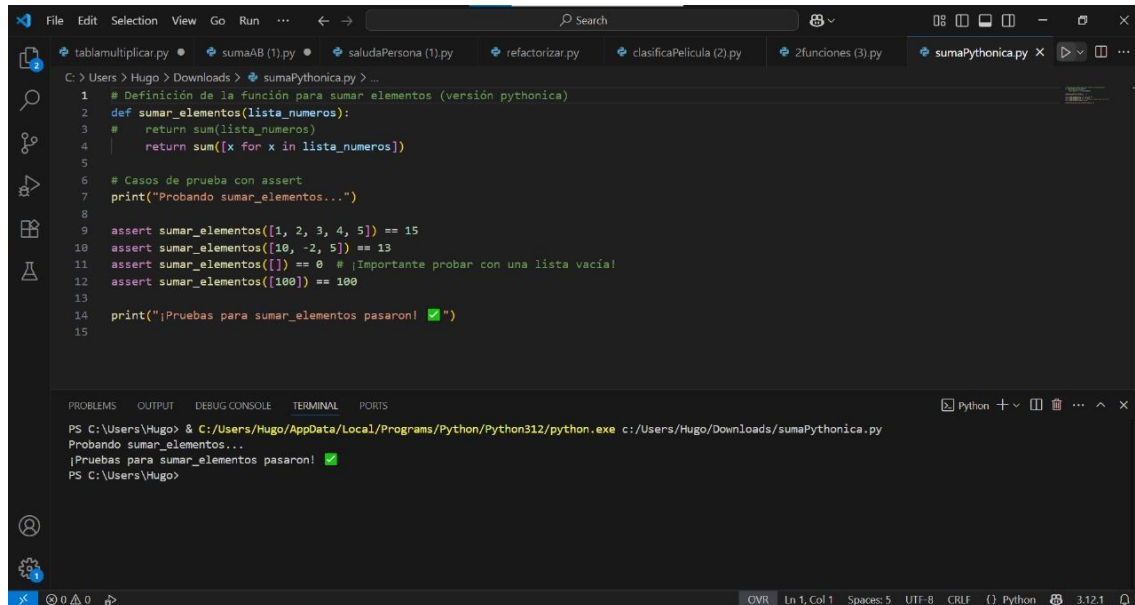
#### Pruebas de la función sumar():

- **Caso 1:** sumar(2, 3) → 5 (números positivos)
- **Caso 2:** sumar(-4, 4) → 0 (números negativos)

**3. Manejo de errores:** Usa try: para capturar errores en entrada numérica y mensajes de error personalizados en pruebas

## EJERCICIO 9

### Sumapythonica



```
File Edit Selection View Go Run ... Search
C:\Users\Hugo\Downloads> sumaPythonica.py ...
1 # Definición de la función para sumar elementos (versión pythonica)
2 def sumar_elementos(lista_numeros):
3     # return sum(lista_numeros)
4     return sum([x for x in lista_numeros])
5
6 # Casos de prueba con assert
7 print("Probando sumar_elementos...")
8
9 assert sumar_elementos([1, 2, 3, 4, 5]) == 15
10 assert sumar_elementos([10, -2, 5]) == 13
11 assert sumar_elementos([]) == 0 # ¡Importante probar con una lista vacía!
12 assert sumar_elementos([100]) == 100
13
14 print("¡Pruebas para sumar_elementos pasaron! ✅")
15

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python Python 3.12.1
PS C:\Users\Hugo> & C:/Users/Hugo/AppData/Local/Programs/Python/Python312/python.exe c:/Users/Hugo/Downloads/sumaPythonica.py
Probando sumar_elementos...
¡Pruebas para sumar_elementos pasaron! ✅
PS C:\Users\Hugo>
```

El programa verifica que la función para sumar los elementos de una lista funciona correctamente con diferentes tipos de listas (normales, vacías, con negativos, con un solo elemento). Esto es útil para asegurarse de que la función se comporta bien en todos los casos posibles.

