UNIVERSIDAD PRIVADA
DOMINGO SAVIO

PROYECTO EN TINKERCAD

Materia: Programación II

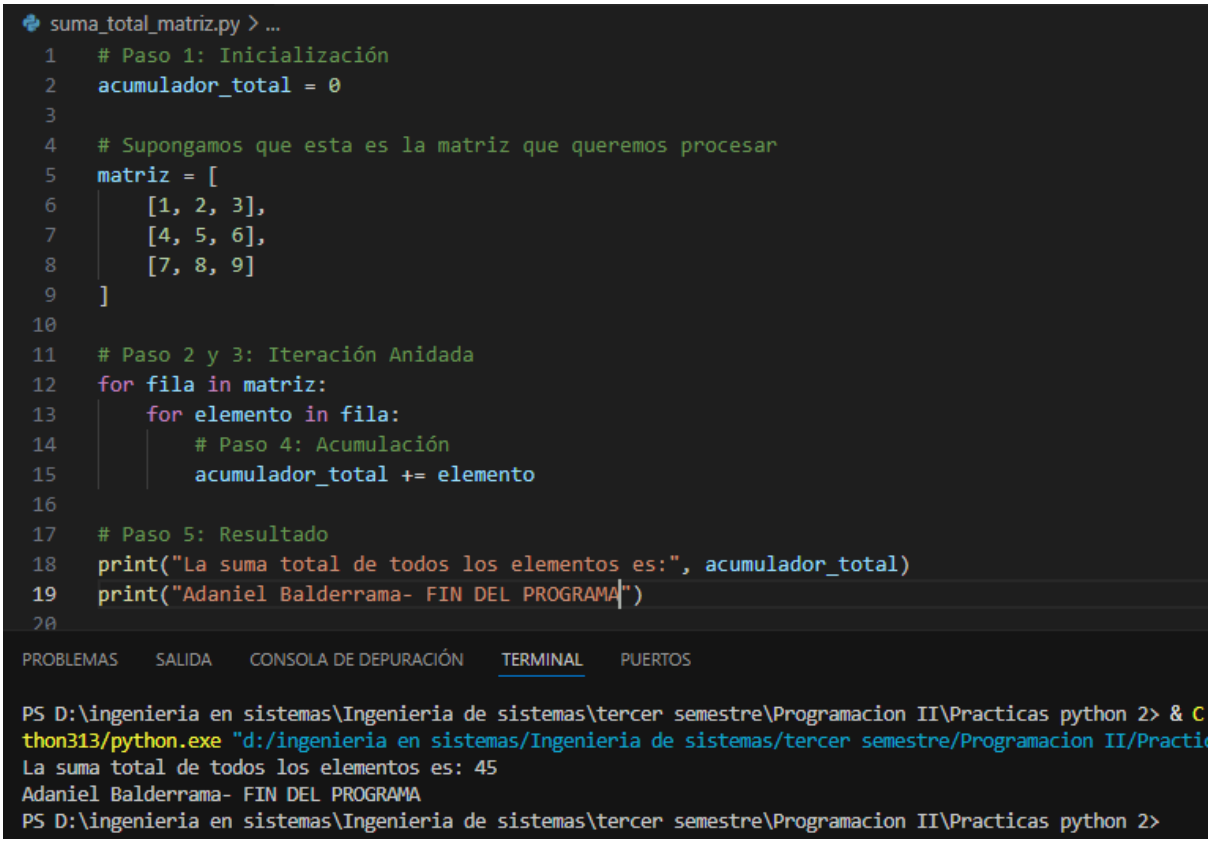
Docente: Jimmy Nataniel Requena LLorentty

Estudiantes:

- Adaniel Balderrama Orellana

Santa Cruz- Bolivia

Nombre del programa : Suma Total de la Matriz



Este código calcula la suma de todos los elementos en una **matriz (lista de listas)**. Emplea **bucles `for` anidados** para **iterar** primero sobre cada fila y luego sobre cada elemento dentro de esa fila, **acumulando la suma total**, demostrando el procesamiento de estructuras de datos multidimensionales.

Nombre del programa : Suma por Filas

Receta 2_Sumar por Filas.py > ...

```python
# Paso 1: Inicialización
sumas_por_fila = []

# Supongamos que esta es la matriz que queremos procesar
matriz = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]

# Paso 2: Iteración exterior (por fila)
for fila in matriz:
    # Paso 3a: Inicializar acumulador de fila
    acumulador_fila = 0

    # Paso 3b y 3c: Recorrer elementos y acumular
    for elemento in fila:
        acumulador_fila += elemento

    # Paso 4: Guardar resultado de la fila
    sumas_por_fila.append(acumulador_fila)

# Paso 5: Resultado final
print("La suma de cada fila es:", sumas_por_fila)
print("Adaniel Balderrama- FIN DEL PROGRAMA")
```

```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PUERTOS

PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\Practicas python 2>
thon313/python.exe "d:/ingenieria en sistemas/Ingenieria de sistemas/tercer semestre/Programacion II/Pra
La suma de cada fila es: [6, 15, 24]
Adaniel Balderrama- FIN DEL PROGRAMA
PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\Practicas python 2>
```

Este código calcula la suma de elementos para cada fila individualmente en una **matriz (lista de listas)**. Utiliza **bucles `for` anidados**: el externo itera por cada fila, y el interno acumula la suma de sus elementos. Cada suma de fila se **agrega a una nueva lista**, demostrando procesamiento por subconjuntos.

Nombre del programa : Suma por Columnas

Receta 3_ Sumar por Columnas.py > ...

```python
# Supongamos que esta es la matriz que queremos procesar
matriz = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]

# Paso 1: Inicialización
num_filas = len(matriz)
num_columnas = len(matriz[0])
sumas_por_columna = [0] * num_columnas

# Paso 2: Iteración anidada con índices
for i in range(num_filas):
    for j in range(num_columnas):
        # Paso 3: Acumulación por índice
        sumas_por_columna[j] += matriz[i][j]

# Paso 4: Resultado final
print("La suma de cada columna es:", sumas_por_columna)
print("Adaniel Balderrama- FIN DEL PROGRAMA")
```

```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PUERTOS

PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\Practicas python 2> & C
thon313/python.exe "d:/ingenieria en sistemas/Ingenieria de sistemas/tercer semestre/Programacion II/Practi
La suma de cada columna es: [12, 15, 18]
Adaniel Balderrama- FIN DEL PROGRAMA
PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\Practicas python 2>
```

Este código calcula la suma de los elementos de cada **columna** en una matriz. Demuestra la **inicialización de un vector de sumas** del tamaño de las columnas. Utiliza **bucles `for` anidados con índices** (`i` para filas, `j` para columnas) para acceder y **acumular los valores verticalmente**, mostrando cómo procesar matrices de manera diferente a por filas.

