

# 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

```
suma_total_matriz.py > ...
1  # Paso 1: Inicialización
2  acumulador_total = 0
3
4  # Supongamos que esta es la matriz que queremos procesar
5  matriz = [
6      [1, 2, 3],
7      [4, 5, 6],
8      [7, 8, 9]
9  ]
10
11 # Paso 2 y 3: Iteración Anidada
12 for fila in matriz:
13     for elemento in fila:
14         # Paso 4: Acumulación
15         acumulador_total += elemento
16
17 # Paso 5: Resultado
18 print("La suma total de todos los elementos es:", acumulador_total)
19 print("Adaniel Balderrama- FIN DEL PROGRAMA")
20
```

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 total de todos los elementos es: 45
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 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 > ...
1  # Paso 1: Inicialización
2  sumas_por_fila = []
3
4  # Supongamos que esta es la matriz que queremos procesar
5  matriz = [
6      [1, 2, 3],
7      [4, 5, 6],
8      [7, 8, 9]
9  ]
10
11 # Paso 2: Iteración exterior (por fila)
12 for fila in matriz:
13     # Paso 3a: Inicializar acumulador de fila
14     acumulador_fila = 0
15
16     # Paso 3b y 3c: Recorrer elementos y acumular
17     for elemento in fila:
18         acumulador_fila += elemento
19
20     # Paso 4: Guardar resultado de la fila
21     sumas_por_fila.append(acumulador_fila)
22
23 # Paso 5: Resultado final
24 print("La suma de cada fila es:", sumas_por_fila)
25 print("Adaniel Balderrama- FIN DEL PROGRAMA")
26
```

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 > ...
1  # Supongamos que esta es la matriz que queremos procesar
2  matriz = [
3      [1, 2, 3],
4      [4, 5, 6],
5      [7, 8, 9]
6  ]
7
8  # Paso 1: Inicialización
9  num_filas = len(matriz)
10 num_columnas = len(matriz[0])
11 sumas_por_columna = [0] * num_columnas
12
13 # Paso 2: Iteración anidada con índices
14 for i in range(num_filas):
15     for j in range(num_columnas):
16         # Paso 3: Acumulación por índice
17         sumas_por_columna[j] += matriz[i][j]
18
19 # Paso 4: Resultado final
20 print("La suma de cada columna es:", sumas_por_columna)
21 print("Adaniel Balderrama- FIN DEL PROGRAMA")
22
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
1  # Definimos la función que suma todos los elementos de una matriz
2  def sumar_total_matriz(matriz):
3      """
4      Esta función recibe una matriz (lista de listas)
5      y retorna la suma total de todos sus elementos.
6      Ejemplo:
7      matriz = [[1, 2], [3, 4]]
8      resultado = 10
9      """
10     total = 0
11     for fila in matriz:
12         for elemento in fila:
13             total += elemento
14     return total
15 # Función para probar que sumar_total_matriz funciona correctamente
16 def probar_suma_total():
17     print("Probando sumar_total_matriz...")
18     # Caso 1: matriz normal
19     m1 = [[5, 2, 3], [4, 5, 6]]
20     assert sumar_total_matriz(m1) == 25 # 1+2+3+4+5+6 = 21
21     # Caso 2: matriz con negativos y ceros
22     m2 = [[-1, 0, 1], [10, -5, 5]]
23     assert sumar_total_matriz(m2) == 10 # -1+0+1+10-5+5 = 10
24     # Casos borde o límites
25     assert sumar_total_matriz([]) == 0 # Matriz con una fila vacía
26     assert sumar_total_matriz([[]]) == 0 # Matriz completamente vacía
27     assert sumar_total_matriz([[42]]) == 42 # Matriz de un solo elemento
28     print("¡Pruebas para sumar_total_matriz pasaron! ")
29     # Llamamos a la función de pruebas
30     probar_suma_total()
31     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 > ...
1  # Definimos la función que suma los elementos por cada fila de la matriz
2  def sumar_por_filas(matriz):
3      """
4      Esta función recibe una matriz (lista de listas)
5      y devuelve una lista con la suma de cada fila.
6      Ejemplo:
7      matriz = [[1, 2, 3], [4, 5, 6]]
8      resultado = [6, 15]
9      """
10     resultado = []
11     for fila in matriz:
12         suma_fila = sum(fila) # Suma todos los elementos de la fila
13         resultado.append(suma_fila)
14     return resultado
15
16 # Función de prueba para verificar que sumar_por_filas funciona correctamente
17 def probar_suma_por_filas():
18     print("\nProbando sumar_por_filas...")
19     # Caso 1: matriz con 3 filas y 3 columnas
20     m1 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
21     assert sumar_por_filas(m1) == [6, 15, 24] # 1+2+3, 4+5+6, 7+8+9
22     # Caso 2: matriz con pares repetidos
23     m2 = [[10, 10], [20, 20], [30, 30]]
24     assert sumar_por_filas(m2) == [20, 40, 60]
25     # Caso borde: matriz vacía
26     assert sumar_por_filas([]) == [] # No hay filas que sumar
27     print("¡Pruebas para sumar_por_filas pasaron!")
28
29 # Llamamos a la función para ejecutar las pruebas
30 probar_suma_por_filas()
31 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 > ...
1  # Definimos la función que suma los elementos de la diagonal principal de una matriz cuadrada
2  def sumar_diagonal_principal(matriz):
3      """
4      Esta función recibe una matriz cuadrada (misma cantidad de filas y columnas)
5      y retorna la suma de los elementos en su diagonal principal.
6      Ejemplo:
7      matriz = [[1, 2],
8                [3, 4]]
9      diagonal principal: 1 y 4 → suma = 5
10     """
11     suma = 0
12     for i in range(len(matriz)):
13         suma += matriz[i][i] # Accede al elemento en la posición (i, i)
14     return suma
15     # Función de prueba para verificar que sumar_diagonal_principal funciona correctamente
16     def probar_suma_diagonal_principal():
17         print("\nProbando sumar_diagonal_principal...")
18         # Caso 1: matriz 3x3 con números consecutivos
19         m1 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
20         assert sumar_diagonal_principal(m1) == 15 # 1 + 5 + 9
21         # Caso 2: matriz 2x2 con ceros y valores definidos
22         m2 = [[10, 0], [0, 20]]
23         assert sumar_diagonal_principal(m2) == 30 # 10 + 20
24         # Caso borde: matriz 1x1
25         m3 = [[5]]
26         assert sumar_diagonal_principal(m3) == 5 # Solo un elemento en la diagonal
27         print("¡Pruebas para sumar_diagonal_principal pasaron! ")
28         # Llamamos a la función para ejecutar las pruebas
29         probar_suma_diagonal_principal()
30     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:\ingeniería en sistemas\Ingeniería 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