#### 1.Función principal: sumar\_elementos(lista\_numeros)

- **Entrada:** una lista llamada lista\_numeros, que contiene números enteros o decimales.
- **Proceso:** utiliza la función incorporada sum() para calcular el total de todos los elementos. La parte [x for x in lista\_numeros] es una comprensión de listas, aunque en este caso no modifica los elementos — simplemente los copia.
- **Salida:** la suma total de los elementos.

#### 2.Pruebas con assert

Estas líneas verifican que la función esté funcionando correctamente:

- assert sumar\_elementos([1, 2, 3, 4, 5]) == 15: comprueba que la suma sea 15.
- assert sumar\_elementos([10, -2, 5]) == 13: verifica que funciona con números negativos.
- assert sumar\_elementos([]) == 0: prueba un caso especial — una lista vacía.

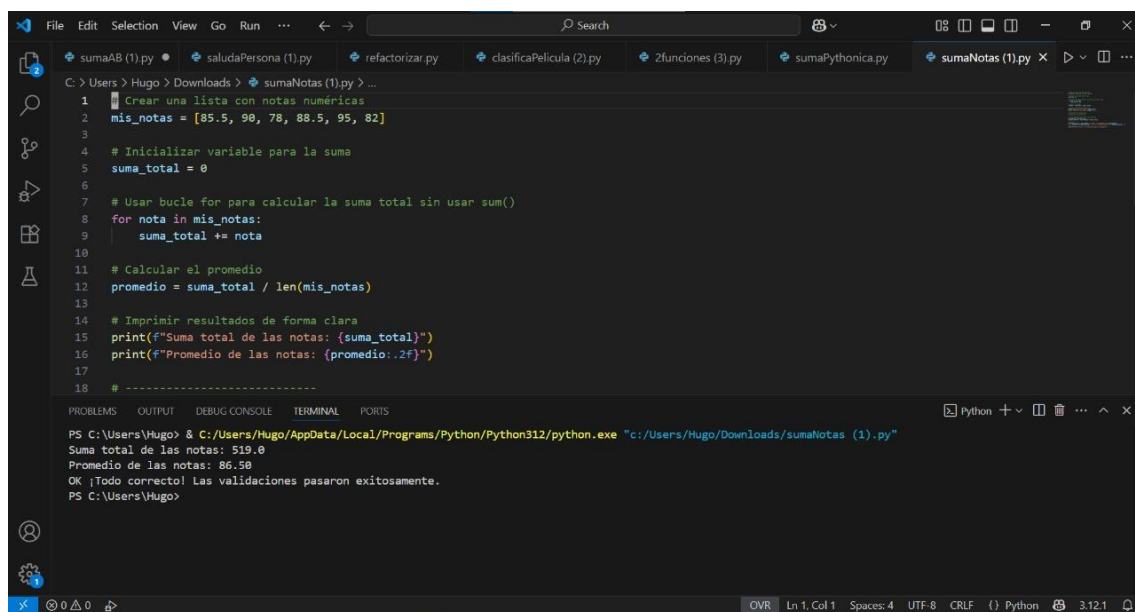
- `assert sumar_elementos([100]) == 100`: prueba con un solo elemento.

### 3.Mensajes en consola

- Muestra mensajes de progreso y éxito con `print()`, como:
  - "Probando sumar\_elementos..." antes de correr las pruebas.
  - "¡Pruebas para sumar\_elementos pasaron! ☒ " si todas las pruebas se ejecutan sin errores.

## EJERCICIO 10

### sumaNotas



```

1  # Crear una lista con notas numéricas
2  mis_notas = [85.5, 90, 78, 88.5, 95, 82]
3
4  # Inicializar variable para la suma
5  suma_total = 0
6
7  # Usar bucle for para calcular la suma total sin usar sum()
8  for nota in mis_notas:
9      suma_total += nota
10
11 # Calcular el promedio
12 promedio = suma_total / len(mis_notas)
13
14 # Imprimir resultados de forma clara
15 print(f"Suma total de las notas: {suma_total}")
16 print(f"Promedio de las notas: {promedio:.2f}")
17
18 # -----

```

```

PS C:\Users\Hugo> & C:\Users\Hugo\AppData\Local\Programs\Python\Python312\python.exe "c:/Users/Hugo/Downloads/sumaNotas (1).py"
Suma total de las notas: 519.0
Promedio de las notas: 86.50
OK ¡Todo correcto! Las validaciones pasaron exitosamente.
PS C:\Users\Hugo>

```

Este programa calcula el promedio de un conjunto de notas numéricas de manera manual, sin usar directamente la función `sum()` para la suma principal.

