

Universidad Privada Domingo Savio



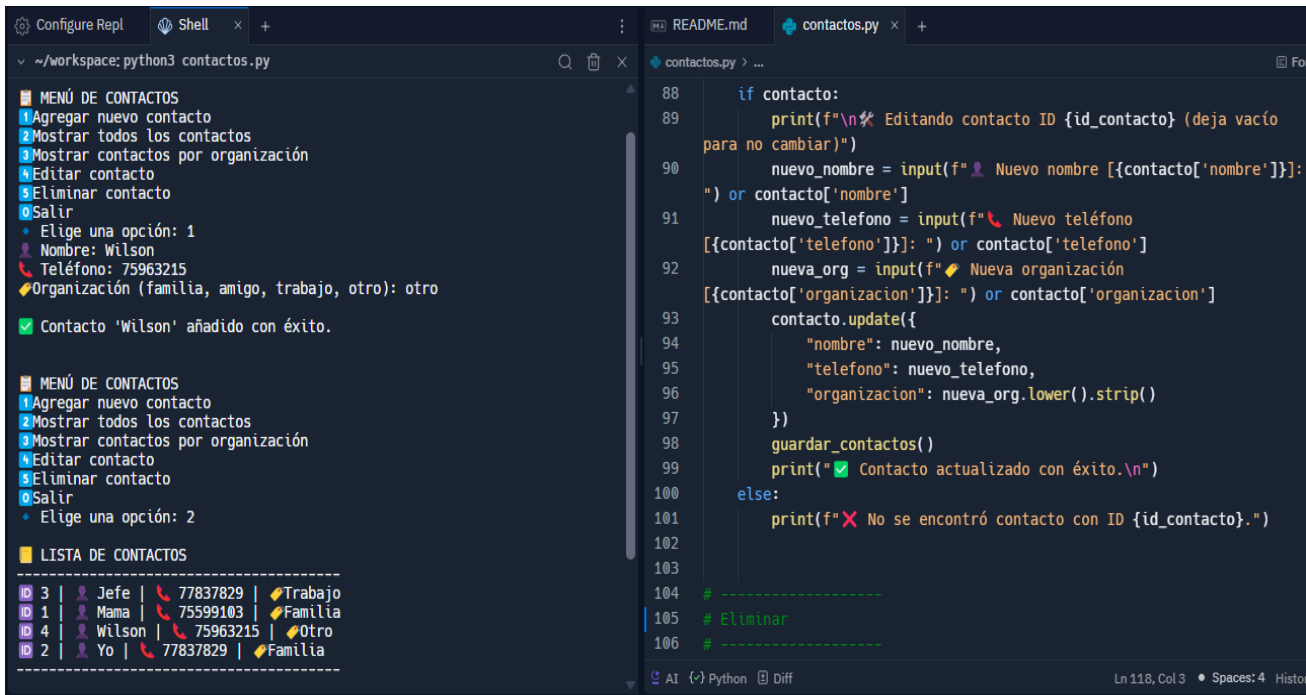
Integrantes :

Flavia Gutiérrez Soliz

Docente:

Jimmy Requena

1. AGENDA



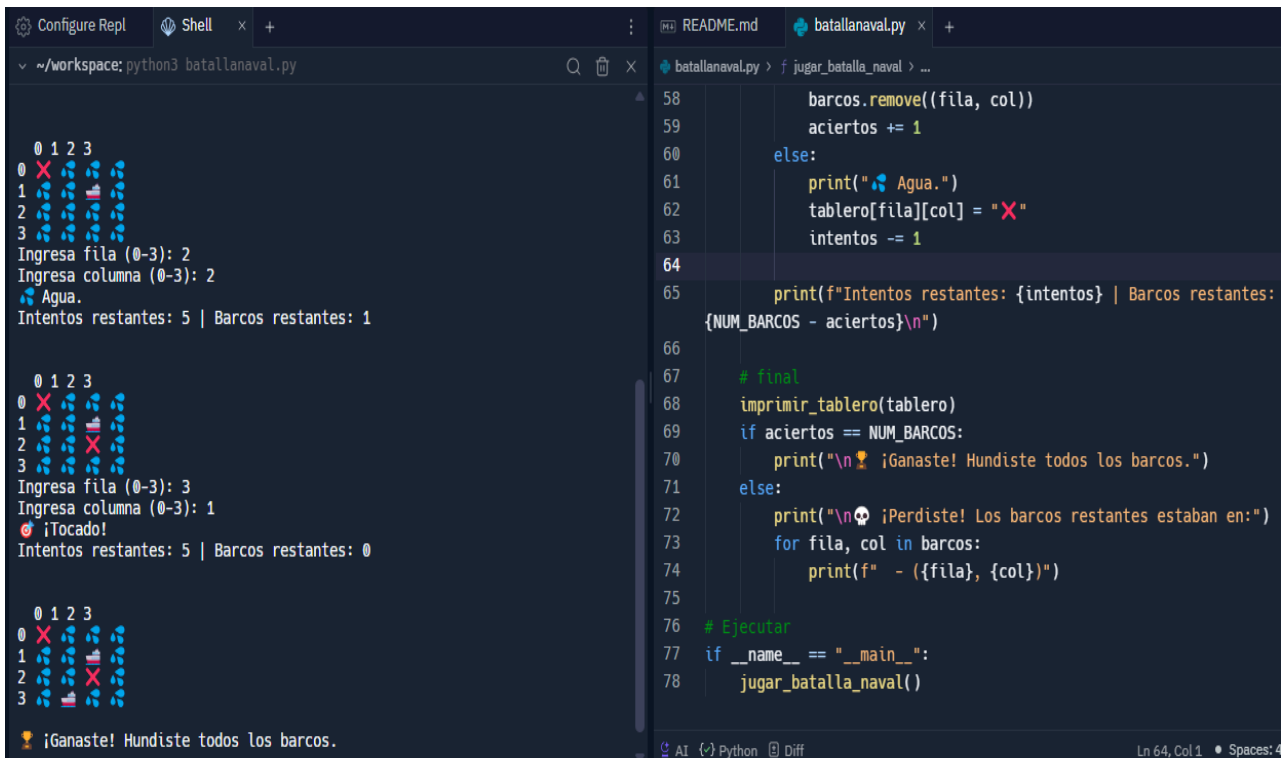
```

88     if contacto:
89         print(f"\n✂ Editando contacto ID {id_contacto} (deja vacío
para no cambiar)")
90         nuevo_nombre = input(f"👤 Nuevo nombre [{contacto['nombre']}:
") or contacto['nombre']
91         nuevo_telefono = input(f"☎ Nuevo teléfono
[{contacto['telefono']}: ") or contacto['telefono']
92         nueva_org = input(f"📁 Nueva organización
[{contacto['organizacion']}: ") or contacto['organizacion']
93         contacto.update({
94             "nombre": nuevo_nombre,
95             "telefono": nuevo_telefono,
96             "organizacion": nueva_org.lower().strip()
97         })
98         guardar_contactos()
99         print("✅ Contacto actualizado con éxito.\n")
100     else:
101         print(f"❌ No se encontró contacto con ID {id_contacto}.")
102
103
104 # -----
105 # Eliminar
106 # -----

```

Este código implementa un gestor de contactos en consola usando Python, que permite agregar, mostrar (con o sin filtro por organización), editar y eliminar contactos, almacenándolos en un archivo JSON. Cada contacto tiene un ID único, y los cambios se guardan automáticamente tras cada operación.