Nombre del programa : Clase10_operaciones_matrices.

clase10_operaciones_matrices.py > sumar_total_matriz

```python
# Definimos la función que suma todos los elementos de una matriz
def sumar_total_matriz(matriz):
    """
    Esta función recibe una matriz (lista de listas)
    y retorna la suma total de todos sus elementos.
    Ejemplo:
    matriz = [[1, 2], [3, 4]]
    resultado = 10
    """
    total = 0
    for fila in matriz:
        for elemento in fila:
            total += elemento
    return total
# Función para probar que sumar_total_matriz funciona correctamente
def probar_suma_total():
    print("Probando sumar_total_matriz...")
    # Caso 1: matriz normal
    m1 = [[5, 2, 3], [4, 5, 6]]
    assert sumar_total_matriz(m1) == 25 # 1+2+3+4+5+6 = 21
    # Caso 2: matriz con negativos y ceros
    m2 = [[-1, 0, 1], [10, -5, 5]]
    assert sumar_total_matriz(m2) == 10 # -1+0+1+10-5+5 = 10
    # Casos borde o límites
    assert sumar_total_matriz([[]]) == 0 # Matriz con una fila vacía
    assert sumar_total_matriz([]) == 0 # Matriz completamente vacía
    assert sumar_total_matriz([[42]]) == 42 # Matriz de un solo elemento
    print("¡Pruebas para sumar_total_matriz pasaron! ")
# Llamamos a la función de pruebas
probar_suma_total()
print("Adaniel Balderrama- FIN DEL PROGRAMA")
```

```
PROBLEMAS   SALIDA   CONSOLA DE DEPURACIÓN   TERMINAL   PUERTOS

thon313/python.exe "d:/ingenieria en sistemas/Ingenieria de sistemas/tercer semestre/Programacion II/Practica
Probando sumar_total_matriz...
¡Pruebas para sumar_total_matriz pasaron!
Adaniel Balderrama- FIN DEL PROGRAMA
PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\Practicas python 2>
```

Estos códigos cubren la entrada/salida, condicionales y bucles. Exploran la manipulación de **listas (vectores)**, incluyendo indexación, iteración y funciones para sumas, conteos y encontrar máximos. Demuestran **modularización con funciones**, **refactorización**, y algoritmos esenciales como **búsqueda (lineal y binaria)** y **ordenamiento (burbuja)**. Finalmente, abordan el procesamiento de **matrices (listas anidadas)** por filas y columnas.

Nombre del programa : Ejercicio 2: Función de Suma por Filas

Ejercicio 2_Función de Suma por Filas.py > ...

```python
# Definimos la función que suma los elementos por cada fila de la matriz
def sumar_por_filas(matriz):
    """
    Esta función recibe una matriz (lista de listas)
    y devuelve una lista con la suma de cada fila.
    Ejemplo:
    matriz = [[1, 2, 3], [4, 5, 6]]
    resultado = [6, 15]
    """
    resultado = []
    for fila in matriz:
        suma_fila = sum(fila)  # Suma todos los elementos de la fila
        resultado.append(suma_fila)
    return resultado

# Función de prueba para verificar que sumar_por_filas funciona correctamente
def probar_suma_por_filas():
    print("\nProbando sumar_por_filas...")
    # Caso 1: matriz con 3 filas y 3 columnas
    m1 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
    assert sumar_por_filas(m1) == [6, 15, 24]  # 1+2+3, 4+5+6, 7+8+9
    # Caso 2: matriz con pares repetidos
    m2 = [[10, 10], [20, 20], [30, 30]]
    assert sumar_por_filas(m2) == [20, 40, 60]
    # Caso borde: matriz vacía
    assert sumar_por_filas([]) == []  # No hay filas que sumar
    print("¡Pruebas para sumar_por_filas pasaron!")

# Llamamos a la función para ejecutar las pruebas
probar_suma_por_filas()
print("Adaniel Balderrama- FIN DEL PROGRAMA")
```

```
PROBLEMAS   SALIDA   CONSOLA DE DEPURACIÓN   TERMINAL   PUERTOS

PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\Practicas python 2> &
thon313/python.exe "d:/ingenieria en sistemas/Ingenieria de sistemas/tercer semestre/Programacion II/Prac
ilas.py"

Probando sumar_por_filas...
¡Pruebas para sumar_por_filas pasaron!
Adaniel Balderrama- FIN DEL PROGRAMA
PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\Practicas python 2>
```

Este código define una **función** `sumar_por_filas` que calcula la suma de elementos para cada fila de una matriz, utilizando `sum()` en cada sublista. Incluye una **función de pruebas (**`probar_suma_por_filas`**) con** `assert` para validar su correcto funcionamiento en matrices normales, con repeticiones y vacías, asegurando su robustez.