```
sumar_diagonal_secundaria.py > ...
5  # Función que suma los elementos de la diagonal secundaria de una matriz cuadrada
6  def sumar_diagonal_secundaria(matriz):
7      """
8      Recibe una matriz cuadrada (misma cantidad de filas y columnas)
9      y devuelve la suma de los elementos en la diagonal secundaria.
10     La diagonal secundaria va desde la esquina superior derecha
11     hasta la esquina inferior izquierda.
12     Por ejemplo, en una matriz 3x3:
13         [[a, b, c],
14          [d, e, f],
15          [g, h, i]]
16     La diagonal secundaria está en las posiciones: (0,2), (1,1), (2,0)
17     y su suma sería: c + e + g
18     """
19     n = len(matriz) # Número de filas (y columnas, ya que es cuadrada)
20     suma = 0
21     for i in range(n):
22         suma += matriz[i][n - 1 - i] # Accede al elemento en la posición (i, n-1-i)
23     return suma
24
25 # Función de pruebas para validar que sumar_diagonal_secundaria funciona correctamente
26 def probar_suma_diagonal_secundaria():
27     print("\nProbando sumar_diagonal_secundaria...")
28
29     # Caso 1: matriz 3x3 normal
30     m1 = [
31         [1, 2, 3],
32         [4, 5, 6],
33         [7, 8, 9]
34     ]
35     # Diagonal secundaria: 3 + 5 + 7 = 15
36     assert sumar_diagonal_secundaria(m1) == 15
```

```
sumar_diagonal_secundaria.py > ...
26 def probar_suma_diagonal_secundaria():
36     assert sumar_diagonal_secundaria(m1) == 15
37
38     # Caso 2: matriz 2x2
39     m2 = [
40         [10, 1],
41         [2, 20]
42     ]
43     # Diagonal secundaria: 1 + 2 = 3
44     assert sumar_diagonal_secundaria(m2) == 3
45
46     # Caso 3: matriz 1x1
47     m3 = [[42]]
48     # Solo hay un elemento: 42
49     assert sumar_diagonal_secundaria(m3) == 42
50
51     print("¡Pruebas para sumar_diagonal_secundaria pasaron! ✅")
52
53 # Llamamos a la función de prueba
54 probar_suma_diagonal_secundaria()
55 print("Adaniel Balderrama- FIN DEL PROGRAMA")