#### 1.Define una lista de notas:

```
mis_notas = [85.5, 90, 78, 88.5, 95, 82]
```

Es una lista con seis calificaciones.

**2.Calcula la suma manualmente con un bucle for:** Recorre la lista sumando cada nota a `suma_total`.

**3.Calcula el promedio:** Divide la suma total entre la cantidad de elementos:

```
promedio = suma_total / len(mis_notas)
```

**4.Imprime los resultados de forma clara:** Usando `print()` con formato redondeado a dos decimales:

```
print(f"Suma total de las notas: {suma_total}")
```

```
print(f"Promedio de las notas: {promedio:.2f}")
```

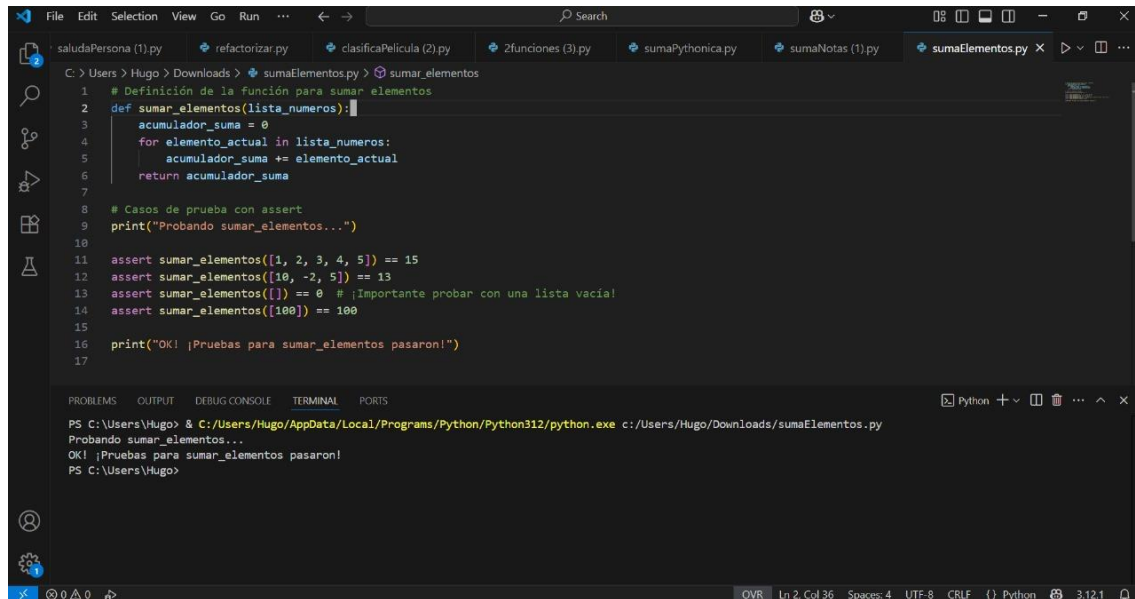
**5. Valida los cálculos con assert:** Usa sum() como referencia para verificar que el promedio y la suma calculados manualmente son correctos. Si hay errores, se muestra un mensaje personalizado con f"".

**6. Si todo está bien:** Se imprime:

OK ¡Todo correcto! Las validaciones pasaron exitosamente.

## EJERCICIO 11

### sumaelementos



```
1 # Definición de la función para sumar elementos
2 def sumar_elementos(lista_numeros):
3     acumulador_suma = 0
4     for elemento_actual in lista_numeros:
5         acumulador_suma += elemento_actual
6     return acumulador_suma
7
8 # Casos de prueba con assert
9 print("Probando sumar_elementos...")
10
11 assert sumar_elementos([1, 2, 3, 4, 5]) == 15
12 assert sumar_elementos([10, -2, 5]) == 13
13 assert sumar_elementos([]) == 0 # ¡Importante probar con una lista vacía!
14 assert sumar_elementos([100]) == 100
15
16 print("OK! ¡Pruebas para sumar_elementos pasaron!")
17
```

```
PS C:\Users\Hugo> & C:\Users\Hugo\AppData\Local\Programs\Python\Python312\python.exe c:/Users/Hugo/Downloads/sumaElementos.py
Probando sumar_elementos...
OK! ¡Pruebas para sumar_elementos pasaron!
PS C:\Users\Hugo>
```

Este programa en Python tiene como objetivo sumar los elementos de una lista de números que incluyen pruebas automáticas

#### 1. Función sumar\_elementos(lista\_numeros)

Inicializa una variable acumulador\_suma en 0.