Nombre del programa : Ejercicio 3: Suma de la Diagonal Principal

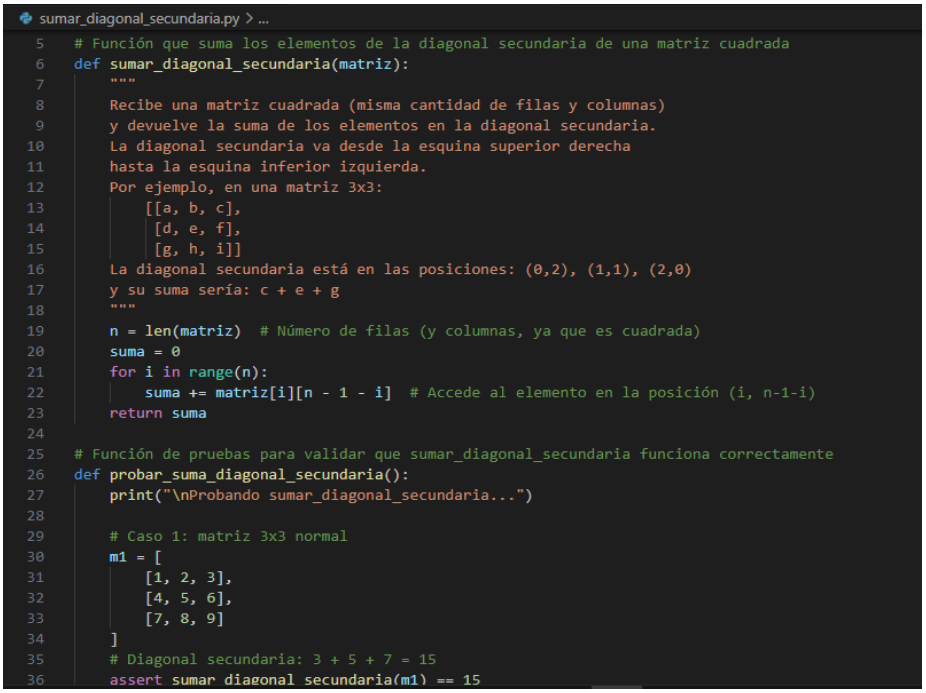
sumar_diagonal_principal.py > ...

```python
# Definimos la función que suma los elementos de la diagonal principal de una matriz cuadrada
def sumar_diagonal_principal(matriz):
    """
    Esta función recibe una matriz cuadrada (misma cantidad de filas y columnas)
    y retorna la suma de los elementos en su diagonal principal.
    Ejemplo:
    matriz = [[1, 2],
    [3, 4]]
    diagonal principal: 1 y 4 → suma = 5
    """
    suma = 0
    for i in range(len(matriz)):
        suma += matriz[i][i] # Accede al elemento en la posición (i, i)
    return suma
# Función de prueba para verificar que sumar_diagonal_principal funciona correctamente
def probar_suma_diagonal_principal():
    print("\nProbando sumar_diagonal_principal...")
    # Caso 1: matriz 3x3 con números consecutivos
    m1 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
    assert sumar_diagonal_principal(m1) == 15 # 1 + 5 + 9
    # Caso 2: matriz 2x2 con ceros y valores definidos
    m2 = [[10, 0], [0, 20]]
    assert sumar_diagonal_principal(m2) == 30 # 10 + 20
    # Caso borde: matriz 1x1
    m3 = [[5]]
    assert sumar_diagonal_principal(m3) == 5 # Solo un elemento en la diagonal
    print("¡Pruebas para sumar_diagonal_principal pasaron! ")
# Llamamos a la función para ejecutar las pruebas
probar_suma_diagonal_principal()
print("Adaniel Balderrama- FIN DEL PROGRAMA")
```

```
PROBLEMAS   SALIDA   CONSOLA DE DEPURACIÓN   TERMINAL   PUERTOS
Probando sumar_diagonal_principal...
¡Pruebas para sumar_diagonal_principal pasaron!
Adaniel Balderrama- FIN DEL PROGRAMA
PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\Practicas python 2>
```

Este código define la función `sumar_diagonal_principal` para calcular la suma de los elementos en la **diagonal principal de una matriz cuadrada**. Utiliza un bucle `for` y un solo índice `i` para acceder a `matriz[i][i]`, sumando eficientemente solo los elementos diagonales. Las **pruebas con `assert`** validan su correcto funcionamiento para varios tamaños de matrices.

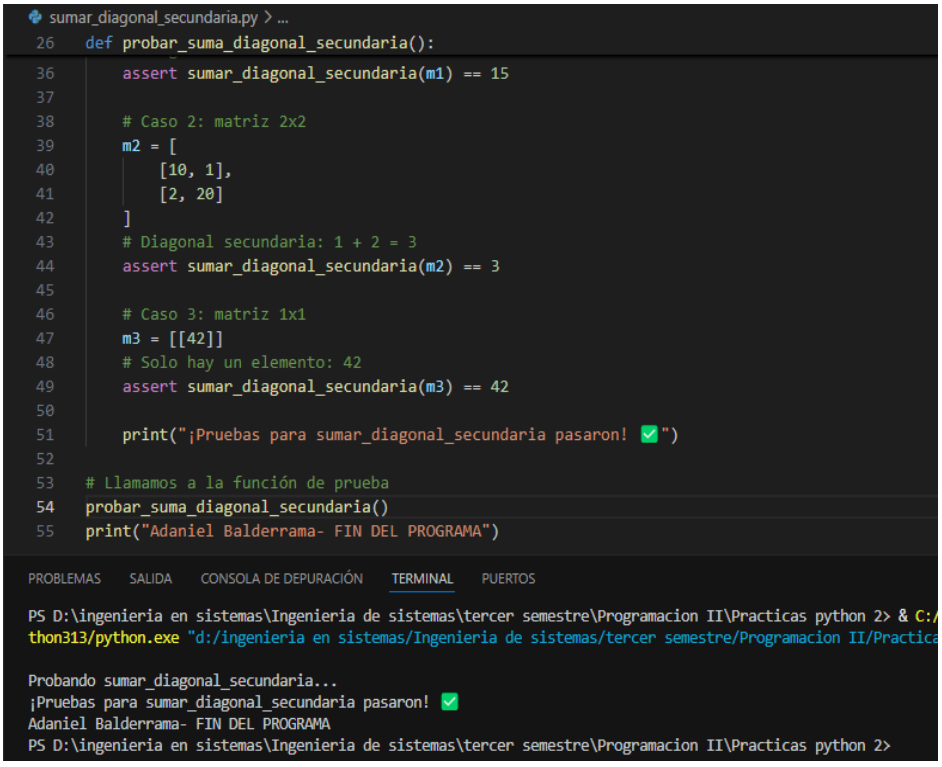
Nombre del programa: Sumar Diagonal Secundaria



```python
# Función que suma los elementos de la diagonal secundaria de una matriz cuadrada
def sumar_diagonal_secundaria(matriz):
    """
    Recibe una matriz cuadrada (misma cantidad de filas y columnas)
    y devuelve la suma de los elementos en la diagonal secundaria.
    La diagonal secundaria va desde la esquina superior derecha
    hasta la esquina inferior izquierda.
    Por ejemplo, en una matriz 3x3:
        [[a, b, c],
         [d, e, f],
         [g, h, i]]
    La diagonal secundaria está en las posiciones: (0,2), (1,1), (2,0)
    y su suma sería: c + e + g
    """
    n = len(matriz)  # Número de filas (y columnas, ya que es cuadrada)
    suma = 0
    for i in range(n):
        suma += matriz[i][n - 1 - i]  # Accede al elemento en la posición (i, n-1-i)
    return suma

# Función de pruebas para validar que sumar_diagonal_secundaria funciona correctamente
def probar_suma_diagonal_secundaria():
    print("\nProbando sumar_diagonal_secundaria...")

    # Caso 1: matriz 3x3 normal
    m1 = [
        [1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]
    ]
    # Diagonal secundaria: 3 + 5 + 7 = 15
    assert sumar_diagonal_secundaria(m1) == 15
```



```python
def probar_suma_diagonal_secundaria():
    assert sumar_diagonal_secundaria(m1) == 15

    # Caso 2: matriz 2x2
    m2 = [
        [10, 1],
        [2, 20]
    ]
    # Diagonal secundaria: 1 + 2 = 3
    assert sumar_diagonal_secundaria(m2) == 3

    # Caso 3: matriz 1x1
    m3 = [[42]]
    # Solo hay un elemento: 42
    assert sumar_diagonal_secundaria(m3) == 42

    print("¡Pruebas para sumar_diagonal_secundaria pasaron! ✅")

# Llamamos a la función de prueba
probar_suma_diagonal_secundaria()
print("Adaniel Balderrama- FIN DEL PROGRAMA")
```

```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PUERTOS

PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\Practicas python 2> & C:/
thon313/python.exe "d:/ingenieria en sistemas/Ingenieria de sistemas/tercer semestre/Programacion II/Practica

Probando sumar_diagonal_secundaria...
¡Pruebas para sumar_diagonal_secundaria pasaron! ✅
Adaniel Balderrama- FIN DEL PROGRAMA
PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\Practicas python 2>
```

