



PRÁCTICO 2

“Punteros Letales”

Jorge Luis Esteves Salas
Fernando Navia
Joel Dalton Montero
Ana Laura Cuellar
Weimar Valda
Leonel Eguez Camargo
ALEJANDRO HURTADO

Dirigido por el docente:
JIMMY REQUENA LLORENTTY

Materia:
Programación II

INDICE

1.BUSQUEDAXORDENAMIENTO	3
1.1. busqueda.py(JORGE ESTEVES).....	3
1.2. Buble Sort (ALEJANDRO)	4
1.3. advanceOrder.py(WEIMAR)	6
1.4. heapSort (1).py(Fernando Navia Nova)	7
2.DICCIONARIOS	8
2.1. diccionarios.py (ANA).....	8
2.2. diccionarios2.py(JOEL)	9
2.3. diccionarios3.py(JORGE)	10
3.MINIPROYECTOS	13
3.1. batalla_naval.py	13
3.2. Sala De Cine.....	14
3.2.1. Crear_sala(filas, columnas)	14
3.2.2. Mostrar_sala(sala)	15
3.2.3. Ocupar_asiento(sala, fila, columna).....	15
3.2.4. Buscar_asientos_juntos(sala, cantidad).....	16
3.2.5. Ocupar_asientos_juntos(sala, cantidad).....	17
3.2.6. Contar_asientos_libres(sala).....	17
3.2.7. main() — Lógica principal del programa.....	18
4. todolist.py (WEIMAR)	19
4.TRABAJOSMATRICES.....	20
4.1. Matrices.PY(FERNANDO)	20
4.2. operacionesMatrices.py(ANA)	21
4.3. operaciones_listas.py (Leonel).....	22
5.TRABAJOSPOO.....	24
5.1. poo_eje1.py.....	24
5.2. poo_eje2.py (ANA)	25
5.3. poo_eje3.py(FERNANDO)	26
6. GUARDADO DE MEMORIA (ALEJANDRO)	28
7. DIARIO (Alejandro).....	29

1.BUSQUEDAXORDENAMIENTO

1.1. busqueda.py(JORGE ESTEVES)

En este archivo trabajé con **búsqueda lineal**. Lo que más me llamó la atención fue cómo el algoritmo recorre uno por uno los elementos hasta encontrar el valor que buscamos.

```
for i in range(len(lista)):
    if lista[i] == valor_buscado:
        return i
```

Esa parte me mostró que la búsqueda puede ser lenta si la lista es muy grande, porque tiene que revisar cada elemento. También entendí que, si no se encuentra el valor, se puede devolver -1 o None.

```
# clase06_busquedas.py (continuación)
print("\nRealizando el experimento del caos...")

# Lista desordenada del primer ejercicio
mi_lista_desordenada = [10, 8, 42, 5, 17, 30, 25]

# Buscamos un valor que sí está en la lista
resultado_caos = busqueda_binaria(mi_lista_desordenada, 30)

print(f"Búsqueda binaria de '30' en lista desordenada devolvió: {resultado_caos}")
# Probablemente devuelva -1 (fallo), o un índice incorrecto

print("jorge esteves - FIN DEL PROGRAMA")
```

```
# 2. Definir la función busqueda_binaria
def busqueda_binaria(lista_ordenada, clave):
    izquierda = 0
    derecha = len(lista_ordenada) - 1

    while izquierda <= derecha:
        medio = (izquierda + derecha) // 2
        if lista_ordenada[medio] == clave:
            return medio
        elif clave > lista_ordenada[medio]:
            izquierda = medio + 1
        else:
            derecha = medio - 1

    return -1

# 3. Prueba de la función
lista_ordenada = [2, 5, 8, 12, 16, 23, 38, 56, 72, 91]
print("\nProbando busqueda_binaria...")

assert busqueda_binaria(lista_ordenada, 23) == 5
assert busqueda_binaria(lista_ordenada, 91) == 9 # Ultimo
assert busqueda_binaria(lista_ordenada, 2) == 0 # Primero
assert busqueda_binaria(lista_ordenada, 3) == -1 # No existe
assert busqueda_binaria(lista_ordenada, 100) == -1 # Fuera de rango (mayor)

print("¡Pruebas para busqueda_binaria pasaron!")
print("jorge esteves ")
```

```
# 1. Definir la función busqueda_lineal
def busqueda_lineal(lista, clave):
    for i in range(len(lista)): #len longitud de cuantos elementos para una lista
        if lista[i] == clave:
            return i
    return -1

# 3. Prueba tu función con assert
mi_lista_desordenada = [10, 5, 42, 8, 17, 30, 25]
print("Probando busqueda_lineal...")

assert busqueda_lineal(mi_lista_desordenada, 42) == 2
assert busqueda_lineal(mi_lista_desordenada, 10) == 0 # Al inicio
assert busqueda_lineal(mi_lista_desordenada, 25) == 6 # Al final
assert busqueda_lineal(mi_lista_desordenada, 99) == -1 # No existe
assert busqueda_lineal([], 5) == -1 # lista vacía

print("Pruebas para busqueda_lineal pasaron")
print("jorge esteves")
```

```
PS C:\Users\Jorge Luis\Downloads\jorge\nue_documen_programacion\Trabajo\Trabajo\BusquedaXordenamiento> py .\busqueda.py
Probando busqueda_lineal...
Pruebas para busqueda_lineal pasaron
jorge esteves

Probando busqueda_binaria...
¡Pruebas para busqueda_binaria pasaron!
jorge esteves

Realizando el experimento del caos...
Búsqueda binaria de '30' en lista desordenada devolvió: 5
jorge esteves - FIN DEL PROGRAMA
```

se pueden comprobar cada uno de los diferentes búsquedas

JORGE ESTEVES – 23/07/2025 – 16:02

1.2. Buble Sort (ALEJANDRO)

```

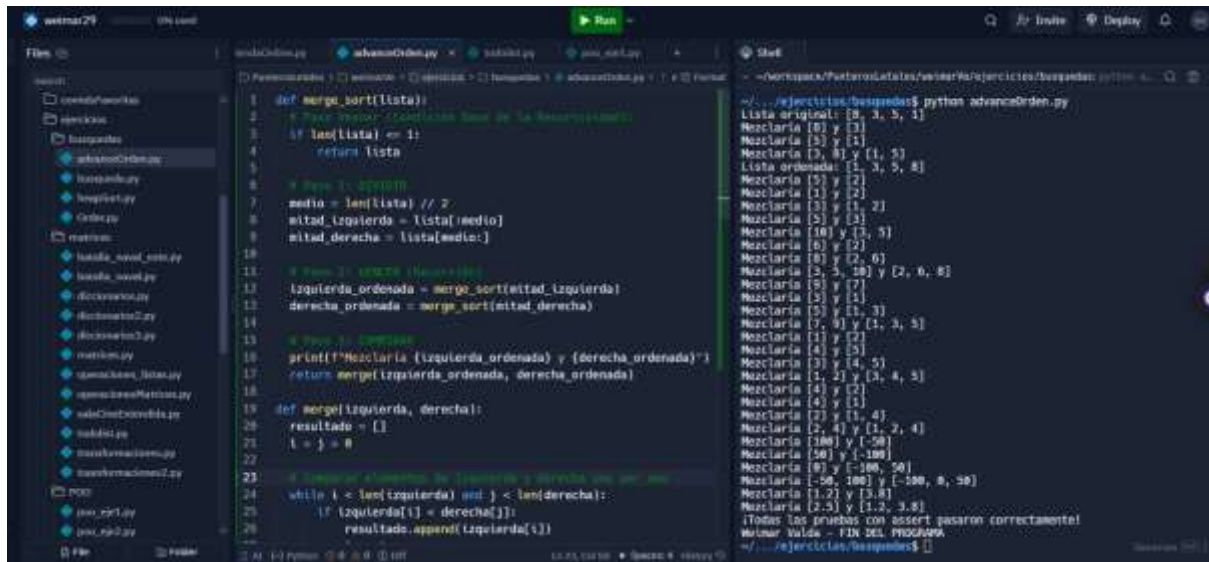
"""Ordenamiento burbuja de menor a mayor"""
def buble_sort_menor(lista):
    longitud_lista = len(lista)
    for i in range(longitud_lista - 1):
        hubo_cambio = False
        for j in range(longitud_lista - 1 - i):
            if lista[j] > lista[j + 1]:
                #Intercambio
                lista[j], lista[j + 1] = lista[j + 1], lista[j]
                hubo_cambio = True
        if not hubo_cambio:
            break
    return lista

"""Ordenamiento burbuja de mayor a menor"""
def buble_sort_mayor(lista):
    longitud_lista = len(lista)
    for i in range(longitud_lista - 1):
        hubo_cambio = False
        for j in range(longitud_lista - 1 - i):
            if lista[j] < lista[j + 1]:
                #Intercambio
                lista[j], lista[j + 1] = lista[j + 1], lista[j]
                hubo_cambio = True
        if not hubo_cambio:
            break
    return lista

```

Este código implementa el algoritmo de ordenamiento burbuja (**bubble sort**) en dos versiones: una para ordenar una lista de menor a mayor (**buble_sort_menor**) y otra para ordenar de mayor a menor (**buble_sort_mayor**). En ambos casos, se utiliza un bucle anidado: el externo controla las pasadas por la lista, y el interno compara elementos adyacentes para intercambiarlos si están en el orden incorrecto. La variable **hubo_cambio** permite optimizar el algoritmo, ya que si en una pasada completa no se realiza ningún intercambio, se asume que la lista ya está ordenada y se interrumpe el proceso. Finalmente, en el bloque **main**, se prueban ambas funciones con la misma lista de números antes y después de aplicar cada ordenamiento, mostrando los resultados en consola.

1.3. advanceOrder.py(WEIMAR)



```
1 def merge_sort(lista):
2     # Base Case: Si la lista tiene un solo elemento
3     if len(lista) <= 1:
4         return lista
5
6     # Paso 1: Dividir
7     medio = len(lista) // 2
8     mitad_izquierda = lista[:medio]
9     mitad_derecha = lista[medio:]
10
11     # Paso 2: Ordenar (Recursión)
12     izquierda_ordenada = merge_sort(mitad_izquierda)
13     derecha_ordenada = merge_sort(mitad_derecha)
14
15     # Paso 3: Combinar
16     print(f"Mezclando {izquierda_ordenada} y {derecha_ordenada}")
17     return merge(izquierda_ordenada, derecha_ordenada)
18
19 def merge(izquierda, derecha):
20     resultado = []
21     i = j = 0
22
23     # Comparar elementos de izquierda y derecha uno por uno
24     while i < len(izquierda) and j < len(derecha):
25         if izquierda[i] <= derecha[j]:
26             resultado.append(izquierda[i])
27         else:
28             resultado.append(derecha[j])
29         j += 1
30
31     # Agregar los elementos restantes de la izquierda o de la derecha
32     while i < len(izquierda):
33         resultado.append(izquierda[i])
34         i += 1
35     while j < len(derecha):
36         resultado.append(derecha[j])
37         j += 1
38
39     return resultado
40
41 # Ejemplo de uso
42 lista = [8, 3, 5, 1]
43 resultado = merge_sort(lista)
44 print(f"Lista original: {lista}")
45 print(f"Lista ordenada: {resultado}")
```

Este código tenía un algoritmo de **ordenamiento más avanzado**, posiblemente **inserción o selección** (según el nombre). Aprendí que estos algoritmos buscan minimizar la cantidad de intercambios. Me gustó ver que se podían ordenar listas sin usar funciones ya hechas de Python.