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PUERTOS

PS D:\ingeniería en sistemas\Ingeniería de sistemas\tercer semestre\Programación II\Prácticas python 2> & C:/Python313/python.exe "d:/ingeniería en sistemas/Ingeniería de sistemas/tercer semestre/Programación II/Prácticas python 2/sumar_diagonal_secundaria.py"
Probando sumar_diagonal_secundaria...
¡Pruebas para sumar_diagonal_secundaria pasaron! ✅
Adaniel Balderrama- FIN DEL PROGRAMA
PS D:\ingeniería en sistemas\Ingeniería de sistemas\tercer semestre\Programación II\Prácticas 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 > ...
1 def transponer_matriz(matriz):
2     if not matriz or not matriz[0]:
3         return []
4     num_filas = len(matriz)
5     num_columnas = len(matriz[0])
6     # Inicializamos la transpuesta la tranpuesta con la estructura correcta (o la construimos dinamicamente)
7     matriz_transpuesta = []
8     for j in range(num_columnas): # El bucle exterior itera sobre las columnas originales
9         nueva_fila = []
10        for i in range (num_filas): # El bucle interior itera sobre las Filas originales
11            nueva_fila.append(matriz[i][j])
12            matriz_transpuesta.append(nueva_fila)
13    return matriz_transpuesta
14
15
16 # Prueba tu funcion rigurosamente (¡incluye matrices no cuadradas!):
17
18 m1 = [[1, 2, 3 ], [4, 5, 6]] #2x3
19 t1 = transponer_matriz (m1)
20 assert t1 == [[1,4],[2,5],[3,6]] # Debe ser 3x2
21 print ("Prueba 1 (2x3) pasada !")
22 # Añade mas pruebas con assert ...
23 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/darthon313/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 > ...
1  # Función que verifica si una matriz es simétrica
2  def es_simetrica(matriz):
3      # Requisito 1: Debe ser cuadrada
4      num_filas = len(matriz)
5      if num_filas == 0:
6          return True # Una matriz vacía es trivialmente simétrica
7
8      for i in range(num_filas):
9          if len(matriz[i]) != num_filas:
10             return False # No es cuadrada
11
12     # Requisito 2: Comparar matriz[i][j] con matriz[j][i]
13     for i in range(num_filas):
14         for j in range(i + 1, num_filas): # Solo chequear la triangular superior
15             if matriz[i][j] != matriz[j][i]:
16                 return False # ¡Con una diferencia es suficiente!
17
18     return True # Si no encontramos diferencias, es simétrica
19
20 # Pruebas de la función
21 sim = [
22     [1, 7, 3],
23     [7, -4, -5],
24     [3, -5, 6]
25 ]
26
27 no_sim = [
28     [1, 2, 3],
29     [4, 5, 6],
30     [7, 8, 9]
31 ]
32
33 no_cuadrada = [
34     [1, 2],
35     [3, 4],
36     [5, 6]
37 ]
38
39 assert es_simetrica(sim) == True
40 assert es_simetrica(no_sim) == False
41 assert es_simetrica(no_cuadrada) == False
42
43 print("¡Pruebas para es_simetrica pasaron! ✅")
44 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:/Python313/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
1  # Función 1: Crear la sala llena de 'L' (libres)
2  def crear_sala(filas, columnas):
3      return [['L' for _ in range(columnas)] for _ in range(filas)]
4
5  # Función 2: Mostrar visualmente la sala
6  def mostrar_sala(sala):
7      print(" " + " ".join(f"{i}" for i in range(len(sala[0])))) # Números de columna
8      for i, fila in enumerate(sala):
9          print(f"{i:2} " + " ".join(fila))
10
11 # Función 3: Ocupar un asiento
12 def ocupar_asiento(sala, fila, columna):
13     filas = len(sala)
14     columnas = len(sala[0])
15
16     # Verifica si coordenadas son válidas
17     if 0 <= fila < filas and 0 <= columna < columnas:
18         if sala[fila][columna] == 'L':
19             sala[fila][columna] = 'O'
20             print(f"Asiento ({fila}, {columna}) ocupado exitosamente.")
21             return True
22         else:
23             print("Ese asiento ya está ocupado.")
24             return False
25     else:
26         print("Coordenadas fuera de rango.")
27         return False
28
29 # Función 4 (Opcional): Contar asientos libres
30 def contar_asientos_libres(sala):
31     return sum(fila.count('L') for fila in sala)
```