2. BATALLA NAVAL



```

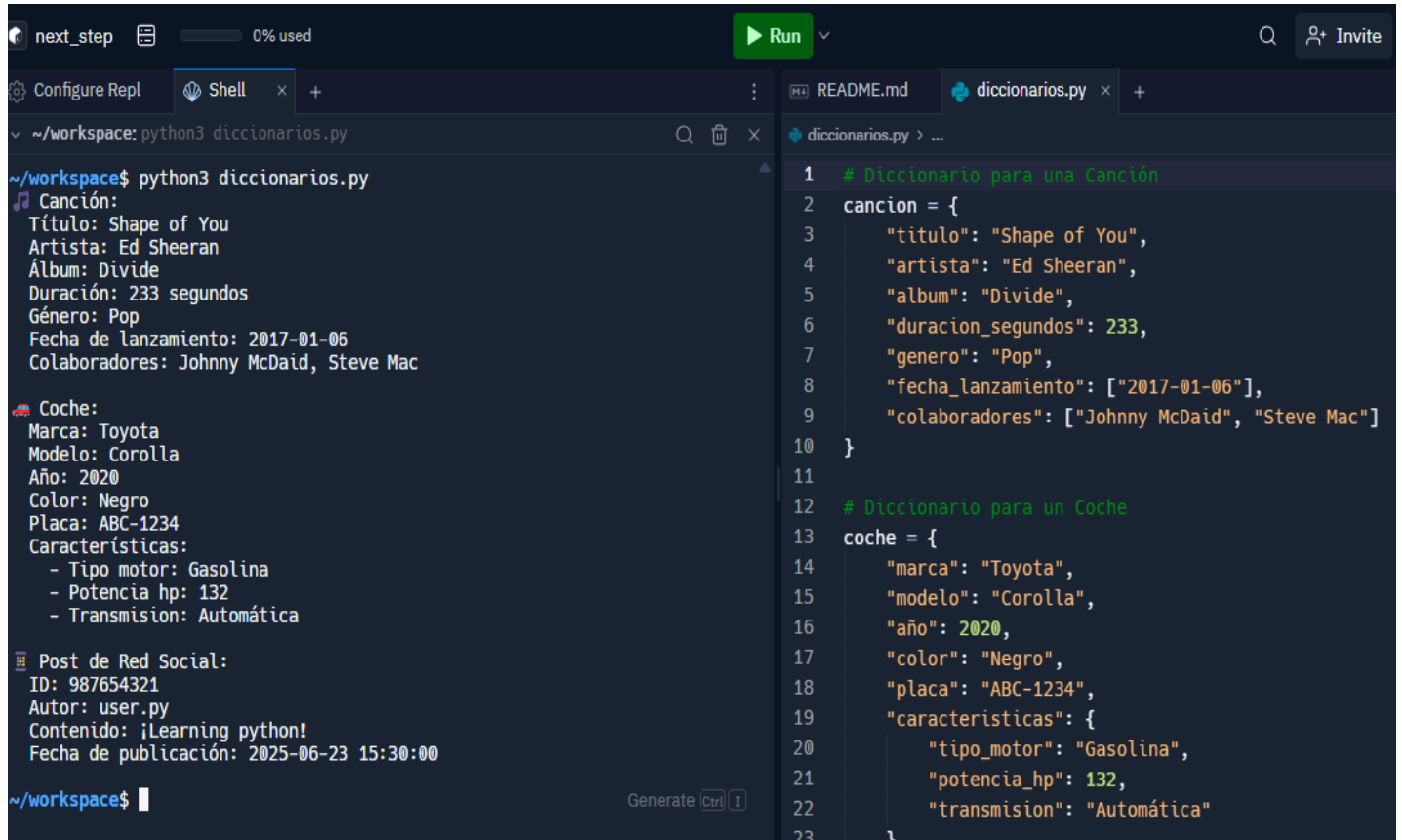
58     barcos.remove((fila, col))
59     aciertos += 1
60     else:
61         print("💧 Agua.")
62         tablero[fila][col] = "X"
63         intentos -= 1
64
65     print(f"Intentos restantes: {intentos} | Barcos restantes:
{NUM_BARCOS - aciertos}\n")
66
67     # final
68     imprimir_tablero(tablero)
69     if aciertos == NUM_BARCOS:
70         print("\n🏆 ¡Ganaste! Hundiste todos los barcos.")
71     else:
72         print("\n💀 ¡Perdiste! Los barcos restantes estaban en:")
73         for fila, col in barcos:
74             print(f" - {fila}, {col}")
75
76     # Ejecutar
77     if __name__ == "__main__":
78         jugar_batalla_naval()