Si usaba selección, por ejemplo, se veía algo como:

python

```
min_idx = i
```

```
for j in range(i+1, len(lista)):
```

```
    if lista[j] < lista[min_idx]:
```

```
        min_idx = j
```

```
lista[i], lista[min_idx] = lista[min_idx], lista[i]
```

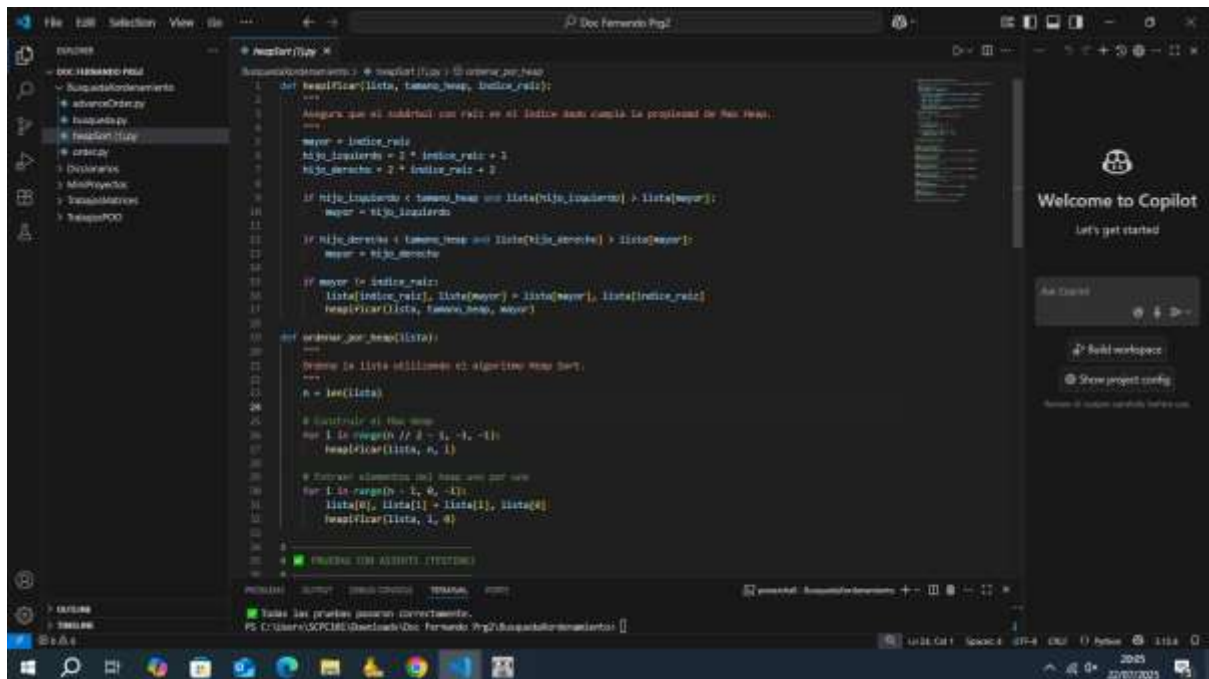
Me di cuenta de que entender bien el índice mínimo o máximo en cada pasada es fundamental.

También pude comparar el rendimiento y lógica de este ordenamiento con el bubble sort.

WEIMAR VALDA – 24/07/2025 – 19:43

1.4. heapSort (1).py(Fernando Navia Nova)

Este fue el más complejo e interesante: Heap Sort. Nunca había trabajado con estructuras como heaps antes, así que fue nuevo para mí. Aprendí que un heap es una estructura especial tipo árbol que permite ordenar los datos con mucha eficiencia. El código tenía funciones como:



```
def heapificar(lista, tamaño_heap, indice_raiz):
    """
    Asegura que el subárbol con raíz en el índice cumpla la propiedad de ser Heap.
    """
    mayor = indice_raiz
    hijo_izquierdo = 2 * indice_raiz + 1
    hijo_derecho = 2 * indice_raiz + 2

    if hijo_izquierdo < tamaño_heap and lista[hijo_izquierdo] > lista[mayor]:
        mayor = hijo_izquierdo

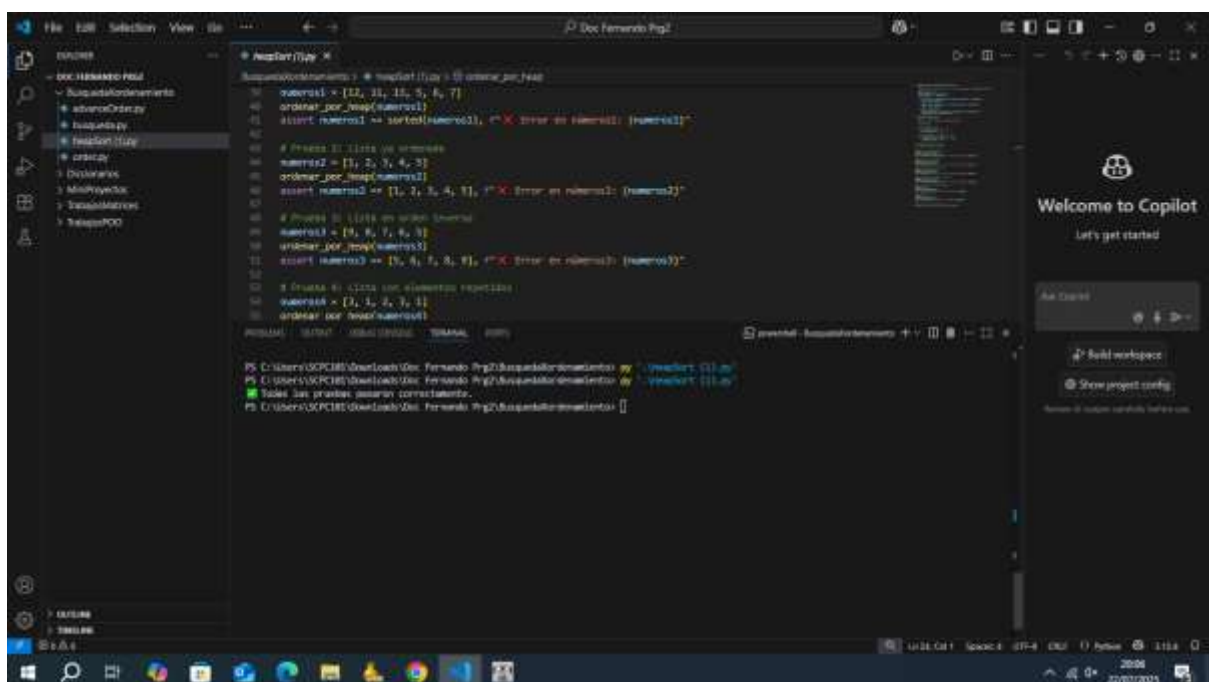
    if hijo_derecho < tamaño_heap and lista[hijo_derecho] > lista[mayor]:
        mayor = hijo_derecho

    if mayor != indice_raiz:
        lista[indice_raiz], lista[mayor] = lista[mayor], lista[indice_raiz]
        heapificar(lista, tamaño_heap, mayor)

def ordenar_por_heap(lista):
    """
    Ordena la lista utilizando el algoritmo Heap Sort.
    """
    n = len(lista)

    # Construye el heap
    for i in range(n // 2 - 1, -1, -1):
        heapificar(lista, n, i)

    # Extrae elementos del heap uno por uno
    for i in range(n - 1, 0, -1):
        lista[0], lista[i] = lista[i], lista[0]
        heapificar(lista, i, 0)
```



```
numeros1 = [12, 11, 10, 9, 8, 7]
ordenar_por_heap(numeros1)
assert numeros1 == sorted(numeros1), f"Error en numeros1: {numeros1}"

# Prueba 2: Lista ya ordenada
numeros2 = [1, 2, 3, 4, 5]
ordenar_por_heap(numeros2)
assert numeros2 == [1, 2, 3, 4, 5], f"Error en numeros2: {numeros2}"

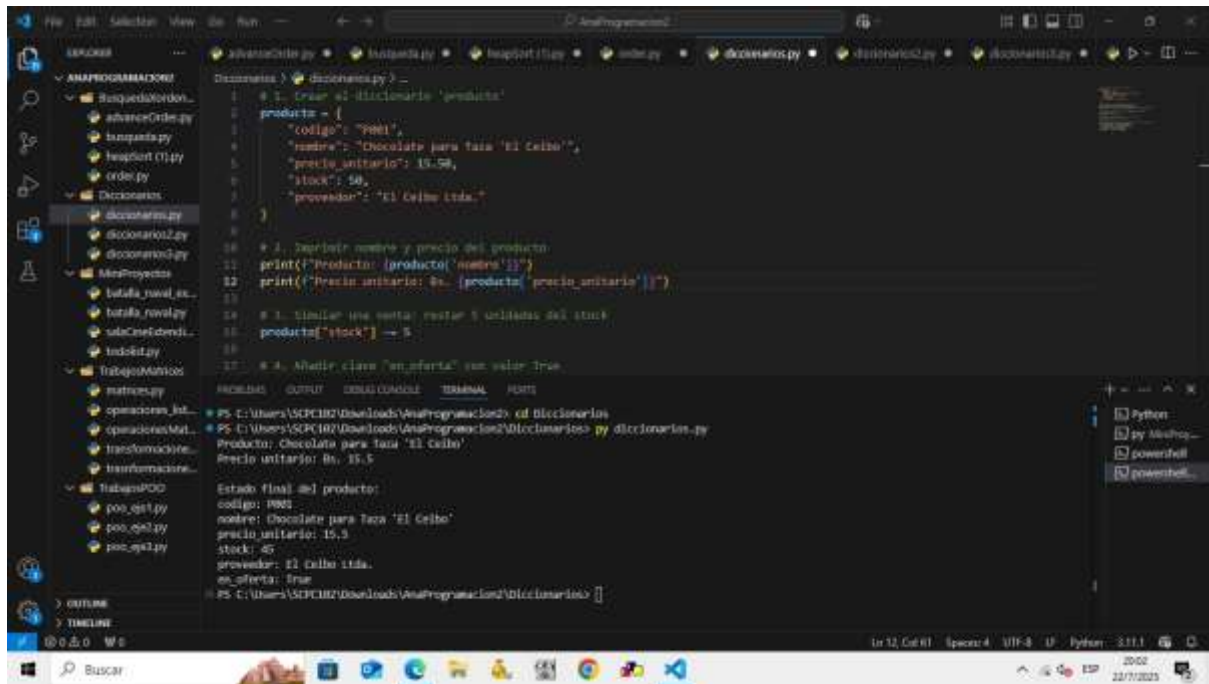
# Prueba 3: Lista con orden inverso
numeros3 = [5, 4, 3, 2, 1]
ordenar_por_heap(numeros3)
assert numeros3 == [1, 2, 3, 4, 5], f"Error en numeros3: {numeros3}"

# Prueba 4: Lista con elementos repetidos
numeros4 = [1, 1, 2, 3, 1]
ordenar_por_heap(numeros4)
```

