

Practico III

Dead Services

Integrantes:

Jose Alejandro Zabala Romero

Leonardo Montenegro

Junior Pacajes Banegas

Nicodemus Berry Quiroga Gonzales

Lucas Mateo Diaz Badani

Richard Iver Hurtado Cotocari

Materia: Programación II

Docente: Jimmy Nataniel Requena

```
the end -add back the deselected mirror mod
```

```
1
```

```
objects.active = modifier_ob
str(modifier_ob)) # modifier ob is the active object
selected_objects[0]
mymirror.select = 1
```

Índice del contenido

Capítulo I.....	4
1.1 Ordenamiento Mergesort	4
1.2 Ordenamiento Burbuja e Inserción.....	6
1.3 Ordenamiento con Heap Sort.....	8
Capítulo II	9
2.1 Introducción a las Matrices.....	9
2.1.1 Matrices Tablero.....	9
2.1.2 Modelando un Teclado Numérico	11
2.1.3 Imprimir la Matriz como una Cuadrícula	12
2.1.4 bucles anidados	13
2.1.5 batalla naval	14
2.1.6 batalla naval extendido	17
2.2 Operaciones Básicas con Matrices.....	20
2.2.1 Suma Total y por Ejes	20
2.2.2 Suma todinigos	21
2.2.3 Suma por columnas	22
2.2.4 suma por filas.....	23
2.2.5 Encontrar el Elemento Máximo/Mínimo.....	24
2.3 Análisis Matricial	25
2.3.1 Función Suma Total.....	25
2.3.2 Función de Suma por Filas	26
2.3.3 Suma de la Diagonal Principal	27
2.3.4 Función que suma los elementos de la diagonal secundaria	28
2.3.5 Suma los elementos de la diagonal secundaria de una matriz cuadrada	29
2.3.6 Matrix Pathfinder.....	30
2.4 Implementación de Algoritmos para el Manejo de Matrices.....	31
2.4.1 Función para Transponer una Matriz.....	31
2.4.2 Función para Verificar Identidad.....	32
2.4.3 Función para Verificar Simetría	33
2.4.4 Gestión de Asientos de Cine.....	34

Capítulo III.....	36
3.1 Introducción a los Registros	36
3.1.1 Nuestro Estudiante	36
3.1.2 Recorriendo un Diccionario con un Bucle for.....	36
3.1.3 Modelando un Producto.....	37
3.1.4 Creando un Inventario	38
3.1.5 Modelando el Mundo Real con Diccionarios	39
3.2. Funciones y Operaciones Avanzadas con Registros	40
3.2.1 Mini-Aplicación: La "To-Do List.....	40
3.2.2 Gestor de Contactos 2.0.....	45
3.3 Introducción a Clases y Objetos	47
3.3.1 Métodos	47
3.3.2 Refactorizando un Libro de Diccionario a Clase.....	48
3.3.3 Creación de instancias y métodos de comportamiento.....	49
3.3.4 Usando Listas de Objetos	51
3.3.5 Diseño de Clases.....	53
Capítulo IV	55
4.1 Introducción al Manejo de Archivos (Texto).....	55
4.1.1 Creando y Escribiendo Nuestro Diario.....	55
4.1.2 Leer Nuestro Diario	56
4.1.3 Continuar Nuestro Diario	57
4.1.4 Diario Persistente.....	58
4.2 Guardando y Cargando Estructuras de Datos	59
4.2.1 CSV (Valores Separados por Comas)	59
4.2.2 JSON (JavaScript Object Notation).....	62
4.3 Implementa Programas Utilizando Archivos	65
4.3.1 Preparando Nuestro Código Base.....	65
4.3.2 Implementando la Función de Carga.....	65
4.3.3 Implementando la Función de Guardado.....	66
4.3.4 Funciones adicionales.....	67
4.3.5 Programa principal	68

Capítulo I

Métodos de Ordenamiento más Eficientes

1.1 Ordenamiento Mergesort

```

1 def merge_sort(lista):
2     # Paso Vencer (Condición Base de la Recursividad):
3     if len(lista) <= 1:
4         return lista
5     # Paso 1: DIVIDIR
6     medio = len(lista) // 2
7     mitad_izquierda = lista[:medio]
8     mitad_derecha = lista[medio:]
9     # Paso 2: VENCER (Recursión)
10    izquierda_ordenada = merge_sort(mitad_izquierda)
11    derecha_ordenada = merge_sort(mitad_derecha)
12    # Paso 3: COMBINAR
13    print(f"Mezclaría {izquierda_ordenada} y {derecha_ordenada}")
14    return merge(izquierda_ordenada, derecha_ordenada)
15 def merge(izquierda, derecha):
16     resultado = []
17     i = j = 0
18     # Comparar elementos de izquierda y derecha uno por uno
19     while i < len(izquierda) and j < len(derecha):
20         if izquierda[i] < derecha[j]:
21             resultado.append(izquierda[i])
22             i += 1
23         else:
24             resultado.append(derecha[j])
25             j += 1
26     # Agregar cualquier elemento restante
27     resultado.extend(izquierda[i:])
28     resultado.extend(derecha[j:])
29     return resultado
30 # --- Prueba ---
31 lista_prueba = [8, 3, 5, 1]
32 print("Lista original:", lista_prueba)
33 resultado = merge_sort(lista_prueba)
34 print("Lista ordenada:", resultado)
35 # --- Pruebas automatizadas ---
36 assert merge_sort([]) == []                      # Lista vacía
37 assert merge_sort([1]) == [1]                      # Lista con un solo elemento
38 assert merge_sort([5, 2]) == [2, 5]                # Lista con dos elementos
39 assert merge_sort([3, 1, 2]) == [1, 2, 3]          # Lista con tres elementos desordenados
40 assert merge_sort([10, 5, 3, 8, 6, 2]) == [2, 3, 5, 6, 8, 10] # Lista par
41 assert merge_sort([9, 7, 5, 3, 1]) == [1, 3, 5, 7, 9] # Lista en orden descendente
42 assert merge_sort([1, 2, 3, 4, 5]) == [1, 2, 3, 4, 5] # Lista ya ordenada
43 assert merge_sort([4, 2, 2, 4, 1]) == [1, 2, 2, 4, 4] # Lista con elementos repetidos
44 assert merge_sort([-100, -50, 0, 50, -100]) == [-100, -50, 0, 50, 100] # Lista con enteros negativos y positivos
45 assert merge_sort([2.5, 1.2, 3.8]) == [1.2, 2.5, 3.8] # Lista con flotantes
46 print("¡Todas las pruebas con assert pasaron correctamente!")
47 print("Jose Alejandro Zabala Romero - FIN DEL PROGRAMA")

```

El código implementa el algoritmo de ordenamiento Merge Sort, que sigue la estrategia divide y vencerás. La función merge_sort divide recursivamente la lista en mitades hasta que cada sublistas tiene uno o cero elementos (caso base). Luego, combina estas sublistas ordenadas usando la función merge, que fusiona dos listas ordenadas comparando elemento por elemento. Además, el código incluye pruebas con assert que verifican su correcto funcionamiento en distintos casos, incluyendo listas vacías, ordenadas, con repetidos, negativos y flotantes.

Ejecución

```
Listado original: [8, 3, 5, 1]
Mezclaría [8] y [3]
Mezclaría [5] y [1]
Mezclaría [3, 8] y [1, 5]
Listado ordenada: [1, 3, 5, 8]
Mezclaría [5] y [2]
Mezclaría [1] y [2]
Mezclaría [3] y [1, 2]
Mezclaría [5] y [3]
Mezclaría [10] y [3, 5]
Mezclaría [6] y [2]
Mezclaría [8] y [2, 6]
Mezclaría [3, 5, 10] y [2, 6, 8]
Mezclaría [9] y [7]
Mezclaría [3] y [1]
Mezclaría [5] y [1, 3]
Mezclaría [7, 9] y [1, 3, 5]
Mezclaría [1] y [2]
Mezclaría [4] y [5]
Mezclaría [3] y [4, 5]
Mezclaría [1, 2] y [3, 4, 5]
Mezclaría [4] y [2]
Mezclaría [4] y [1]
Mezclaría [2] y [1, 4]
Mezclaría [2, 4] y [1, 2, 4]
Mezclaría [100] y [-50]
Mezclaría [50] y [-100]
Mezclaría [0] y [-100, 50]
Mezclaría [-50, 100] y [-100, 0, 50]
Mezclaría [1.2] y [3.8]
Mezclaría [2.5] y [1.2, 3.8]
¡Todas las pruebas con assert pasaron correctamente!
Jose Alejandro Zabala Romero - FIN DEL PROGRAMA
```

1.2 Ordenamiento Burbuja e Inserción

```

1 def ordenamiento_burbuja(lista):
2     """
3         Ordena una lista en orden ascendente utilizando el algoritmo de burbuja.
4         Modifica la lista original (in-place) y también la retorna por conveniencia.
5     """
6     n = len(lista)
7     for i in range(n - 1):
8         hubo_intercambio = False
9         for j in range(n - 1 - i):
10            if lista[j] > lista[j + 1]:
11                lista[j], lista[j + 1] = lista[j + 1], lista[j]
12                hubo_intercambio = True
13            if not hubo_intercambio:
14                break
15     return lista
16 if __name__ == "__main__":
17     numeros = [6, 3, 8, 2, 5]
18     print("Antes:", numeros)
19     ordenamiento_burbuja(numeros)
20     print("Después Ordenamiento Burbuja:", numeros)
21 # --- Pruebas burbuja ---
22 lista1 = [6, 3, 8, 2, 5]
23 ordenamiento_burbuja(lista1)
24 assert lista1 == [2, 3, 5, 6, 8]
25 lista2 = [1, 2, 3, 4, 5]
26 ordenamiento_burbuja(lista2)
27 assert lista2 == [1, 2, 3, 4, 5]
28 lista3 = [5, 4, 3, 2, 1]
29 ordenamiento_burbuja(lista3)
30 assert lista3 == [1, 2, 3, 4, 5]
31 lista4 = [5, 1, 4, 2, 8, 5, 2]
32 ordenamiento_burbuja(lista4)
33 assert lista4 == [1, 2, 2, 4, 5, 5, 8]
34 assert ordenamiento_burbuja([]) == []
35 assert ordenamiento_burbuja([42]) == [42]
36 print("¡Todas las pruebas ordenamiento burbuja pasaron! ✓")
37 print("JOSE ALEJANDRO ZABALA ROMERO - FIN PROGRAMA ORDENAMIENTO BURBUJA")

```

```

38 # ALGORITMO DE INSERCIÓN
39 def ordenamiento_insercion(lista):
40     """
41         Ordena la lista in-place usando el algoritmo de inserción.
42         Retorna la misma lista por conveniencia.
43     """
44     for i in range(1, len(lista)):
45         valor_actual = lista[i]
46         posicion_actual = i
47         while posicion_actual > 0 and lista[posicion_actual - 1] > valor_actual:
48             lista[posicion_actual] = lista[posicion_actual - 1]
49             posicion_actual -= 1
50         lista[posicion_actual] = valor_actual
51     return lista
52 # --- Pruebas inserción ---
53 lista1 = [6, 3, 8, 2, 5]
54 print("Antes:", lista1)
55 ordenamiento_insercion(lista1)
56 print("Después Ordenamiento Inserción:", lista1)
57 assert lista1 == [2, 3, 5, 6, 8]
58 lista2 = [1, 2, 3, 4, 5]
59 print("Antes:", lista2)
60 ordenamiento_insercion(lista2)
61 print("Después Ordenamiento Inserción:", lista2)
62 assert lista2 == [1, 2, 3, 4, 5]
63 lista3 = [5, 4, 3, 2, 1]
64 ordenamiento_insercion(lista3)
65 assert lista3 == [1, 2, 3, 4, 5]
66 lista4 = [5, 1, 4, 2, 8, 5, 2]
67 ordenamiento_insercion(lista4)
68 assert lista4 == [1, 2, 2, 4, 5, 5, 8]
69 assert ordenamiento_insercion([]) == []
70 assert ordenamiento_insercion([42]) == [42]
71 print("¡Todas las pruebas del ordenamiento por inserción pasaron! ✅")
72 print("JOSE ALEJANDRO ZABALA ROMERO - FIN PROGRAMA ORDENAMIENTO INSERCIÓN")

```

Este código implementa dos algoritmos de ordenamiento: burbuja e inserción, utilizando listas, bucles for, condicionales y control de índices. Ambos métodos ordenan los elementos in-place (sin crear nuevas listas) y están bien estructurados. La parte compleja fue entender cómo los elementos se intercambian o desplazan dentro del ciclo interno, especialmente al manejar condiciones como si ya está ordenado o hay elementos repetidos.

Ejecución

```

~/.../Práctico_1/busquedas$ python ordenamiento_burbuja.py
Antes: [6, 3, 8, 2, 5]
Después Ordenamiento Burbuja: [2, 3, 5, 6, 8]
¡Todas las pruebas ordenamiento burbuja pasaron! ✅
JOSE ALEJANDRO ZABALA ROMERO - FIN PROGRAMA ORDENAMIENTO BURBUJA
Antes: [6, 3, 8, 2, 5]
Después Ordenamiento Inserción: [2, 3, 5, 6, 8]
Antes: [1, 2, 3, 4, 5]
Después Ordenamiento Inserción: [1, 2, 3, 4, 5]
¡Todas las pruebas del ordenamiento por inserción pasaron! ✅
JOSE ALEJANDRO ZABALA ROMERO - FIN PROGRAMA ORDENAMIENTO INSERCIÓN

```

1.3 Ordenamiento con Heap Sort

```

1 def heapificar(lista, tamano_heap, indice_raiz):
2     """ Asegura que el subárbol con raíz en el índice dado cumpla la propiedad de Max Heap. """
3     mayor = indice_raiz
4     hijo_izquierdo = 2 * indice_raiz + 1
5     hijo_derecho = 2 * indice_raiz + 2
6     if hijo_izquierdo < tamano_heap and lista[hijo_izquierdo] > lista[mayor]:
7         mayor = hijo_izquierdo
8     if hijo_derecho < tamano_heap and lista[hijo_derecho] > lista[mayor]:
9         mayor = hijo_derecho
10    if mayor != indice_raiz:
11        lista[indice_raiz], lista[mayor] = lista[mayor], lista[indice_raiz]
12        heapificar(lista, tamano_heap, mayor)
13 def ordenar_por_heap(lista):
14     """ Ordena la lista utilizando el algoritmo Heap Sort. """
15     n = len(lista)
16     # Construir el Max Heap
17     for i in range(n // 2 - 1, -1, -1):
18         heapificar(lista, n, i)
19     # Extraer elementos del heap uno por uno
20     for i in range(n - 1, 0, -1):
21         lista[0], lista[i] = lista[i], lista[0]
22         heapificar(lista, i, 0)
23 # PRUEBAS CON ASSERTS (TESTING)
24 # Prueba 1: Lista desordenada
25 numeros1 = [12, 11, 13, 5, 6, 7]
26 ordenar_por_heap(numeros1)
27 assert numeros1 == sorted(numeros1), f" Error en números1: {numeros1}"
28 # Prueba 2: Lista ya ordenada
29 numeros2 = [1, 2, 3, 4, 5]
30 ordenar_por_heap(numeros2)
31 assert numeros2 == [1, 2, 3, 4, 5], f" Error en números2: {numeros2}"
32 # Prueba 3: Lista en orden inverso
33 numeros3 = [9, 8, 7, 6, 5]
34 ordenar_por_heap(numeros3)
35 assert numeros3 == [5, 6, 7, 8, 9], f" Error en números3: {numeros3}"
36 # Prueba 4: Lista con elementos repetidos
37 numeros4 = [3, 1, 2, 3, 1]
38 ordenar_por_heap(numeros4)
39 assert numeros4 == sorted(numeros4), f" Error en números4: {numeros4}"
40 # Prueba 5: Lista con un solo elemento
41 numeros5 = [42]
42 ordenar_por_heap(numeros5)
43 assert numeros5 == [42], f" Error en números5: {numeros5}"
44 # Prueba 6: Lista vacía
45 numeros6 = []
46 ordenar_por_heap(numeros6)
47 assert numeros6 == [], f" Error en números6: {numeros6}"
48 print(" Todas las pruebas pasaron correctamente.")
49 print("Fin del programa----Jose Alejandro Zabala Romero")

```

Este programa implementa el algoritmo de ordenamiento Heap Sort usando un enfoque basado en montículos (Max Heap). Incluye pruebas automáticas con listas variadas. La parte compleja fue entender la función 'heapificar', que ajusta el subárbol para mantener la propiedad del heap.