Recorre cada elemento de la lista lista\_numeros usando un bucle for.

Va sumando cada elemento al acumulador.

Devuelve el resultado final como la suma total.

#### 2. Pruebas de validación con assert

```
assert sumar_elementos([1, 2, 3, 4, 5]) == 15
```

```
assert sumar_elementos([10, -2, 5]) == 13
```

```
assert sumar_elementos([]) == 0
```

```
assert sumar_elementos([100]) == 100
```

Se prueban **casos distintos**, incluyendo:

- Lista con varios números

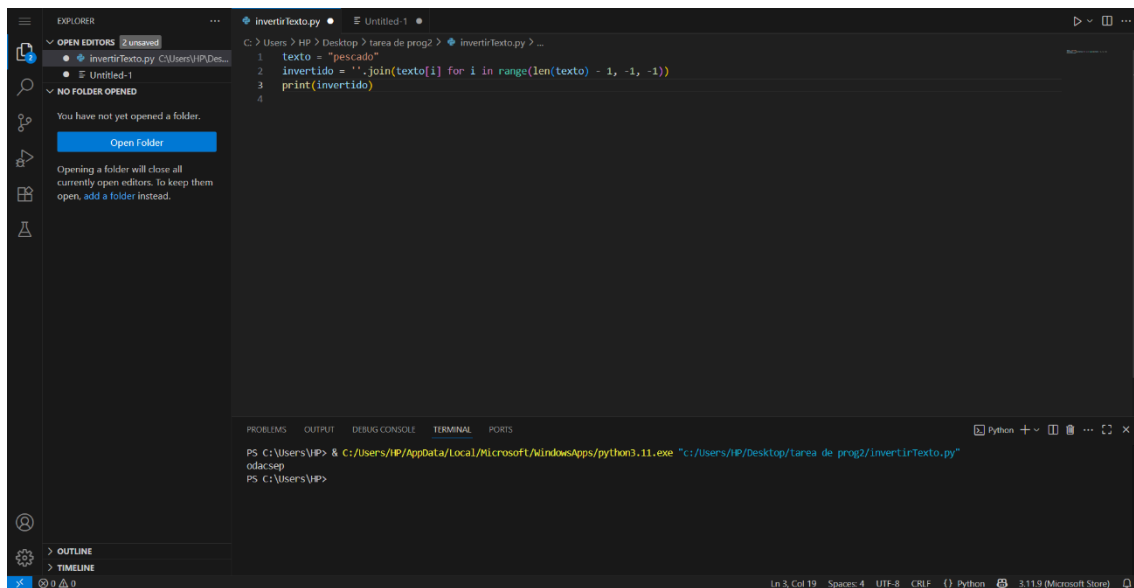
- Lista con números negativos
- Lista vacía (importante para verificar robustez)
- Lista con un solo número

Si alguna prueba falla, assert lanza un error indicando que el resultado no coincide con lo esperado. Si todo va bien, imprime:

```
print("OK! ¡Pruebas para sumar_elementos pasaron!")
```

## EJERCICIO 12

### invertirTexto



Este programa está diseñado para invertir el contenido de una cadena de texto, es decir, dar vuelta un string como si lo leyeras al revés

**Define la cadena original:** texto = "pescado" — una palabra simple.

**Invierte la cadena manualmente:**

- Usa un bucle que recorre el índice desde el final (len(texto) - 1) hasta el principio (-1), dando pasos hacia atrás (-1).
- Para cada índice i, extrae el carácter correspondiente texto[i].
- ".join(...)" combina todos los caracteres en una nueva cadena.