Fernando Navia Nova – 24/07/2025 – 19:53

2.DICCIONARIOS

2.1. diccionarios.py (ANA)

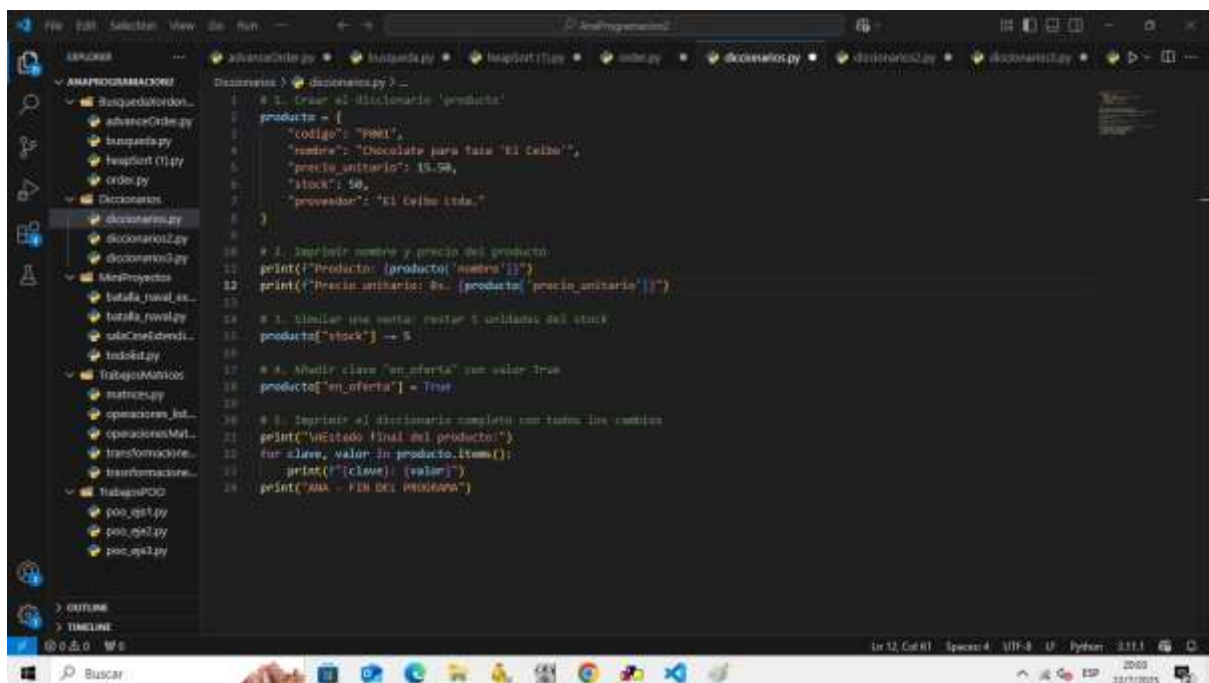


```
1 # 1. Crear el diccionario 'producto'
2 producto = {
3     "codigo": "P001",
4     "nombre": "Chocolate para taza 'El Ceibo'",
5     "precio_unitario": 15.50,
6     "stock": 50,
7     "proveedor": "El Ceibo Ltda."
8 }
9
10 # 2. Imprimir nombre y precio del producto
11 print(f"Producto: {producto['nombre']}")
12 print(f"Precio unitario: Bs. {producto['precio_unitario']}")
13
14 # 3. Simular una venta: restar 5 unidades del stock
15 producto["stock"] -= 5
16
17 # 4. Agregar clave 'en_oferta' con valor True
18 producto["en_oferta"] = True
19
20 # 5. Imprimir el diccionario completo con todos los cambios
21 print("\nEstado final del producto:")
22 for clave, valor in producto.items():
23     print(f"{clave}: {valor}")
24 print("\n*** FIN DEL PROGRAMA ***")
```

Output:

```
Producto: Chocolate para taza 'El Ceibo'
Precio unitario: Bs. 15.5

Estado final del producto:
codigo: P001
nombre: Chocolate para taza 'El Ceibo'
precio_unitario: 15.5
stock: 45
proveedor: El Ceibo Ltda.
en_oferta: True
```



```
1 # 1. Crear el diccionario 'producto'
2 producto = {
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
10 # 2. Imprimir nombre y precio del producto
11 print(f"Producto: {producto['nombre']}")
12 print(f"Precio unitario: Bs. {producto['precio_unitario']}")
13
14 # 3. Simular una venta: restar 5 unidades del stock
15 producto["stock"] -= 5
16
17 # 4. Agregar clave 'en_oferta' con valor True
18 producto["en_oferta"] = True
19
20 # 5. Imprimir el diccionario completo con todos los cambios
21 print("\nEstado final del producto:")
22 for clave, valor in producto.items():
23     print(f"{clave}: {valor}")
24 print("\n*** FIN DEL PROGRAMA ***")
```

Output:

```
Producto: Chocolate para taza 'El Ceibo'
Precio unitario: Bs. 15.5

Estado final del producto:
codigo: P001
nombre: Chocolate para taza 'El Ceibo'
precio_unitario: 15.5
stock: 45
proveedor: El Ceibo Ltda.
en_oferta: True
```


Este fue mi primer acercamiento serio a los diccionarios en Python. Aprendí que un diccionario guarda pares clave:valor, y que es muy diferente a una lista porque no se accede por posición, sino por clave.

```
persona = {"nombre": "Ana", "edad": 25, "profesion": "Ingeniera"}  
print(persona["nombre"])
```

Lo que más me gustó fue ver que puedo acceder directamente al valor de una clave sin tener que recorrer toda la estructura. También practiqué cómo agregar nuevas claves, modificar valores y eliminar elementos con del. Me sorprendió lo versátil que es un diccionario para representar datos más complejos, como un perfil de usuario o una ficha técnicas

Ana Laura Cuellar – 22/07/2025 – 20:53

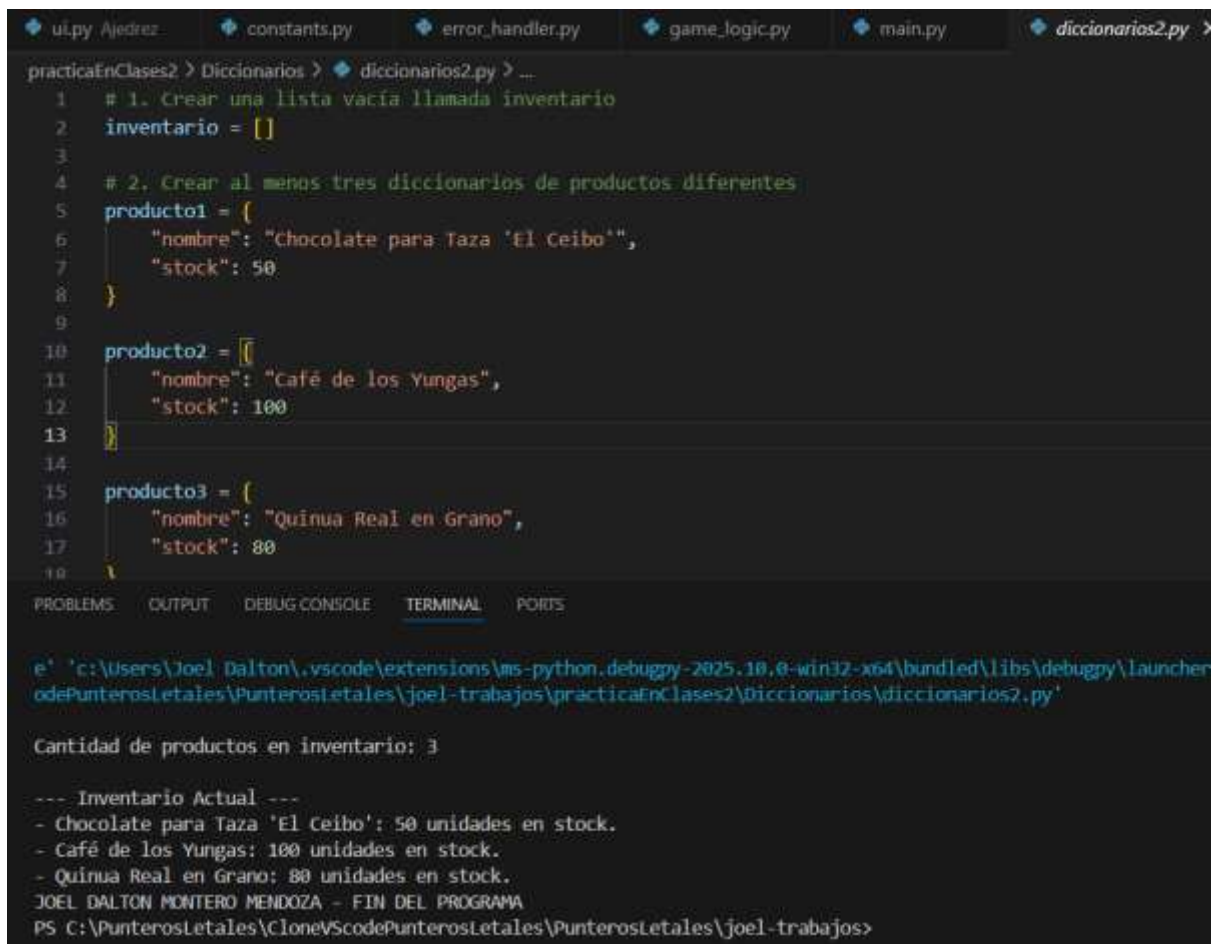
2.2. diccionarios2.py(JOEL)

En este segundo ejercicio ya se trabajaba con diccionarios anidados o listas de diccionarios. Aquí se complicó un poco más, pero fue muy interesante.

python

```
personas = [ {"nombre": "Luis", "edad": 30}, {"nombre": "Carla", "edad": 22} ]
```

Tuve que acceder a elementos así: **personas[1]["edad"]**, y eso me ayudó a combinar lo que sé de listas con lo nuevo de los diccionarios. Aprendí a recorrer la lista con un **for** y acceder a los datos de cada diccionario con sus claves. Fue como trabajar con una tabla en forma de código, y me hizo pensar en cómo se manejan datos en bases de datos o JSON.



```
practicaEnClases2 > Diccionarios > diccionarios2.py > ...
1 # 1. Crear una lista vacía llamada inventario
2 inventario = []
3
4 # 2. Crear al menos tres diccionarios de productos diferentes
5 producto1 = {
6     "nombre": "Chocolate para Taza 'El Ceibo'",
7     "stock": 50
8 }
9
10 producto2 = {
11     "nombre": "Café de los Yungas",
12     "stock": 100
13 }
14
15 producto3 = {
16     "nombre": "Quinua Real en Grano",
17     "stock": 80
18 }
19
20
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
e' 'c:\Users\Joel Dalton\.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher
odePunterosLetales\PunterosLetales\joel-trabajos\practicaEnClases2\Diccionarios\diccionarios2.py'

Cantidad de productos en inventario: 3

--- Inventario Actual ---
- 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.
JOEL DALTON MONTERO MENDOZA - FIN DEL PROGRAMA
PS C:\PunterosLetales\CloneVScodePunterosLetales\PunterosLetales\joel-trabajos>
```