```

salaCine.py > ocupar_asiento
33 # Programa Principal
34 def main():
35     filas, columnas = 5, 8 # Tamaño predeterminado
36     sala = crear_sala(filas, columnas)
37
38     while True:
39         print("\n🎬 Estado actual de la sala de cine:")
40         mostrar_sala(sala)
41         print(f"Asientos libres: {contar_asientos_libres(sala)}")
42         print("\nMenú:")
43         print("1. Ocupar asiento")
44         print("0. Salir")
45
46         opcion = input("Elige una opción: ")
47
48         if opcion == '1':
49             try:
50                 fila = int(input("Ingresa el número de fila: "))
51                 columna = int(input("Ingresa el número de columna: "))
52                 ocupar_asiento(sala, fila, columna)
53             except ValueError:
54                 print("❌ Entrada inválida. Por favor ingresa números.")
55         elif opcion == '0':
56             print("👋 Gracias por usar el sistema de reserva. ¡Hasta luego!")
57             break
58         else:
59             print("❌ Opción no válida. Intenta nuevamente.")
60
61     # Ejecutar el programa principal
62     if __name__ == "__main__":
63         main()
64     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 0 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

```
class_diccionarios.py > ...
1  producto = {'codigo': 'P002', 'nombre': 'Coca Cola', 'precio': 15, 'stock': 50}
2  print("\n--- Claves del producto ---")
3  for clave in producto: # Por defecto, itera sobre las CLAVES
4      print(clave)
5  print("\n--- Clave y Valor ---")
6  for clave in producto:
7      valor = producto[clave]
8      print(f"{clave.capitalize()}: {valor}")
9
10 producto1 = {'codigo': 'P003', 'nombre': 'Pizza', 'precio': 54, 'stock': 25}
11 print("\n--- Claves del producto ---")
12 for clave in producto1: # Por defecto, itera sobre las CLAVES
13     print(clave)
14 print("\n--- Clave y Valor ---")
15 for clave in producto1:
16     valor = producto1[clave]
17     print(f"{clave.capitalize()}: {valor}")
18
19 producto2 = {'codigo': 'P004', 'nombre': 'Hamburguesa', 'precio': 25, 'stock': 75}
20 print("\n--- Claves del producto ---")
21 for clave in producto2: # Por defecto, itera sobre las CLAVES
22     print(clave)
23 print("\n--- Clave y Valor ---")
24 for clave in producto2:
25     valor = producto2[clave]
26     print(f"{clave.capitalize()}: {valor}")
27 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 > ...
1  # 1. Definición de la función
2  def es_simetrica(matriz):
3      # Requisito 1: Debe ser cuadrada
4      num_filas = len(matriz)
5      if num_filas == 0:
6          return True # Una matriz vacía es trivialmente simétrica
7
8      for i in range(num_filas):
9          if len(matriz[i]) != num_filas:
10             return False # No es cuadrada
11
12     # Requisito 2: Comparar matriz[i][j] con matriz[j][i]
13     for i in range(num_filas):
14         for j in range(i + 1, num_filas): # Solo chequear la triangular superior
15             if matriz[i][j] != matriz[j][i]:
16                 return False # ¡Con una diferencia es suficiente!
17
18     return True # Si no encontramos diferencias, es simétrica
19
20 # 2. Pruebas
21 sim = [
22     [1, 7, 3],
23     [7, -4, -5],
24     [3, -5, 6]
25 ]
26
27 no_sim = [
28     [1, 2, 3],
29     [4, 5, 6],
30     [7, 8, 9]
31 ]
```

```
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 > ...
1  # Paso 1: Variables Globales
2  lista_de_tareas = []
3  proximo_id_tarea = 1 # Para generar IDs únicos
4
5
6  # Paso 2: Implementar agregar_tarea
7  def agregar_tarea(descripcion, prioridad="media"):
8      global proximo_id_tarea # ¡Necesario para modificar una variable global!
9      nueva_tarea = {
10         "id": proximo_id_tarea,
11         "descripcion": descripcion,
12         "completada": False,
13         "prioridad": prioridad
14     }
15     lista_de_tareas.append(nueva_tarea)
16     proximo_id_tarea += 1
17     print(f"✅ Tarea '{descripcion}' añadida con éxito.")
18
19
20 # Paso 3: Implementar mostrar_tareas
21 def mostrar_tareas():
22     print("\n--- LISTA DE TAREAS ---")
23     if not lista_de_tareas:
24         print("¡No hay tareas pendientes! ¡A disfrutar!")
25         return
26
27     for tarea in lista_de_tareas:
28         estado = "✓" if tarea["completada"] else "⌚"
29         print(f"{estado} ID: {tarea['id']} | {tarea['descripcion']} (Prioridad: {tarea['prioridad']})")
30     print("-----")
```

```
# 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 python
✅ 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 > ...
1  # clase13_todolist.py
2
3  # Paso 1: Variables Globales
4  lista_de_tareas = []
5  proximo_id_tarea = 1 # Para generar IDs únicos
6
7
8  # Paso 2: Implementar agregar_tarea
9  def agregar_tarea(descripcion, prioridad="media"):
10     global proximo_id_tarea
11     nueva_tarea = {
12         "id": proximo_id_tarea,
13         "descripcion": descripcion,
14         "completada": False,
15         "prioridad": prioridad
16     }
17     lista_de_tareas.append(nueva_tarea)
18     proximo_id_tarea += 1
19     print(f"✅ Tarea '{descripcion}' añadida con éxito.")
20
21
22 # Paso 3: Implementar mostrar_tareas
23 def mostrar_tareas():
24     print("\n--- LISTA DE TAREAS ---")
25     if not lista_de_tareas:
26         print("¡No hay tareas pendientes! ¡A disfrutar!")
27         return
28
29     for tarea in lista_de_tareas:
30         estado = "✓" if tarea["completada"] else "⌚"
31         print(f"{estado} ID: {tarea['id']} | {tarea['descripcion']} (Prioridad: {tarea['prioridad']})")
32     print("-----")
```

```
clase13_todolist complet.py > ...
35 # Paso 4: Implementar buscar_tarea_por_id
36 def buscar_tarea_por_id(id_buscado):
37     for tarea in lista_de_tareas:
38         if tarea["id"] == id_buscado:
39             return tarea
40     return None
41
42
43 # Paso 5: Implementar marcar_tarea_completada
44 def marcar_tarea_completada(id_tarea):
45     tarea = buscar_tarea_por_id(id_tarea)
46     if tarea:
47         tarea["completada"] = True
48         print(f"✅ Tarea '{tarea['descripcion']}' marcada como completada.")
49     else:
50         print(f"❌ Error: No se encontró la tarea con ID {id_tarea}.")
51
52
53 # Paso 6: Implementar eliminar_tarea
54 def eliminar_tarea(id_tarea):
55     tarea = buscar_tarea_por_id(id_tarea)
56     if tarea:
57         lista_de_tareas.remove(tarea)
58         print(f"🗑️ Tarea '{tarea['descripcion']}' eliminada.")
59     else:
60         print(f"❌ Error: No se encontró la tarea con ID {id_tarea}.")
61
62
63 # -----
64 # Prueba de todo el flujo
65 # -----
```



```

53 # -----
54 # Prueba de todo el flujo
55 # -----
56 agregar_tarea("Estudiar para el examen de Cálculo")
57 agregar_tarea("Hacer las compras", prioridad="alta")
58
59 mostrar_tareas()
60
61 marcar_tarea_completada(1)
62 mostrar_tareas() # Debería mostrar la tarea 1 como completada
63
64 eliminar_tarea(2)
65 mostrar_tareas() # La tarea 2 ya no debería aparecer
66
67 marcar_tarea_completada(99) # Probar con un ID que no existe
68 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:/Python313/python.exe "d:/ingenieria en sistemas/Ingenieria de sistemas/tercer semestre/Programacion II/Practicas python 2.py"
✓ 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_ej1.py > ...
1  class Libro:
2      """
3      Clase que representa un libro con sus atributos principales.
4      """
5
6      def __init__(self, titulo, autor, isbn, paginas):
7          """
8          Constructor de la clase Libro.
9
10         Args:
11             titulo (str): Título del libro
12             autor (str): Autor del libro
13             isbn (str): ISBN del libro
14             paginas (int): Número de páginas del libro
15         """
16         # Crear los atributos de instancia correspondientes
17         self.titulo = titulo
18         self.autor = autor
19         self.isbn = isbn
20         self.paginas = paginas
21
22         # Atributo extra que se inicializa siempre por defecto
23         self.disponible = True
24
25     def mostrar_info(self):
26         """
27         Método que imprime todos los atributos del libro de forma clara y formateada.
28         """
29         print("=" * 50)
30         print("INFORMACIÓN DEL LIBRO")
31         print("=" * 50)
32         print(f"Título: {self.titulo}")
33
34         print(f"Autor: {self.autor}")
35         print(f"ISBN: {self.isbn}")
36         print(f"Páginas: {self.paginas}")
37         print(f"Disponible: {'Sí' if self.disponible else 'No'}")
38         print("=" * 50)
39
40     # Ejemplo de uso (no te preocupes por crear objetos todavía, ¡solo define la clase!)
41     if __name__ == "__main__":
42         # Creación de ejemplo para demostrar el funcionamiento
43         libro1 = Libro("Cien años de soledad", "Gabriel García Márquez", "978-0307474728", 417)
44         libro1.mostrar_info()
45
46         # Cambiar disponibilidad
47         libro1.disponible = False
48         print("\nDespués de cambiar disponibilidad:")
49         libro1.mostrar_info()
50
51     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 > ...
1  # Ejercicio 2: Creación de instancias y métodos de comportamiento
2
3  class Libro:
4      """
5      Clase que representa un libro con sus atributos principales.
6      """
7
8      def __init__(self, titulo, autor, isbn, paginas):
9          """
10         Constructor de la clase Libro.
11         """
12         self.titulo = titulo
13         self.autor = autor
14         self.isbn = isbn
15         self.paginas = paginas
16         self.disponible = True
17
18     def mostrar_info(self):
19         """
20         Método que imprime todos los atributos del libro de forma clara y formateada.
21         """
22         print("=" * 50)
23         print("INFORMACIÓN DEL LIBRO")
24         print("=" * 50)
25         print(f"Título: {self.titulo}")
26         print(f"Autor: {self.autor}")
27         print(f"ISBN: {self.isbn}")
28         print(f"Páginas: {self.paginas}")
29         print(f"Disponible: {'Sí' if self.disponible else 'No'}")
30         print("=" * 50)
31
32     def prestar_libro(self):
33         """
34         Método para prestar el libro. Cambia disponible a False si está disponible.
35         """
36         if self.disponible == True:
37             self.disponible = False
38             print(f"El libro '{self.titulo}' ha sido prestado.")
39         else:
40             print(f"El libro '{self.titulo}' ya está prestado.")
41
42     def devolver_libro(self):
43         """
44         Método para devolver el libro. Cambia disponible a True si estaba prestado.
45         """
46         if self.disponible == False:
47             self.disponible = True
48             print(f"El libro '{self.titulo}' ha sido devuelto.")
49         else:
50             print(f"El libro '{self.titulo}' ya estaba disponible.")
51
52
53 # 1. Definición de clase Libro (arriba) ✓
54
55 # 2. Crear dos objetos Libro diferentes
56 libro1 = Libro("El Principito", "Antoine de Saint-Exupéry", "978-3-14-046401-7", 120)
57 libro2 = Libro("Raza de Bronce", "Alcides Arguedas", "978-99905-2-213-9", 250)
58
59 # 3. Acceder y mostrar algunos de sus atributos directamente
60 print(f"\nEl autor del primer libro es: {libro1.autor}")
61 print(f"El ISBN del segundo libro es: {libro2.isbn}")
```