**Imprime el resultado:**

- Muestra "odacsep" — la versión invertida de "pescado".

## EJERCICIO 13

### for tradicionalpythonico

```
1 # CALCULO DEL CUADRADO DE LOS ELEMENTOS DE UNA LISTA DEL 0 AL 4
2 cuadrados = []
3 # TRADICIONAL
4 for x in range(5):
5     cuadrados.append(x * x)
6 print(cuadrados) # [0, 1, 4, 9, 16]
7 #PYTHONICO
8 cuadrados = [x * x for x in range(5)]
9 print(cuadrados) # [0, 1, 4, 9, 16]
10
```

```
PS C:\Users\Hugo> & C:/Users/Hugo/AppData/Local/Programs/Python/Python312/python.exe c:/Users/Hugo/Downloads/invertirLista.py
Probando invertir lista...
¡Pruebas para invertir lista pasaron! ✓
PS C:\Users\Hugo> & C:/Users/Hugo/AppData/Local/Programs/Python/Python312/python.exe c:/Users/Hugo/Downloads/forTradicionalVsPythonico.py
[0, 1, 4, 9, 16]
[0, 1, 4, 9, 16]
PS C:\Users\Hugo>
```

Este programa muestra dos formas distintas de calcular los cuadrados de los números del 0 al 4 en Python: una versión tradicional con bucle for y una más compacta y elegante usando comprensión de listas.

### Método tradicional

```
cuadrados = []
```

```
for x in range(5):
```

```
    cuadrados.append(x * x)
```

- Se crea una lista vacía cuadrados.
- Se recorre cada número x desde 0 hasta 4.
- Se calcula  $x * x$  y se agrega al final de la lista con `append()`.

Resultado: [0, 1, 4, 9, 16]

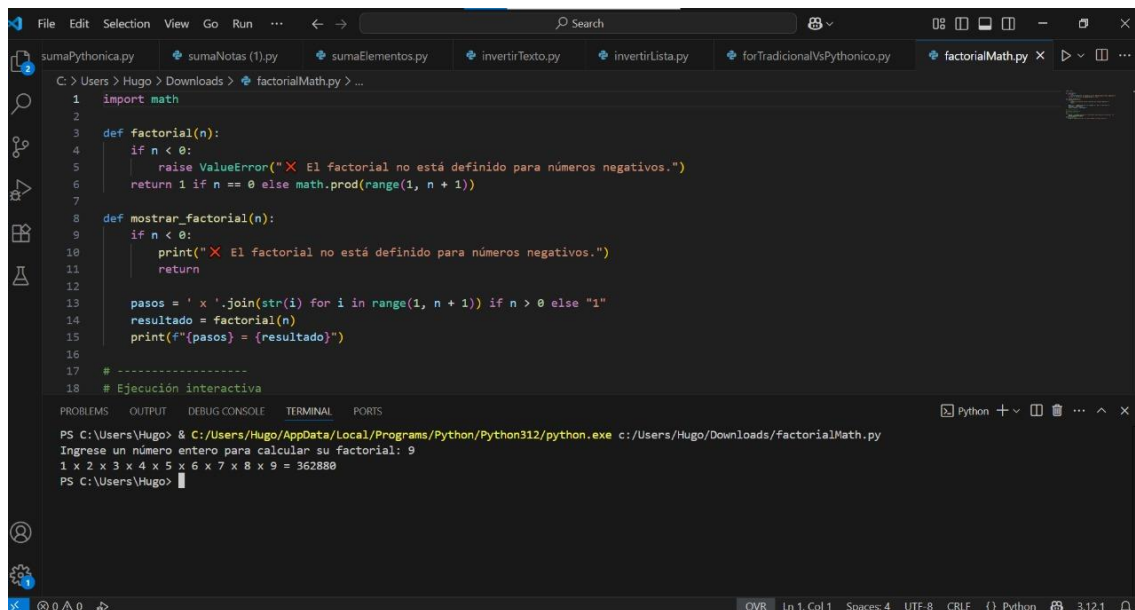
### Método pythonico

```
cuadrados = [x * x for x in range(5)]
```

- Este enfoque hace lo mismo, pero de forma más compacta y elegante.
- Usa una **list comprehension**, que es muy común en Python para transformar listas de manera concisa.