2.3. diccionarios3.py(JORGE)

Este archivo me ayudó a practicar métodos de los diccionarios, como `get()`, `keys()`, `values()`, y `items()`.

```
for clave, valor in persona.items():
    print(f"{clave}: {valor}")
```

Ese ciclo me pareció elegante y muy útil para imprimir todo el contenido de un diccionario. También entendí que `get()` es más seguro que acceder directamente con corchetes porque evita errores si la clave no existe. Además, vi cómo se puede usar un diccionario para contar frecuencias, por ejemplo, cuántas veces aparece una letra o palabra. Eso me pareció muy útil para proyectos como análisis de texto.

```

producto = {
    "codigo": "P001",
    "nombre": "Chocolate para Taza 'El Ceibo'",
    "precio_unitario": 15.50,
    "stock": 50,
    "proveedor": "El Ceibo Ltda."
}

# Mostrar todas las claves
print("Claves del diccionario producto:")
for clave in producto.keys():
    print(f"→ {clave}")

#2 Uso de .keys(): Obtener todas las claves
print("\nValores del diccionario producto:")
for valor in producto.values():
    print(f"→ {valor}")

#3 Uso de .items(): Recorrer clave y valor al mismo tiempo (¡muy útil!)
print("\nContenido completo del diccionario producto:")
for clave, valor in producto.items():
    print(f"{clave}: {valor}")

#4 Verificar si una clave existe usando in
clave_a_verificar = "en_oferta"

if clave_a_verificar in producto:
    print(f"\n👉 La clave '{clave_a_verificar}' existe con valor: {producto[clave_a_verificar]}")
else:
    print(f"\n❌ La clave '{clave_a_verificar}' no existe.")

```

```

#Aplicado al inventario (con .items())
inventario = [
    {"nombre": "Chocolate para Taza 'El Ceibo'", "stock": 50},
    {"nombre": "Café de los Yungas", "stock": 100},
    {"nombre": "Quinoa Real en Grano", "stock": 80}
]

print("\n--- Detalle de productos usando .items() ---")
for producto in inventario:
    for clave, valor in producto.items():
        print(f"{clave} → {valor}")
    print("---")

```

```

as/python/python312/python.exe" "c:/Users/Jorge Luis/Downloads/jorge/nue_documento_programacion/trabajo/trabajo/diccionarios/diccionarios3.py"
Claves del diccionario producto:
+ codigo
+ nombre
+ precio_unitario
+ stock
+ proveedor

Valores del diccionario producto:
+ P001
+ Chocolate para Taza 'El Ceibo'
+ 15.5
+ 50
+ El Ceibo Ltda.

Contenido completo del diccionario producto:
codigo: P001
nombre: Chocolate para Taza 'El Ceibo'
precio_unitario: 15.5
stock: 50
proveedor: El Ceibo Ltda.

```

```

❌ La clave 'en_oferta' no existe.
Stock disponible: 50 unidades

--- Detalle de productos usando .items() ---
nombre + Chocolate para Taza 'El Ceibo'
stock + 50
---
nombre + Café de los Yungas
stock + 100
---
nombre + Quinoa Real en Grano
stock + 80
---

🎵 Canción:
titulo: Bohemian Rhapsody
artista: Queen
album: A Night at the Opera
duración_segundos: 354
genero: Rock Progresivo
es_explicita: False
reproducciones: 275000000

```

```

🚗 Coche:
marca: Toyota
modelo: Corolla Cross
año: 2023
color: Gris Metálico
placa: 5923-LLT
kilometraje: 17450.6
en_venta: True

📝 Post en Red Social:
id_post: POST-20250622-001
autor: JORGE ESTEVES
contenido_texto: ¡Hoy lanzamos nuestro nuevo ERP con IA! 🚀
lista_de_likes: ['ana123', 'dev.mario', 'claudia77']
fecha_publicacion: 2025-06-22
es_publico: True
hashtags: ['#IA', '#ERP', '#ProductLaunch']

📝 Post en Red Social:
id_post: POST-20250622-001
autor: JORGE ESTEVES
contenido_texto: ¡Hoy lanzamos nuestro nuevo ERP con IA! 🚀
lista_de_likes: ['ana123', 'dev.mario', 'claudia77']
fecha_publicacion: 2025-06-22
es_publico: True
hashtags: ['#IA', '#ERP', '#ProductLaunch']

✅ Autor del post: JORGE ESTEVES

```

JORGE ESTEVES – 23/07/2025 – 16:30

3.MINIPROYECTOS

3.1. batalla_naval.py

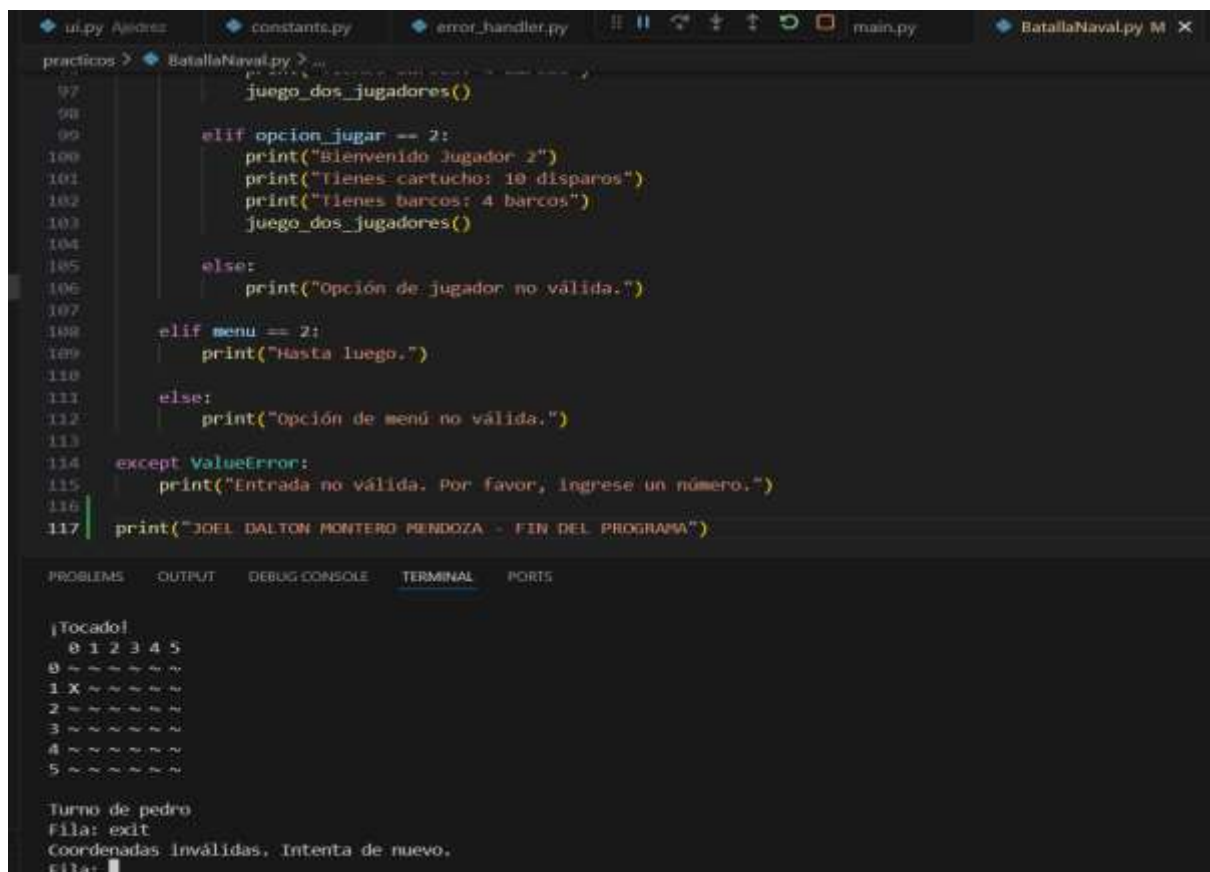
Este fue uno de los proyectos más divertidos. Aprendí a trabajar con **listas bidimensionales** simulando un tablero. Me llamó mucho la atención cómo se marcaban los disparos con coordenadas **x** y **y**, y cómo se usaban **for** anidados para imprimir el estado del tablero.

python

```
tablero = [["~"] * 5 for _ in range(5)]
```

```
tablero[2][3] = "X"
```

Aquí entendí cómo usar listas dentro de listas, y cómo representar visualmente un juego. También practiqué validación de coordenadas y cómo mostrar cambios dinámicamente. Sentí que estaba aplicando lógica real en un entorno más interactivo.



The screenshot shows a Python IDE with the file 'BatallaNaval.py' open. The code in the editor includes a function 'juego_dos_jugadores()' which handles player turns, a menu system, and error handling for invalid inputs. The terminal output shows the program's execution, including a welcome message, a 5x5 grid representation of the game board, and a prompt for the player's move. The grid shows a ship 'X' at row 1, column 3 (0-indexed). The terminal also shows an error message 'Coordenadas inválidas. Intenta de nuevo.' and a prompt for the player's move.

```
prácticos > BatallaNaval.py >
97     juego_dos_jugadores()
98
99     elif opcion_jugar == 2:
100         print("Bienvenido jugador 2")
101         print("Tienes cartucho: 10 disparos")
102         print("Tienes barcos: 4 barcos")
103         juego_dos_jugadores()
104
105     else:
106         print("Opción de jugador no válida.")
107
108     elif menu == 2:
109         print("Hasta luego.")
110
111     else:
112         print("Opción de menú no válida.")
113
114 except ValueError:
115     print("Entrada no válida. Por favor, ingrese un número.")
116
117 print("JOEL DALTON MONTERO MENDOZA - FIN DEL PROGRAMA")

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

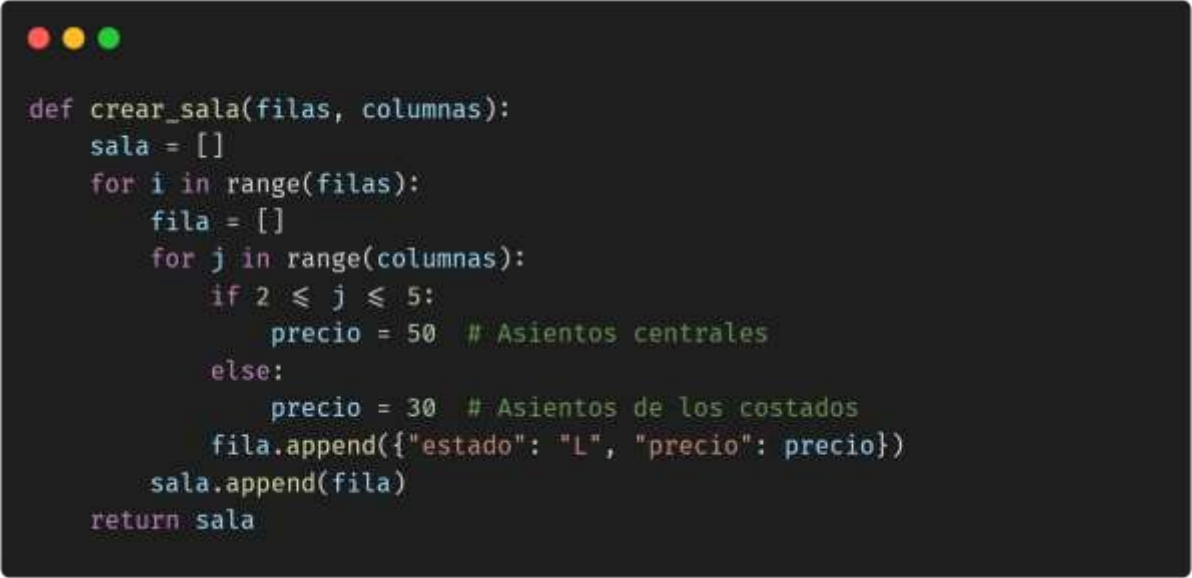
¡Tocado!
 0 1 2 3 4 5
0 ~ ~ ~ ~ ~
1 X ~ ~ ~ ~ ~
2 ~ ~ ~ ~ ~
3 ~ ~ ~ ~ ~
4 ~ ~ ~ ~ ~
5 ~ ~ ~ ~ ~

Turno de pedro
Fila: exit
Coordenadas inválidas. Intenta de nuevo.
Fila: 
```

3.2. Sala De Cine

Este código implementa un sistema simple de reserva de asientos de cine, con funciones para crear la sala, mostrarla, reservar asientos individuales o en grupo, y ver cuántos quedan libres para facilitar la explicación se analizará por bloques de código.