Ejecución

```

~/.../Practico_I/busquedas$ python heapsort.py
Todas las pruebas pasaron correctamente.
Fin del programa----Jose Alejandro Zabala Romero

```

Capítulo II

Arreglos Bidimensionales (Matrices)

2.1 Introducción a las Matrices

2.1.1 Matrices Tablero

```

1  # Definimos una matriz de 3 filas y 4 columnas
2  matriz = [
3      [10, 20, 30, 40],  # Fila 0
4      [50, 60, 70, 80],  # Fila 1
5      [90, 91, 92, 93]   # Fila 2
6  ]
7  # Acceder al número 70
8  valor = matriz[1][2]
9  print(f"El valor es: {valor}")  # Resultado: 70
10 # Modificar el valor 90 por un 0
11 matriz[2][0] = 0
12 print("\n--- Recorrido con índices ---")
13 num_filas = len(matriz)
14 num_columnas = len(matriz[0])
15 for i in range(num_filas):
16     for j in range(num_columnas):
17         elemento = matriz[i][j]
18         print(f"Elemento en ({i},{j}) es {elemento}")
19 print("\n--- Recorrido Pythonico ---")
20 for fila_actual in matriz:
21     for elemento in fila_actual:
22         print(elemento, end=" ")
23     print() # Salto de línea entre filas
24 # --- TABLERO DE TRES EN RAYA ---
25 print("\n--- Tablero de Tres en Raya ---")
26 tablero = [
27     ['X', 'O', 'X'],
28     [' ', 'X', 'O'],
29     ['O', ' ', ' ']
30 ]
31 for i, fila in enumerate(tablero):
32     print(" | ".join(fila))
33     if i < len(tablero) - 1:
34         print("-" * 9)
35 print("fin del programa -----Richard hurtado")

```

Primero se creó una matriz de 3x4 y se accedió a un valor específico usando índices. También se modificó un elemento (el 90 por un 0), lo que me ayudó a entender que las matrices se pueden cambiar directamente. Luego se recorrió la matriz de dos formas: con bucles anidados usando índices (lo cual fue un poco confuso al principio porque tenía que entender bien qué representa cada i y j), y después de forma más sencilla recorriendo directamente cada fila.

Una de las partes que más me costó fue visualizar cómo se organizan los datos en filas y columnas, especialmente al imprimirlas con sus posiciones. Finalmente, se arma un tablero de Tres en Raya con otra matriz y usé " | ".join(fila) para darle formato, lo cual también me tomó algo de tiempo entender cómo funcionaba esa línea. El código está dividido en partes claras: creación, modificación, recorrido y representación visual.

Ejecución

```
~/.Practico_II/matrices$ python matrices_tablero.py
El valor es: 70

--- Recorrido con índices ---
Elemento en (0,0) es 10
Elemento en (0,1) es 20
Elemento en (0,2) es 30
Elemento en (0,3) es 40
Elemento en (1,0) es 50
Elemento en (1,1) es 60
Elemento en (1,2) es 70
Elemento en (1,3) es 80
Elemento en (2,0) es 0
Elemento en (2,1) es 91
Elemento en (2,2) es 92
Elemento en (2,3) es 93

--- Recorrido Pythonico ---
10 20 30 40
50 60 70 80
0 91 92 93

--- Tablero de Tres en Raya ---
X | O | X
-----|-----|-----
| X | O
-----|-----|
O | |
fin del programa -----Richard hurtado
~/.Practico_II/matrices$
```

2.1.2 Modelando un Teclado Numérico

```

1  matriz = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
2  for fila_actual in matriz:
3      for elemento in fila_actual:
4          print(elemento, end=" ")
5      print()
6
7  print()
8
9  matriz = [[col + fila * 5 + 1 for col in range(5)] for fila in range(5)]
10 for fila in matriz:
11     print(" ".join(str(elem) for elem in fila))
12
13 print()
14
15 teclado_numerico = [['1', '2', '3'], ['4', '5', '6'], ['7', '8', '9'],
16                  ['*', '0', '#']]
17 for fila in teclado_numerico:
18     print(" ".join(fila))
19
20 print("Fin del Programa ----- Jose Alejandro Zabala Romero")

```

En el código se practicaron distintas formas de crear y recorrer matrices. Primero se utilizó una matriz 3x3 con números del 1 al 9, recorrida con bucles anidados e imprimiendo los elementos en una sola línea usando `end=" "`. Luego se generó una matriz 5x5 con comprensión de listas, lo cual fue complejo al principio, pero permitió entender cómo crear matrices con números secuenciales. Finalmente, se representó un teclado numérico en forma de matriz e imprimió cada fila con `" ".join(fila)`

Ejecución

```

~/.../Practico_II/matrices$ python matrices_teclado.py
1 2 3
4 5 6
7 8 9

1 2 3 4 5
6 7 8 9 10
11 12 13 14 15
16 17 18 19 20
21 22 23 24 25

1 2 3
4 5 6
7 8 9
* 0 #

Fin del Programa ----- Jose Alejandro Zabala Romero

```

2.1.3 Imprimir la Matriz como una Cuadrícula

```

1 # 1. Una matriz de 3x3 llamada teclado que representa un teclado numérico
2 teclado = [
3     [1, 2, 3],
4     [4, 5, 6],
5     [7, 8, 9]
6 ]
7 # 2. Imprimir la matriz completa
8 print("Matriz original:")
9 for fila in teclado:
10     print(fila)
11 # 3. Mostrar valores específicos usando doble índice
12 print(f"\nNúmero en el centro: {teclado[1][1]}")           # fila 1, columna 1 → 5
13 print(f"Número en la esquina inferior derecha: {teclado[2][2]}") # fila 2, columna 2 → 9
14 # 4. Modificar el número en la esquina superior izquierda (fila 0, columna 0)
15 teclado[0][0] = 0
16 # 5. Imprimir la matriz nuevamente para ver el cambio
17 print("\nMatriz modificada:")
18 for fila in teclado:
19     print(fila)
20 print("\nFin del programa ----- leonardo montenegro")

```

En este código se trabajó con una matriz de 3x3 llamada teclado, que representa un teclado numérico del 1 al 9. Primero se imprimió toda la matriz usando un bucle for. Luego se accedió a elementos específicos usando índices dobles, como el número central (5) y el número en la esquina inferior derecha (9). Esta parte me ayudó a entender mejor cómo se ubican los datos por filas y columnas, aunque al principio me confundía un poco cuál índice representaba la fila y cuál la columna. Después de modificar el valor de la esquina superior izquierda (el 1) y reemplazarlo por un 0. Finalmente, se vuelve a imprimir la matriz para verificar que el cambio se haya aplicado correctamente.

Ejecución

```

Matriz original:
[1, 2, 3]
[4, 5, 6]
[7, 8, 9]

Número en el centro: 5
Número en la esquina inferior derecha: 9

Matriz modificada:
[0, 2, 3]
[4, 5, 6]
[7, 8, 9]

Fin del programa ----- leonardo montenegro

```

2.1.4 bucles anidados

```

1 # Recorriendo Matrices: La Lógica de los Bucles Anidados
2 # Para visitar cada celda de una matriz, necesitamos un bucle ¡DENTRO de otro bucle!
3 matriz = [
4     [1, 2, 3],
5     [4, 5, 6],
6     [7, 8, 9]
7 ]
8 # Opción 1: Recorriendo con índices
9 num_filas = len(matriz)
10 num_columnas = len(matriz[0])
11
12 for i in range(num_filas):      # Bucle exterior para índices de fila (0, 1, 2)
13     for j in range(num_columnas): # Bucle interior para índices de columna (0, 1, 2)
14         elemento = matriz[i][j]
15         print(f"Elemento en ({i},{j}) es {elemento}")
16 # Opción 2: Recorriendo por elemento (más "Pythonico")
17 print("\nRecorrido pythonico:")
18 for fila_actual in matriz:      # Bucle exterior toma cada lista-fila
19     for elemento in fila_actual: # Bucle interior toma cada elemento de esa fila
20         print(elemento, end=" ") # "end=' '" para imprimir en la misma línea
21     print() # Un salto de línea después de cada fila para formatear
22 print("\nFin del programa ----- richard hurtado")

```

En este código se aprendió a recorrer matrices con bucles anidados. Primero se usó una matriz 3x3 y se mostró dos formas de recorrerla: una con índices (`matriz[i][j]`) que permite conocer la posición exacta de cada elemento, y otra más directa y "pythónica", que recorre filas y valores sin usar índices. La primera opción ayudó a entender la estructura interna de la matriz, aunque al inicio fue difícil visualizar el avance de los índices. La segunda fue más sencilla y legible, ideal cuando no se necesita la posición.

Ejecución

```

Elemento en (0,0) es 1
Elemento en (0,1) es 2
Elemento en (0,2) es 3
Elemento en (1,0) es 4
Elemento en (1,1) es 5
Elemento en (1,2) es 6
Elemento en (2,0) es 7
Elemento en (2,1) es 8
Elemento en (2,2) es 9

Recorrido pythonico:
1 2 3
4 5 6
7 8 9

Fin del programa ----- richard hurtado

```

2.1.5 batalla naval

```

1 import random
2 # Tamaño del tablero (por ejemplo 5x5)
3 FILAS = 5
4 COLUMNAS = 5
5 # Crear una matriz con ceros
6 def crear_tablero():
7     return [[0 for _ in range(COLUMNAS)] for _ in range(FILAS)]
8 # Mostrar el tablero en consola
9 def mostrar_tablero(tablero, ocultar_barcos=False):
10    print(" " + " ".join(str(i + 1) for i in range(COLUMNAS)))
11    for i, fila in enumerate(tablero):
12        letra = chr(ord('A') + i)
13        fila_mostrar = []
14        for celda in fila:
15            if ocultar_barcos and celda == 1:
16                fila_mostrar.append("0")
17            elif celda == 0:
18                fila_mostrar.append("0")
19            elif celda == 1:
20                fila_mostrar.append("1")
21            elif celda == 2:
22                fila_mostrar.append("X")
23            elif celda == 3:
24                fila_mostrar.append("*")
25        print(letra + " " + " ".join(fila_mostrar))
26 # Convertir coordenada tipo "A3" a [fila][columna]
27 def coord_a_indices(coord):
28    fila = ord(coord[0].upper()) - ord('A')
29    columna = int(coord[1:]) - 1
30    return fila, columna
31 # Colocar barcos aleatoriamente en el tablero
32 def colocar_barcos(tablero, cantidad):
33    colocados = 0
34    while colocados < cantidad:
35        fila = random.randint(0, FILAS - 1)
36        columna = random.randint(0, COLUMNAS - 1)
37        if tablero[fila][columna] == 0:
38            tablero[fila][columna] = 1
39            colocados += 1
40 # Ejecutar un disparo
41 def disparar(tablero_objetivo, tablero_disparos, coord):
42    fila, columna = coord_a_indices(coord)
43    if tablero_objetivo[fila][columna] == 1:
44        tablero_objetivo[fila][columna] = 2
45        tablero_disparos[fila][columna] = 2
46        print("¡Tocado!")
47    elif tablero_objetivo[fila][columna] in [0, 3]:
48        tablero_objetivo[fila][columna] = 3
49        tablero_disparos[fila][columna] = 3
50        print("Agua...")
51    else:
52        print("Ya disparaste allí.")
53 # Comprobar si hay barcos vivos
54 def quedan_barcos(tablero):
55    for fila in tablero:
56        if 1 in fila:

```

```

57         return True
58     return False
59 # Juego principal
60 def juego():
61     print("== Batalla Naval ==")
62     tablero_jugador = crear_tablero()
63     tablero_cpu = crear_tablero()
64     tablero_disparos_jugador = crear_tablero()
65     tablero_disparos_cpu = crear_tablero()
66     # Colocar barcos
67     colocar_barcos(tablero_jugador, 3)
68     colocar_barcos(tablero_cpu, 3)
69     turno = 1
70     while quedan_barcos(tablero_jugador) and quedan_barcos(tablero_cpu):
71         print(f"\n--- Turno {turno} ---")
72         print("Tu tablero:")
73         mostrar_tablero(tablero_jugador)
74         print("Tus disparos:")
75         mostrar_tablero(tablero_disparos_jugador)
76         # Turno del jugador
77         coord = input("Ingresa coordenada para disparar (ej. A3): ")
78         disparar(tablero_cpu, tablero_disparos_jugador, coord)
79         # Turno del CPU (simplemente al azar)
80         while True:
81             fila_cpu = random.randint(0, FILAS - 1)
82             col_cpu = random.randint(0, COLUMNAS - 1)
83             if tablero_jugador[fila_cpu][col_cpu] in [0, 1]:
84                 break
85         coord_cpu = f"{chr(ord('A') + fila_cpu)}{col_cpu + 1}"
86         print(f"La CPU dispara a {coord_cpu}")
87         disparar(tablero_jugador, tablero_disparos_cpu, coord_cpu)
88         turno += 1
89         if quedan_barcos(tablero_jugador):
90             print("¡Ganaste!")
91         else:
92             print("La CPU gana.")
93         print("\nFin del programa Jose Alejandro Zabala Romero")
94 # Ejecutar el juego
95 juego()

```

Este código implementa un juego modular, dividido en funciones para organizar cada parte del proceso. Se definen funciones para crear y mostrar el tablero, colocar barcos aleatoriamente, convertir coordenadas del usuario en índices, y gestionar disparos. La función principal controla los turnos y verifica si quedan barcos activos. Aunque fue complicado entender cómo se representan los valores en la matriz (0: vacío, 1: barco, 2: tocado, 3: agua) y evitar errores como disparos repetidos,

Ejecución

```

== Batalla Naval ==
--- Turno 1 ---
Tu tablero:
 1 2 3 4 5
A 0 0 0 0 0
B 0 0 1 0 0
C 0 0 0 0 0
D 0 0 0 0 0
E 0 1 0 0 1
Tus disparos:
 1 2 3 4 5
A 0 0 0 0 0
B 0 0 0 0 0
C 0 0 0 0 0
D 0 0 0 0 0
E 0 0 0 0 0
Ingresa coordenada para disparar (ej. A3): a1
Agua...
La CPU dispara a D5
Agua...
--- Turno 2 ---
Tu tablero:
 1 2 3 4 5
A 0 0 0 0 0
B 0 0 1 0 0
C 0 0 0 0 0
D 0 0 0 0 *
E 0 1 0 0 1
Tus disparos:
 1 2 3 4 5
A * 0 0 0 0
B 0 0 0 0 0
C 0 0 0 0 0
D 0 0 0 0 0
E 0 0 0 0 0
La CPU dispara a E3
Agua...
--- Turno 7 ---
Tu tablero:
 1 2 3 4 5
A 0 0 0 * *
B 0 * 1 0 0
C * 0 0 0 0
D 0 0 0 0 *
E 0 1 * 0 1
Tus disparos:
 1 2 3 4 5
A * 0 * 0 *
B * 0 * 0 X
C 0 0 0 0 0
D 0 0 0 0 0
E 0 0 0 0 0
Ingresa coordenada para disparar (ej. A3): c1
Agua...
La CPU dispara a C4
Agua...
--- Turno 8 ---
Tu tablero:
 1 2 3 4 5
A 0 0 0 * *
B 0 * 1 0 0
C * 0 0 * 0
D 0 0 0 0 *
E 0 1 * 0 1
Tus disparos:
 1 2 3 4 5
A * 0 * 0 *
B * 0 * 0 X
C * 0 * 0 *
D 0 * * 0 *
E * 0 * * X
Ingresa coordenada para disparar (ej. A3): a2
¡Tocado!
La CPU dispara a D1
Agua...
¡Ganaste!
Fin del programa Jose Alejandro Zabala Romero

```

Al ejecutar el código del juego “Batalla Naval”, se inicializan los tableros del jugador y la CPU, con tres barcos colocados aleatoriamente para cada uno. En cada turno, se muestra el número de turno, el tablero del jugador y el tablero de disparos. El jugador ingresa coordenadas como “C4” y el juego responde si fue un acierto o no. Luego la CPU realiza su disparo aleatorio y también se informa el resultado. Las ubicaciones de los barcos enemigos no se muestran al jugador, manteniendo el desafío del juego.