poo\_eje2.py > ...

```
62
63 # 4. Llamar al método mostrar_info() para cada uno de los objetos
64 print("\n--- Mostrando información completa ---")
65 libro1.mostrar_info()
66 libro2.mostrar_info()
67
68 # 5. Añadir métodos de comportamiento (ya implementados arriba) ✓
69
70 # 6. Prueba los nuevos métodos con tus objetos libro1 y libro2
71 print("\n--- Probando métodos de comportamiento ---")
72
73 # Prestar libro1
74 libro1.prestar_libro()
75
76 # Intentar prestar libro1 otra vez (ya prestado)
77 libro1.prestar_libro()
78
79 # Devolver libro1
80 libro1.devolver_libro()
81
82 # Intentar devolver libro1 otra vez (ya disponible)
83 libro1.devolver_libro()
84
85 print("\n--- Probando con libro2 ---")
86
87 # Prestar libro2
88 libro2.prestar_libro()
89
90 # Devolver libro2
91 libro2.devolver_libro()
92
93 print("\n--- Estado final de los libros ---")
```

```
93 print("\n--- Estado final de los libros ---")
94 libro1.mostrar_info()
95 libro2.mostrar_info()
96 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:\Ingeniería en sistemas\Ingeniería de sistemas\tercer semestre\Programacion II\Practicas python 2>
```