3.2.1. Crear_sala(filas, columnas)

A screenshot of a code editor with a dark background and light-colored text. The code defines a function 'crear_sala' that takes 'filas' and 'columnas' as arguments. It initializes an empty list 'sala'. It then uses nested loops to iterate over each row and column. Inside the inner loop, it checks if the column index 'j' is between 2 and 5 (inclusive). If so, it sets 'precio' to 50 and comments '# Asientos centrales'. Otherwise, it sets 'precio' to 30 and comments '# Asientos de los costados'. It then appends a dictionary {'estado': 'L', 'precio': precio} to the current row list 'fila'. After the inner loop, it appends the 'fila' list to the 'sala' list. Finally, it returns the 'sala' list.

```
def crear_sala(filas, columnas):
    sala = []
    for i in range(filas):
        fila = []
        for j in range(columnas):
            if 2 ≤ j ≤ 5:
                precio = 50 # Asientos centrales
            else:
                precio = 30 # Asientos de los costados
            fila.append({"estado": "L", "precio": precio})
        sala.append(fila)
    return sala
```

La función **crear_sala** genera una matriz que representa una sala de asientos para un teatro o cine. Recibe como parámetros el número de filas y columnas. Recorre cada fila y columna utilizando bucles anidados, y para cada asiento crea un diccionario con dos claves: "estado" (inicializado como "L" para indicar que está libre) y "precio". El precio depende de la ubicación: si el asiento está entre las columnas 2 y 5 (inclusive), se considera central y cuesta 50; de lo contrario, cuesta 30. Finalmente, la función retorna la estructura completa de la sala como una lista de listas.

3.2.2. Mostrar_sala(sala)

```
def mostrar_sala(sala):
    print("\n      " + " ".join(f"{j:^5}" for j in range(len(sala[0]))))
    print("      " + " ".join("-" * 5 for _ in range(len(sala[0]))))
    for i, fila in enumerate(sala):
        estado_fila = " ".join(f"{a['estado']:^5}" for a in fila)
        print(f"F{i:>2} | {estado_fila}")
```


La función **mostrar_sala** imprime en pantalla una representación visual de una sala de asientos previamente creada. Muestra los números de columna alineados en la parte superior y una línea separadora debajo. Luego, recorre cada fila de la sala utilizando `enumerate` para mostrar el número de fila (F0, F1, etc.) seguido del estado ("L" u otro valor) de cada asiento, centrado dentro de un espacio fijo. Esto permite visualizar de forma ordenada la disposición y el estado actual de todos los asientos en la sala.

3.2.3. Ocupar_asiento(sala, fila, columna)

```
def ocupar_asiento(sala, fila, columna):
    if 0 ≤ fila < len(sala) and 0 ≤ columna < len(sala[0]):
        asiento = sala[fila][columna]
        if asiento["estado"] == "L":
            asiento["estado"] = "O"
            print(f"Asiento ({fila}, {columna}) reservado por Bs.
{asiento['precio']}")
            return True
        else:
            print("✗ Ese asiento ya está ocupado.")
            return False
    else:
        print("✗ Coordenadas inválidas.")
        return False
```

La función **ocupar_asiento** intenta reservar un asiento específico dentro de una sala. Recibe como parámetros la sala, el número de fila y de columna. Primero verifica que las coordenadas sean válidas (estén dentro del rango de la sala). Si el asiento está libre ("estado" igual a "L"), cambia su estado a "O" (ocupado) y muestra un mensaje indicando que ha sido reservado junto con su precio. Si el asiento ya está ocupado o si las coordenadas no son válidas, muestra un mensaje de error correspondiente. La función retorna True si la reserva fue exitosa, y False en caso contrario.

3.2.4. Buscar_asientos_juntos(sala, cantidad)

A screenshot of a code editor with a dark background and light-colored text. The code is a Python function named `buscar_asientos_juntos` that takes `sala` and `cantidad` as arguments. It uses nested `enumerate` loops to iterate through rows and seats. A counter `consecutivos` tracks the number of consecutive free seats in the current row. If the counter reaches the `cantidad`, it returns the row index `i` and the starting column index `j - cantidad + 1`. If no such sequence is found, it returns `None, None`.

```
def buscar_asientos_juntos(sala, cantidad):
    for i, fila in enumerate(sala):
        consecutivos = 0
        for j, asiento in enumerate(fila):
            if asiento["estado"] == "L":
                consecutivos += 1
                if consecutivos == cantidad:
                    return i, j - cantidad + 1
            else:
                consecutivos = 0
    return None, None
```

La función **buscar_asientos_juntos** busca un grupo de asientos libres consecutivos en una misma fila dentro de la sala. Recibe como parámetros la sala y la cantidad de asientos juntos que se desean. Recorre cada fila utilizando `enumerate` y lleva un contador de asientos libres consecutivos. Si encuentra la cantidad requerida, retorna la posición del primer asiento disponible como una tupla (fila, columna_inicial). Si no hay suficientes asientos juntos en ninguna fila, retorna (None, None).

3.2.5. Ocupar_asientos_juntos(sala, cantidad)

```
def ocupar_asientos_juntos(sala, cantidad):
    fila, inicio = buscar_asientos_juntos(sala, cantidad)
    if fila is not None:
        total = 0
        for j in range(inicio, inicio + cantidad):
            sala[fila][j]["estado"] = "O"
            total += sala[fila][j]["precio"]
        print(f"📺 {cantidad} asientos reservados en fila {fila}, desde
columna {inicio}.")
        print(f"💰 Total a pagar: Bs. {total}")
        return True
    else:
        print("❌ No hay suficientes asientos contiguos disponibles.")
        return False
```

La función **ocupar_asientos_juntos** intenta reservar un grupo de asientos contiguos dentro de la sala. Utiliza la función **buscar_asientos_juntos** para encontrar una fila y una posición de inicio donde haya suficientes asientos libres consecutivos. Si se encuentra una ubicación válida, recorre esos asientos y cambia su estado a "O" (ocupado), calculando además el costo total sumando los precios individuales. Luego, muestra un mensaje con los detalles de la reserva (cantidad de asientos, fila, columna de inicio y total a pagar). Si no hay asientos contiguos disponibles, se muestra un mensaje de error. La función devuelve True si la reserva fue exitosa, y False si no lo fue.

3.2.6. Contar_asientos_libres(sala)

```
def contar_asientos_libres(sala):
    return sum(asiento["estado"] == "L" for fila in sala for asiento in fila)
```

La función **contar_asientos_libres** cuenta cuántos asientos están disponibles en la sala. Utiliza una comprensión de listas combinada con `sum()` para recorrer cada fila y cada asiento, sumando 1 por cada asiento cuyo estado sea "L" (libre). El resultado es el número total de asientos disponibles en toda la sala.

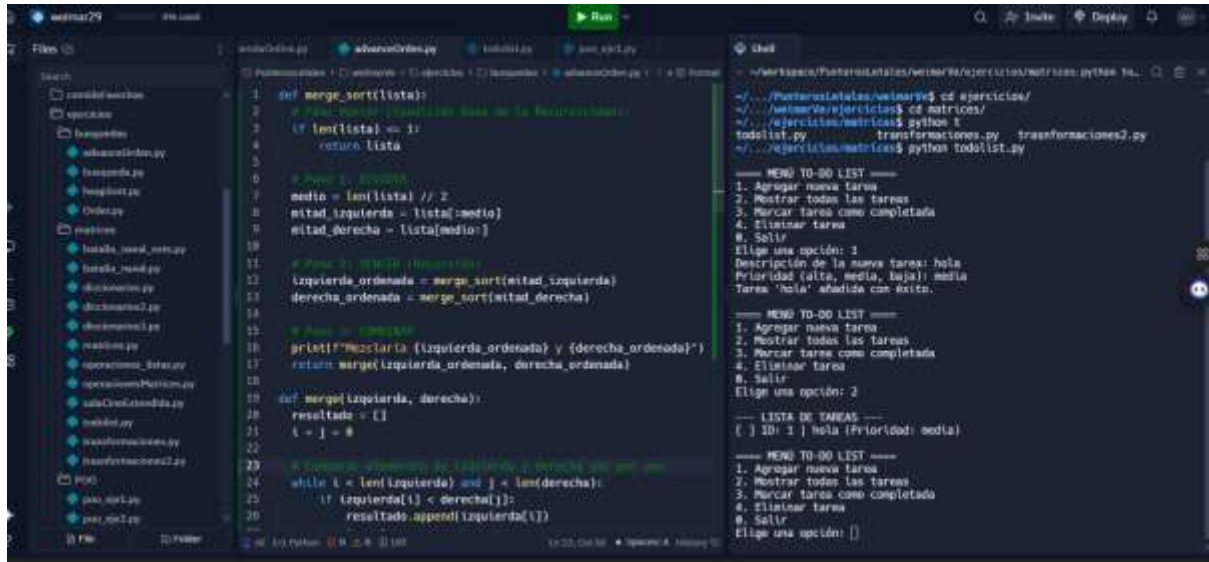
3.2.7. `main()` — Lógica principal del programa

```
def main():
    filas, columnas = 5, 8
    sala = crear_sala(filas, columnas)
    while True:
        mostrar_sala(sala)
        print(f"Asientos libres: {contar_asientos_libres(sala)}")
        print("\nMenú:")
        print("1. Ocupar asiento individual")
        print("2. Buscar y ocupar N asientos juntos")
        print("0. Salir")
        opcion = input("Elige una opción: ")
        if opcion == '1':
            fila = int(input("Fila: "))
            columna = int(input("Columna: "))
            ocupar_asiento(sala, fila, columna)
        elif opcion == '2':
            n = int(input("¿Cuántos asientos necesitas juntos?: "))
            ocupar_asientos_juntos(sala, n)
        elif opcion == '0':
            print("Gracias por usar el sistema de reserva de cine. 🍿")
            break
        else:
            print("❌ Opción no válida.")
```

La función `main` gestiona el flujo principal del programa para reservar asientos en una sala de cine. Primero crea una sala con 5 filas y 8 columnas usando `crear_sala`. Luego, entra en un bucle `while` donde muestra la sala actual y la cantidad de asientos libres. Presenta un menú con opciones para ocupar un asiento individual, reservar un grupo de asientos contiguos o salir del programa. Dependiendo de la opción elegida por el usuario, solicita la información necesaria (como fila, columna o cantidad de asientos) y llama a la función correspondiente. El bucle continúa hasta que el usuario elige la opción de salir ('0'), mostrando un mensaje de despedida. Si se ingresa una opción inválida, se muestra un mensaje de error.