```

Batalla Naval en consola utilizando una matriz (lista de listas) para representar un tablero de 4x4. Los barcos se colocan aleatoriamente como coordenadas en un conjunto, y el jugador tiene hasta 7 intentos para encontrarlos ingresando posiciones. El tablero se actualiza visualmente con símbolos

según los aciertos o fallos, y el juego termina cuando se hunden todos los barcos o se acaban los intentos.

3. DICCIONARIO 1



The screenshot shows a code editor with a dark theme. On the left, a terminal window displays the output of running a Python script. On the right, the editor shows the source code of the script, which defines three dictionaries and functions to print them in a structured format.

```
~/workspace$ python3 diccionarios.py
🎵 Canción:
Título: Shape of You
Artista: Ed Sheeran
Álbum: Divide
Duración: 233 segundos
Género: Pop
Fecha de lanzamiento: 2017-01-06
Colaboradores: Johnny McDaid, Steve Mac

🚗 Coche:
Marca: Toyota
Modelo: Corolla
Año: 2020
Color: Negro
Placa: ABC-1234
Características:
- Tipo motor: Gasolina
- Potencia hp: 132
- Transmisión: Automática

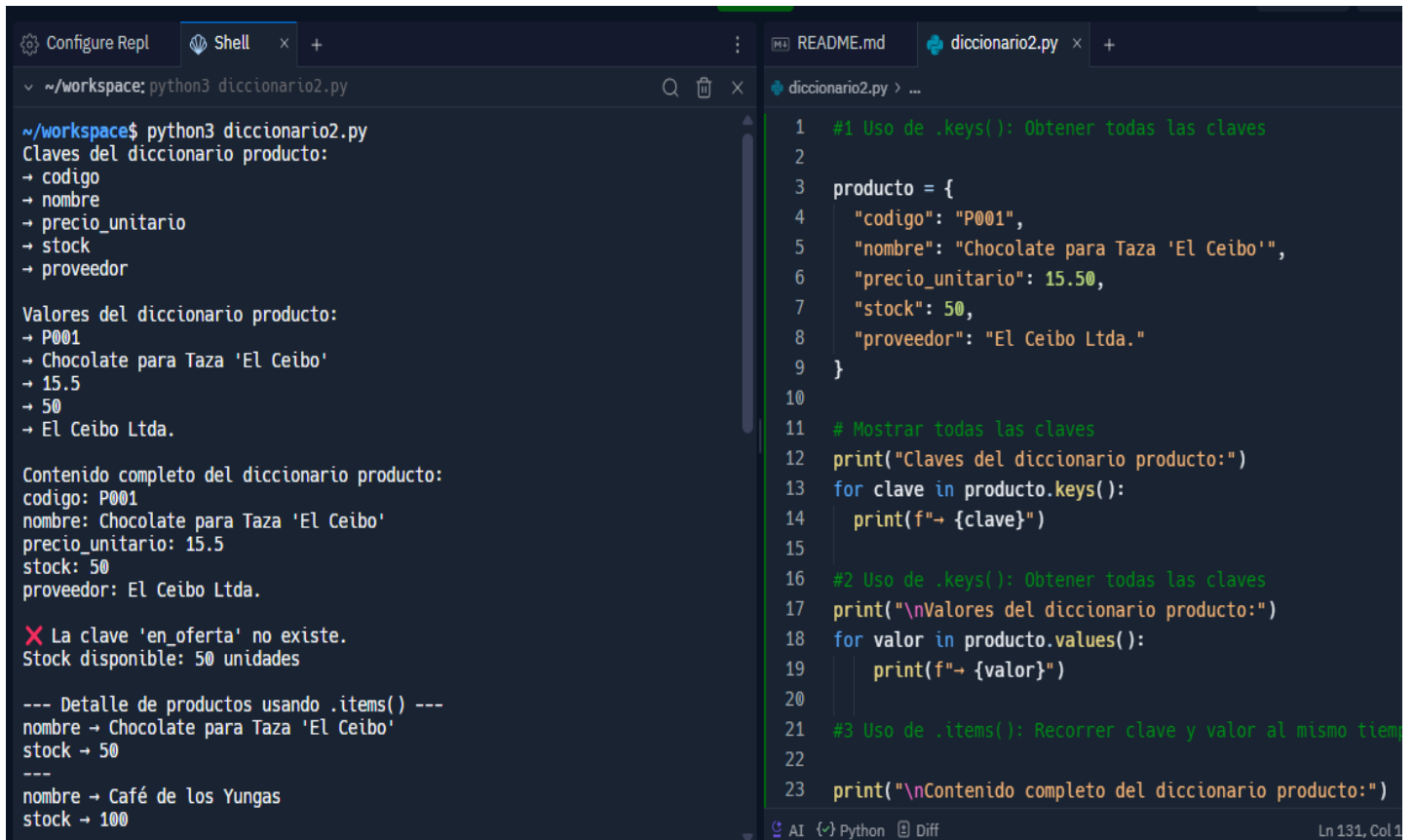
📱 Post de Red Social:
ID: 987654321
Autor: user.py
Contenido: ¡Learning python!
Fecha de publicación: 2025-06-23 15:30:00

~/workspace$
```

```
1 # Diccionario para una Canción
2 cancion = {
3     "titulo": "Shape of You",
4     "artista": "Ed Sheeran",
5     "album": "Divide",
6     "duracion_segundos": 233,
7     "genero": "Pop",
8     "fecha_lanzamiento": ["2017-01-06"],
9     "colaboradores": ["Johnny McDaid", "Steve Mac"]
10 }
11
12 # Diccionario para un Coche
13 coche = {
14     "marca": "Toyota",
15     "modelo": "Corolla",
16     "año": 2020,
17     "color": "Negro",
18     "placa": "ABC-1234",
19     "caracteristicas": {
20         "tipo_motor": "Gasolina",
21         "potencia_hp": 132,
22         "transmision": "Automática"
23     }
```