2.1.6 batalla naval extendido

```

1 import random
2 FILAS = 4
3 COLUMNAS = 2
4 BARCOS = 3
5 # Crear un tablero vacío
6 def crear_tablero():
7     return [[0 for _ in range(COLUMNAS)] for _ in range(FILAS)]
8 # Mostrar el tablero
9 def mostrar_tablero(tablero, ocultar_barcos=False):
10    print(" " + " ".join(str(i + 1) for i in range(COLUMNAS)))
11    for i, fila in enumerate(tablero):
12        letra = chr(ord('A') + i)
13        fila_mostrar = []
14        for celda in fila:
15            if ocultar_barcos and celda == 1:
16                fila_mostrar.append("0")
17            elif celda == 0:
18                fila_mostrar.append("0")
19            elif celda == 1:
20                fila_mostrar.append("1")
21            elif celda == 2:
22                fila_mostrar.append("X")
23            elif celda == 3:
24                fila_mostrar.append("*")
25        print(letra + " " + " ".join(fila_mostrar))
26 # Convertir coordenada tipo "A3" a índices [fila][columna]
27 def coord_a_indices(coord):
28     fila = ord(coord[0].upper()) - ord('A')
29     columna = int(coord[1:]) - 1
30     return fila, columna
31 # Colocar barcos aleatoriamente
32 def colocar_barcos(tablero, cantidad):
33     colocados = 0
34     while colocados < cantidad:
35         fila = random.randint(0, FILAS - 1)
36         columna = random.randint(0, COLUMNAS - 1)
37         if tablero[fila][columna] == 0:
38             tablero[fila][columna] = 1
39             colocados += 1
40 # Ejecutar disparo con validación
41 def disparar(tablero_objetivo, tablero_disparos, coord, nombre):
42     try:
43         fila, columna = coord_a_indices(coord)
44     except (IndexError, ValueError):
45         print(f"{nombre}, formato de coordenada inválido. Usa ejemplo: A1, B2.")
46         return False # disparo inválido
47     # Validar rangos
48     if fila < 0 or fila >= FILAS or columna < 0 or columna >= COLUMNAS:
49         print(f"{nombre}, coordenada fuera de rango. Intenta de nuevo.")
50         return False # disparo inválido
51     # Verificar estado de la celda
52     if tablero_objetivo[fila][columna] == 1:
53         tablero_objetivo[fila][columna] = 2
54         tablero_disparos[fila][columna] = 2
55         print(f"{nombre} hizo ¡Tocado!")
56         return True

```

```

57     elif tablero_objetivo[fila][columna] == 0:
58         tablero_objetivo[fila][columna] = 3
59         tablero_disparos[fila][columna] = 3
60         print(f"{nombre} disparó al agua.")
61         return True
62     else:
63         print(f"{nombre}, ya disparaste allí. Elige otra coordenada.")
64         return False # disparo inválido, repetir
65 # Comprobar si quedan barcos
66 def quedan_barcos(tablero):
67     for fila in tablero:
68         if 1 in fila:
69             return True
70     return False
71 # Guardar puntuación
72 def guardar_puntuacion(nombre):
73     with open("puntuaciones.txt", "a") as archivo:
74         archivo.write(f"{nombre} ganó la partida.\n")
75 # Menú principal del juego
76 def juego():
77     print("== Batalla Naval ==")
78     print("1. Jugar contra la CPU")
79     print("2. Jugar contra otro jugador")
80     modo = input("Selecciona modo (1 o 2): ")
81     if modo == "1":
82         nombre_jugador = input("Tu nombre: ")
83         nombre_cpu = "CPU"
84         tablero_jugador = crear_tablero()
85         tablero_cpu = crear_tablero()
86         disparos_jugador = crear_tablero()
87         disparos_cpu = crear_tablero()
88         colocar_barcos(tablero_jugador, BARCOS)
89         colocar_barcos(tablero_cpu, BARCOS)
90         turno = 1
91         while quedan_barcos(tablero_jugador) and quedan_barcos(tablero_cpu):
92             print(f"\n--- Turno {turno} ---")
93             print("Tu tablero:")
94             mostrar_tablero(tablero_jugador)
95             print("Tus disparos:")
96             mostrar_tablero(disparos_jugador)
97             # Turno jugador con validación de disparo
98             while True:
99                 coord = input("Dispara (ej. A1): ")
100                if disparar(tablero_cpu, disparos_jugador, coord, nombre_jugador):
101                    break
102                # Turno CPU (simple, sin repetir disparos)
103                while True:
104                    fila_cpu = random.randint(0, FILAS - 1)
105                    col_cpu = random.randint(0, COLUMNAS - 1)
106                    if tablero_jugador[fila_cpu][col_cpu] in [0, 1]:
107                        break
108                    coord_cpu = f"{chr(ord('A') + fila_cpu)}{col_cpu + 1}"
109                    print(f"{nombre_cpu} dispara a {coord_cpu}")
110                    disparar(tablero_jugador, disparos_cpu, coord_cpu, nombre_cpu)
111                    turno += 1
112                if quedan_barcos(tablero_jugador):

```

```

113     print(f"¡{nombre_jugador} gana!")
114     guardar_puntuacion(nombre_jugador)
115 else:
116     print("¡La CPU gana!")
117 elif modo == "2":
118     nombre1 = input("Nombre del Jugador 1: ")
119     nombre2 = input("Nombre del Jugador 2: ")
120     tablero1 = crear_tablero()
121     tablero2 = crear_tablero()
122     disparos1 = crear_tablero()
123     disparos2 = crear_tablero()
124     colocar_barcos(tablero1, BARCOS)
125     colocar_barcos(tablero2, BARCOS)
126     turno = 1
127     while quedan_barcos(tablero1) and quedan_barcos(tablero2):
128         print(f"\n--- Turno {turno} ---")
129         # Turno jugador 1 con validación
130         print(f"\n{nombre1}, este es tu turno")
131         mostrar_tablero(disparos1)
132         while True:
133             coord = input("Dispara (ej. A1): ")
134             if disparar(tablero2, disparos1, coord, nombre1):
135                 break
136             if not quedan_barcos(tablero2):
137                 break
138             # Turno jugador 2 con validación
139             print(f"\n{nombre2}, este es tu turno")
140             mostrar_tablero(disparos2)
141             while True:
142                 coord = input("Dispara (ej. A1): ")
143                 if disparar(tablero1, disparos2, coord, nombre2):
144                     break
145                 turno += 1
146             if quedan_barcos(tablero2):
147                 print(f"¡{nombre1} gana!")
148                 guardar_puntuacion(nombre1)
149             else:
150                 print(f"¡{nombre2} gana!")
151                 guardar_puntuacion(nombre2)
152         else:
153             print("Opción inválida. Reinicia el programa.")
154     print("\nFin del programa Jose Alejandro Zabala Romero")
155 juego()

```

Este código es una versión mejorada del juego Batalla Naval. Reduce el tamaño del tablero a 4x2 y permite jugar contra otro jugador además de la CPU, incluyendo la opción de ingresar nombres. El código está organizado en funciones que manejan tareas como crear tableros, mostrar contenido, colocar barcos, validar disparos, convertir coordenadas y verificar si quedan barcos. También guarda al ganador en un archivo de texto. Aunque trabajar con matrices bidimensionales y coordinar acciones entre tableros fue lo más desafiante, la estructura modular facilitó el desarrollo y mantenimiento del juego.

Ejecución

```
~/.../Practico_II/matrices$ python batalla_naval_texto.py
== Batalla Naval ==
1. Jugar contra la CPU
2. Jugar contra otro jugador
Selecciona modo (1 o 2): 2
Nombre del Jugador 1: jose
Nombre del Jugador 2: alejandro

--- Turno 1 ---
jose, este es tu turno
 1 2
A 0 0
B 0 0
C 0 0
D 0 0
Dispara (ej. A1): a1
jose disparó al agua.

alejandro, este es tu turno
 1 2
A 0 0
B 0 0
C 0 0
D 0 0
Dispara (ej. A1): b5
alejandro, coordenada fuera de rango. Intenta de nuevo.
Dispara (ej. A1): b2
alejandro hizo ¡Tocado!

--- Turno 2 ---
jose, este es tu turno
 1 2
A * 0
B 0 0
C 0 0
D 0 0
Dispara (ej. A1): b1

A * 0
B X X
C * 0
D 0 *
Dispara (ej. A1): a2
jose disparó al agua.

alejandro, este es tu turno
 1 2
A 0 0
B 0 X
C * X
D * *
Dispara (ej. A1): b1
alejandro disparó al agua.

--- Turno 7 ---
jose, este es tu turno
 1 2
A * *
B X X
C * 0
D 0 *
Dispara (ej. A1): c2
jose disparó al agua.

alejandro, este es tu turno
 1 2
A 0 0
B * X
C * X
D * *
Dispara (ej. A1): a2
alejandro hizo ¡Tocado!
¡jose gana!

Fin del programa Jose Alejandro Zabala Romero
```

2.2 Operaciones Básicas con Matrices

2.2.1 Suma Total y por Ejes

```
1 matriz = [
2   [1, 2, 3],
3   [4, 5, 6],
4   [7, 8, 9]
5 ]
6 # 1. Inicialización
7 acumulador_total = 0
8 # 2. Iteración Anidada
9 for fila in matriz:
10   # 3. Recorremos cada elemento en la fila actual
11   for elemento in fila:
12     # 4. Acumulación
13     acumulador_total += elemento
14 # 5. Resultado
15 print(f"La suma total de todos los elementos es: {acumulador_total}")
16 print("\nFin del programa ---- Jose Alejandro Zabala Romero")
```

Ejecución

```
~/.../Practico_II/matrices$ python sumatotal_porejes.py
La suma total de todos los elementos es: 45

Fin del programa ---- Jose Alejandro Zabala Romero
~/.../Practico_II/matrices$
```

Este código calcula la suma total de todos los elementos de una matriz bidimensional. Para lograrlo, primero se declaré una matriz de 3x3 con números del 1 al 9. Luego se inicializa un acumulador en cero, que será el encargado de ir sumando cada uno de los elementos que recorra.

La lógica principal está en el uso de dos bucles anidados: el primero recorre cada fila de la matriz, y el segundo recorre cada elemento dentro de esa fila. A medida que se avanza, se va sumando cada valor al acumulador utilizando acumulador_total += elemento.

2.2.2 Suma todinigos

```

1  matriz = [
2      [10, 2, 3, 4, 5],
3      [6, 20, 8, 9, 10],
4      [11, 12, 30, 14, 15],
5      [16, 17, 18, 40, 20],
6      [21, 22, 23, 24, 50]
7  ]
8  # Suma total
9  suma = sum(sum(fila) for fila in matriz)
10 print("Suma total:", suma)
11 # Suma por filas
12 print("Suma por filas:")
13 for i, fila in enumerate(matriz):
14     print(f"Fila {i}: {sum(fila)}")
15 # Suma por columnas
16 print("Suma por columnas:")
17 for j in range(5):
18     col = sum(matriz[i][j] for i in range(5))
19     print(f"Columna {j}: {col}")
20 # MÁximo y MÍnimo
21 todos = [num for fila in matriz for num in fila]
22 print("Máximo:", max(todos))
23 print("Mínimo:", min(todos))
24 print("Diagonal principal:")
25 print(' '.join(str(matriz[i][i]) for i in range(5)))
26 print("Diagonal secundaria:")
27 print(' '.join(str(matriz[i][4 - i]) for i in range(5)))
28 print("Fin del programa --- Richard hurtado")

```

Ejecución

```

~.../Practico_II/matrices$ python Suma_todinigos.py
Suma total: 410
Suma por filas:
Fila 0: 24
Fila 1: 53
Fila 2: 82
Fila 3: 111
Fila 4: 140
Suma por columnas:
Columna 0: 64
Columna 1: 73
Columna 2: 82
Columna 3: 91
Columna 4: 100
Máximo: 50
Mínimo: 2
Diagonal principal:
10 20 30 40 50
Diagonal secundaria:
5 9 30 17 21
Fin del programa --- Richard hurtado

```

Este programa trabaja con una matriz 5x5 representada como una lista de listas, lo que permite organizar y acceder fácilmente a los datos en filas y columnas. Realiza varias operaciones: calcula la suma total de los elementos, la suma por filas y columnas, encuentra el valor máximo y mínimo al convertir la matriz en una lista lineal, y extrae los elementos de la diagonal principal y secundaria.

2.2.3 Suma por columnas

```

1 # Matriz de ejemplo
2 matriz = [
3     [1, 2, 3],
4     [4, 5, 6],
5     [7, 8, 9]
6 ]
7 # 1. Inicialización
8 num_filas = len(matriz)
9 num_columnas = len(matriz[0])
10 sumas_por_columna = [0] * num_columnas # lista con ceros para cada columna
11 # 2. Iteración anidada con índices
12 for i in range(num_filas):
13     for j in range(num_columnas):
14         # 3. Acumulación por índice
15         sumas_por_columna[j] += matriz[i][j]
16 # 4. Resultado final
17 print("Suma por columnas:", sumas_por_columna)
18 print("\nFin del programa ---- Leonardo Montenegro")

```

Ejecución

```

~/.../Practico_II/matrices$ python sumar_columnas.py
Suma por columnas: [12, 15, 18]

Fin del programa ---- Leonardo Montenegro
~/.../Practico_II/matrices$ █

```

Este programa utiliza una estructura bidimensional, representada en Python como una lista de listas, para formar una matriz de 3x3. Esta estructura permite acceder a los datos mediante dos índices: uno para las filas y otro para las columnas. Calcula la suma de cada columna de la matriz. Para ello, primero se determina la cantidad de filas y columnas usando `len()`. Luego se crea una lista llamada `sumas_por_columna`, inicializada con ceros, donde cada posición representa una

columna y acumulará su suma. Utiliza dos bucles anidados con índices: el bucle externo recorre cada fila, y el interno recorre cada elemento dentro de esa fila (es decir, cada columna). A medida que recorro la matriz, acumulo cada valor en la posición correspondiente de la lista sumas_por_columna usando el índice de columna j.

2.2.4 suma por filas

```

1  matriz = [
2      [1, 2, 3],
3      [4, 5, 6],
4      [7, 8, 9]
5  ]
6  # 1. Inicialización
7  sumas_por_fila = []
8  # 2. Iteración Exterior (por fila)
9  for fila in matriz:
10     # 3a. Inicializa acumulador para esta fila
11     acumulador_fila = 0
12     # 3b y 3c. Suma cada elemento de la fila
13     for elemento in fila:
14         acumulador_fila += elemento
15     # 4. Guardar suma de la fila
16     sumas_por_fila.append(acumulador_fila)
17 # 5. Resultado final
18 print("Suma por filas:", sumas_por_fila)
19 print("\nFin del programa ----- Leonardo Montenegro")

```

Este programa trabaja con una matriz 3x3 usando listas de listas para organizar los datos. Calcula la suma de los elementos de cada fila por separado. Para ello, usa un bucle externo que recorre cada fila, y dentro de este, un bucle interno que acumula los valores de esa fila en una variable. Luego, el resultado se guarda en una lista llamada sumas_por_fila. Al final, se imprime esta lista con las sumas individuales.

Ejecución

```

~/.../Practico_II/matrices$ python sumar_porfilas.py
Suma por filas: [6, 15, 24]
Fin del programa ----- Leonardo Montenegro
~/.../Practico_II/matrices$ 

```

2.2.5 Encontrar el Elemento Máximo/Mínimo

```

1 # Matriz de ejemplo (cuadrada 3x3)
2 matriz = [
3     [5, 2, 9],
4     [1, 8, 3],
5     [4, 7, 6]
6 ]
7 # 1. Suposición inicial (verificando que la matriz no esté vacía)
8 if not matriz or not matriz[0]:
9     print("La matriz está vacía.")
10 else:
11     num_filas = len(matriz)
12     num_columnas = len(matriz[0])
13     # Asumimos el primer elemento como máximo temporal
14     mayor_temporal = matriz[0][0]
15     # 2 y 3. Iteración anidada para recorrer todos los elementos
16     for i in range(num_filas):
17         for j in range(num_columnas):
18             if matriz[i][j] > mayor_temporal:
19                 mayor_temporal = matriz[i][j]
20     print(f"El valor máximo en la matriz es: {mayor_temporal}")
21     # Si la matriz es cuadrada (N x N)
22     if num_filas == num_columnas:
23         N = num_filas
24         diagonal_principal = []
25         diagonal_secundaria = []
26         # Recorrido de la diagonal principal y secundaria
27         for i in range(N):
28             diagonal_principal.append(matriz[i][i])
29             diagonal_secundaria.append(matriz[i][N - 1 - i])
30         print("Diagonal principal:", diagonal_principal)
31         print("Diagonal secundaria:", diagonal_secundaria)
32 print("\nFin del programa ----- Richard Hurtado")

```

Este programa trabaja con una matriz cuadrada 3x3 y comienza verificando que no esté vacía. Usa dos bucles anidados para encontrar el valor máximo, comparando cada elemento con un valor temporal. Luego, si la matriz es cuadrada, extrae los elementos de la diagonal principal y secundaria usando sus índices, y los guarda en listas. Finalmente, imprime el valor máximo y ambas diagonales.

Ejecución

```

~/.../Practico_II/matrices$ python encont_max_min.py
El valor máximo en la matriz es: 9
Diagonal principal: [5, 8, 6]
Diagonal secundaria: [9, 8, 4]

Fin del programa ----- Richard Hurtado
~/.../Practico_II/matrices

```

2.3 Análisis Matricial

2.3.1 Función Suma Total

```

1  def sumar_total_matriz(matriz):
2      """
3          Suma todos los elementos de una matriz (lista de listas).
4          Retorna 0 si la matriz está vacía o si las filas están vacías.
5      """
6      if not matriz:
7          return 0
8      # Suma todos los elementos con comprensión de listas
9      return sum(elemento for fila in matriz if fila for elemento in fila)
10 def probar_suma_total():
11     print("Probando sumar_total_matriz...")
12     m1 = [[1, 2, 3], [4, 5, 6]]
13     assert sumar_total_matriz(m1) == 21
14     m2 = [[-1, 0, 1], [10, -5, 5]]
15     assert sumar_total_matriz(m2) == 10
16     assert sumar_total_matriz([]) == 0
17     assert sumar_total_matriz([[]]) == 0
18     assert sumar_total_matriz([[42]]) == 42
19     assert sumar_total_matriz([], [1, 2, 3], []) == 6
20     print("¡Pruebas para sumar_total_matriz pasaron!")
21 if __name__ == "__main__":
22     probar_suma_total()
23     print("\nFin del programa ---- Junior Pacajes Banegas")

```

Este programa define la función `sumar_total_matriz`, que calcula la suma total de los elementos de una matriz. Incluye validaciones para evitar errores con matrices vacías o filas sin datos. Usa una comprensión de listas anidada para sumar de forma eficiente. También implementa `probar_suma_total`, una función de prueba con varios casos (matrices vacías, con positivos y negativos), usando aserciones para validar los resultados. El bloque `if __name__ == "__main__"` permite ejecutar las pruebas al correr el script directamente.

Ejecución

```

~/.../Practico_II/analizis_matricial$ python funcion_sumaT.py
Probando sumar_total_matriz...
¡Pruebas para sumar_total_matriz pasaron!

Fin del programa ---- Junior Pacajes Banegas
~/.../Practico_II/analizis_matricial$ █

```

2.3.2 Función de Suma por Filas

```

1 def sumar_por_filas(matriz):
2     """
3         Esta función recibe una matriz (lista de listas)
4         y devuelve una lista con la suma de cada fila.
5     """
6     # Usamos comprensión de listas para sumar cada fila
7     return [sum(fila) for fila in matriz]
8
9     def probar_suma_por_filas():
10        print("\nProbando sumar_por_filas...")
11        m1 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
12        assert sumar_por_filas(m1) == [6, 15, 24]
13        m2 = [[10, 10], [20, 20], [30, 30]]
14        assert sumar_por_filas(m2) == [20, 40, 60]
15        assert sumar_por_filas([]) == []
16        print("¡Pruebas para sumar_por_filas pasaron! ")
17
18    if __name__ == "__main__":
19        probar_suma_por_filas()
20
21    print("\nFin del programa ---- Junior Pacajes Banegas")

```

Este programa define la función `sumar_por_filas` que recibe una matriz y devuelve una lista con la suma de cada fila usando comprensión de listas y la función `sum()`. Incluye una función de prueba `probar_suma_por_filas` que valida distintos casos, como matrices con números y matrices vacías, usando `assert`. El bloque `if __name__ == "__main__":` ejecuta las pruebas al correr el script.