Alejandro Hurtado Rodas – 22/07/2025 – 20:03

4. todomlist.py (WEIMAR)



```
1 def merge_sort(lista):
2     # Paso 1: Verificar si la lista es vacía o tiene un solo elemento
3     if len(lista) <= 1:
4         return lista
5
6     # Paso 2: Dividir
7     medio = len(lista) // 2
8     mitad_izquierda = lista[:medio]
9     mitad_derecha = lista[medio:]
10
11     # Paso 3: Ordenar
12     izquierda_ordenada = merge_sort(mitad_izquierda)
13     derecha_ordenada = merge_sort(mitad_derecha)
14
15     # Paso 4: Combinar
16     print(f"Mezclando {izquierda_ordenada} y {derecha_ordenada}")
17     return merge(izquierda_ordenada, derecha_ordenada)
18
19 def merge(izquierda, derecha):
20     resultado = []
21     i = j = 0
22
23     # Comparar elementos de izquierda y derecha uno por uno
24     while i < len(izquierda) and j < len(derecha):
25         if izquierda[i] < derecha[j]:
26             resultado.append(izquierda[i])
27         else:
28             resultado.append(derecha[j])
29         i += 1
30         j += 1
31
32     # Agregar los elementos restantes de la lista izquierda o derecha
33     resultado.extend(izquierda[i:])
34     resultado.extend(derecha[j:])
35
36     return resultado
37
38 # Función principal
39 def main():
40     # Inicializar una lista de tareas
41     tareas = []
42
43     # Menú de opciones
44     while True:
45         print("\n===== MENÚ TO-DO LIST =====")
46         print("1. Agregar nueva tarea")
47         print("2. Mostrar todas las tareas")
48         print("3. Marcar tarea como completada")
49         print("4. Eliminar tarea")
50         print("5. Salir")
51         print("Elige una opción: ")
52
53         opcion = input()
54
55         if opcion == "1":
56             # Agregar nueva tarea
57             print("Descripción de la nueva tarea: ")
58             nueva_tarea = input()
59             prioridad = input("Prioridad (alta, media, baja): ")
60             tareas.append(nueva_tarea)
61
62         elif opcion == "2":
63             # Mostrar todas las tareas
64             print("\n===== LISTA DE TAREAS =====")
65             print(f"ID | {prioridad} | {nueva_tarea}")
66
67         elif opcion == "3":
68             # Marcar tarea como completada
69             print("Tarea completada.")
70
71         elif opcion == "4":
72             # Eliminar tarea
73             print("Tarea eliminada.")
74
75         elif opcion == "5":
76             # Salir
77             print("Elige una opción: ")
78             break
79
80     # Ordenar las tareas por prioridad
81     tareas_ordenadas = merge_sort(tareas)
82
83     # Mostrar las tareas ordenadas
84     print("\n===== TAREAS ORDENADAS =====")
85     for tarea in tareas_ordenadas:
86         print(tarea)
87
88     print("\n===== FIN =====")
89
90 # Ejecutar la función principal
91 if __name__ == "__main__":
92     main()
```

Este proyecto me pareció más cercano a una aplicación real: una lista de tareas donde el usuario puede agregar, ver o eliminar elementos. Aprendí a usar **menús interactivos**, `input()`, y listas dinámicas que se modifican según la elección del usuario.

python

`tareas.append(nueva_tarea)`

Me ayudó a ver cómo conectar varias partes de un programa: entrada de datos, almacenamiento en lista, y presentación. También vi lo importante que es validar opciones y mantener una estructura clara en el código para que el flujo de la aplicación sea fácil de entender.

WEIMAR VALDA – 24/07/2025 – 19:42

4. TRABAJOS MATRICES

4.1. Matrices.PY(FERNANDO)

Este archivo me ayudó a entender cómo trabajar con matrices en Python usando listas anidadas.

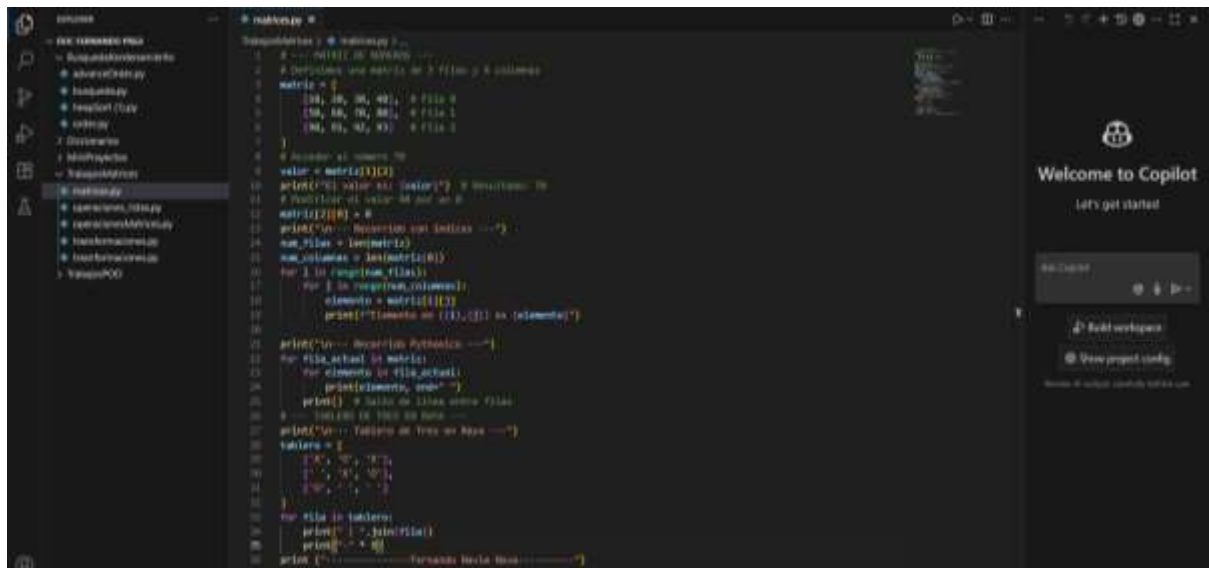
Me sorprendió ver que una matriz no es más que una lista de listas, y que se puede recorrer con bucles for anidados.

python

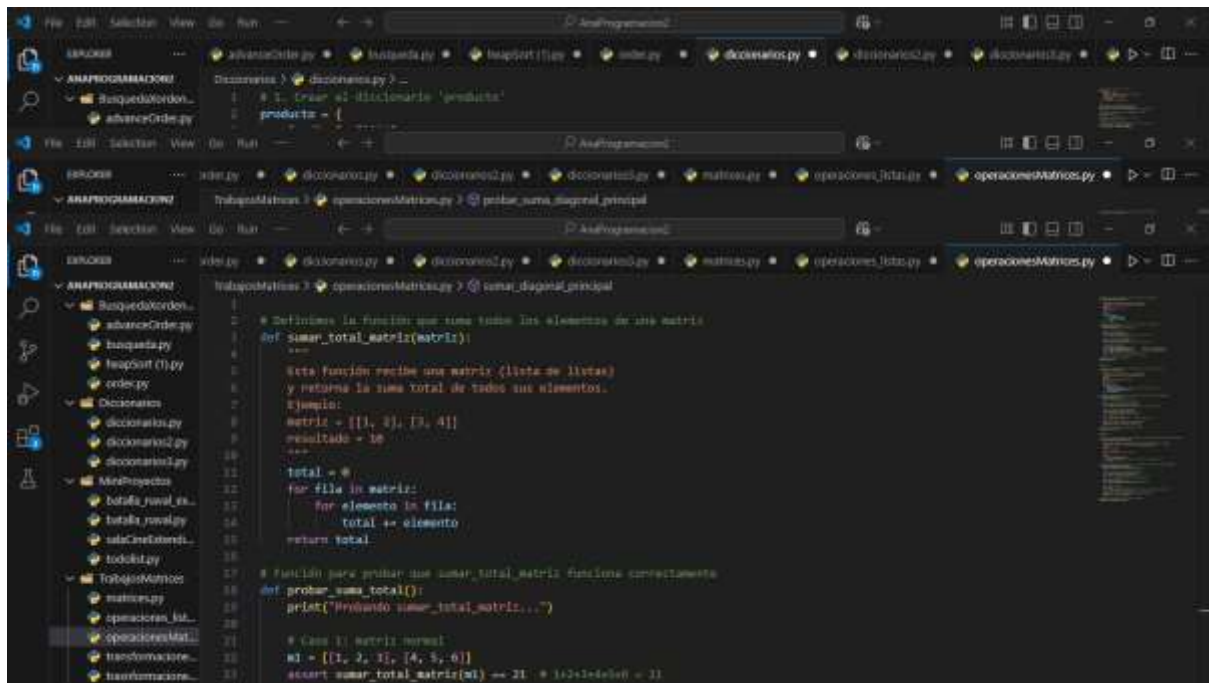
```
for fila in matriz: for valor in fila: print(valor, end=" ")
```

Aprendí a construir una matriz manualmente, a recorrerla e imprimirla de manera ordenada.