### Nombre del programa : clase 14 poo ejemplo 3

```
poo_eje3.py > ...
1  class Libro:
2      """
3      Clase que representa un libro con sus atributos principales.
4      """
5
6      def __init__(self, titulo, autor, isbn, paginas):
7          """
8          Constructor de la clase Libro.
9
10         Args:
11             titulo (str): Título del libro
12             autor (str): Autor del libro
13             isbn (str): ISBN del libro
14             paginas (int): Número de páginas del libro
15         """
16         self.titulo = titulo
17         self.autor = autor
18         self.isbn = isbn
19         self.paginas = paginas
20         self.disponible = True
21
22     def mostrar_info(self):
23         """
24         Método que imprime todos los atributos del libro de forma clara y formateada.
25         """
26         print("=" * 50)
27         print("INFORMACIÓN DEL LIBRO")
28         print("=" * 50)
29         print(f"Título: {self.titulo}")
30         print(f"Autor: {self.autor}")
31         print(f"ISBN: {self.isbn}")
32         print(f"Páginas: {self.paginas}")
33
34         print(f"Disponible: {'Sí' if self.disponible else 'No'}")
35         print("=" * 50)
36
37     if __name__ == "__main__":
38         # Crear objetos de tipo Libro
39         libro1 = Libro("Cien años de soledad", "Gabriel García Márquez", "978-0307474728", 417)
40         libro2 = Libro("1984", "George Orwell", "978-0451524935", 328)
41         libro3 = Libro("El Principito", "Antoine de Saint-Exupéry", "978-0156013987", 96)
42
43         # Crear una lista vacía
44         mi_biblioteca = []
45
46         # Añadir libros a la lista
47         mi_biblioteca.append(libro1)
48         mi_biblioteca.append(libro2)
49         mi_biblioteca.append(libro3)
50
51         # Mostrar el inventario completo
52         print("\n\n--- INVENTARIO COMPLETO DE LA BIBLIOTECA ---")
53         for libro_actual in mi_biblioteca:
54             libro_actual.mostrar_info()
55             print("=" * 20) # Separador
56         print("Adaniel Balderrama- FIN DEL PROGRAMA")
```

## Prueba en consola

```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PUERTOS

--- INVENTARIO COMPLETO DE LA BIBLIOTECA ---
=====
INFORMACIÓN DEL LIBRO
=====
Título: Cien años de soledad
Autor: Gabriel García Márquez
ISBN: 978-0307474728
Páginas: 417
Disponible: Sí
=====
=====
=====
INFORMACIÓN DEL LIBRO
=====
Título: 1984
Autor: George Orwell
ISBN: 978-0451524935
Páginas: 328
Disponible: Sí
=====
=====
=====
INFORMACIÓN DEL LIBRO
=====
Título: El Principito
Autor: Antoine de Saint-Exupéry
ISBN: 978-0156013987
Páginas: 96
Disponible: Sí
=====
=====
Adaniel Balderrama- FIN DEL PROGRAMA
```

Nombre del programa : clase 15 archivos

```
clase15_archivos.py > ...
1  # Paso 1: Definimos el nombre del archivo
2  nombre_archivo = "mi_diario.txt"
3
4  # Paso 2 y 3: Abrimos el archivo en modo escritura y escribimos entradas de diario
5  with open(nombre_archivo, 'w') as diario_file:
6      diario_file.write("Querido diario,\n")
7      diario_file.write("Hoy aprendí sobre archivos en Python.\n")
8      diario_file.write("El modo 'w' borra todo antes de escribir. ¡Qué miedo!\n")
9      diario_file.write("Pero también es muy útil para guardar información nueva.\n")
10
11 # Paso 4: Confirmamos que la escritura se completó
12 print("¡Se ha escrito el diario correctamente en", nombre_archivo, "!")
13 print("Adaniel Balderrama - FIN DEL PROGRAMA")
```

Prueba de consola:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Prueba la nueva tecnología PowerShell multiplataforma https://aka.ms/pscore6