Ejecución

```

~/.../Practico_II/analisis_matricial$ python suma_porfilas.py

Probando sumar_por_filas...
¡Pruebas para sumar_por_filas pasaron!

Fin del programa ---- Junior Pacajes Banegas
~/.../Practico_II/analisis_matricial$ █

```

2.3.3 Suma de la Diagonal Principal

```

1 def sumar_diagonal_principal(matriz):
2     """
3         Esta función recibe una matriz cuadrada (misma cantidad de filas y columnas)
4         y retorna la suma de los elementos en su diagonal principal.
5     """
6     return sum(matriz[i][i] for i in range(len(matriz)))
7 def probar_suma_diagonal_principal():
8     print("\nProbando sumar_diagonal_principal...")
9     m1 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
10    assert sumar_diagonal_principal(m1) == 15 # 1 + 5 + 9
11    m2 = [[10, 0], [0, 20]]
12    assert sumar_diagonal_principal(m2) == 30 # 10 + 20
13    m3 = [[5]]
14    assert sumar_diagonal_principal(m3) == 5 # Solo un elemento en la diagonal
15    print("¡Pruebas para sumar_diagonal_principal pasaron!")
16 if __name__ == "__main__":
17     probar_suma_diagonal_principal()
18     print("\nFin del programa ---- Jose Alejandro Zabala Romero")

```

Este programa define la función `sumar_diagonal_principal` que recibe una matriz cuadrada y devuelve la suma de sus elementos en la diagonal principal usando una comprensión de listas con un solo índice. Incluye pruebas con matrices de distintos tamaños en la función `probar_suma_diagonal_principal`, que utiliza `assert` para validar resultados. El bloque `if __name__ == "__main__":` ejecuta las pruebas al correr el script.

Ejecución

```

~/.../Practico_II/analisis_matricial$ python S_diag_princ.py
Probando sumar_diagonal_principal...
¡Pruebas para sumar_diagonal_principal pasaron!
Fin del programa ---- Jose Alejandro Zabala Romero
~/.../Practico_II/analisis_matricial$ █

```

2.3.4 Función que suma los elementos de la diagonal secundaria

```

1  def sumar_diagonal_secundaria(matriz):
2      """
3          Esta función recibe una matriz cuadrada y retorna la suma
4          de los elementos de su diagonal secundaria (de derecha a izquierda).
5      """
6      suma = 0
7      n = len(matriz)
8      for i in range(n):
9          suma += matriz[i][n - 1 - i]
10     return suma
11 def probar_suma_diagonal_secundaria():
12     print("\nProbando sumar_diagonal_secundaria...")
13     m1 = [[1, 2, 3],
14           [4, 5, 6],
15           [7, 8, 9]]
16     assert sumar_diagonal_secundaria(m1) == 15 # 3 + 5 + 7
17     m2 = [[0, 0, 1],
18           [0, 2, 0],
19           [3, 0, 0]]
20     assert sumar_diagonal_secundaria(m2) == 6 # 1 + 2 + 3
21     m3 = [[8]]
22     assert sumar_diagonal_secundaria(m3) == 8 # Solo un elemento
23     print("¡Pruebas para sumar_diagonal_secundaria pasaron! ")
24 if __name__ == "__main__":
25     probar_suma_diagonal_secundaria()
26     print("\nFin del programa ----- Nicodemus Berny Quiroga González")

```

El programa define la función `sumar_diagonal_secundaria`, que calcula la suma de los elementos de la diagonal secundaria de una matriz cuadrada usando un índice `i` y la expresión `matriz[i][n - 1 - i]`. Incluye la función `probar_suma_diagonal_secundaria` con casos de prueba para matrices 3x3 y 1x1, usando `assert` para validar resultados. El bloque `if __name__ == "__main__":` ejecuta las pruebas al correr el script.

Ejecución

```

~/.../Práctico_II/analisis_matricial$ python Su_Di_Se.py
Probando sumar_diagonal_secundaria...
¡Pruebas para sumar_diagonal_secundaria pasaron!

Fin del programa ----- Nicodemus Berny Quiroga González
~/.../Práctico_II/analisis_matricial$ 

```

2.3.5 Suma los elementos de la diagonal secundaria de una matriz cuadrada

```

1 # Función que suma los elementos de la diagonal secundaria de una matriz cuadrada
2 def sumar_diagonal_secundaria(matriz):
3     n = len(matriz)
4     return sum(matriz[i][n - 1 - i] for i in range(n))
5 # Funciones de prueba para validar que sumar_diagonal_secundaria funciona correctamente
6 def probar_suma_diagonal_secundaria():
7     print("\nProbando sumar_diagonal_secundaria...")
8     # Caso 1: matriz 3x3 normal
9     m1 = [[1, 2, 3],
10           [4, 5, 6],
11           [7, 8, 9]]
12     # Diagonal secundaria: 3 + 5 + 7 = 15
13     assert sumar_diagonal_secundaria(m1) == 15
14     # Caso 2: matriz 2x2
15     m2 = [[10, 1],
16           [2, 20]]
17     # Diagonal secundaria: 1 + 2 = 3
18     assert sumar_diagonal_secundaria(m2) == 3
19     # Caso 3: matriz 1x1
20     m3 = [[42]]
21     # Solo hay un elemento: 42
22     assert sumar_diagonal_secundaria(m3) == 42
23     print("¡Pruebas para sumar_diagonal_secundaria pasaron! ")
24 # Llamamos a la función de prueba
25 probar_suma_diagonal_secundaria()
26 print("Fin del programa --- Lucas Mateo Diaz Badano")

```

Este programa define la función `sumar_diagonal_secundaria`, que calcula la suma de la diagonal secundaria de una matriz cuadrada usando una comprensión de listas y la expresión `matriz[i][n - 1 - i]`. Incluye pruebas con matrices de tamaños 3x3, 2x2 y 1x1 en la función `probar_suma_diagonal_secundaria`, que utiliza `assert` para validar resultados. Al final, se ejecutan las pruebas para asegurar el correcto funcionamiento.

Ejecución

```

~/.../Practico_II/analisis_matricial$ python suma_elemt.py
Probando sumar_diagonal_secundaria...
¡Pruebas para sumar_diagonal_secundaria pasaron!
Fin del programa --- Lucas Mateo Diaz Badano
~/.../Practico_II/analisis_matricial$ 

```

2.3.6 Matrix Pathfinder

```

1 def hay_camino(matriz):
2     """
3     Determina si existe un camino desde la esquina superior izquierda (0, 0)
4     hasta la esquina inferior derecha (N-1, N-1) en una matriz.
5     0 = camino libre, 1 = pared.
6     Usa búsqueda en profundidad (DFS).
7     """
8     if not matriz or matriz[0][0] == 1:
9         return False
10    n = len(matriz)
11    visitado = [[False]*n for _ in range(n)]
12
13    def dfs(x, y):
14        if x < 0 or x >= n or y < 0 or y >= n:
15            return False
16        if matriz[x][y] == 1 or visitado[x][y]:
17            return False
18        if (x, y) == (n-1, n-1):
19            return True
20        visitado[x][y] = True
21
22        # Explorar en las 4 direcciones
23        return (dfs(x+1, y) or dfs(x-1, y) or dfs(x, y+1) or dfs(x, y-1))
24    return dfs(0, 0)
25 # ----- Pruebas -----
26 print("Pruebas del camino en la matriz:")
27 matriz1 = [
28     [0, 1, 0],
29     [0, 0, 0],
30     [1, 1, 0]
31 ]
32 print(hay_camino(matriz1)) # True, hay camino desde (0,0) a (2,2)
33 matriz2 = [
34     [0, 1, 1],
35     [1, 1, 1],
36     [1, 1, 0]
37 ]
38 print(hay_camino(matriz2)) # False, bloqueado
39 matriz3 = [
40     [0, 0, 0],
41     [1, 1, 0],
42     [1, 1, 0]
43 ]
44 print(hay_camino(matriz3)) # True

```

Este programa implementa el desafío "Matrix Pathfinder", que verifica si existe un camino desde la celda inicial [0][0] hasta la final [N-1][N-1] en una matriz que representa un laberinto (0 = camino libre, 1 = pared). Usa una búsqueda en profundidad (DFS) con una matriz auxiliar para marcar celdas visitadas y evitar ciclos. La función recursiva explora las cuatro direcciones posibles, asegurándose de no salir de los límites ni visitar paredes o celdas repetidas, hasta encontrar el camino o agotar opciones.

Ejecución

```

~/.../Práctico_II/análisis_matricial$ python matrix_Pathfinder.py
Pruebas del camino en la matriz:
True
False
True

```

2.4 Implementación de Algoritmos para el Manejo de Matrices

2.4.1 Función para Transponer una Matriz

```

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     matriz_transpuesta = []
7     for j in range(num_columnas): # Recorremos columnas de la original
8         nueva_fila = []
9         for i in range(num_filas): # Recorremos filas de la original
10            nueva_fila.append(matriz[i][j])
11        matriz_transpuesta.append(nueva_fila)
12    return matriz_transpuesta
13 # Prueba 1: Matriz 2x3
14 m1 = [[1, 2, 3], [4, 5, 6]]
15 t1 = transponer_matriz(m1)
16 assert t1 == [[1, 4], [2, 5], [3, 6]]
17 print("Prueba 1 (2x3) pasada!")
18 # Prueba 2: Matriz cuadrada 2x2
19 m2 = [[7, 8], [9, 10]]
20 t2 = transponer_matriz(m2)
21 assert t2 == [[7, 9], [8, 10]]
22 print("Prueba 2 (2x2) pasada!")
23 # Prueba 3: Matriz 3x1
24 m3 = [[1], [2], [3]]
25 t3 = transponer_matriz(m3)
26 assert t3 == [[1, 2, 3]]
27 print("Prueba 3 (3x1) pasada!")
28 # Prueba 4: Matriz 1x3
29 m4 = [[10, 20, 30]]
30 t4 = transponer_matriz(m4)
31 assert t4 == [[10], [20], [30]]
32 print("Prueba 4 (1x3) pasada!")
33 m5 = []
34 t5 = transponer_matriz(m5)
35 assert t5 == []
36 print("Prueba 5 (matriz vacía) pasada!")
37 print("Fin del programa ---- Jose Alejandro Zabala Romero")

```

Este programa define la función `transponer_matriz` que recibe una matriz (lista de listas) y devuelve su transpuesta, es decir, intercambia filas por columnas. Primero verifica que la matriz no esté vacía para evitar errores, obtiene las dimensiones con `len()`, y luego usa dos bucles anidados: el externo recorre columnas y el interno filas, construyendo fila por fila la matriz transpuesta. Al final, devuelve la nueva matriz transpuesta.

Ejecución

```

~/.../Practico_II/implmnt_algort$ python transp_matrix.py
Prueba 1 (2x3) pasada!
Prueba 2 (2x2) pasada!
Prueba 3 (3x1) pasada!
Prueba 4 (1x3) pasada!
Prueba 5 (matriz vacía) pasada!
Fin del programa ---- Jose Alejandro Zabala Romero
~/.../Practico_II/implmnt_algort$ 

```

2.4.2 Función para Verificar Identidad

```

1  def es_identidad(matriz):
2      num_filas = len(matriz)
3      if num_filas == 0:
4          return True # Una matriz vacía es identidad por convención
5      for i in range(num_filas):
6          if len(matriz[i]) != num_filas:
7              return False # No es cuadrada
8
9      for i in range(num_filas):
10         for j in range(num_filas):
11             if i == j:
12                 if matriz[i][j] != 1:
13                     return False # La diagonal debe tener 1
14             else:
15                 if matriz[i][j] != 0:
16                     return False # Fuera de la diagonal debe haber 0
17         return True
18 # ===== PRUEBAS DE LA FUNCIÓN =====
19 identidad = [[1, 0, 0], [0, 1, 0], [0, 0, 1]]
20 no_identidad = [[1, 0, 0], [0, 2, 0], [0, 0, 1]]
21 no_cuadrada = [[1, 0], [0, 1], [0, 0]]
22 assert es_identidad(identidad) == True
23 assert es_identidad(no_identidad) == False
24 assert es_identidad(no_cuadrada) == False
25 print("Todas las pruebas pasaron correctamente.")
26 print("Fin del programa ---- leonardo montenegro")

```

Este programa define una función llamada `es_identidad` que verifica si una matriz es una matriz identidad. Primero, se asegura de que la matriz sea cuadrada, ya que este tipo de matriz debe tener el mismo número de filas y columnas. Luego, se recorren todos los elementos de la matriz utilizando bucles anidados, donde `i` representa el índice de fila y `j` el de columna. En cada posición, si el elemento está en la diagonal principal (es decir, si `i == j`), su valor debe ser 1. En cambio, si el elemento no está en la diagonal, debe ser 0. Si alguna de estas condiciones no se cumple, la función retorna `False`, indicando que no es una matriz identidad. Si todos los valores cumplen los criterios correctamente, la función retorna `True`.

Ejecución

```

~/.../Practico_II/implemt_algort$ python identidad.py
Todas las pruebas pasaron correctamente.
Fin del programa ---- leonardo montenegro
~/.../Practico_II/implemt_algort$ 

```

2.4.3 Función para Verificar Simetría

```

1 def es_simetrica(matriz):
2     num_filas = len(matriz)
3     if num_filas == 0:
4         return True # Una matriz vacía es simétrica por convención
5     for i in range(num_filas):
6         if len(matriz[i]) != num_filas:
7             return False # No es cuadrada
8     for i in range(num_filas):
9         for j in range(i + 1, num_filas): # Solo triangular superior
10            if matriz[i][j] != matriz[j][i]:
11                return False # Encontramos una diferencia
12    return True
13 # ===== PRUEBAS DE LA FUNCIÓN =====
14 sim = [[1, 7, 3], [7, 4, -5], [3, -5, 6]]
15 no_sim = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
16 no_cuadrada = [[1, 2], [3, 4], [5, 6]]
17 assert es_simetrica(sim) == True
18 assert es_simetrica(no_sim) == False
19 assert es_simetrica(no_cuadrada) == False
20 print("¡Pruebas para es_simetrica pasaron!")
21 print("Fin del programa ----- jose alejandro zabala romero")

```

Ejecución

```

$ ./Practico_II/implent_algort$ python v_simetria.py
¡Pruebas para es_simetrica pasaron!
Fin del programa ----- jose alejandro zabala romero
$ ./Practico_II/implent_algort$ 

```

El programa define una función llamada `es_simetrica` que verifica si una matriz es simétrica. Para ello, primero se comprueba que la matriz sea cuadrada, ya que solo las matrices cuadradas pueden ser simétricas. Esta verificación se realiza revisando que todas las filas tengan la misma longitud que el número total de filas. Luego, la función recorre únicamente la parte triangular superior de la matriz (excluyendo la diagonal) usando bucles anidados, comparando cada elemento con su correspondiente en la posición simétrica de la triangular inferior (`matriz[i][j]` con `matriz[j][i]`). Si encuentra alguna diferencia entre estas posiciones, retorna `False`, indicando que la matriz no es simétrica. Si todas las comparaciones son iguales, la función devuelve `True`, indicando que la matriz cumple la condición de simetría.