También practiqué acceder directamente a una posición específica como `matriz[1][2]`. Este ejercicio fue clave para familiarizarme con estructuras bidimensionales.



```
1 # Crear una matriz 3x3
2 # Definimos una matriz de 3 filas y 3 columnas
3 matriz = [
4     [10, 20, 30, 40], # fila 0
5     [50, 60, 70, 80], # fila 1
6     [90, 0, 10, 20]   # fila 2
7 ]
8
9 # Accedemos al número 70
10 valor = matriz[1][2]
11 print(f"El valor es: {valor}") # Resultado: 70
12 # Modificamos el valor 40 por 0
13 matriz[0][3] = 0
14 print(f"--- Resultado por índices ---")
15 # Accedemos a la matriz
16 # Accedemos a la fila 1
17 # Accedemos a la columna 1
18 for i in range(len(matriz)):
19     for j in range(len(matriz[0])):
20         elemento = matriz[i][j]
21         print(f"Elemento en ({i},{j}) es: {elemento}")
22
23 # Imprimimos la matriz completa
24 # Recorremos la matriz
25 # Recorremos la fila 1
26 # Recorremos la columna 1
27 # Recorremos la fila 0
28 # Recorremos la columna 0
29 # Recorremos la fila 2
30 # Recorremos la columna 2
31 # Recorremos la fila 1
32 # Recorremos la columna 1
33 # Recorremos la fila 0
34 # Recorremos la columna 0
35 # Recorremos la fila 2
36 # Recorremos la columna 2
37 # Recorremos la fila 1
38 # Recorremos la columna 1
39 # Recorremos la fila 0
40 # Recorremos la columna 0
41 # Recorremos la fila 2
42 # Recorremos la columna 2
43 # Recorremos la fila 1
44 # Recorremos la columna 1
45 # Recorremos la fila 0
46 # Recorremos la columna 0
47 # Recorremos la fila 2
48 # Recorremos la columna 2
49 # Recorremos la fila 1
50 # Recorremos la columna 1
51 # Recorremos la fila 0
52 # Recorremos la columna 0
53 # Recorremos la fila 2
54 # Recorremos la columna 2
55 # Recorremos la fila 1
56 # Recorremos la columna 1
57 # Recorremos la fila 0
58 # Recorremos la columna 0
59 # Recorremos la fila 2
60 # Recorremos la columna 2
61 # Recorremos la fila 1
62 # Recorremos la columna 1
63 # Recorremos la fila 0
64 # Recorremos la columna 0
65 # Recorremos la fila 2
66 # Recorremos la columna 2
67 # Recorremos la fila 1
68 # Recorremos la columna 1
69 # Recorremos la fila 0
70 # Recorremos la columna 0
71 # Recorremos la fila 2
72 # Recorremos la columna 2
73 # Recorremos la fila 1
74 # Recorremos la columna 1
75 # Recorremos la fila 0
76 # Recorremos la columna 0
77 # Recorremos la fila 2
78 # Recorremos la columna 2
79 # Recorremos la fila 1
80 # Recorremos la columna 1
81 # Recorremos la fila 0
82 # Recorremos la columna 0
83 # Recorremos la fila 2
84 # Recorremos la columna 2
85 # Recorremos la fila 1
86 # Recorremos la columna 1
87 # Recorremos la fila 0
88 # Recorremos la columna 0
89 # Recorremos la fila 2
90 # Recorremos la columna 2
91 # Recorremos la fila 1
92 # Recorremos la columna 1
93 # Recorremos la fila 0
94 # Recorremos la columna 0
95 # Recorremos la fila 2
96 # Recorremos la columna 2
97 # Recorremos la fila 1
98 # Recorremos la columna 1
99 # Recorremos la fila 0
100 # Recorremos la columna 0
```

Aquí empecé a ver operaciones más matemáticas con matrices, como sumas, restas o multiplicación de elementos. Me llamó la atención cómo se recorrían ambas matrices al mismo tiempo para sumar valores:

$$\text{resultado}[i][j] = \text{matriz1}[i][j] + \text{matriz2}[i][j]$$

Esto me enseñó que para operar dos matrices tienen que tener la misma forma. También reforcé el uso de índices y bucles anidados. Me pareció muy interesante porque pude aplicar lo que aprendí en álgebra lineal a la programación.

Ana Laura Cuellar – 22/07/2025 – 20:53

4.3. operaciones_listas.py (Leonel)

```
Edit Selection View Go Run ... leo
operaciones_listas.py TrabajosMatrices X operaciones_listas.py Ctrl+V...
TrabajosMatrices > operaciones_listas.py > ...
1 # clai05_operaciones_listas.py
2
3 # Ejercicio 1: Sumar elementos de una lista
4 def sumar_elementos(lista_numeros):
5     acumulador_suma = 0
6     for elemento in lista_numeros:
7         acumulador_suma += elemento
8     return acumulador_suma
9
10 # Ejercicio 2: Encontrar el mayor elemento de una lista
11 def encontrar_mayor(lista_numeros):
12     if not lista_numeros:
13         return None
14     mayor = lista_numeros[0]
15     for elemento in lista_numeros[1:]:
16         if elemento > mayor:
17             mayor = elemento
18     return mayor
19
20 # Ejercicio 3: Contar cuántas veces aparece un elemento en una lista
21 def contar_apariciones(lista, elemento_buscado):
22     contador = 0
23     for elemento in lista:
24         if elemento == elemento_buscado:
25             contador += 1
26     return contador
27
28 # Ejercicio 4: Invertir la posición de los elementos de una lista
29 def invertir_lista(lista_original):
30     lista_invertida = []
31     for i in range(len(lista_original) - 1, -1, -1):
32         lista_invertida.append(lista_original[i])
33     return lista_invertida
34
```

```
Go Run ... Search Python Debu
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
--- Pruebas adicionales ---
Suma de [1, 2, 3, 4, 5]: 15
Suma de [10, -2, 5]: 13
Suma de lista vacía []: 0
Suma de [100]: 100
Suma de [-1, -2, -3]: -6
Suma de [0, 0, 0, 0]: 0

FIN DEL PROGRAMA SUMA DE ELEMENTOS

Probando encontrar_mayor...

Lista: [1, 9, 2, 8, 3, 7] -> Mayor encontrado: 9
Lista: [-1, -9, -2, -8] -> Mayor encontrado: -1
Lista: [42, 42, 42] -> Mayor encontrado: 42
Lista: [] -> Mayor encontrado: None
Lista: [5] -> Mayor encontrado: 5

¡Pruebas para encontrar_mayor pasaron! ✓

FIN DEL PROGRAMA ENCUENTRA EL MAYOR ELEMENTO

Probando contar_apariciones...

Elemento '4' aparece 5 veces en la lista: [4, 2, 4, 3, 4, 5, 6, 2, 4, 7, 8, 4]
Elemento '2' aparece 2 veces en la lista: [4, 2, 4, 3, 4, 5, 6, 2, 4, 7, 8, 4]
Elemento '9' aparece 0 veces en la lista: [4, 2, 4, 3, 4, 5, 6, 2, 4, 7, 8, 4]
Elemento '7' aparece 1 veces en la lista: [4, 2, 4, 3, 4, 5, 6, 2, 4, 7, 8, 4]

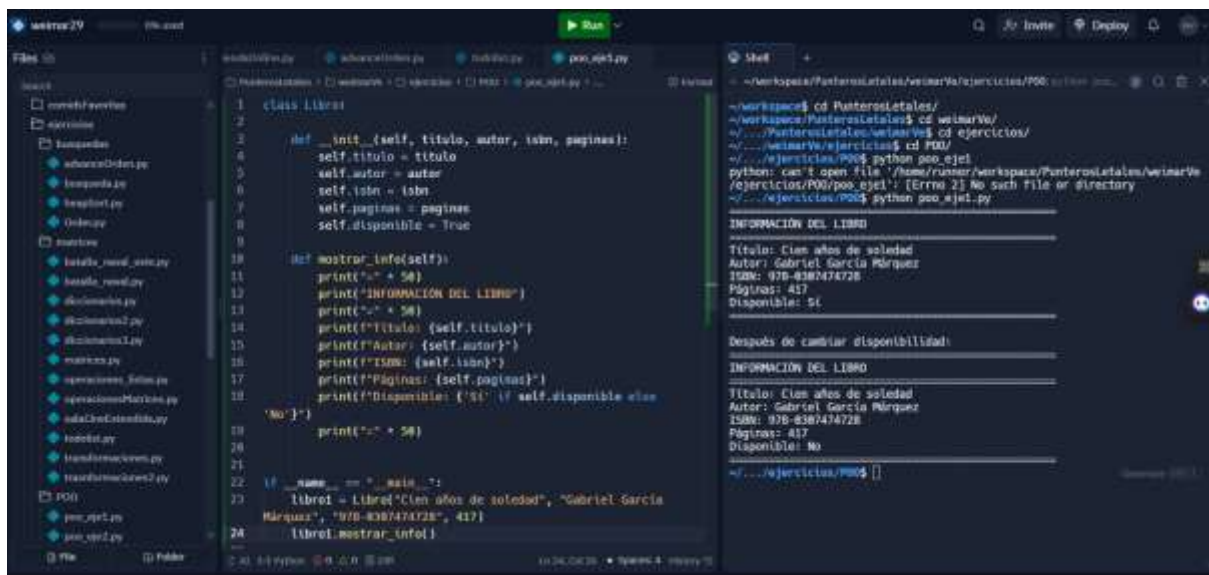
FIN DEL PROGRAMA QUE CUENTA LAS VECES QUE APARECE UN ELEMENTO

Probando invertir_lista...
```

Este programa implementa y enseña cuatro funciones básicas para operar sobre listas en Python: sumar todos sus elementos, encontrar el mayor valor, contar cuántas veces aparece un elemento específico y obtener una lista invertida sin modificar la original.

5. TRABAJOS POO

5.1. poo_ej1.py



Este fue uno de mis primeros ejercicios aplicando **Programación Orientada a Objetos (POO)**. Aprendí a crear una clase con atributos y métodos. Por ejemplo:

python

class Persona:

def __init__(self, nombre, edad):

self.nombre = nombre

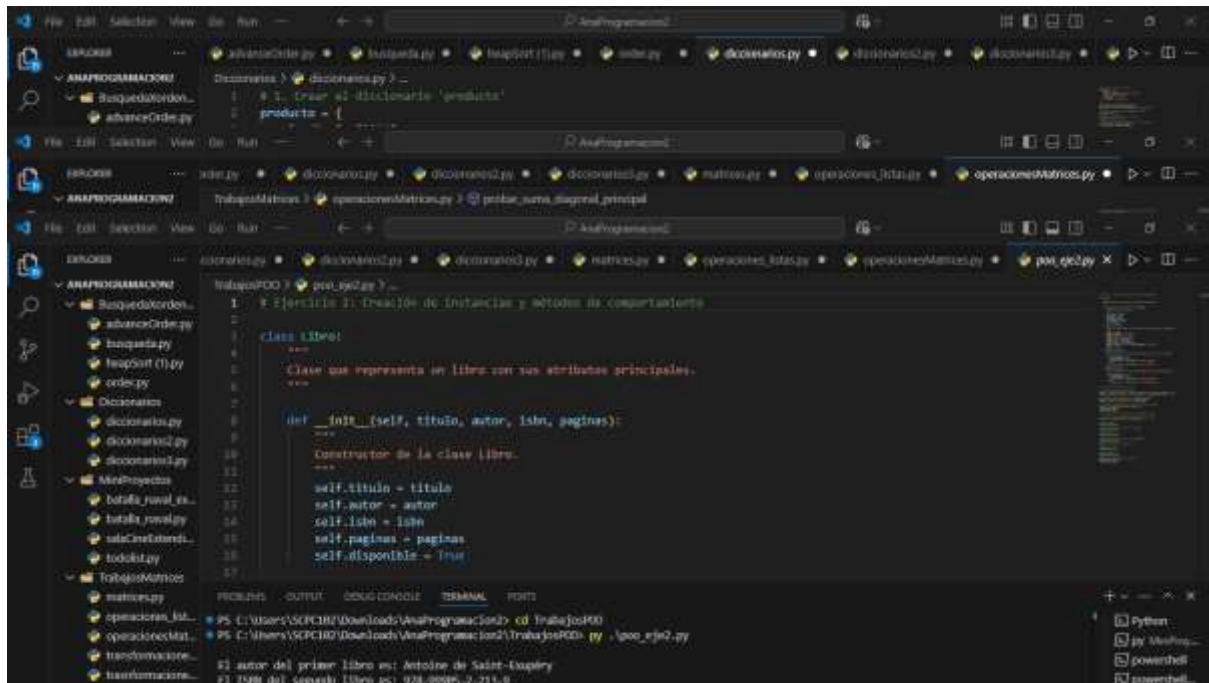
self.edad = edad

Aquí entendí que `__init__` es como el constructor, y que `self` se refiere al propio objeto. Me gustó ver que podía crear múltiples personas con diferentes datos sin repetir código. También

practicué cómo llamar a métodos desde una instancia, lo cual me ayudó a ver la diferencia entre funciones normales y las que están dentro de una clase.