Este código define la función `sumar_diagonal_secundaria` para calcular la suma de los elementos en la **diagonal secundaria de una matriz cuadrada**. Utiliza un bucle `for` y la relación `matriz[i][n - 1 - i]` para acceder a los elementos correctos. Las **pruebas con `assert`** validan su funcionamiento para matrices de diferentes tamaños, demostrando el acceso preciso a elementos diagonales específicos.

Nombre del programa: : Clase 11 Transformaciones

clase11_transformaciones.py > ...

```python
def transponer_matriz(matriz):
    if not matriz or not matriz[0]:
        return[]
    num_filas = len(matriz)
    num_columnas = len(matriz[0])
    # Inicializamos la transpuesta la tranpuesta con la estructura correcta (o la construimos dinamicamente)
    matriz_transpuesta = []
    for j in range(num_columnas): # El bucle exterior itera sobre las columnas originales
        nueva_fila = []
        for i in range (num_filas): # El bucle interior itera sobre las Filas originales
            nueva_fila.append(matriz[i][j])
        matriz_transpuesta.append(nueva_fila)
    return matriz_transpuesta


# Prueba tu funcion rigurosamente (¡incluye matrices no cuadradas!):

m1 = [[1, 2, 3 ], [4, 5, 6]] #2x3
t1 = transponer_matriz (m1)
assert t1 == [[1,4],[2,5],[3,6]] # Debe ser 3x2
print ("Prueba 1 (2x3) pasada !")
# Añade mas pruebas con assert ...
print("Adaniel Balderrama- FIN DEL PROGRAMA")
```

```
PROBLEMAS   SALIDA   CONSOLA DE DEPURACIÓN   TERMINAL   PUERTOS

PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\Practicas python 2> & C:/Users/dar
thon313/python.exe "d:/ingenieria en sistemas/Ingenieria de sistemas/tercer semestre/Programacion II/Practicas python
Prueba 1 (2x3) pasada !
Adaniel Balderrama- FIN DEL PROGRAMA
PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\Practicas python 2>
```

Este código define la función `transponer_matriz` que invierte filas y columnas de una matriz, creando una nueva matriz transpuesta. Inicializa la `matriz_transpuesta` y utiliza **bucles anidados** que iteran sobre las columnas originales (externo) y las filas originales (interno) para construir la nueva estructura. Incluye validación para matrices vacías y pruebas con `assert` para verificar su corrección, incluso con matrices no cuadradas.

Nombre de programa: Ejercicio 2: Función para Verificar Identidad

Ejercicio 2_Función para Verificar Identidad.py > ...

```python
# Función que verifica si una matriz es simétrica
def es_simetrica(matriz):
    # Requisito 1: Debe ser cuadrada
    num_filas = len(matriz)
    if num_filas == 0:
        return True  # Una matriz vacía es trivialmente simétrica

    for i in range(num_filas):
        if len(matriz[i]) != num_filas:
            return False  # No es cuadrada

    # Requisito 2: Comparar matriz[i][j] con matriz[j][i]
    for i in range(num_filas):
        for j in range(i + 1, num_filas):  # Solo chequear la triangular superior
            if matriz[i][j] != matriz[j][i]:
                return False  # ¡Con una diferencia es suficiente!

    return True  # Si no encontramos diferencias, es simétrica

# Pruebas de la función
sim = [
    [1, 7, 3],
    [7, -4, -5],
    [3, -5, 6]
]

no_sim = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]

no_cuadrada = [
    [1, 2],
    [3, 4],
    [5, 6]
]

assert es_simetrica(sim) == True
assert es_simetrica(no_sim) == False
assert es_simetrica(no_cuadrada) == False

print("¡Pruebas para es_simetrica pasaron! ✅")
print("Adaniel Balderrama- FIN DEL PROGRAMA")
```

```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PUERTOS
PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\Practicas python 2> & C:/U
thon313/python.exe "d:/ingenieria en sistemas/Ingenieria de sistemas/tercer semestre/Programacion II/Practicas
r Identidad.py"
¡Pruebas para es_simetrica pasaron! ✅
Adaniel Balderrama- FIN DEL PROGRAMA
PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\Practicas python 2>
```

Este código define la función `es_simetrica` que verifica si una matriz es simétrica. Primero valida que sea **cuadrada**. Luego, usa **bucles anidados** para comparar `matriz[i][j]` con `matriz[j][i]`, iterando eficientemente solo sobre la mitad superior de la matriz. Si encuentra alguna desigualdad, retorna `False`; de lo contrario, `True`. Las **pruebas con `assert`** cubren matrices simétricas, no simétricas y no cuadradas.