## EJERCICIO 14

factorialmath



```
1 import math
2
3 def factorial(n):
4     if n < 0:
5         raise ValueError("X El factorial no está definido para números negativos.")
6     return 1 if n == 0 else math.prod(range(1, n + 1))
7
8 def mostrar_factorial(n):
9     if n < 0:
10        print("X El factorial no está definido para números negativos.")
11        return
12
13    pasos = ' x '.join(str(i) for i in range(1, n + 1)) if n > 0 else "1"
14    resultado = factorial(n)
15    print(f"{pasos} = {resultado}")
16
17 # Ejecución interactiva
18
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Hugo> & C:/Users/Hugo/AppData/Local/Programs/Python/Python312/python.exe c:/Users/Hugo/Downloads/factorialMath.py
Ingrese un número entero para calcular su factorial: 9
1 x 2 x 3 x 4 x 5 x 6 x 7 x 8 x 9 = 362880
PS C:\Users\Hugo>
```

Este programa permite calcular el factorial de un número entero no negativo de forma interactiva. Incluye manejo de errores, validaciones, visualización de pasos, y un cálculo eficiente usando la biblioteca estándar.

### Función `factorial(n)`

- Verifica si el número es negativo. Si lo es, lanza un error con `raise ValueError`.
- Si es 0, devuelve 1 (por definición matemática).
- Si es positivo, usa `math.prod()` con `range(1, n + 1)` para calcular el factorial eficientemente.

### Función `mostrar_factorial(n)`

- Si el número es negativo, imprime un mensaje de error y termina la ejecución.
- Si no, genera una cadena como "1 x 2 x 3 x ... x n" para mostrar los pasos del cálculo.
- Calcula el factorial usando la función anterior.
- Imprime el proceso completo y el resultado.

### Ejecución interactiva

- Solicita al usuario que introduzca un número entero con `input()`.
- Intenta convertirlo en entero con `int()`.
- Si hay un error (por ejemplo, si el usuario escribe texto o un número decimal), captura el error y muestra un mensaje amigable.

## EJERCICIO 15

### factorialfunctools



```
1 from functools import reduce
2
3 def factorial(n):
4     if n < 0:
5         raise ValueError("X El factorial no está definido para números negativos.")
6     return 1 if n == 0 else reduce(lambda x, y: x * y, range(1, n + 1))
7
8 assert factorial(0) == 1
9 assert factorial(1) == 1
10 assert factorial(5) == 120
11 assert factorial(6) == 720
12 print("¡Pruebas de factoriales pasaron! ✅")
13
14 # Solicita un número al usuario
15 try:
16     numero = int(input("Ingrese un número entero para calcular su factorial: "))
17     resultado = factorial(numero)
18     print(f"El factorial de {numero} es: {resultado}")
```

PS C:\Users\Hugo> & C:\Users\Hugo\AppData\Local\Programs\Python\Python312\python.exe c:\Users\Hugo\Downloads\factorialFuncToolsReduce.py  
¡Pruebas de factoriales pasaron! ✅  
Ingrese un número entero para calcular su factorial: 8  
El factorial de 8 es: 40320  
PS C:\Users\Hugo>

Este programa en Python está diseñado para calcular el factorial de un número entero ingresado por el usuario.

### 1. Definición de la función factorial(n)

Usa reduce para multiplicar todos los números del 1 al n.

Si n es negativo, lanza un error.

Si n es 0, devuelve 1.

### 2. Pruebas automáticas

Verifica que la función funciona correctamente para algunos valores conocidos.

Si alguna prueba falla, el programa se detiene.