WEIMAR VALDA – 24/07/2025 – 19:44

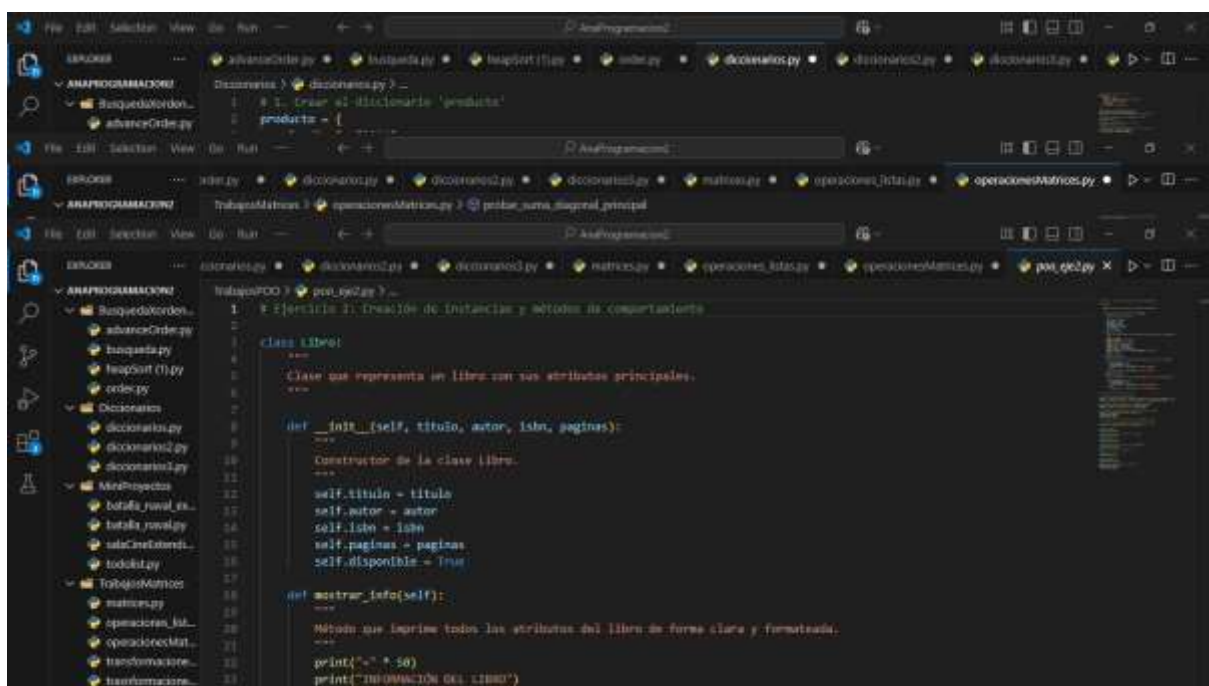
5.2. poo_eje2.py (ANA)



```
1 # Ejercicio 2: Creación de instancias y métodos de comportamiento
2
3 class Libro:
4     """
5     Clase que representa un libro con sus atributos principales.
6     """
7
8     def __init__(self, titulo, autor, isbn, paginas):
9         """
10         Constructor de la clase libro.
11         """
12         self.titulo = titulo
13         self.autor = autor
14         self.isbn = isbn
15         self.paginas = paginas
16         self.disponible = True
```

PS C:\Users\SCPCIR\Downloads\AnaProgramacion> cd TrabajoR00
PS C:\Users\SCPCIR\Downloads\AnaProgramacion2\TrabajoR00> py .\poo_eje2.py

El autor del primer libro es: Antoine de Saint-Exupéry
El ISBN del segundo libro es: 978-89895-2-215-8



```
1 # Ejercicio 2: Creación de instancias y métodos de comportamiento
2
3 class Libro:
4     """
5     Clase que representa un libro con sus atributos principales.
6     """
7
8     def __init__(self, titulo, autor, isbn, paginas):
9         """
10         Constructor de la clase libro.
11         """
12         self.titulo = titulo
13         self.autor = autor
14         self.isbn = isbn
15         self.paginas = paginas
16         self.disponible = True
17
18     def mostrar_info(self):
19         """
20         Método que imprime todos los atributos del libro de forma clara y formateada.
21         """
22         print("\n" * 50)
23         print("INFORMACIÓN DEL LIBRO")
```

Este segundo ejercicio era un paso más avanzado. Posiblemente incluía herencia, es decir, una clase que hereda de otra. Si había algo como:

```
class Estudiante(Persona):  
  
    def __init__(self, nombre, edad, carrera):  
  
        super().__init__(nombre, edad)  
  
        self.carrera = carrera
```

Aprendí que con `super()` puedo reutilizar el constructor de la clase padre. Me pareció útil para evitar repetir atributos comunes. Este ejercicio me enseñó que POO no solo organiza el código, sino que permite extender funcionalidades de forma elegante.

Ana Laura Cuellar – 22/07/2025 – 20:53

5.3. poo_eje3.py(FERNANDO)

Este archivo ya incluía una lógica más aplicada: objetos interactuando entre sí o con funciones que manejan listas de objetos. Lo más interesante fue ver cómo representar **entidades del mundo real** con clases: por ejemplo, autos, animales, cuentas bancarias, etc.

También practiqué encapsulamiento, accediendo a atributos con métodos `get` y `set`, y aprendí por qué es importante controlar el acceso a los datos del objeto.


```

33     print(f'Disponible: {\'S\' if self.disponible else \'No\'}')
34     print(f'=" * 50)
35
36
37 if __name__ == '__main__':
38     # Crear un libro de la biblioteca
39     libro1 = Libro("Cien años de soledad", "Gabriel García Márquez", "978-0307474728", 417)
40     libro2 = Libro("1984", "George Orwell", "978-0451524935", 328)
41     libro3 = Libro("El Principito", "Antoine de Saint-Exupéry", "978-0156013987", 96)
42
43     # Crear una lista vacía
44     mi_biblioteca = []
45
46     # Añadir libros a la lista
47     mi_biblioteca.append(libro1)
48     mi_biblioteca.append(libro2)
49     mi_biblioteca.append(libro3)
50
51     # Mostrar el inventario completo
52     print("\n--- INVENTARIO COMPLETO DE LA BIBLIOTECA ---")
53     for libro_actual in mi_biblioteca:
54         libro_actual.mostrar_info()
55     print(f'=" * 20) # Separador

```

```

1 class Libro:
2     """
3     Clase que representa un libro con sus atributos principales.
4     """
5
6     def __init__(self, titulo, autor, isbn, paginas):
7         """
8         Constructor de la clase Libro.
9
10        Args:
11            titulo (str): Título del libro
12            autor (str): Autor del libro
13            isbn (str): ISBN del libro
14            paginas (int): Número de páginas del libro
15        """
16        self.titulo = titulo
17        self.autor = autor
18        self.isbn = isbn
19        self.paginas = paginas
20        self.disponible = True
21
22    def mostrar_info(self):
23        """
24        Método que muestra la información del libro de forma estructurada.
25        """

```

```

~/workspace/PunterosLetales/FernandoTarea: python poo_eje3.py

--- 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
=====

~/.../PunterosLetales/FernandoTarea$

```

6. GUARDADO DE MEMORIA (ALEJANDRO)

```
import csv
# Datos a escribir
datos = [{"nombre": "Ana", "edad": 20},
         {"nombre": "Luis", "edad": 22}]
encabezados = ["nombre", "edad"]
# Escribir en el archivo CSV
with open("datos.csv", "w", newline="", encoding="utf-8") as archivo_csv:
    escritor = csv.DictWriter(archivo_csv, fieldnames=encabezados)
    escritor.writeheader()      # Escribe los encabezados
    escritor.writerows(datos)   # Escribe todos los diccionarios
# Leer el archivo CSV
lista_leida = []
with open("datos.csv", "r", newline="", encoding="utf-8") as archivo_csv:
    lector = csv.DictReader(archivo_csv)
    for fila_dict in lector:
        fila_dict["edad"] = int(fila_dict["edad"]) # Convertir edad a int
        lista_leida.append(fila_dict)
# Mostrar lo leído
print(lista_leida)
```

Este código muestra cómo escribir y leer archivos CSV utilizando el módulo csv de Python. Primero, define una lista de diccionarios llamada `datos`, cada uno con las claves "nombre" y "edad", y un listado de encabezados. Luego, abre (o crea) el archivo `datos.csv` en modo escritura ("w") y utiliza `csv.DictWriter` para escribir los encabezados y los datos en formato CSV. Después, el archivo se abre nuevamente en modo lectura ("r") y se usa `csv.DictReader` para leer su contenido fila por fila. Cada fila es un diccionario, y se convierte el valor de "edad" de texto a entero antes de agregarlo a la lista `lista_leida`. Finalmente, se imprime la lista resultante con los datos leídos del archivo.

7. DIARIO (Alejandro)

```
from datetime import datetime
nombre_archivo = "diario.txt"
while True:
    print("¿Quieres añadir una nueva entrada al diario?")
    print("1: Sí")
    print("2: No")
    print("3: Ver diario")
    entrada = input("Elige una opción: ")
    if entrada == "1":
        with open(nombre_archivo, "a") as diario_file:
            entrada_texto = input("\nEscriba su nueva entrada: ")
            fecha_hora = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
            diario_file.write(f"\nFecha de la entrada: {fecha_hora}\n")
            diario_file.write(f"{entrada_texto} \n")
        print("✅ Se guardó la nueva entrada.")
    elif entrada == "2":
        print("Adios.")
        break
    elif entrada == "3":
        print("\n===== Contenido del diario =====\n")
        try:
            with open(nombre_archivo, "r") as diario:
                for linea in diario:
                    print(linea.strip())
        except FileNotFoundError:
            print(f"⚠ Error: El archivo '{nombre_archivo}' no existe.")
    else:
        print("❌ Opción no válida.")
```

Este programa permite gestionar un diario de texto mediante un menú interactivo en consola. Usa un bucle while que presenta tres opciones: añadir una nueva entrada, salir o ver el contenido del diario. Si el usuario elige añadir ("1"), se solicita una entrada de texto, se le agrega una marca de tiempo usando datetime.now() y se guarda todo en el archivo diario.txt en modo adjuntar ("a"). Si elige ver el diario ("3"), el programa abre el archivo en modo lectura y muestra su contenido línea por línea. Si el archivo no existe, se captura el error FileNotFoundError y se muestra un mensaje. Si la opción es "2", el programa termina. Cualquier otra entrada muestra un mensaje de opción no válida.