Nombre de programa: Gestión de Asientos de Cine

salaCine.py > ocupar_asiento

```python
# Función 1: Crear la sala llena de 'L' (libres)
def crear_sala(filas, columnas):
    return [['L' for _ in range(columnas)] for _ in range(filas)]

# Función 2: Mostrar visualmente la sala
def mostrar_sala(sala):
    print("   " + " ".join(f"{i}" for i in range(len(sala[0]))))  # Números de columna
    for i, fila in enumerate(sala):
        print(f"{i:2} " + " ".join(fila))

# Función 3: Ocupar un asiento
def ocupar_asiento(sala, fila, columna):
    filas = len(sala)
    columnas = len(sala[0])

    # Verifica si coordenadas son válidas
    if 0 <= fila < filas and 0 <= columna < columnas:
        if sala[fila][columna] == 'L':
            sala[fila][columna] = 'O'
            print(f"Asiento ({fila}, {columna}) ocupado exitosamente.")
            return True
        else:
            print("Ese asiento ya está ocupado.")
            return False
    else:
        print("Coordenadas fuera de rango.")
        return False

# Función 4 (Opcional): Contar asientos libres
def contar_asientos_libres(sala):
    return sum(fila.count('L') for fila in sala)
```

salaCine.py > ocupar_asiento

```python
# Programa Principal
def main():
    filas, columnas = 5, 8  # Tamaño predeterminado
    sala = crear_sala(filas, columnas)

    while True:
        print("\n🎥 Estado actual de la sala de cine:")
        mostrar_sala(sala)
        print(f"Asientos libres: {contar_asientos_libres(sala)}")
        print("\nMenú:")
        print("1. Ocupar asiento")
        print("0. Salir")

        opcion = input("Elige una opción: ")

        if opcion == '1':
            try:
                fila = int(input("Ingresa el número de fila: "))
                columna = int(input("Ingresa el número de columna: "))
                ocupar_asiento(sala, fila, columna)
            except ValueError:
                print("❌ Entrada inválida. Por favor ingresa números.")
        elif opcion == '0':
            print("🎬 Gracias por usar el sistema de reserva. ¡Hasta luego!")
            break
        else:
            print("❌ Opción no válida. Intenta nuevamente.")

# Ejecutar el programa principal
if __name__ == "__main__":
    main()
print("Adaniel Balderrama- FIN DEL PROGRAMA")
```

```
🎥 Estado actual de la sala de cine:
   0 1 2 3 4 5 6 7
 0 L L L L L L L L
 1 L L L L L L L L
 2 L L L L L L L L
 3 L L L L L L L L
 4 L L L L L L L L
Asientos libres: 40

Menú:
1. Ocupar asiento
0. Salir
Elige una opción: 1
Ingresa el número de fila: 2
Ingresa el número de columna: 5
Asiento (2, 5) ocupado exitosamente.

🎥 Estado actual de la sala de cine:
   0 1 2 3 4 5 6 7
 0 L L L L L L L L
 1 L L L L L L L L
 2 L L L L L O L L
 3 L L L L L L L L
 4 L L L L L L L L
Asientos libres: 39

Menú:
1. Ocupar asiento
0. Salir
Elige una opción: 0
🎬 Gracias por usar el sistema de reserva. ¡Hasta luego!
Adaniel Balderrama- FIN DEL PROGRAMA
PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\Practicas python 2>
```

Nombre del programa: Diccionario con un Bucle for

clase_diccionarios.py > ...

```python
producto = {'codigo': 'P002', 'nombre': 'Coca Cola', 'precio': 15, 'stock': 50}
print("\n--- Claves del producto ---")
for clave in producto: # Por defecto, itera sobre las CLAVES
    print(clave)
print("\n--- Clave y Valor ---")
for clave in producto:
    valor = producto[clave]
    print(f"{clave.capitalize()}: {valor}")

    producto1 = {'codigo': 'P003', 'nombre': 'Pizza', 'precio': 54, 'stock': 25}
print("\n--- Claves del producto ---")
for clave in producto1: # Por defecto, itera sobre las CLAVES
    print(clave)
print("\n--- Clave y Valor ---")
for clave in producto1:
    valor = producto1[clave]
    print(f"{clave.capitalize()}: {valor}")

    producto2 = {'codigo': 'P004', 'nombre': 'Hamburguesa', 'precio': 25, 'stock': 75}
print("\n--- Claves del producto ---")
for clave in producto2: # Por defecto, itera sobre las CLAVES
    print(clave)
print("\n--- Clave y Valor ---")
for clave in producto2:
    valor = producto2[clave]
    print(f"{clave.capitalize()}: {valor}")
    print("Adaniel Balderrama- FIN DEL PROGRAMA")
```

```
--- Claves del producto ---
codigo
nombre
precio
stock

--- Clave y Valor ---
Codigo: P004
Nombre: Hamburguesa
Precio: 25
Stock: 75
Adaniel Balderrama- FIN DEL PROGRAMA
PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\Practicas python 2>
```

Este código demuestra la manipulación e iteración de **diccionarios (`dict`)** en Python, que almacenan datos en pares `clave: valor`. Muestra cómo **acceder a las claves** directamente (`for clave in diccionario:`) y cómo luego **obtener los valores asociados** (`diccionario[clave]`) para imprimir ambos, ejemplificando la estructura y el recorrido de datos

Nombre del programa: Clase 12 Diccionarios

clase12_diccionarios.py > ...

```python
# 1. Definición de la función
def es_simetrica(matriz):
    # Requisito 1: Debe ser cuadrada
    num_filas = len(matriz)
    if num_filas == 0:
        return True  # Una matriz vacía es trivialmente simétrica

    for i in range(num_filas):
        if len(matriz[i]) != num_filas:
            return False  # No es cuadrada

    # Requisito 2: Comparar matriz[i][j] con matriz[j][i]
    for i in range(num_filas):
        for j in range(i + 1, num_filas):  # Solo chequear la triangular superior
            if matriz[i][j] != matriz[j][i]:
                return False  # ¡Con una diferencia es suficiente!

    return True  # Si no encontramos diferencias, es simétrica

# 2. Pruebas
sim = [
    [1, 7, 3],
    [7, -4, -5],
    [3, -5, 6]
]

no_sim = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]
```

```python
no_cuadrada = [
    [1, 2],
    [3, 4],
    [5, 6]
]

# 3. Verificación
assert es_simetrica(sim) == True
assert es_simetrica(no_sim) == False
assert es_simetrica(no_cuadrada) == False

print("¡Pruebas para es_simetrica pasaron! ✅")
print("Adaniel Balderrama- FIN DEL PROGRAMA")
```

```
PROBLEMAS   SALIDA   CONSOLA DE DEPURACIÓN   TERMINAL   PUERTOS
PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\Practicas python 2> & C:/Use
thon313/python.exe "d:/ingenieria en sistemas/Ingenieria de sistemas/tercer semestre/Programacion II/Practicas p
¡Pruebas para es_simetrica pasaron! ✅
Adaniel Balderrama- FIN DEL PROGRAMA
PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\Practicas python 2>
```