2.4.4 Gestión de Asientos de Cine

```

1 def crear_sala(filas, columnas):
2     return [{"estado": "L", "precio": 50} for _ in range(columnas)] for _ in range(filas)]
3 def mostrar_sala(sala):
4     print("\n💡 Estado actual de la sala:")
5     for i, fila in enumerate(sala):
6         print("Fila", i + 1, end=" -> ")
7         for asiento in fila:
8             simbolo = "0" if asiento["estado"] == "0" else "L"
9             print(simbolo, end=" ")
10    print()
11    print("L = Libre, 0 = Ocupado")
12
13 def ocupar_asiento(sala, fila, columna):
14     try:
15         asiento = sala[fila][columna]
16         if asiento["estado"] == "0":
17             print(" Ese asiento ya está ocupado.")
18             return False
19         else:
20             asiento["estado"] = "0"
21             print(f" Asiento en Fila {fila + 1}, Columna {columna + 1} reservado.")
22             return True
23     except IndexError:
24         print(" Asiento inválido.")
25         return False
26 def contar_asientos_libres(sala):
27     return sum(asiento["estado"] == "L" for fila in sala for asiento in fila)
28 def buscar_asientos_juntos(sala, cantidad):
29     for i, fila in enumerate(sala):
30         consecutivos = 0
31         inicio = 0
32         for j, asiento in enumerate(fila):
33             if asiento["estado"] == "L":
34                 if consecutivos == 0:
35                     inicio = j
36                     consecutivos += 1
37                     if consecutivos == cantidad:
38                         return i, inicio
39                 else:
40                     consecutivos = 0
41     return None, None
42 def ocupar_asientos_juntos(sala, fila, inicio_columna, cantidad):
43     for j in range(inicio_columna, inicio_columna + cantidad):
44         sala[fila][j]["estado"] = "0"
45     print(f"👉 {cantidad} asientos reservados en fila {fila + 1}, columnas {inicio_columna + 1} a {inicio_columna + cantidad}.")
46 # Ejecución del sistema de cine
47 filas = 5
48 columnas = 7
49 sala = crear_sala(filas, columnas)
50 while True:
51     print("\n💡 MENÚ DE OPCIONES")
52     print("1. Ver sala")
53     print("2. Reservar un asiento")
54     print("3. Reservar varios asientos juntos")
55     print("4. Contar asientos libres")
56     print("5. Salir")
57     opcion = input("Seleccione una opción: ")
58     if opcion == "1":
59         mostrar_sala(sala)
60     elif opcion == "2":
61         fila = int(input("Ingrese número de fila (1-5): ")) - 1
62         columna = int(input("Ingrese número de columna (1-7): ")) - 1
63         ocupar_asiento(sala, fila, columna)
64     elif opcion == "3":
65         cantidad = int(input("¿Cuántos asientos quiere reservar juntos?: "))
66         fila, inicio = buscar_asientos_juntos(sala, cantidad)
67         if fila is not None:
68             ocupar_asientos_juntos(sala, fila, inicio, cantidad)
69         else:
70             print("❌ No hay suficientes asientos contiguos disponibles.")
71     elif opcion == "4":
72         libres = contar_asientos_libres(sala)
73         print(f"💡 Asientos disponibles: {libres}")
74     elif opcion == "5":
75         print("👋 Gracias por usar el sistema de reservas.")
76         break
77     else:
78         print("⚠️ Opción inválida. Intente nuevamente.")
79 print(" fin del programa ----- leonardo montenegro")

```

Ejecución

Asientos disponibles: 29

■ MENÚ DE OPCIONES

1. Ver sala
2. Reservar un asiento
3. Reservar varios asientos juntos
4. Contar asientos libres
5. Salir

Seleccione una opción: 2

Ingrese número de fila (1-5): 4

Ingrese número de columna (1-7): 6

Asiento en Fila 4, Columna 6 reservado.

■ MENÚ DE OPCIONES

1. Ver sala
2. Reservar un asiento
3. Reservar varios asientos juntos
4. Contar asientos libres
5. Salir

Seleccione una opción: 1

Estado actual de la sala:

Fila 1 -> O O O O O O L

Fila 2 -> L L L L L L L

Fila 3 -> L L L L L L L

Fila 4 -> L L L L L O L

Fila 5 -> L L L L L L L

L = Libre, O = Ocupado

■ MENÚ DE OPCIONES

1. Ver sala
2. Reservar un asiento
3. Reservar varios asientos juntos
4. Contar asientos libres
5. Salir

Seleccione una opción: 5

Gracias por usar el sistema de reservas.

El programa implementa un sistema de reservas para una sala de cine utilizando una matriz (lista de listas) donde cada asiento es representado por un diccionario con dos claves: "estado" (libre "L" u ocupado "O") y "precio". La función crear_sala genera la sala con comprensión de listas. mostrar_sala imprime el estado actual de todos los asientos.

Para reservar, ocupar_asiento permite seleccionar un asiento específico validando su disponibilidad y controlando errores con try-except. La función contar_asientos_libres recorre la sala y cuenta los disponibles.

Para reservas grupales, buscar_asientos_juntos localiza una secuencia de asientos contiguos libres en una misma fila, y ocupar_asientos_juntos los marca como ocupados. El programa corre en un bucle con menú interactivo, permitiendo ver la sala, reservar asientos (individual o grupal), consultar disponibilidad o salir.

Capítulo III

Registros (Diccionario)

3.1 Introducción a los Registros

3.1.1 Nuestro Estudiante

```

1 estudiante_alejandro = {
2     "nombre": " Alejandro Romero",
3     "edad": 20,
4     "carrera": "Ing. en Redes y Telecomunicaciones",
5     "esta_activa": True,
6     "materias_inscritas": ["Prog II", "Cálculo I", "Física II"]
7 }
8 print(estudiante_alejandro)
9 print ("fin del programa ----- jose alejandro zabala romero")

```

Este código crea un diccionario llamado estudiante_alejandro que almacena información personal y académica sobre mí, como mi nombre, edad, carrera, si estoy activo o no, y una lista de materias inscritas. Luego imprime todo el contenido del diccionario en pantalla.

3.1.2 Recorriendo un Diccionario con un Bucle for

```

1 producto = {'codigo': 'P001', 'nombre': 'Café', 'precio': 38.0, 'stock': 100}
2 print("\n--- Claves del producto ---")
3 for clave in producto: # Itera sobre las claves por defecto
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 # métodos para más control:
10 print("\n--- Usando producto.keys() ---")
11 for clave in producto.keys():
12     print(clave)
13 print("\n--- Usando producto.values() ---")
14 for valor in producto.values():
15     print(valor)
16 print("\n--- Usando producto.items() ---")
17 for clave, valor in producto.items():
18     print(f"{clave}: {valor}")
19 # Uso de .get() para acceso seguro y evitar errores
20 descuento = producto.get('descuento', 0.0) # 'descuento' no existe en el diccionario
21 print(f"\nDescuento aplicable: {descuento}") # Imprime 0.0, el valor por defecto
22 print("\nFin del programa ----- jose alejandro zabala romero")

```

En este código define un diccionario llamado producto con información como código, nombre, precio y stock. Usa varios ciclos for para recorrer el diccionario de distintas formas: primero directamente sobre las claves, luego accediendo a los valores mediante las claves, y después usando los métodos keys(), .values() y items() para tener mayor control en la iteración. Estos ciclos permiten mostrar la información de forma ordenada. También el uso de get() para acceder clave inexistente y asignarle un valor por defecto.

Ejecución

```
~/.../Practico_II/diccionario$ python b_for.py
--- Claves del producto ---
codigo
nombre
precio
stock

--- Clave y Valor ---
Codigo: P001
Nombre: Café
Precio: 38.0
Stock: 100

--- Usando producto.keys() ---
codigo
nombre
precio
stock

--- Usando producto.values() ---
P001
Café
38.0
100

--- Usando producto.items() ---
codigo: P001
nombre: Café
precio: 38.0
stock: 100

Descuento aplicable: 0.0

Fin del programa ----- jose alejandro zabala romero
```

3.1.3 Modelando un Producto

```
1 producto = {
2     "codigo": "P001",
3     "nombre": "Chocolate para Taza 'El Ceibo'",
4     "precio_unitario": 15.50,
5     "stock": 50,
6     "proveedor": "El Ceibo Ltda."
7 }
8 print(f"Producto: {producto['nombre']}")
9 print(f"Precio unitario: Bs {producto['precio_unitario']} ")
10 producto["stock"] -= 5
11 producto["en_oferta"] = True
12 print("\nDatos actualizados del producto:")
13 for clave, valor in producto.items():
14     print(f"{clave}: {valor}")
15 print("\nFin del programa ----- jose alejandro zabala romero")
```

En este código define un diccionario llamado producto que almacena información relevante de un artículo: su código, nombre, precio unitario, cantidad en stock y proveedor. Al inicio, imprime directamente el nombre del producto y su precio, accediendo a las claves específicas del diccionario. Luego simula una venta restando 5 unidades del stock usando producto["stock"] -= 5, lo que demuestra cómo se pueden modificar valores dentro del diccionario. Después agrega una nueva clave llamada en_oferta con el valor True, ampliando así la estructura del diccionario dinámicamente. Para mostrar toda la información actualizada, se utiliza con un ciclo for junto al método .items().

Ejecución

```
~/.../Practico_II/diccionario$ python m_prodt.py
Producto: Chocolate para Taza 'El Ceibo'
Precio unitario: Bs 15.5
Datos actualizados del producto:
codigo: P001
nombre: Chocolate para Taza 'El Ceibo'
precio_unitario: 15.5
stock: 45
proveedor: El Ceibo Ltda.
en_oferta: True
Fin del programa ----- jose alejandro zabala romero
```

3.1.4 Creando un Inventario

```
1 inventario = []
2 producto1 = {
3     "codigo": "P001",
4     "nombre": "Chocolate para Taza 'El Ceibo'",
5     "precio_unitario": 15.50,
6     "stock": 50,
7     "proveedor": "El Ceibo Ltda."
8 }
9 producto2 = {
10    "codigo": "P002",
11    "nombre": "Café de los Yungas",
12    "precio_unitario": 40.00,
13    "stock": 100,
14    "proveedor": "Cafés Andinos"
15 }
16 producto3 = {
17    "codigo": "P003",
18    "nombre": "Quinua Real en Grano",
19    "precio_unitario": 20.75,
20    "stock": 80,
21    "proveedor": "Granos Andinos"
22 }
23 inventario.append(producto1)
24 inventario.append(producto2)
25 inventario.append(producto3)
26 print("--- Inventario Actual ---")
27 print(f"Cantidad de productos diferentes: {len(inventario)}")
28 for producto in inventario:
29     print(f"- {producto['nombre']}: {producto['stock']} unidades en stock.")
30 print("\nFin del programa ----- jose alejandro zabala romero")
```

En este código crea un sistema de inventario usando listas y diccionarios. Primero se define una lista vacía llamada inventario, donde luego agrega tres productos representados como diccionarios. Cada producto contiene claves como codigo, nombre, precio_unitario, stock y proveedor, lo que permite organizar la información de forma clara. Uso .append() para añadir cada producto a la lista. Luego imprime la cantidad de productos con len() y uso un ciclo for para recorrer la lista e imprimir el nombre y stock de cada producto.

Ejecución

```
--- Inventario Actual ---
Cantidad de productos diferentes: 3
- Chocolate para Taza 'El Ceibo': 50 unidades en stock.
- Café de los Yungas: 100 unidades en stock.
- Quinua Real en Grano: 80 unidades en stock.

Fin del programa ----- jose alejandro zabala romero
```

3.1.5 Modelando el Mundo Real con Diccionarios

```
1 post_red_social = {
2     "id_post": "A123X",
3     "autor": {
4         "usuario": "Alejandro-X-911",
5         "nombre_completo": "José Alejandro Zabala Romero"
6     },
7     "contenido_texto": "¡Hoy aprendí a usar diccionarios en Python! #programacion",
8     "lista_de_likes": ["ana_s", "marco99", "coder_girl", "python_fan"],
9     "fecha_publicacion": "2025-07-19",
10    "comentarios": [
11        {"usuario": "ana_s", "texto": "¡Qué buena onda! "},
12        {"usuario": "marco99", "texto": "Python es genial "},
13    ],
14    "es_publico": True
15 }
16 print("--- Detalles del Post ---")
17 print(f"Author: {post_red_social['autor']['nombre_completo']}")
18 print(f"Publicado el: {post_red_social['fecha_publicacion']}")
19 print(f"Contenido: {post_red_social['contenido_texto']}")
20 print(f"Likes: {len(post_red_social['lista_de_likes'])} usuarios")
21 print(f"Comentarios: {len(post_red_social['comentarios'])} comentarios")
22 print("\nFin del programa ----- jose alejandro zabala romero")
```

El código define un diccionario llamado post_red_social que modela una publicación típica en una red social. La estructura es compleja (Para mi) porque combina distintos tipos de datos: cadenas de texto para el ID del post, el contenido y la fecha; un diccionario anidado para la información del autor con su usuario y nombre completo; listas para los likes y los comentarios; y un valor booleano para indicar si el post es público.

Ejecución

```
~/.../Practico_II/diccionario$ python post.py
--- Detalles del Post ---
Autor: José Alejandro Zabala Romero
Publicado el: 2025-07-19
Contenido: ¡Hoy aprendí a usar diccionarios en Python! #programacion
Likes: 4 usuarios
Comentarios: 2 comentarios

Fin del programa ----- jose alejandro zabala romero
```

3.2. Funciones y Operaciones Avanzadas con Registros

3.2.1 Mini-Aplicación: La "To-Do List"

➤ *tololist*

```
1 # === PASO 1: Crear estructura base ===
2 tareas = []
3 id_actual = 1
4 # === PASO 2: Función para agregar tarea ===
5 def agregar_tarea(descripcion, prioridad="media"):
6     global id_actual
7     nueva_tarea = {
8         "id": id_actual,
9         "descripcion": descripcion,
10        "prioridad": prioridad,
11        "completada": False
12    }
13    tareas.append(nueva_tarea)
14    print(f" Tarea '{descripcion}' añadida con éxito.")
15    id_actual += 1
16 # === PASO 3: Función para mostrar tareas ===
17 def mostrar_tareas():
18     print("\n--- LISTA DE TAREAS ---")
19     if not tareas:
20         print(" No hay tareas registradas.")
21     else:
22         for tarea in tareas:
23             estado = "✓" if tarea["completada"] else "■"
24             print(f"{estado} ID: {tarea['id']} | {tarea['descripcion']} (Prioridad: {tarea['prioridad']})")
25 # Prueba paso 3:
26 agregar_tarea("Estudiar para el examen de Cálculo")
27 agregar_tarea("Hacer las compras", prioridad="alta")
28 mostrar_tareas()
```

En esta primera parte se creó una lista vacía llamada tareas para almacenar las tareas y una variable id_actual para asignar identificadores únicos a cada tarea. Se definió una función agregar_tarea que recibe una descripción y una prioridad (con valor por defecto "media"), crea un diccionario con esos datos más un estado inicial de no completada, lo añade a la lista tareas y actualiza el ID para la siguiente tarea. Luego, la función mostrar_tareas imprime la lista de tareas con su estado (completada o no), ID, descripción y prioridad, mostrando un mensaje si la lista está vacía. Finalmente, se prueba ambas funciones agregando dos tareas y mostrando la lista.

➤ *buscar tarea por id*

```

29 # === PASO 4: Función para buscar tarea por ID ===
30 def buscar_tarea_por_id(id_busqueda):
31     for tarea in tareas:
32         if tarea["id"] == id_busqueda:
33             return tarea
34     return None
35 # Prueba paso 4:
36 tarea_encontrada = buscar_tarea_por_id(1)
37 if tarea_encontrada:
38     print(f"\n🟢 Búsqueda exitosa: {tarea_encontrada['descripcion']}")
```

```

39 else:
40     print("\n🔴 Búsqueda fallida: Tarea no encontrada.")
41 tarea_fantasma = buscar_tarea_por_id(99)
42 if not tarea_fantasma:
43     print("✅ Búsqueda de tarea inexistente funcionó correctamente.")
```

En esta sección define la función buscar_tarea_por_id que recibe un identificador y recorre la lista de tareas para encontrar una que coincida con ese ID. Si la encuentra, devuelve el diccionario de la tarea; si no, devuelve None. Esto permite buscar tareas específicas de forma rápida. Luego, se prueba la función buscando una tarea existente (ID 1) y mostrando su descripción, y también se prueba una búsqueda con un ID que no existe (99) para asegurar que la función maneje correctamente el caso cuando no encuentra nada.

➤ Funciones de Actualización y Borrado

```

44 # ===== PASO 5: Función para marcar como completada =====
45 def marcar_tarea_completada(id_tarea):
46     tarea = buscar_tarea_por_id(id_tarea)
47     if tarea:
48         tarea["completada"] = True
49         print(f"✓ Tarea '{tarea['descripcion']}' marcada como completada.")
50     else:
51         print("✗ Error: No se encontró la tarea con ese ID.")
52 # ===== PASO 6: Función para eliminar tarea =====
53 def eliminar_tarea(id_tarea):
54     global tareas
55     tarea = buscar_tarea_por_id(id_tarea)
56     if tarea:
57         tareas = [t for t in tareas if t["id"] != id_tarea]
58         print(f"✓ Tarea '{tarea['descripcion']}' eliminada.")
59     else:
60         print("✗ Error: No se encontró la tarea con ese ID.")
61 # Prueba paso 6:
62 mostrar_tareas()
63 marcar_tarea_completada(1)
64 mostrar_tareas() # La tarea 1 debería mostrarse como completada
65 eliminar_tarea(2)
66 mostrar_tareas() # La tarea 2 ya no debería aparecer
67 marcar_tarea_completada(99) # ID que no existe

```

En esta sección se añade dos funciones para la gestión de tareas. La función marcar_tarea_completada recibe un ID, busca la tarea correspondiente y, si la encuentra, cambia su estado a completada, mostrando un mensaje de confirmación. Si no encuentra la tarea, avisa con un mensaje de error. La función eliminar_tarea también usa el ID para buscar la tarea y, si existe, la elimina de la lista tareas filtrando todos los elementos excepto el que coincide con el ID, y confirma la eliminación. Si no encuentra la tarea, muestra un mensaje de error. Después se prueba estas funciones mostrando las tareas antes y después de marcar una como completada y eliminar otra, además de intentar marcar como completada una tarea que no existe para verificar el manejo de errores.