### 3. Interacción con el usuario

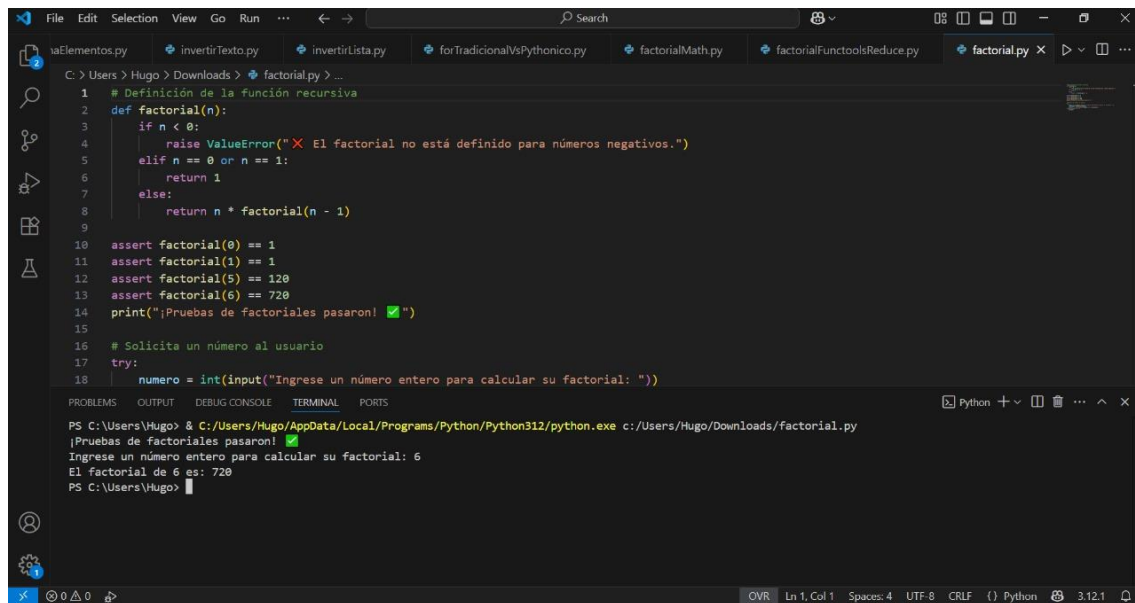
Pide al usuario un número.

Calcula el factorial usando la función.

Muestra el resultado o un mensaje de error si el número es negativo o no es entero.

## EJERCICIO 16

### factorial



```
1 # Definición de la función recursiva
2 def factorial(n):
3     if n < 0:
4         raise ValueError("X El factorial no está definido para números negativos.")
5     elif n == 0 or n == 1:
6         return 1
7     else:
8         return n * factorial(n - 1)
9
10 assert factorial(0) == 1
11 assert factorial(1) == 1
12 assert factorial(5) == 120
13 assert factorial(6) == 720
14 print("¡Pruebas de factoriales pasaron! ✅")
15
16 # Solicita un número al usuario
17 try:
18     numero = int(input("Ingrese un número entero para calcular su factorial: "))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python + -