Este código define la función `es_simetrica` para determinar si una matriz es simétrica. Primero, valida que la matriz sea **cuadrada**. Luego, utiliza **bucles anidados** para comparar eficientemente `matriz[i][j]` con `matriz[j][i]`, iterando solo sobre la parte superior de la matriz. Si encuentra una discrepancia, retorna `False`; de lo contrario, `True`. Las **pruebas (`assert`)** verifican su precisión con matrices simétricas, asimétricas y no cuadradas.

Nombre del programa : clase 13 todolist

clase13_todolist.py > ...

```python
# Paso 1: Variables Globales
lista_de_tareas = []
proximo_id_tarea = 1  # Para generar IDs únicos


# Paso 2: Implementar agregar_tarea
def agregar_tarea(descripcion, prioridad="media"):
    global proximo_id_tarea  # ¡Necesario para modificar una variable global!
    nueva_tarea = {
        "id": proximo_id_tarea,
        "descripcion": descripcion,
        "completada": False,
        "prioridad": prioridad
    }
    lista_de_tareas.append(nueva_tarea)
    proximo_id_tarea += 1
    print(f"✅ Tarea '{descripcion}' añadida con éxito.")


# Paso 3: Implementar mostrar_tareas
def mostrar_tareas():
    print("\n--- LISTA DE TAREAS ---")
    if not lista_de_tareas:
        print("¡No hay tareas pendientes! ¡A disfrutar!")
        return

    for tarea in lista_de_tareas:
        estado = "✓" if tarea["completada"] else "⏳"
        print(f"{estado} ID: {tarea['id']} | {tarea['descripcion']} (Prioridad: {tarea['prioridad']})")
    print("-------------------------")
```

```python
# Prueba de las funciones
agregar_tarea("Estudiar para el examen de Cálculo")
agregar_tarea("Hacer las compras", prioridad="alta")
mostrar_tareas()
print("Adaniel Balderrama- FIN DEL PROGRAMA")
```

```
PROBLEMAS   SALIDA   CONSOLA DE DEPURACIÓN   TERMINAL   PUERTOS
PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\Practicas python 2> & C:/Users/d
thon313/python.exe "d:/ingenieria en sistemas/Ingenieria de sistemas/tercer semestre/Programacion II/Practicas pytho
✅ Tarea 'Estudiar para el examen de Cálculo' añadida con éxito.
✅ Tarea 'Hacer las compras' añadida con éxito.

--- LISTA DE TAREAS ---
⏳ ID: 1 | Estudiar para el examen de Cálculo (Prioridad: media)
⏳ ID: 2 | Hacer las compras (Prioridad: alta)
-------------------------
Adaniel Balderrama- FIN DEL PROGRAMA
PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\Practicas python 2>
```

Este código implementa un sistema básico de **lista de tareas** utilizando **variables globales** (`lista_de_tareas`, `proximo_id_tarea`). Define la función `agregar_tarea` para añadir diccionarios de tareas (con ID, descripción, estado y prioridad) a la lista, e `incrementa el ID`. La función `mostrar_tareas` itera sobre esta lista para imprimir el estado de cada tarea. Ilustra el uso de **estructuras de datos complejas (listas de diccionarios)** y la gestión de estado global en una aplicación sencilla.

Nombre del programa : Clase 13 todolist complet

clase13_todolist complet.py > ...

```python
# clase13_todolist.py

# Paso 1: Variables Globales
lista_de_tareas = []
proximo_id_tarea = 1  # Para generar IDs únicos


# Paso 2: Implementar agregar_tarea
def agregar_tarea(descripcion, prioridad="media"):
    global proximo_id_tarea
    nueva_tarea = {
        "id": proximo_id_tarea,
        "descripcion": descripcion,
        "completada": False,
        "prioridad": prioridad
    }
    lista_de_tareas.append(nueva_tarea)
    proximo_id_tarea += 1
    print(f"✅ Tarea '{descripcion}' añadida con éxito.")


# Paso 3: Implementar mostrar_tareas
def mostrar_tareas():
    print("\n--- LISTA DE TAREAS ---")
    if not lista_de_tareas:
        print("¡No hay tareas pendientes! ¡A disfrutar!")
        return

    for tarea in lista_de_tareas:
        estado = "✓" if tarea["completada"] else "⏳"
        print(f"{estado} ID: {tarea['id']} | {tarea['descripcion']} (Prioridad: {tarea['prioridad']})")
    print("-------------------------")
```