PS D:\pruebas> & C:/Users/SCPC114/AppData/Local/Programs/Python/Python312/python.exe d:/pruebas/clase15_archivos.py
¡Se ha escrito el diario correctamente en mi_diario.txt !
Adaniel Balderrama - FIN DEL PROGRAMA
PS D:\pruebas>
```

Este código sencillo me ayudó a entender los principios básicos del trabajo con archivos en Python. Me di cuenta de que escribir en archivos no es complicado, pero sí requiere atención, especialmente por los efectos del modo 'w'. También valoro el uso de `with`, ya que facilita la gestión de recursos de forma segura. aprendí a crear un archivo de texto, escribir contenido dentro de él y asegurarme de que todo se maneje correctamente sin errores. Esto es muy útil para guardar datos de manera permanente, como registros, diarios, o configuraciones en cualquier tipo de programa.



## Nombre del programa : clase 15 archivos, ejercicio 2

```
class 15 archivos, ejercicio 2.py > ...
1  # --- Ejercicio 1: Escribir el diario ---
2  nombre_archivo = "mi_diario.txt"
3
4  with open(nombre_archivo, 'w') as diario_file:
5      diario_file.write("Querido diario,\n")
6      diario_file.write("Hoy aprendí sobre archivos en Python.\n")
7      diario_file.write("El modo 'w' borra todo antes de escribir. ¡Qué miedo!\n")
8      diario_file.write("Pero también es muy útil para guardar información nueva.\n")
9
10 print("¡Se ha escrito el diario correctamente en", nombre_archivo, "!")
11
12 # --- Ejercicio 2: Leer el diario ---
13
14 # Opción A: Leer todo de golpe (comentada)
15 # with open(nombre_archivo, 'r') as diario_file:
16 #     contenido = diario_file.read()
17 # print("\n--- Contenido completo del diario ---")
18 # print(contenido)
19
20 # Opción B: Leer línea por línea con manejo de errores
21 print("\n--- Contenido del diario (línea por línea) ---")
22 try:
23     with open(nombre_archivo, 'r') as diario_file:
24         for linea in diario_file:
25             print(linea.strip()) # .strip() elimina los saltos de línea extra
26 except FileNotFoundError:
27     print(f"Error: El archivo '{nombre_archivo}' no existe.")
28 print("Adaniel Balderrama - FIN DEL PROGRAMA")
```

Prueba de consola:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

--- Contenido del diario (línea por línea) ---
Querido diario,
Hoy aprendí sobre archivos en Python.
El modo 'w' borra todo antes de escribir. ¡Qué miedo!
Pero también es muy útil para guardar información nueva.
Adaniel Balderrama - FIN DEL PROGRAMA
PS D:\pruebas>
```

## Nombre del programa : clase 15 archivos, ejercicio 3

```
clase 15 archivos, ejercicio 3.py > ...
1  # --- Ejercicio 1: Escribir el diario desde cero (modo 'w') ---
2  nombre_archivo = "mi_diario.txt"
3
4  with open(nombre_archivo, 'w') as diario_file:
5      diario_file.write("Querido diario,\n")
6      diario_file.write("Hoy aprendí sobre archivos en Python.\n")
7      diario_file.write("El modo 'w' borra todo antes de escribir. ¡Qué miedo!\n")
8      diario_file.write("Pero también es muy útil para guardar información nueva.\n")
9
10 print("¡Se ha escrito el diario correctamente en", nombre_archivo, "!")
11
12 # --- Ejercicio 2: Leer el diario (línea por línea con manejo de errores) ---
13
14 # Opción A: Leer todo de golpe (comentada)
15 # with open(nombre_archivo, 'r') as diario_file:
16 #     contenido = diario_file.read()
17 # print("\n--- Contenido completo del diario ---")
18 # print(contenido)
19
20 print("\n--- Contenido del diario (línea por línea) ---")
21 try:
22     with open(nombre_archivo, 'r') as diario_file:
23         for línea in diario_file:
24             print(línea.strip()) # Elimina espacios y saltos de línea extra
25 except FileNotFoundError:
26     print(f"Error: El archivo '{nombre_archivo}' no existe.")
27
28 # --- Ejercicio 3: Añadir nuevas entradas (modo 'a') ---
29 print("\nAñadiendo nuevas entradas al diario...")
30 with open(nombre_archivo, 'a') as diario_file:
31     diario_file.write("\n--- Entrada del 20 de Junio de 2025 ---\n")
32     diario_file.write("El modo 'a' es genial para no perder datos.\n")
33     diario_file.write("Ahora mi diario puede crecer cada día.\n")
34 print("¡Nuevas entradas guardadas!")
```

```
36 # Verificación final: Releer el diario con Opción B (línea por línea)
37 print("\n--- Contenido del diario actualizado ---")
38 try:
39     with open(nombre_archivo, 'r') as diario_file:
40         for línea in diario_file:
41             print(línea.strip())
42 except FileNotFoundError:
43     print(f"Error: El archivo '{nombre_archivo}' no existe.")
44
45 print("Adaniel Balderrama - FIN DEL PROGRAMA")
```