➤ *Orquestando con un Menú Principal*

```

68 # === PASO 7: Menú interactivo ===
69 def menu():
70     while True:
71         print("\n==== MENÚ TO-DO LIST ====")
72         print("1. Agregar nueva tarea")
73         print("2. Mostrar todas las tareas")
74         print("3. Marcar tarea como completada")
75         print("4. Eliminar tarea")
76         print("0. Salir")
77         opcion = input("Elige una opción: ")
78
79         if opcion == "1":
80             descripción = input("Descripción de la nueva tarea: ")
81             prioridad = input("Prioridad (alta, media, baja): ").lower()
82             agregar_tarea(descripción, prioridad)
83         elif opcion == "2":
84             mostrar_tareas()
85         elif opcion == "3":
86             try:
87                 id_tarea = int(input("ID de la tarea a completar: "))
88                 marcar_tarea_completada(id_tarea)
89             except ValueError:
90                 print("X Ingresa un número válido.")
91         elif opcion == "4":
92             try:
93                 id_tarea = int(input("ID de la tarea a eliminar: "))
94                 eliminar_tarea(id_tarea)
95             except ValueError:
96                 print("X Ingresa un número válido.")
97         elif opcion == "0":
98             print("¡Hasta pronto!")
99             break
100        else:
101            print("X Opción no válida. Intenta de nuevo.")
102    menu()

```

En esta sección implementa un menú interactivo que permite al usuario gestionar la lista de tareas. Usa un ciclo while True para mostrar las opciones hasta que el usuario decida salir. Según la opción que elija, se ejecutan las funciones definidas anteriormente: agregar una nueva tarea pidiendo descripción y prioridad, mostrar todas las tareas, marcar una tarea como completada o eliminar una tarea, solicitando en estos casos el ID correspondiente. Incluye manejo de errores para asegurar que el usuario ingrese números válidos al pedir los IDs, evitando que el programa falle por entradas incorrectas. Si el usuario elige salir, se rompe el ciclo.

Ejecución

```

~/.../Practico_II/registro$ python to-dolist.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)

🔍 Búsqueda exitosa: Estudiar para el examen de Cálculo
✅ Búsqueda de tarea inexistente funcionó correctamente.

--- 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)
✗ Error: No se encontró la tarea con ese ID.

===== MENÚ TO-DO LIST =====
1. Agregar nueva tarea
2. Mostrar todas las tareas
3. Marcar tarea como completada
4. Eliminar tarea
0. Salir
Elige una opción: 1
Descripción de la nueva tarea: estudiar para el segundo examen de programacion_II
Prioridad (alta, media, baja): alta
Tarea 'estudiar para el segundo examen de programacion_II' añadida con éxito.

===== MENÚ TO-DO LIST =====
1. Agregar nueva tarea
2. Mostrar todas las tareas
3. Marcar tarea como completada
4. Eliminar tarea
0. Salir
Elige una opción: 2

--- LISTA DE TAREAS ---
■ ID: 3 | estudiar para el segundo examen de programacion_II (Prioridad: alta)

===== MENÚ TO-DO LIST =====
1. Agregar nueva tarea
2. Mostrar todas las tareas
3. Marcar tarea como completada
4. Eliminar tarea
0. Salir
Elige una opción: 0
¡Hasta pronto!

```

Este programa es una aplicación de gestión de tareas o *to-do list*, desarrollada en Python.

A lo largo del código se construye una estructura que me permite agregar, mostrar, buscar, completar y eliminar tareas usando diccionarios y listas. Cada tarea tiene un ID único, una descripción, una prioridad y un estado (completada o no). Además, implementa un menú interactivo que facilita al usuario ejecutar todas estas funciones desde la consola. Este ejercicio me ayudó a practicar la creación de estructuras de datos, el uso de funciones, ciclos, condicionales, y el manejo de entradas del usuario, todo dentro de un caso práctico.

3.2.2 Gestor de Contactos 2.0

```

1 # Lista principal que almacena todos los contactos
2 agenda_contactos = []
3 # Función para agregar un contacto nuevo
4 def agregar_contacto(contacto):
5     agenda_contactos.append(contacto)
6     print(f"✓ Contacto '{contacto['nombre']}' agregado.")
7 # Función para buscar contactos por nombre (parcial o completo)
8 def buscar_por_nombre(nombre):
9     resultados = [c for c in agenda_contactos if nombre.lower() in c["nombre"].lower()]
10    return resultados
11 # Función para editar un campo de un contacto existente
12 def editar_contacto(nombre_original, campo, nuevo_valor):
13     for c in agenda_contactos:
14         if c["nombre"].lower() == nombre_original.lower():
15             c[campo] = nuevo_valor
16             print(f"✓ Contacto '{c['nombre']}' actualizado.")
17     return
18     print("✗ Contacto no encontrado.")
19 # Función para eliminar un contacto por nombre
20 def eliminar_contacto(nombre):
21     global agenda_contactos
22     agenda_contactos = [c for c in agenda_contactos if c["nombre"].lower() != nombre.lower()]
23     print(f"✗ Contacto '{nombre}' eliminado (si existía).")
24 # Función para mostrar todos los contactos en pantalla
25 def mostrar_contactos():
26     if not agenda_contactos:
27         print("■ Agenda vacía.")
28     else:
29         print("\n--- ■ Contactos ---")
30         for c in agenda_contactos:
31             print(f"■ {c['nombre']}")
32             print(f"  ☎ Teléfonos: {', '.join(c['telefonos'])}")
33             print(f"  📩 Email: {c['email']}")
34             print(f"  ⌂ Dirección: {c['direccion']}")
35             print(f"  █ Notas: {c['notas']} \n")
36 # ===== SIMULACIÓN DE USO =====
37 # Crear algunos contactos de ejemplo
38 contacto1 = {
39     "nombre": "Alejandro Romero",
40     "telefonos": ["78945612", "76543210"],
41     "email": "alejandro@mail.com",
42     "direccion": "Av. Principal 123",
43     "notas": "Estudiante de informática"
44 }
45 contacto2 = {
46     "nombre": "Lucía Fernández",
47     "telefonos": ["76512345"],
48     "email": "luciaf@gmail.com",
49     "direccion": "Calle Secundaria 45",
50     "notas": "Vecina"
51 }
52 # Agregar contactos
53 agregar_contacto(contacto1)
54 agregar_contacto(contacto2)
55 # Mostrar todos los contactos
56 mostrar_contactos()
57 # Buscar por nombre
58 print("🔎 Buscando 'lucía':")
59 resultados = buscar_por_nombre("lucía")
60 for r in resultados:
61     print(f"✓ Encontrado: {r['nombre']} - {r['email']}")
62 # Editar un campo
63 editar_contacto("Lucía Fernández", "email", "lucia.fernandez@outlook.com")
64 # Mostrar los contactos actualizados
65 mostrar_contactos()
66 # Eliminar un contacto
67 eliminar_contacto("Alejandro Romero")
68 # Mostrar la agenda tras eliminar
69 mostrar_contactos()

```

Este código está estructurado alrededor de una lista principal llamada `agenda_contactos`, que almacena todos los contactos como diccionarios. Cada contacto contiene varias claves: nombre (una cadena), teléfonos (una lista de cadenas), email, dirección y notas. El programa está dividido en funciones específicas, la función `agregar_contacto` simplemente añade un nuevo diccionario de contacto a la lista. `buscar_por_nombre` utiliza comprensión de listas para filtrar los contactos cuyo nombre coincida parcial o completamente con el valor buscado, sin importar mayúsculas o minúsculas.

La función `editar_contacto` recorre la lista y actualiza un campo específico de un contacto si encuentra una coincidencia exacta por nombre. Luego está `eliminar_contacto`, que vuelve a asignar a la lista solo aquellos contactos cuyo nombre no coincide con el que se quiere eliminar, utilizando también comprensión de listas. Finalmente, `mostrar_contactos` imprime en pantalla todos los datos organizados de cada contacto.

Ejecución

```
~/.../Práctico_II/registro$ python g_cont.py
Contacto 'Alejandro Romero' agregado.
Contacto 'Lucía Fernández' agregado.

--- Contactos ---
Alejandro Romero
Teléfonos: 78945612, 76543210
Email: alejandro@email.com
Dirección: Av. cumavi
Notas: Estudiante de informática

Lucía Fernández
Teléfonos: 76512345
Email: luciaf@gmail.com
Dirección: Av cumavi
Notas: Vecina

Buscando 'lucía':
Encontrado: Lucía Fernández - luciaf@gmail.com
Contacto 'Lucía Fernández' actualizado.

--- Contactos ---
Alejandro Romero
Teléfonos: 78945612, 76543210
Email: alejandro@email.com
Dirección: Av. cumavi
Notas: Estudiante de informática

Lucía Fernández
Teléfonos: 76512345
Email: lucia.fernandez@outlook.com
Dirección: Av cumavi
Notas: Vecina

Contacto 'Alejandro Romero' eliminado (si existía).

--- Contactos ---
Lucía Fernández
Teléfonos: 76512345
Email: lucia.fernandez@outlook.com
Dirección: Av cumavi
Notas: Vecina
```

3.3 Introducción a Clases y Objetos

3.3.1 Métodos

```

1  class Estudiante:
2      def __init__(self, nombre, edad, carrera):
3          self.nombre = nombre
4          self.edad = edad
5          self.carrera = carrera
6          self.materias = []
7      # Método para presentarse
8      def presentarse(self):
9          print(f"Hello, my name is {self.nombre}, I am {self.edad} years old and I study {self.carrera}.")
10     # Método para inscribirse en materias
11     def inscribir_materia(self, nombre_materia):
12         self.materias.append(nombre_materia)
13         print(f"{self.nombre} has successfully registered in {nombre_materia}!")
14     # Ejemplo de uso
15 estudiante1 = Estudiante("Alejandro Romero", 21, "Ing. en Redes y Telecomunicaciones")
16 estudiante1.presentarse()
17 estudiante1.inscribir_materia("Programación II")
18 estudiante1.inscribir_materia("Taller de Sistemas Operativos I")
19 print(f"Materials of Alejandro: {estudiante1.materias}")
20 print("End of program ----- Richard hurtado")

```

Este código es un ejemplo de cómo se puede utilizar la programación orientada a objetos para modelar entidades del mundo real, como un estudiante. La clase Estudiante encapsula tanto los atributos (nombre, edad, carrera, materias) como los métodos que permiten al objeto interactuar con sus propios datos. El método presentarse permite imprimir una presentación personalizada, mientras que inscribir_materia modifica el estado interno del objeto agregando nuevas materias a su lista.

Ejecución

```

~/.../Práctico_II/Poo$ python métodos.py
Hello, my name is Alejandro Romero, I am 21 years old and I study Ing. in Networks and Telecommunications.
Alejandro Romero has successfully registered in Programación II!
Alejandro Romero has successfully registered in Taller de Sistemas Operativos I!
Materials of Alejandro: ['Programación II', 'Taller de Sistemas Operativos I']
End of program ----- Richard hurtado
~/.../Práctico_II/Poo$ 

```

3.3.2 Refactorizando un Libro de Diccionario a Clase

```

1  class Libro:
2      """
3          Clase que representa un libro con sus atributos principales.
4      """
5      def __init__(self, titulo, autor, isbn, paginas):
6          # Crear los atributos de instancia correspondientes
7          self.titulo = titulo
8          self.autor = autor
9          self.isbn = isbn
10         self.paginas = paginas
11         # Atributo extra que se inicializa siempre por defecto
12         self.disponible = True
13     def mostrar_info(self):
14         """
15             Método que imprime todos los atributos del libro de forma clara y formateada.
16         """
17         print("=" * 50)
18         print("INFORMACIÓN DEL LIBRO")
19         print("=" * 50)
20         print(f"Título: {self.titulo}")
21         print(f"Autor: {self.autor}")
22         print(f"ISBN: {self.isbn}")
23         print(f"Páginas: {self.paginas}")
24         print(f"Disponible: {'Sí' if self.disponible else 'No'}")
25         print("=" * 50)
26     # Ejemplo de uso (no te preocupes por crear objetos todavía, solo define la clase!)
27     if __name__ == "__main__":
28         # Creación de ejemplo para demostrar el funcionamiento
29         libro1 = Libro("Cien años de soledad", "Gabriel García Márquez", "978-0307474728", 417)
30         libro1.mostrar_info()
31         # Cambiar disponibilidad
32         libro1.disponible = False
33         print("\nDespués de cambiar disponibilidad:")
34         libro1.mostrar_info()
35         print ("fin del programa ----- Lucas Mateo Diaz Badano")

```

Este programa define una clase Libro que representa un libro mediante atributos como título, autor, ISBN, número de páginas y disponibilidad (disponible, por defecto True). Está documentada con docstrings. El constructor `__init__` inicializa los atributos, y el método `mostrar_info` imprime sus valores con formato claro usando f-strings, incluyendo una condición para mostrar "Sí" o "No" según la disponibilidad. En el bloque `if __name__ == "__main__":` se crea una instancia, se muestra su información, se modifica su disponibilidad a False y se vuelve a imprimir para reflejar el cambio.

Ejecución

```
~/.../Practico_II/Poo$ python poo1.py
=====
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
=====
```

3.3.3 Creación de instancias y métodos de comportamiento

```
1  class Libro:
2      def __init__(self, titulo, autor, isbn, paginas):
3          self.titulo = titulo
4          self.autor = autor
5          self.isbn = isbn
6          self.paginas = paginas
7          self.disponible = True
8      def mostrar_info(self):
9          print("=" * 50)
10         print("INFORMACIÓN DEL LIBRO")
11         print("=" * 50)
12         print(f"Título: {self.titulo}")
13         print(f" Autor: {self.autor}")
14         print(f" ISBN: {self.isbn}")
15         print(f" Páginas: {self.paginas}")
16         print(f" Disponible: {'Sí' if self.disponible else 'No'}")
17         print("=" * 50)
18     def prestar_libro(self):
19         if self.disponible:
20             self.disponible = False
21             print(f"El libro '{self.titulo}' ha sido prestado.")
22         else:
23             print(f"El libro '{self.titulo}' ya está prestado.")
24     def devolver_libro(self):
25         if not self.disponible:
26             self.disponible = True
27             print(f"El libro '{self.titulo}' ha sido devuelto.")
28         else:
29             print(f"El libro '{self.titulo}' ya estaba disponible.")
30 # Crear objetos
31 libro1 = Libro("El Principito", "Antoine de Saint-Exupéry", "978-3-14-046401-7", 120)
32 libro2 = Libro("Raza de Bronce", "Alcides Arguedas", "978-99905-2-213-9", 250)
33 # Acceso directo a atributos
34 print(f"\nEl autor del primer libro es: {libro1.autor}")
35 print(f"El ISBN del segundo libro es: {libro2.isbn}")
36 # Mostrar info
37 print("\n--- Mostrando información completa ---")
```

```

38 libro1.mostrar_info()
39 libro2.mostrar_info()
40 # Probar métodos de comportamiento
41 print("\n--- Probando métodos de comportamiento ---")
42 libro1.prestar_libro()
43 libro1.prestar_libro()
44 libro1.devolver_libro()
45 libro1.devolver_libro()
46 print("\n--- Probando con libro2 ---")
47 libro2.prestar_libro()
48 libro2.devolver_libro()
49 print("\n--- Estado final de los libros ---")
50 libro1.mostrar_info()
51 libro2.mostrar_info()
52 print ("Fin del programa ----- Lucas Mateo Diaz Badan")

```

El programa define una clase Libro que representa un libro con los atributos: título, autor, ISBN, número de páginas y un indicador de disponibilidad (disponible). El constructor `__init__` se encarga de inicializar estos atributos cuando se crea un objeto.

Dentro de la clase se implementan tres métodos. El primero, `mostrar_info()`, imprime todos los datos del libro de forma clara, incluyendo si el libro está disponible o no. El segundo, `prestar_libro()`, cambia el estado del libro a “no disponible” si aún no ha sido prestado, mostrando un mensaje que confirma la acción o advierte si ya estaba prestado.

El tercero, `devolver_libro()`, vuelve a marcar el libro como disponible, también mostrando un mensaje adecuado según el caso. Después de la definición de la clase, el código crea dos objetos: uno representa el libro *El Principito* y el otro *Raza de Bronce*. Se accede directamente a algunos de sus atributos, se imprime su información completa y se prueban los métodos de préstamo y devolución.

Ejecución

```
~/.../Practico_II/Poo$ python poo2.py
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.

--- 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
```

3.3.4 Usando Listas de Objetos

```
1  class Libro:
2      """ Clase que representa un libro con sus atributos principales. """
3      def __init__(self, titulo, autor, isbn, paginas):
4          """Constructor de la clase Libro. """
5          self.titulo = titulo
6          self.autor = autor
7          self.isbn = isbn
8          self.paginas = paginas
9          self.disponible = True
10     def mostrar_info(self):
11         """ Método que imprime todos los atributos del libro de forma clara y formateada. """
12         print("=" * 50)
13         print("INFORMACIÓN DEL LIBRO")
14         print("=" * 50)
15         print(f"Tituloo: {self.titulo}")
16         print(f"Autoo: {self.autor}")
17         print(f"ISBN: {self.isbn}")
18         print(f"Páginas: {self.paginas}")
19         print(f"Disponible: {'Sí' if self.disponible else 'No'}")
20         print("=" * 50)
21     if __name__ == "__main__":
22         # Crear objetos de tipo Libro
23         libro1 = Libro("Cien años de soledad", "Gabriel García Márquez", "978-0307474728", 417)
24         libro2 = Libro("1984", "George Orwell", "978-0451524935", 328)
25         libro3 = Libro("El Principito", "Antoine de Saint-Exupéry", "978-0156013987", 96)
26         # Crear una lista vacía
27         mi_biblioteca = []
28         # Añadir libros a la lista
29         mi_biblioteca.append(libro1)
30         mi_biblioteca.append(libro2)
31         mi_biblioteca.append(libro3)
32         # Mostrar el inventario completo
33         print("\n\n--- INVENTARIO COMPLETO DE LA BIBLIOTECA ---")
34         for libro_actual in mi_biblioteca:
35             libro_actual.mostrar_info()
36             print("=" * 20) # Separador
37     print ("Fin del programa ----- Jose Alejandro Zabala Romero")
```

El código define una clase llamada Libro con atributos clave: título, autor, ISBN, número de páginas y un estado disponible que se inicializa en True. Incluye un método mostrar_info que imprime toda la información del libro de forma clara y ordenada, usando líneas y formato para mejorar la presentación.