clase13_todolist complet.py > ...

```python
# Paso 4: Implementar buscar_tarea_por_id
def buscar_tarea_por_id(id_buscado):
    for tarea in lista_de_tareas:
        if tarea["id"] == id_buscado:
            return tarea
    return None


# Paso 5: Implementar marcar_tarea_completada
def marcar_tarea_completada(id_tarea):
    tarea = buscar_tarea_por_id(id_tarea)
    if tarea:
        tarea["completada"] = True
        print(f"☑️ Tarea '{tarea['descripcion']}' marcada como completada.")
    else:
        print(f"❌ Error: No se encontró la tarea con ID {id_tarea}.")


# Paso 6: Implementar eliminar_tarea
def eliminar_tarea(id_tarea):
    tarea = buscar_tarea_por_id(id_tarea)
    if tarea:
        lista_de_tareas.remove(tarea)
        print(f"🗑️ Tarea '{tarea['descripcion']}' eliminada.")
    else:
        print(f"❌ Error: No se encontró la tarea con ID {id_tarea}.")


# --------------------------------
# Prueba de todo el flujo
# --------------------------------
```

```python
# ------------------------------
# Prueba de todo el flujo
# ------------------------------
agregar_tarea("Estudiar para el examen de Cálculo")
agregar_tarea("Hacer las compras", prioridad="alta")

mostrar_tareas()

marcar_tarea_completada(1)
mostrar_tareas()  # Debería mostrar la tarea 1 como completada

eliminar_tarea(2)
mostrar_tareas()  # La tarea 2 ya no debería aparecer

marcar_tarea_completada(99)  # Probar con un ID que no existe
print("Adaniel Balderrama- FIN DEL PROGRAMA")
```

```
PROBLEMAS   SALIDA   CONSOLA DE DEPURACIÓN   TERMINAL   PUERTOS

PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\Practicas python 2> & C:/
thon313/python.exe "d:/ingenieria en sistemas/Ingenieria de sistemas/tercer semestre/Programacion II/Practica
✅ Tarea 'Estudiar para el examen de Cálculo' añadida con éxito.
✅ Tarea 'Hacer las compras' añadida con éxito.

--- LISTA DE TAREAS ---
⏳ ID: 1 | Estudiar para el examen de Cálculo (Prioridad: media)
⏳ ID: 2 | Hacer las compras (Prioridad: alta)
-----------------------
☑Tarea 'Estudiar para el examen de Cálculo' marcada como completada.

--- LISTA DE TAREAS ---
✓ ID: 1 | Estudiar para el examen de Cálculo (Prioridad: media)
⏳ ID: 2 | Hacer las compras (Prioridad: alta)
-----------------------
🗑Tarea 'Hacer las compras' eliminada.

--- LISTA DE TAREAS ---
✓ ID: 1 | Estudiar para el examen de Cálculo (Prioridad: media)
-----------------------
❌ Error: No se encontró la tarea con ID 99.
Adaniel Balderrama- FIN DEL PROGRAMA
PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\Practicas python 2>
```

Este código implementa un **sistema de gestión de tareas** robusto, utilizando **variables globales** (`lista_de_tareas`, `proximo_id_tarea`) para mantener el estado. Define funciones para:

- `agregar_tarea`: Añade tareas como **diccionarios** con ID, descripción, estado y prioridad, usando `global` para actualizar el ID único.
- `mostrar_tareas`: Lista las tareas con su estado visual (✔ o ⏳).
- `buscar_tarea_por_id`: Localiza una tarea por su ID.
- `marcar_tarea_completada`: Cambia el estado de una tarea a completada.
- `eliminar_tarea`: Remueve una tarea de la lista. Estas funciones demuestran **operaciones CRUD (Crear, Leer, Actualizar, Borrar)** básicas sobre una colección de datos. El código incluye un **flujo de prueba** que valida cada funcionalidad y maneja casos de tareas inexistentes.

Nombre del programa: clase 14 poo ejemplo 1

poo_eje1.py > ...

```python
class Libro:
    """
    Clase que representa un libro con sus atributos principales.
    """

    def __init__(self, titulo, autor, isbn, paginas):
        """
        Constructor de la clase Libro.

        Args:
            titulo (str): Título del libro
            autor (str): Autor del libro
            isbn (str): ISBN del libro
            paginas (int): Número de páginas del libro
        """
        # Crear los atributos de instancia correspondientes
        self.titulo = titulo
        self.autor = autor
        self.isbn = isbn
        self.paginas = paginas

        # Atributo extra que se inicializa siempre por defecto
        self.disponible = True

    def mostrar_info(self):
        """
        Método que imprime todos los atributos del libro de forma clara y formateada.
        """
        print("=" * 50)
        print("INFORMACIÓN DEL LIBRO")
        print("=" * 50)
        print(f"Título: {self.titulo}")
        print(f"Autor: {self.autor}")
        print(f"ISBN: {self.isbn}")
        print(f"Páginas: {self.paginas}")
        print(f"Disponible: {'Sí' if self.disponible else 'No'}")
        print("=" * 50)


# Ejemplo de uso (no te preocupes por crear objetos todavía, ¡solo define la clase!)
if __name__ == "__main__":
    # Creación de ejemplo para demostrar el funcionamiento
    libro1 = Libro("Cien años de soledad", "Gabriel García Márquez", "978-0307474728", 417)
    libro1.mostrar_info()

    # Cambiar disponibilidad
    libro1.disponible = False
    print("\nDespués de cambiar disponibilidad:")
    libro1.mostrar_info()

print("Adaniel Balderrama- FIN DEL PROGRAMA")
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS

```
PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\Practicas python 2> & C
thon313/python.exe "d:/ingenieria en sistemas/Ingenieria de sistemas/tercer semestre/Programacion II/Practi
==================================================
INFORMACIÓN DEL LIBRO
==================================================
Título: Cien años de soledad
Autor: Gabriel García Márquez
ISBN: 978-0307474728
Páginas: 417
Disponible: Sí
==================================================

Después de cambiar disponibilidad:
==================================================
INFORMACIÓN DEL LIBRO
==================================================
Título: Cien años de soledad
Autor: Gabriel García Márquez
ISBN: 978-0307474728
Páginas: 417
Disponible: No
==================================================
Adaniel Balderrama- FIN DEL PROGRAMA
PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\Practicas python 2>
```

Nombre del programa : Clase 14 poo ejemplo 2

poo_eje2.py > ...

```python
# Ejercicio 2: Creación de instancias y métodos de comportamiento

class Libro:
    """
    Clase que representa un libro con sus atributos principales.
    """

    def __init__(self, titulo, autor, isbn, paginas):
        """
        Constructor de la clase Libro.
        """
        self.titulo = titulo
        self.autor = autor
        self.isbn = isbn
        self.paginas = paginas
        self.disponible = True

    def mostrar_info(self):
        """
        Método que imprime todos los atributos del libro de forma clara y formateada.
        """
        print("=" * 50)
        print("INFORMACIÓN DEL LIBRO")
        print("=" * 50)
        print(f"Título: {self.titulo}")
        print(f"Autor: {self.autor}")
        print(f"ISBN: {self.isbn}")
        print(f"Páginas: {self.paginas}")
        print(f"Disponible: {'Sí' if self.disponible else 'No'}")
        print("=" * 50)

    def prestar_libro(self):
        """
        Método para prestar el libro. Cambia disponible a False si está disponible.
        """
        if self.disponible == True:
            self.disponible = False
            print(f"El libro '{self.titulo}' ha sido prestado.")
        else:
            print(f"El libro '{self.titulo}' ya está prestado.")

    def devolver_libro(self):
        """
        Método para devolver el libro. Cambia disponible a True si estaba prestado.
        """
        if self.disponible == False:
            self.disponible = True
            print(f"El libro '{self.titulo}' ha sido devuelto.")
        else:
            print(f"El libro '{self.titulo}' ya estaba disponible.")