```
PS C:\Users\Hugo> & C:\Users\Hugo\AppData\Local\Programs\Python\Python312\python.exe c:/Users/Hugo/Downloads/factorial.py
¡Pruebas de factoriales pasaron! ✅
Ingrese un número entero para calcular su factorial: 6
El factorial de 6 es: 720
PS C:\Users\Hugo>
```

Este programa está diseñado para calcular el factorial de un número entero utilizando una función recursiva.

## 1. Definición de la función factorial(n)

Si  $n$  es menor que 0, lanza una excepción con un mensaje claro.

Si  $n$  es 0 o 1, retorna 1 (por definición matemática).

Si  $n$  es mayor que 1, retorna  $n * \text{factorial}(n - 1)$ , lo que genera una cadena de llamadas recursivas hasta llegar al caso base.

## 2. Pruebas automáticas con assert

Estas líneas verifican que la función factorial devuelve los resultados esperados.

Si alguna prueba falla, el programa se detiene con un error.

Si todas pasan, imprime un mensaje de éxito.

## 3. Interacción con el usuario

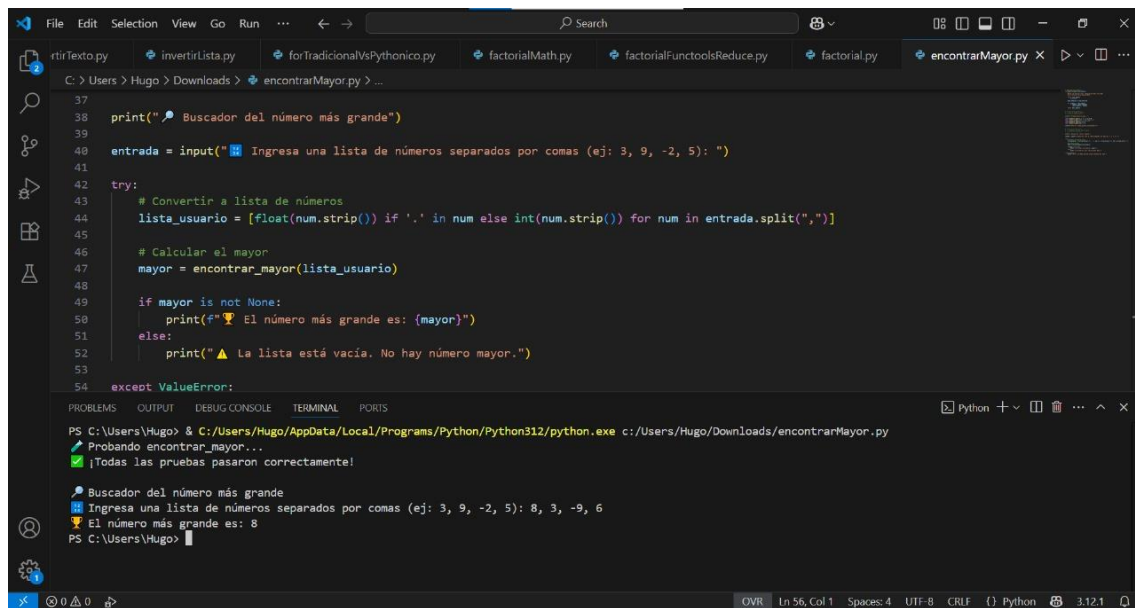
Solicita al usuario un número entero.

Intenta calcular el factorial usando la función recursiva.

Si el usuario ingresa un valor no válido (como texto o número negativo), muestra un mensaje de error amigable.

## EJERCICIO 17

encontrarmayor



The image shows a screenshot of a code editor (VS Code) with a Python file named `encontrarMayor.py` open. The code is a function `encontrar_mayor` that takes a list of numbers and returns the maximum value. It includes a try-except block to handle empty lists. The terminal shows the execution of the script, which prompts the user to enter a list of numbers separated by commas. The user enters `8, 3, -9, 6`, and the program outputs `El número más grande es: 8`.

```
37
38 print("🔍 Buscador del número más grande")
39
40 entrada = input("📝 Ingrese una lista de números separados por comas (ej: 3, 9, -2, 5): ")
41
42 try:
43     # Convertir a lista de números
44     lista_usuario = [float(num.strip()) if '.' in num else int(num.strip()) for num in entrada.split(",")]
45
46     # Calcular el mayor
47     mayor = encontrar_mayor(lista_usuario)
48
49     if mayor is not None:
50         print(f"🏆 El número más grande es: {mayor}")
51     else:
52         print("⚠️ La lista está vacía. No hay número mayor.")
53
54 except ValueError:
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\Hugo> & C:\Users\Hugo\AppData\Local\Programs\Python\Python312\python.exe c:/Users/Hugo/Downloads/encontrarMayor.py
🏆 Probando encontrar_mayor...
✅ ¡Todas las pruebas pasaron correctamente!

🔍 Buscador del número más grande
📝 Ingrese una lista de números separados por comas (ej: 3, 9, -2, 5): 8, 3, -9, 6
🏆 El número más grande es: 8
PS C:\Users\Hugo>
```

Este programa está diseñado para encontrar el número más grande dentro de una lista de números.

## 1. Definición de la función `encontrar_mayor(lista_numeros)`

**Caso especial:** si la lista está vacía, devuelve `None`.

**Inicializa** el primer elemento como el "mayor temporal".

**Recorre** la lista comparando cada elemento con el actual mayor.

**Actualiza** el mayor si encuentra uno más grande.

**Devuelve** el número más grande encontrado.

## 2. Pruebas automáticas con `assert`

Verifica que la función funciona correctamente con distintos tipos de listas:

- Lista con varios números.
- Lista con números negativos.
- Lista con números iguales.
- Lista vacía.
- Lista con un solo número.