En la parte principal, se crean tres objetos de la clase con datos distintos y se agregan a una lista llamada mi_biblioteca. Luego, se recorre esta lista y se llama al método mostrar_info de cada libro para mostrar sus datos uno por uno, separándolos con líneas para facilitar la lectura.

La parte más compleja está en el método mostrar_info, donde se debe asegurar imprimir correctamente todos los atributos, incluyendo mostrar “Sí” o “No” según el estado disponible,

Ejecución

```
--- 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
=====

Fin del programa ---- Jose Alejandro Zabala Romero
```

3.3.5 Diseño de Clases

```

1  class Producto:
2      def __init__(self, id, nombre, precio, stock):
3          self.id = id
4          self.nombre = nombre
5          self.precio = precio
6          self.stock = stock
7      def actualizar_stock(self, cantidad):
8          self.stock += cantidad
9      def esta_disponible(self):
10         return self.stock > 0
11     def mostrar_info(self):
12         print(f"Producto: {self.nombre} | Precio: ${self.precio} | Stock: {self.stock}")
13 class Cliente:
14     def __init__(self, id, nombre, email, direccion_envio):
15         self.id = id
16         self.nombre = nombre
17         self.email = email
18         self.direccion_envio = direccion_envio
19     def actualizar_direccion(self, nueva_direccion):
20         self.direccion_envio = nueva_direccion
21     def mostrar_info(self):
22         print(f"Cliente: {self.nombre} | Email: {self.email} | Dirección: {self.direccion_envio}")
23 class CarritoDeCompras:
24     def __init__(self, cliente):
25         self.cliente = cliente
26         self.productos = {} # clave: id del producto, valor: [producto, cantidad]
27     def agregar_producto(self, producto, cantidad):
28         if producto.id in self.productos:
29             self.productos[producto.id][1] += cantidad
30         else:
31             self.productos[producto.id] = [producto, cantidad]
32     def calcular_total(self):
33         total = 0
34         for producto, cantidad in self.productos.values():
35             total += producto.precio * cantidad
36         return total
37     def vaciar_carrito(self):
38         self.productos.clear()
39     def mostrar_carrito(self):
40         print(f"\nCarrito de {self.cliente.nombre}:")
41         for producto, cantidad in self.productos.values():
42             print(f"- {producto.nombre}: {cantidad} unidades x ${producto.precio} = ${producto.precio * cantidad}")
43         print(f"Total a pagar: ${self.calcular_total()}")
44 # BLOQUE DE PRUEBA
45 if __name__ == "__main__":
46     # Crear productos
47     p1 = Producto(1, "Laptop", 1000, 5)
48     p2 = Producto(2, "Mouse", 25, 20)
49     p3 = Producto(3, "Auriculares", 50, 10)
50     # Mostrar productos disponibles
51     print("== PRODUCTOS DISPONIBLES ==")
52     p1.mostrar_info()
53     p2.mostrar_info()
54     p3.mostrar_info()
55     # Crear cliente (Alejandro Romero)
56     cliente1 = Cliente(1, "Alejandro Romero", "alejandro@example.com", "Av. Central 456")
57     # Crear carrito para el cliente
58     carrito = CarritoDeCompras(cliente1)
59     # Agregar productos al carrito
60     carrito.agregar_producto(p1, 1) # 1 Laptop
61     carrito.agregar_producto(p2, 2) # 2 Mouse
62     carrito.agregar_producto(p3, 1) # 1 Auriculares
63     # Mostrar el contenido del carrito
64     carrito.mostrar_carrito()

```

En este código se diseñó un sistema para una tienda online. A partir del escenario planteado, se identificó tres clases: Producto, Cliente y CarritoDeCompras. Cada una de estas clases cuenta con atributos representativos y métodos que permiten simular acciones comunes dentro de una tienda virtual. La clase Producto representa un artículo que se puede vender, incluyendo su nombre, precio, código y stock disponible. La clase Cliente modela al usuario que realiza las compras, con datos como su nombre, correo y dirección. Por último, la clase CarritoDeCompras simula el carrito donde se agregan productos, relacionando los objetos Producto con un Cliente en particular. En cuanto a las funciones, Producto y Cliente tienen métodos para mostrar su información. CarritoDeCompras permite agregar productos al carrito, validando si hay suficiente stock antes de hacerlo, y también muestra el resumen del carrito con los productos añadidos y el total acumulado. Para probar el sistema, usamos un cliente llamado Alejandro Romero y se agregaron dos productos al carrito. La lógica de control de stock dentro del carrito fue uno de los aspectos más complejos del diseño, ya que requiere verificar condiciones antes de aceptar una operación.

Ejecución

```
== PRODUCTOS DISPONIBLES ==
Producto: Laptop | Precio: $1000 | Stock: 5
Producto: Mouse | Precio: $25 | Stock: 20
Producto: Auriculares | Precio: $50 | Stock: 10

Carrito de Alejandro Romero:
- Laptop: 1 unidades x $1000 = $1000
- Mouse: 2 unidades x $25 = $50
- Auriculares: 1 unidades x $50 = $50
Total a pagar: $1100
```

Capítulo IV

Guardando y Recuperando Datos desde Archivos

4.1 Introducción al Manejo de Archivos (Texto)

4.1.1 Creando y Escribiendo Nuestro Diario

```

1 # 1. Definimos el nombre del archivo
2 nombre_archivo = "mi_diario.txt"
3 # 2. Abrimos el archivo en modo escritura ('w')
4 with open(nombre_archivo, 'w') as diario_file:
5     # 3. Escribimos varias entradas en el archivo
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 # 4. Confirmamos que el archivo fue escrito
10 print("Diario guardado correctamente en 'mi_diario.txt'")
11 print ("fin del programa --- Richard hurtado")

```

Este código ejemplifica el proceso de creación y escritura en un archivo de texto. Se define primero el nombre del archivo como una cadena, y luego se utiliza la instrucción `with open(..., 'w')` para abrir el archivo en modo escritura. Este modo permite escribir desde cero, eliminando cualquier contenido previo si el archivo ya existe. Dentro del bloque `with`, se emplea el método `.write()` para registrar varias líneas de texto, y se utiliza `\n` al final de cada línea para asegurar una correcta separación al momento de la lectura. Fuera del bloque, se incluye un `print()` que actúa como confirmación visual de que la operación de escritura se realizó.

Ejecución

```

~/worksapce$ cd practico_II/archivos
~/.../Practico_II/archivos$ python diario.py
Diario guardado correctamente en 'mi_diario.txt'
fin del programa --- Richard hurtado
~/.../Practico_II/archivos$ █

```

```

1 Querido diario,
2 Hoy aprendí sobre archivos en Python.
3 El modo 'w' borra todo antes de escribir. ¡Qué miedo!
4

```

4.1.2 Leer Nuestro Diario

```

1 # Definimos el nombre del archivo
2 nombre_archivo = "mi_diario.txt"
3 # Escritura
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 print("Diario guardado correctamente.")
9 # Lectura (línea por línea con manejo de errores)
10 print("\n--- Contenido del diario (línea por línea) ---")
11 try:
12     with open(nombre_archivo, 'r') as diario_file:
13         for linea in diario_file:
14             print(linea.strip())
15 except FileNotFoundError:
16     print(f"Error: El archivo '{nombre_archivo}' no existe.")
17 print("\n--- Fin del programa ---- Leonardo Montenegro")

```

Este código trabaja con un archivo de texto llamado "mi_diario.txt". Primero abre el archivo en modo escritura, lo que borra cualquier contenido previo, y escribe algunas líneas simulando un diario personal. Luego cierra automáticamente el archivo y muestra un mensaje indicando que el diario fue guardado. Después, intenta abrir el mismo archivo en modo lectura para mostrar su contenido línea por línea. Utiliza un manejo de errores para evitar que el programa falle si el archivo no existe. Cada línea leída se imprime sin los saltos de línea extras.

Ejecución

```

~/.../Practico_II/archivos$ python diario2.py
Diario guardado correctamente.

--- Contenido del diario (línea por línea) ---
Querido diario,
Hoy aprendí sobre archivos en Python.
El modo 'w' borra todo antes de escribir. Qué miedo!

--- Fin del programa ---- Leonardo Montenegro
~/.../Practico_II/archivos$ █

```

4.1.3 Continuar Nuestro Diario

```

1 # Definimos el nombre del archivo
2 nombre_archivo = "mi_diario.txt"
3 # Escritura inicial (modo 'w' borra todo y escribe)
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 print("Diario guardado correctamente.")
9 # Añadimos nuevas entradas sin borrar las anteriores (modo 'a')
10 print("\nAñadiendo nuevas entradas al diario...")
11 with open(nombre_archivo, 'a') as diario_file:
12     diario_file.write("\n--- Entrada del 20 de Junio de 2025 ---\n")
13     diario_file.write("El modo 'a' es genial para no perder datos.\n")
14     diario_file.write("Ahora mi diario puede crecer cada día.\n")
15 print("¡Nuevas entradas guardadas!")
16 # Lectura línea por línea con manejo de errores (Opción B)
17 print("\n--- Contenido del diario (línea por línea) ---")
18 try:
19     with open(nombre_archivo, 'r') as diario_file:
20         for linea in diario_file:
21             print(linea.strip())
22 except FileNotFoundError:
23     print(f"Error: El archivo '{nombre_archivo}' no existe.")
24 print("\n--- Fin del programa ---- Leonardo Montenegro")

```

Este código maneja un archivo de texto llamado "mi_diario.txt" realizando tres acciones principales. Primero, escribe contenido inicial en el archivo usando el modo 'w', que borra cualquier dato previo antes de escribir. Luego, añade nuevas entradas al archivo en modo añadir ('a'), lo que permite conservar el contenido existente y agregar más texto al final. Finalmente, intenta leer y mostrar todo el contenido del archivo línea por línea, utilizando un bloque try-except para manejar la posible ausencia del archivo y evitar que el programa falle.

Ejecución

```

~/.../Practico_II/archivos$ python diario3.py
Diario guardado correctamente.

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

--- Contenido del diario (línea por línea) ---
Querido diario,
Hoy aprendí sobre archivos en Python.
El modo 'w' borra todo antes de escribir. ¡Qué miedo!

--- Entrada del 20 de Junio de 2025 ---
El modo 'a' es genial para no perder datos.
Ahora mi diario puede crecer cada día.

--- Fin del programa ---- Leonardo Montenegro
~/.../Practico_II/archivos$ █

```

4.1.4 Diario Persistente

```

1  nombre_archivo = "mi_diario.txt"
2  # Verificamos si el archivo existe; si no, lo creamos vacío
3  import os
4  if not os.path.exists(nombre_archivo):
5      with open(nombre_archivo, 'w') as f:
6          f.write("")
7  # Pedimos al usuario que escriba su nueva entrada
8  entrada = input("Escribe tu nueva entrada para el diario:\n")
9  # Añadimos un separador y la entrada del usuario al archivo en modo añadir ('a')
10 with open(nombre_archivo, 'a') as diario_file:
11     diario_file.write("\n--- Nueva entrada ---\n")
12     diario_file.write(entrada + "\n")
13 print("¡Tu entrada ha sido guardada!")
14 # Mostrar todo el contenido actualizado
15 print("\n--- Contenido completo del diario ---")
16 try:
17     with open(nombre_archivo, 'r') as diario_file:
18         for linea in diario_file:
19             print(linea.strip())
20 except FileNotFoundError:
21     print(f"Error: El archivo '{nombre_archivo}' no existe.")
22 print("\n--- Fin del programa ---- Leonardo Montenegro")

```

El script implementa una aplicación de diario interactiva y persistente. Utiliza `input()` para permitir que el usuario escriba nuevas entradas, y almacena esta información en el archivo `mi_diario.txt` usando el modo `'a'`, lo cual garantiza que las entradas anteriores no se pierdan. También incluye una función de lectura que muestra el contenido completo del diario, haciendo que el usuario pueda revisar sus anotaciones pasadas.

Ejecucion

```

~/.Práctico_II/archivos$ python D_P.py
Escribe tu nueva entrada para el diario:
jum125
¡Tu entrada ha sido guardada!

--- Contenido completo del diario ---
Querido diario,
Hoy aprendí sobre archivos en Python.
El modo 'w' borra todo antes de escribir. ¡Qué miedo!

--- Entrada del 20 de Junio de 2025 ---
El modo 'a' es genial para no perder datos.
Ahora mi diario puede crecer cada día.

--- Nueva entrada ---
jum125

--- Fin del programa ---- Leonardo Montenegro
~/.Práctico_II/archivos$ 

```

```

1  Querido diario,
2  Hoy aprendí sobre archivos en Python.
3  El modo 'w' borra todo antes de escribir. ¡Qué miedo!
4
5  --- Entrada del 20 de Junio de 2025 ---
6  El modo 'a' es genial para no perder datos.
7  Ahora mi diario puede crecer cada día.
8
9  --- Nueva entrada ---
10 jum125
11

```

4.2 Guardando y Cargando Estructuras de Datos

4.2.1 CSV (Valores Separados por Comas)

➤ *El Módulo csv de Python: Nuestro Ayudante para CSV*

```

1  import csv
2
3  # Datos a guardar en el archivo
4  datos = [{"nombre": "Alejandro", "edad": 20},
5             {"nombre": "Richard", "edad": 22}]
6  encabezados = ["nombre", "edad"]
7
8  # Escritura del archivo CSV
9  with open("datos.csv", "w", newline="", encoding="utf-8") as archivo_csv:
10     escritor = csv.DictWriter(archivo_csv, fieldnames=encabezados)
11     escritor.writeheader()           # Escribe los encabezados: nombre, edad
12     escritor.writerows(datos)       # Escribe cada diccionario como una fila
13
14 # Lectura del archivo CSV
15 lista_leida = []
16 with open("datos.csv", "r", newline="", encoding="utf-8") as archivo_csv:
17     lector = csv.DictReader(archivo_csv)
18     for fila_dict in lector:
19         fila_dict['edad'] = int(fila_dict['edad']) # Convierte edad a entero
20         lista_leida.append(fila_dict)
21
22 # Mostramos lo leído
23 print(lista_leida)
24 print ("fin del programa ---- richard hurtado")

```

El código guarda y lee datos en un archivo CSV usando el módulo csv. Primero, escribe una lista de diccionarios con DictWriter, incluyendo encabezados. Luego, lee el archivo con DictReader, convirtiendo el campo 'edad' de texto a entero. Es útil para manejar datos tipo tabla sin preocuparse por el formato manual de comas o comillas.

Ejecución

```
~/.../Practico_II/CSV$ python modulo.py
[{'nombre': 'Alejandro', 'edad': 20}, {'nombre': 'Richard', 'edad': 22}]
fin del programa ---- richard hurtado
~/.../Practico_II/CSV$
```

	nombre	edad
✓	Alejandro	20
✓	Richard	22
✓		

➤ Guardar y Cargar Inventario con CSV

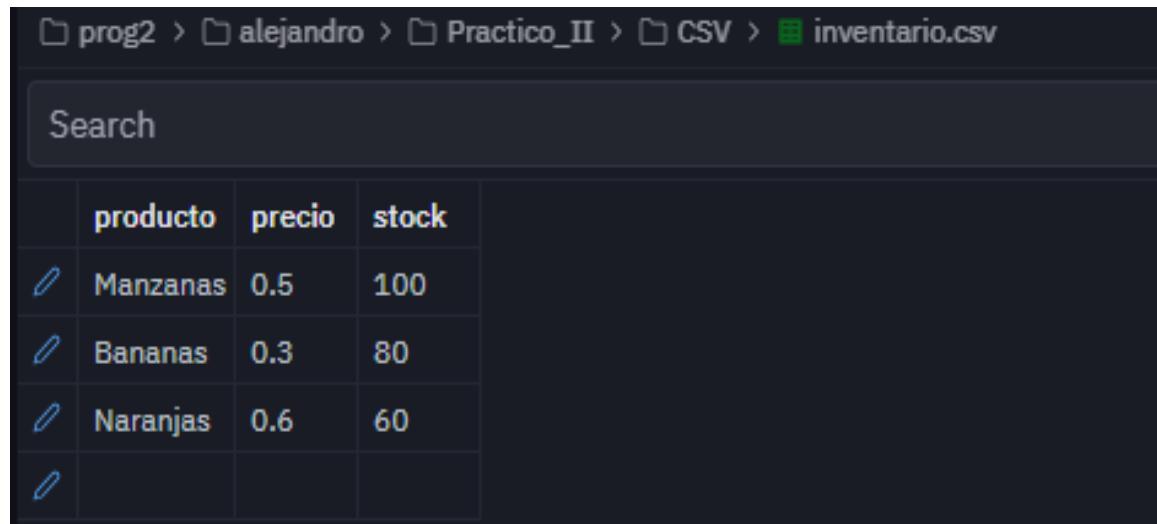
```
1 import csv
2 import os
3 # ----- PARTE 1: GUARDAR EL INVENTARIO EN CSV -----
4 # Lista del inventario original
5 inventario = [ {"producto": "Manzanas", "precio": 0.5, "stock": 100},
6                 {"producto": "Bananas", "precio": 0.3, "stock": 80},
7                 {"producto": "Naranjas", "precio": 0.6, "stock": 60}]
8 # Encabezados (claves de los diccionarios)
9 encabezados = ["producto", "precio", "stock"]
10 # Guardar en archivo CSV
11 nombre_archivo = "inventario.csv"
12 with open(nombre_archivo, mode='w', newline='') as archivo_csv:
13     escritor_csv = csv.DictWriter(archivo_csv, fieldnames=encabezados)
14     escritor_csv.writeheader()
15     escritor_csv.writerows(inventario)
16 print(" Inventario guardado exitosamente en 'inventario.csv' .")
17 # ----- PARTE 2: CARGAR EL INVENTARIO DESDE CSV -----
18 # Lista vacía para cargar el contenido
19 inventario_cargado_csv = []
20 # Leer el archivo CSV
21 with open(nombre_archivo, mode='r') as archivo_csv:
22     lector_csv = csv.DictReader(archivo_csv)
23     for fila in lector_csv:
24         # Convertimos tipos correctamente
25         producto = fila["producto"]
26         precio = float(fila["precio"])
27         stock = int(fila["stock"])
28         # Añadimos a la lista cargada
29         producto_dic = { "producto": producto,
30                           "precio": precio,
31                           "stock": stock }
32         inventario_cargado_csv.append(producto_dic)
33 # Mostrar el contenido cargado
34 print("\n▣ Inventario cargado desde CSV:")
35 for item in inventario_cargado_csv:
36     print(item)
37 print("\n--- Fin del programa ---- Jose Alejandro Zabala Romero")
```