# 1. Definición de clase Libro (arriba) ✓

# 2. Crear dos objetos Libro diferentes
libro1 = Libro("El Principito", "Antoine de Saint-Exupéry", "978-3-14-046401-7", 120)
libro2 = Libro("Raza de Bronce", "Alcides Arguedas", "978-99905-2-213-9", 250)

# 3. Acceder y mostrar algunos de sus atributos directamente
print(f"\nEl autor del primer libro es: {libro1.autor}")
print(f"El ISBN del segundo libro es: {libro2.isbn}")
```

poo_eje2.py > ...

```python
# 4. Llamar al método mostrar_info() para cada uno de los objetos
print("\n--- Mostrando información completa ---")
libro1.mostrar_info()
libro2.mostrar_info()

# 5. Añadir métodos de comportamiento (ya implementados arriba) ✓

# 6. Prueba los nuevos métodos con tus objetos libro1 y libro2
print("\n--- Probando métodos de comportamiento ---")

# Prestar libro1
libro1.prestar_libro()

# Intentar prestar libro1 otra vez (ya prestado)
libro1.prestar_libro()

# Devolver libro1
libro1.devolver_libro()

# Intentar devolver libro1 otra vez (ya disponible)
libro1.devolver_libro()

print("\n--- Probando con libro2 ---")

# Prestar libro2
libro2.prestar_libro()

# Devolver libro2
libro2.devolver_libro()

print("\n--- Estado final de los libros ---")
```

```python
print("\n--- Estado final de los libros ---")
libro1.mostrar_info()
libro2.mostrar_info()
print("Adaniel Balderrama- FIN DEL PROGRAMA")
```

Prueba en la consola

```
El autor del primer libro es: Antoine de Saint-Exupéry
El ISBN del segundo libro es: 978-99905-2-213-9

--- Mostrando información completa ---
==================================================
INFORMACIÓN DEL LIBRO
==================================================
Título: El Principito
Autor: Antoine de Saint-Exupéry
ISBN: 978-3-14-046401-7
Páginas: 120
Disponible: Sí
==================================================
==================================================
INFORMACIÓN DEL LIBRO
==================================================
Título: Raza de Bronce
Autor: Alcides Arguedas
ISBN: 978-99905-2-213-9
Páginas: 250
Disponible: Sí
==================================================

--- Probando métodos de comportamiento ---
El libro 'El Principito' ha sido prestado.
El libro 'El Principito' ya está prestado.
El libro 'El Principito' ha sido devuelto.
El libro 'El Principito' ya estaba disponible.

--- Probando con libro2 ---
El libro 'Raza de Bronce' ha sido prestado.
El libro 'Raza de Bronce' ha sido devuelto.
```

```
--- Probando métodos de comportamiento ---
El libro 'El Principito' ha sido prestado.
El libro 'El Principito' ya está prestado.
El libro 'El Principito' ha sido devuelto.
El libro 'El Principito' ya estaba disponible.

--- Probando con libro2 ---
El libro 'Raza de Bronce' ha sido prestado.
El libro 'Raza de Bronce' ha sido devuelto.

--- Estado final de los libros ---
==================================================
INFORMACIÓN DEL LIBRO
==================================================
Título: El Principito
Autor: Antoine de Saint-Exupéry
ISBN: 978-3-14-046401-7
Páginas: 120
Disponible: Sí
==================================================
==================================================
INFORMACIÓN DEL LIBRO
==================================================
Título: Raza de Bronce
Autor: Alcides Arguedas
ISBN: 978-99905-2-213-9
Páginas: 250
Disponible: Sí
==================================================
Adaniel Balderrama- FIN DEL PROGRAMA
PS D:\ingenieria en sistemas\Ingenieria de sistemas\tercer semestre\Programacion II\Practicas python 2>
```

Nombre del programa : clase 14 poo ejemplo 3

poo_eje3.py > ...

```python
class Libro:
    """
    Clase que representa un libro con sus atributos principales.
    """

    def __init__(self, titulo, autor, isbn, paginas):
        """
        Constructor de la clase Libro.

        Args:
            titulo (str): Título del libro
            autor (str): Autor del libro
            isbn (str): ISBN del libro
            paginas (int): Número de páginas del libro
        """
        self.titulo = titulo
        self.autor = autor
        self.isbn = isbn
        self.paginas = paginas
        self.disponible = True

    def mostrar_info(self):
        """
        Método que imprime todos los atributos del libro de forma clara y formateada.
        """
        print("=" * 50)
        print("INFORMACIÓN DEL LIBRO")
        print("=" * 50)
        print(f"Título: {self.titulo}")
        print(f"Autor: {self.autor}")
        print(f"ISBN: {self.isbn}")
        print(f"Páginas: {self.paginas}")
        print(f"Disponible: {'Sí' if self.disponible else 'No'}")
        print("=" * 50)


if __name__ == "__main__":
    # Crear objetos de tipo Libro
    libro1 = Libro("Cien años de soledad", "Gabriel García Márquez", "978-0307474728", 417)
    libro2 = Libro("1984", "George Orwell", "978-0451524935", 328)
    libro3 = Libro("El Principito", "Antoine de Saint-Exupéry", "978-0156013987", 96)

    # Crear una lista vacía
    mi_biblioteca = []

    # Añadir libros a la lista
    mi_biblioteca.append(libro1)
    mi_biblioteca.append(libro2)
    mi_biblioteca.append(libro3)

    # Mostrar el inventario completo
    print("\n\n--- INVENTARIO COMPLETO DE LA BIBLIOTECA ---")
    for libro_actual in mi_biblioteca:
        libro_actual.mostrar_info()
        print("=" * 20)  # Separador
print("Adaniel Balderrama- FIN DEL PROGRAMA")
```

Prueba en consola