## Prueba de consola:

```
--- Contenido del diario (línea por línea) ---
Querido diario,
Hoy aprendí sobre archivos en Python.
El modo 'w' borra todo antes de escribir. ¡Qué miedo!
Pero también es muy útil para guardar información nueva.

Añadiendo nuevas entradas al diario...
¡Nuevas entradas guardadas!

--- Contenido del diario actualizado ---
Querido diario,
Hoy aprendí sobre archivos en Python.
El modo 'w' borra todo antes de escribir. ¡Qué miedo!
Pero también es muy útil para guardar información nueva.

--- Entrada del 20 de Junio de 2025 ---
El modo 'a' es genial para no perder datos.
Ahora mi diario puede crecer cada día.
Adaniel Balderrama - FIN DEL PROGRAMA
PS D:\pruebas>
```

## Nombre del programa : Clase 15 Desafío

```
class15_desafio.py > ...
1 # -----
2 # ✅ Ejercicio 1: Crear y escribir diario
3 # -----
4
5 # Paso 1: Definimos el nombre del archivo
6 nombre_archivo = "mi_diario.txt"
7
8 # Paso 2: Creamos el archivo desde cero (modo 'w') y escribimos líneas predefinidas
9 with open(nombre_archivo, "w") as diario_file:
10     diario_file.write("Querido diario,\n")
11     diario_file.write("Hoy aprendí sobre archivos en Python.\n")
12     diario_file.write("El modo 'w' borra todo antes de escribir. ¡Qué miedo!\n")
13     diario_file.write("¡Pero también es muy útil para comenzar desde cero!\n")
14
15 # Confirmamos al usuario
16 print("✅ Diario creado y primeras entradas guardadas.")
17
18 # -----
19 # ✅ Ejercicio 2: Leer el diario (modo A y B)
20 # -----
21
22 # Opción A: Leer todo el contenido de una sola vez
23 print("\n--- Contenido completo del diario (modo A - read()) ---")
24 try:
25     with open(nombre_archivo, "r") as diario_file:
26         contenido = diario_file.read()
27         print(contenido)
28 except FileNotFoundError:
29     print(f"❌ Error: El archivo '{nombre_archivo}' no existe.")
30
31 # Opción B: Leer el archivo línea por línea (más controlado)
32 print("\n--- Contenido del diario (modo B - línea por línea) ---")
33 try:
34     with open(nombre_archivo, "r") as diario_file:
35         for linea in diario_file:
36             print(linea.strip()) # .strip() elimina los saltos de línea '\n'
37 except FileNotFoundError:
38     print(f"❌ Error: El archivo '{nombre_archivo}' no existe.")
39
40 # -----
41 # ✅ Ejercicio 3: Añadir entrada desde input()
42 # -----
43
44 # Aquí el usuario puede ingresar su propia entrada personalizada
45 print("\n📝 Vamos a añadir una nueva entrada a tu diario.")
46 print("Escribe tu entrada (usa varias líneas si quieres). Escribe una línea vacía para terminar.")
47
48 # Usamos una lista para recoger múltiples líneas del usuario
49 lineas_usuario = []
50 while True:
51     linea = input()
52     if linea == "":
53         break # Al ingresar una línea vacía, se termina la entrada
54     lineas_usuario.append(linea)
55
56 # Añadimos las nuevas líneas al diario, en modo 'a' para no borrar lo anterior
57 print("\n📝 Añadiendo tu entrada al diario...")
58 with open(nombre_archivo, "a") as diario_file:
59     diario_file.write("\n--- Nueva entrada personalizada ---\n")
60     for linea in lineas_usuario:
61         diario_file.write(linea + "\n")
62
63 print("✅ ¡Tu entrada fue añadida con éxito!")
64
65 # -----
66 # ✅ Verificación final: Leer todo el contenido actualizado
67 # -----
68
69 print("\n📋 Verificando el contenido final del diario (modo B)...")
70 try:
71     with open(nombre_archivo, "r") as diario_file:
72         for linea in diario_file:
73             print(linea.strip())
74 except FileNotFoundError:
75     print(f"❌ Error: El archivo '{nombre_archivo}' no existe.")
76
77 print("Adaniel Balderrama - FIN DEL PROGRAMA")
```

Prueba en consola :

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.
Follow link (ctrl + click)

Prueba la nueva tecnología PowerShell multiplataforma https://aka.ms/pscore6

PS D:\pruebas> & C:/Users/SCPC114/AppData/Local/Programs/Python/Python312/python.exe d:/pruebas/clase15_desafio.py
✅ Diario creado y primeras entradas guardadas.

--- Contenido completo del diario (modo A - read()) ---
Querido diario,
Hoy aprendí sobre archivos en Python.
El modo 'w' borra todo antes de escribir. ¡Qué miedo!
¡Pero también es muy útil para comenzar desde cero!

--- Contenido del diario (modo B - línea por línea) ---
Querido diario,
Hoy aprendí sobre archivos en Python.
El modo 'w' borra todo antes de escribir. ¡Qué miedo!
¡Pero también es muy útil para comenzar desde cero!

📝 Vamos a añadir una nueva entrada a tu diario.
Escribe tu entrada (usa varias líneas si quieres). Escribe una línea vacía para terminar.
```

Fecha y hora actual: 2025-06-26 09:31:07