Ejecución

```
~/.../Practico_II/CSV$ python Ej1.py
Inventario guardado exitosamente en 'inventario.csv'.

■ Inventario cargado desde CSV:
{'producto': 'Manzanas', 'precio': 0.5, 'stock': 100}
{'producto': 'Bananas', 'precio': 0.3, 'stock': 80}
{'producto': 'Naranjas', 'precio': 0.6, 'stock': 60}

--- Fin del programa ---- Jose Alejandro Zabala Romero
~/.../Practico_II/CSV$
```



The screenshot shows a terminal window at the top with the command `python Ej1.py` and its output. Below it is a file browser window showing the directory structure and the contents of the `inventario.csv` file.

	producto	precio	stock
0	Manzanas	0.5	100
1	Bananas	0.3	80
2	Naranjas	0.6	60
3			

El código que utiliza CSV para guardar y cargar el inventario es una solución práctica y sencilla para manejar datos tabulares. Este formato es ampliamente compatible con diferentes programas y sistemas, como hojas de cálculo, lo que facilita la interoperabilidad. Usar `csv.DictWriter` y `csv.DictReader` permite transformar fácilmente entre listas de diccionarios y archivos CSV. Sin embargo, al leer un archivo CSV, todos los datos se interpretan como texto, por lo que es necesario convertir manualmente los valores numéricos a sus tipos correctos para poder operar con ellos. Además, el formato CSV es limitado en cuanto a estructuras de datos, ya que solo maneja datos planos y puede requerir atención especial para caracteres especiales o delimitadores.

4.2.2 JSON (JavaScript Object Notation)

- *El Módulo json de Python: Simplicidad y Poder*

```

1 import json
2
3 # Datos en formato Python (lista de diccionarios)
4 datos_python = [{"nombre": "richard", "edad": 20}]
5
6 # Escritura del archivo JSON (serialización)
7 with open("datos.json", "w", encoding="utf-8") as archivo_json:
8     json.dump(datos_python, archivo_json, indent=4, ensure_ascii=False)
9
10 # Lectura del archivo JSON (deserialización)
11 with open("datos.json", "r", encoding="utf-8") as archivo_json:
12     datos_cargados = json.load(archivo_json)
13
14 # Verificación de tipos
15 print(type(datos_cargados))           # <class 'list'>
16 print(type(datos_cargados[0]['edad'])) # <class 'int'>
17 print(datos_cargados)                # Muestra el contenido cargado
18 print ("fin del programa --- leonardo montenegro")

```

El código guarda una lista de diccionarios en un archivo JSON usando `json.dump()` y luego la recupera con `json.load()`. Se conserva la estructura y los tipos de datos originales, como enteros. Es una forma práctica de almacenar datos estructurados de forma legible y compatible con otros sistemas.

Ejecución

```

~/.../Práctico_II/CSV$ python modulo2.py
<class 'list'>
<class 'int'>
[{"nombre": "richard", "edad": 20}]
fin del programa --- leonardo montenegro
~/.../Práctico_II/CSV$ █

```

```

prog2 > alejandro > Práctico_II > CSV > datos.json
1 [
2   {
3     "nombre": "richard",
4     "edad": 20
5   }
6 ]

```

➤ *Guardar y Cargar Inventario con JSON*

```

1 import json
2 # ----- PARTE 1: GUARDAR EL INVENTARIO EN JSON -----
3 # Lista del inventario original
4 inventario = [
5     {"producto": "Manzanas", "precio": 0.5, "stock": 100},
6     {"producto": "Bananas", "precio": 0.3, "stock": 80},
7     {"producto": "Naranjas", "precio": 0.6, "stock": 60}
8 ]
9 # Guardar en archivo JSON (modo escritura)
10 nombre_archivo_json = "inventario.json"
11 with open(nombre_archivo_json, mode='w') as archivo_json:
12     json.dump(inventario, archivo_json, indent=4)
13 print(" Inventario guardado exitosamente en 'inventario.json' .")
14 # ----- PARTE 2: CARGAR EL INVENTARIO DESDE JSON -----
15 # Leer el archivo JSON
16 with open(nombre_archivo_json, mode='r') as archivo_json:
17     inventario_cargado_json = json.load(archivo_json)
18 # Mostrar el contenido cargado
19 print("\n Inventario cargado desde JSON:")
20 for item in inventario_cargado_json:
21     print(item)
22 # Verificar tipo de dato de 'stock' del primer producto
23 print("\n Tipo de dato de 'stock':", type(inventario_cargado_json[0]['stock']))
24 print("\n--- Fin del programa ---- Jose Alejandro Zabala Romero")

```

Por otro lado, JSON mantiene automáticamente los tipos de datos originales, como enteros y flotantes, lo que simplifica la lectura y escritura sin necesidad de convertir manualmente los valores. Este formato también es legible para humanos y ampliamente utilizado en aplicaciones modernas, especialmente en la comunicación entre sistemas a través de APIs. Aunque JSON puede no ser tan directamente compatible con hojas de cálculo como CSV y puede ocupar más espacio en archivos muy grandes, es muy útil para estructuras de datos más complejas y anidadas. También requiere cuidado para validar la integridad y formato de los datos cuando se trabaja con archivos externos.

Ejecucion

```
~/.../Practico_II/CSV$ python Ej2.py
Inventario guardado exitosamente en 'inventario.json'.

Inventario cargado desde JSON:
{'producto': 'Manzanas', 'precio': 0.5, 'stock': 100}
{'producto': 'Bananas', 'precio': 0.3, 'stock': 80}
{'producto': 'Naranjas', 'precio': 0.6, 'stock': 60}

Tipo de dato de 'stock': <class 'int'>

--- Fin del programa ---- Jose Alejandro Zabala Romero
~/.../Practico_II/CSV$
```

```
prog2 > alejandro > Practico_II > CSV > inventario.json
1 [
2   {
3     "producto": "Manzanas",
4     "precio": 0.5,
5     "stock": 100
6   },
7   {
8     "producto": "Bananas",
9     "precio": 0.3,
10    "stock": 80
11  },
12  {
13    "producto": "Naranjas",
14    "precio": 0.6,
15    "stock": 60
16  }
17 ]
```

➤ Hola mundo

```
1 class EscritorDeArchivos:
2     def __init__(self, archivo, agregarAlFinal = False):
3         self.escritor = open(archivo, 'w') # Siempre abre en modo escritura, sin importar agregarAlFinal
4     def cerrar(self):
5         self.escritor.close()
6     def escribir(self, texto):
7         datos_escritos = False
8         if not self.escritor.closed:
9             self.escritor.write(texto)
10            datos_escritos = True
11        return datos_escritos
12    def main():
13        escritor = EscritorDeArchivos("Prueba.txt")
14        escritor.escribir("Hola mundo")
15        escritor.cerrar()
16    if __name__ == "__main__":
17        main()
```

El código define una clase EscritorDeArchivos que permite escribir en un archivo de texto.

Abre el archivo en modo escritura ('w'), por lo que borra el contenido anterior si ya existe. Aunque incluye un parámetro agregarAlFinal, este no se usa. La clase tiene métodos para escribir texto

(escribir) y cerrar el archivo (cerrar). En el bloque principal (main), se crea una instancia de la clase, se escribe “Hola mundo” en el archivo “Prueba.txt” y se cierra.

Ejecución

```
lineal.py Ej1.py Shell inventario.csv Ej2.py h_m.py Prueba.txt +
prog2 > alejandro > Practico_II > CSV > Prueba.txt
1 Hola mundo
```

4.3 Implementa Programas Utilizando Archivos

4.3.1 Preparando Nuestro Código Base

```
1 # =====
2 # EJERCICIO 1: Estructura base del sistema (menú + estructura de tareas)
3 # =====
4 def mostrar_menu():
5     print("\n MENÚ DE TAREAS:")
6     print("1. Ver tareas")
7     print("2. Agregar tarea")
8     print("3. Marcar tarea como completada")
9     print("4. Eliminar tarea")
10    print("0. Salir")
```

En este primer ejercicio se implementa la estructura del sistema de gestión de tareas. Se define un menú interactivo que permite al usuario seleccionar distintas operaciones como ver tareas, agregarlas, marcarlas como completadas o eliminarlas. Esta parte es importante para la navegación del programa y para la interacción con el usuario.

4.3.2 Implementando la Función de Carga

```
# =====
# | EJERCICIO 2: Uso de listas de diccionarios para las tareas
# =====
tareas = []
proximo_id = 1
# =====
```

En la segunda sección se inicializan dos variables principales: una lista para almacenar las tareas y un contador para asignar identificadores únicos a cada tarea. La lista contiene elementos que son diccionarios, donde cada diccionario representa una tarea con sus atributos como id, texto y estado. Se emplean estructuras de datos dinámicas que permiten agregar, buscar y modificar elementos.

4.3.3 Implementando la Función de Guardado

```

16 # -----
17 # EJERCICIO 3: Persistencia con archivo JSON
18 # -----
19
20 import json
21 import os
22
23 def cargar_tareas():
24     global tareas, proximo_id
25     if os.path.exists("mis_tareas.json"):
26         with open("mis_tareas.json", "r") as archivo:
27             datos = json.load(archivo)
28             tareas = datos.get("tareas", [])
29             proximo_id = datos.get("proximo_id", 1)
30
31 def guardar_tareas():
32     with open("mis_tareas.json", "w") as archivo:
33         json.dump({
34             "tareas": tareas,
35             "proximo_id": proximo_id
36         }, archivo, indent=4)
37
38

```

En la tercera sección se implementan funciones para la persistencia de datos mediante archivos JSON. Los datos que están en la lista de diccionarios se guardan en un archivo para mantener el estado entre ejecuciones. Al cargar, se reconstruyen las estructuras de datos originales desde el archivo. Esto implica el uso de serialización y deserialización para convertir entre estructuras en memoria y formato de archivo.

4.3.4 Funciones adicionales

```

35 def ver_tareas():
36     if not tareas:
37         print("No hay tareas registradas.")
38         return
39     for tarea in tareas:
40         estado = "✓" if tarea["completada"] else "✗"
41         print(f"{tarea['id']}. {estado} {tarea['texto']}")
42 def agregar_tarea():
43     global proximo_id
44     texto = input("Ingresa la nueva tarea: ")
45     nueva_tarea = {
46         "id": proximo_id,
47         "texto": texto,
48         "completada": False
49     }
50     tareas.append(nueva_tarea)
51     proximo_id += 1
52     print("Tarea agregada.")
53 def marcar_completada():
54     try:
55         id_busqueda = int(input("Ingresa el ID de la tarea completada: "))
56         for tarea in tareas:
57             if tarea["id"] == id_busqueda:
58                 tarea["completada"] = True
59                 print("Tarea marcada como completada.")
60                 return
61         print("ID no encontrado.")
62     except ValueError:
63         print("ID inválido.")
64 def eliminar_tarea():
65     try:
66         id_busqueda = int(input("Ingresa el ID de la tarea a eliminar: "))
67         for tarea in tareas:
68             if tarea["id"] == id_busqueda:
69                 tareas.remove(tarea)
70                 print("Tarea eliminada.")
71                 return
72         print("ID no encontrado.")
73     except ValueError:
74         print("ID inválido.")

```

Las funciones adicionales manipulan la lista de diccionarios para realizar operaciones específicas. Estas funciones recorren la lista, agregan nuevos diccionarios, modifican el estado de elementos existentes o eliminan elementos según un criterio de búsqueda. Se utiliza iteración y acceso a elementos por sus claves para realizar estas tareas.

4.3.5 Programa principal

```

83     cargar_tareas()
84     while True:
85         mostrar_menu()
86         opcion = input("Elige una opción: ")
87
88         if opcion == "1":
89             ver_tareas()
90         elif opcion == "2":
91             agregar_tarea()
92         elif opcion == "3":
93             marcar_completada()
94         elif opcion == "4":
95             eliminar_tarea()
96         elif opcion == "0":
97             guardar_tareas()
98             print("Tareas guardadas. ¡Hasta luego!")
99             break
100        else:
101            print("Opción inválida.")

```

El programa principal contiene un ciclo que permite la interacción continua con el usuario.

En este ciclo se carga la información guardada, se muestra el menú y se ejecutan las funciones correspondientes según la opción seleccionada. Al finalizar, se guarda el estado actualizado en el archivo JSON. La estructura central de datos es la lista de diccionarios que se mantiene y manipula durante toda la ejecución.

Ejecución

```

~/.Práctico_II/p_c_a.py$ python todo_list.py
■ MENÚ DE TAREAS:
1. Ver tareas
2. Agregar tarea
3. Marcar tarea como completada
4. Eliminar tarea
0. Salir
Elige una opción: 1
No hay tareas registradas.

■ MENÚ DE TAREAS:
1. Ver tareas
2. Agregar tarea
3. Marcar tarea como completada
4. Eliminar tarea
0. Salir
Elige una opción: 2
Ingresa la nueva tarea: estudiar para el examen de hoy
Tarea agregada.

■ MENÚ DE TAREAS:
1. Ver tareas
2. Agregar tarea
3. Marcar tarea como completada
4. Eliminar tarea
0. Salir
Elige una opción: 2
Ingresa la nueva tarea: tener listo el proyecto final para el lunes
Tarea agregada.

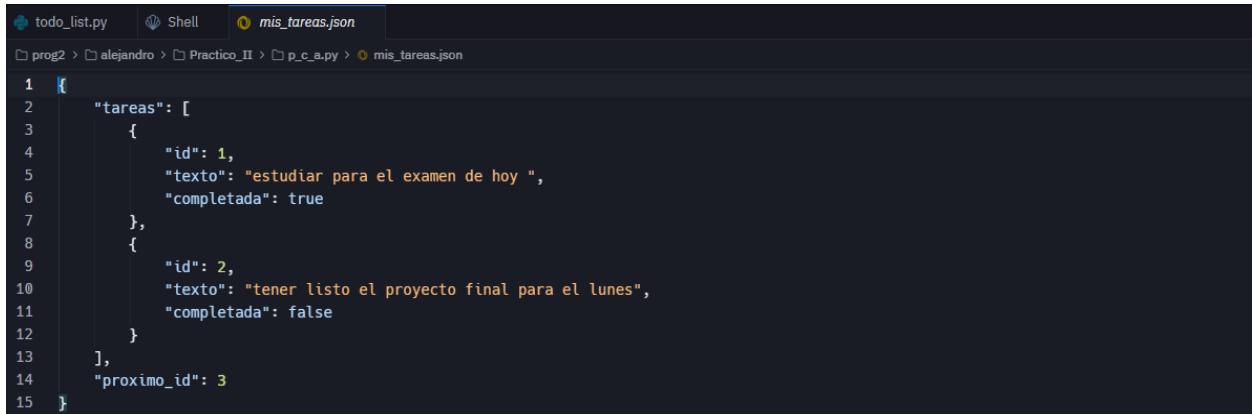
■ MENÚ DE TAREAS:
1. Ver tareas
2. Agregar tarea
3. Marcar tarea como completada
4. Eliminar tarea
0. Salir
Elige una opción: 3
Ingresa el ID de la tarea completada: 1
Tarea marcada como completada.

■ MENÚ DE TAREAS:
1. Ver tareas
2. Agregar tarea
3. Marcar tarea como completada
4. Eliminar tarea
0. Salir

```

```
Elige una opción: 1
1. ✓ estudiar para el examen de hoy
2. ✗ tener listo el proyecto final para el lunes

▣ MENÚ DE TAREAS:
1. Ver tareas
2. Agregar tarea
3. Marcar tarea como completada
4. Eliminar tarea
0. Salir
Elige una opción: 0
Tareas guardadas. ¡Hasta luego!
~/.../Práctico_II/p_c_a.py$
```



The screenshot shows a terminal window with the following details:

- Tab bar: todo_list.py, Shell, mis_tareas.json
- Path: prog2 > alejandro > Práctico_II > p_c_a.py > mis_tareas.json
- Content of mis_tareas.json (copied from the terminal):

```
1  {
2      "tareas": [
3          {
4              "id": 1,
5              "texto": "estudiar para el examen de hoy",
6              "completada": true
7          },
8          {
9              "id": 2,
10             "texto": "tener listo el proyecto final para el lunes",
11             "completada": false
12         }
13     ],
14     "proximo_id": 3
15 }
```