Este código define tres diccionarios en Python que representan una canción, un coche y un post de red social, cada uno con atributos específicos. Luego, se implementan funciones que imprimen los datos de cada diccionario de forma ordenada y legible. Utiliza estructuras anidadas como listas y sub diccionarios (en el caso del coche), y aplica técnicas como `join()` y `for` para formatear la salida. Es un ejemplo claro del uso de diccionarios y funciones para organizar y mostrar información estructurada.

4. DICCIONARIO 2



The screenshot shows a code editor with two panels. The left panel is a terminal window showing the execution of a Python script. The right panel shows the source code of the script, `diccionario2.py`.

Terminal Output:

```
~/workspace$ python3 diccionario2.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

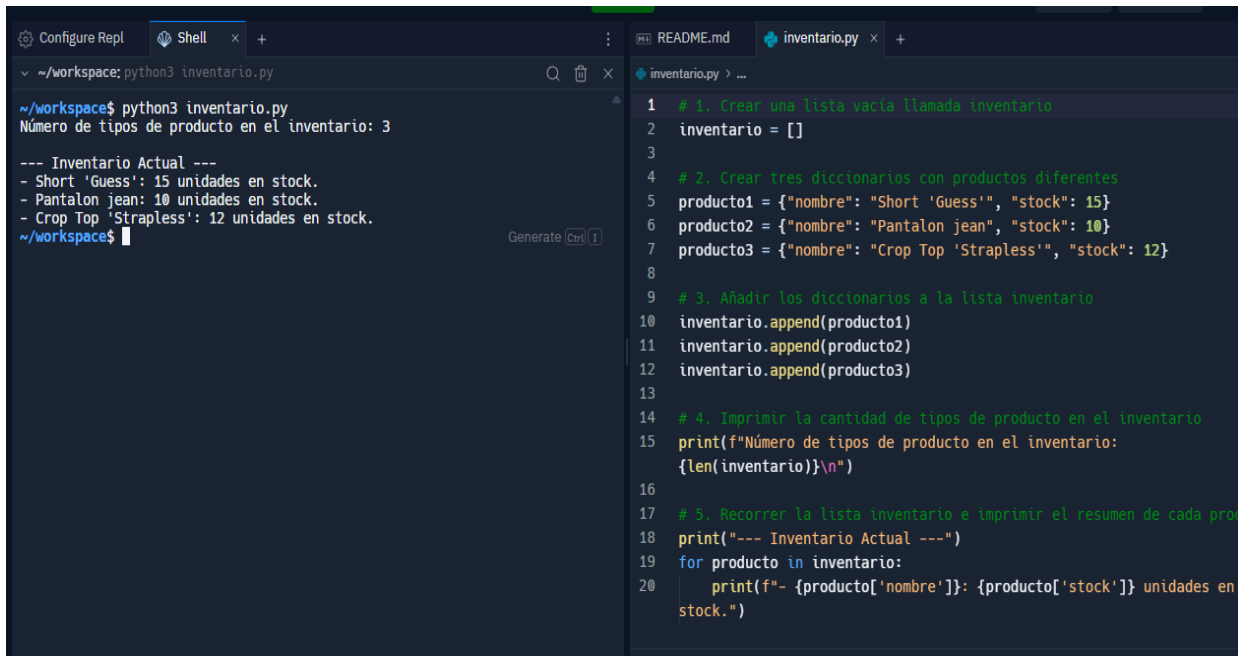
--- Detalle de productos usando .items() ---
nombre → Chocolate para Taza 'El Ceibo'
stock → 50
---
nombre → Café de los Yungas
stock → 100
```

Source Code (diccionario2.py):

```
1  #1 Uso de .keys(): Obtener todas las claves
2
3  producto = {
4      "codigo": "P001",
5      "nombre": "Chocolate para Taza 'El Ceibo'",
6      "precio_unitario": 15.50,
7      "stock": 50,
8      "proveedor": "El Ceibo Ltda."
9  }
10
11 # Mostrar todas las claves
12 print("Claves del diccionario producto:")
13 for clave in producto.keys():
14     print(f"→ {clave}")
15
16 #2 Uso de .keys(): Obtener todas las claves
17 print("\nValores del diccionario producto:")
18 for valor in producto.values():
19     print(f"→ {valor}")
20
21 #3 Uso de .items(): Recorrer clave y valor al mismo tiempo
22
23 print("\nContenido completo del diccionario producto:")
```

Este código muestra cómo trabajar con diccionarios en Python para modelar y manejar información estructurada de manera clara. Se utiliza `.keys()`, `.values()` y `.items()` para acceder a las claves, valores y pares clave-valor, respectivamente. También se demuestra cómo verificar la existencia de claves con `in`. Los ejemplos incluyen datos de un producto, un inventario, una canción, un coche y un post de red social, destacando el uso de diferentes tipos de datos como enteros, booleanos, cadenas, listas y diccionarios anidados. Es una aplicación práctica del manejo de estructuras de datos en Python.

5. INVENTARIO



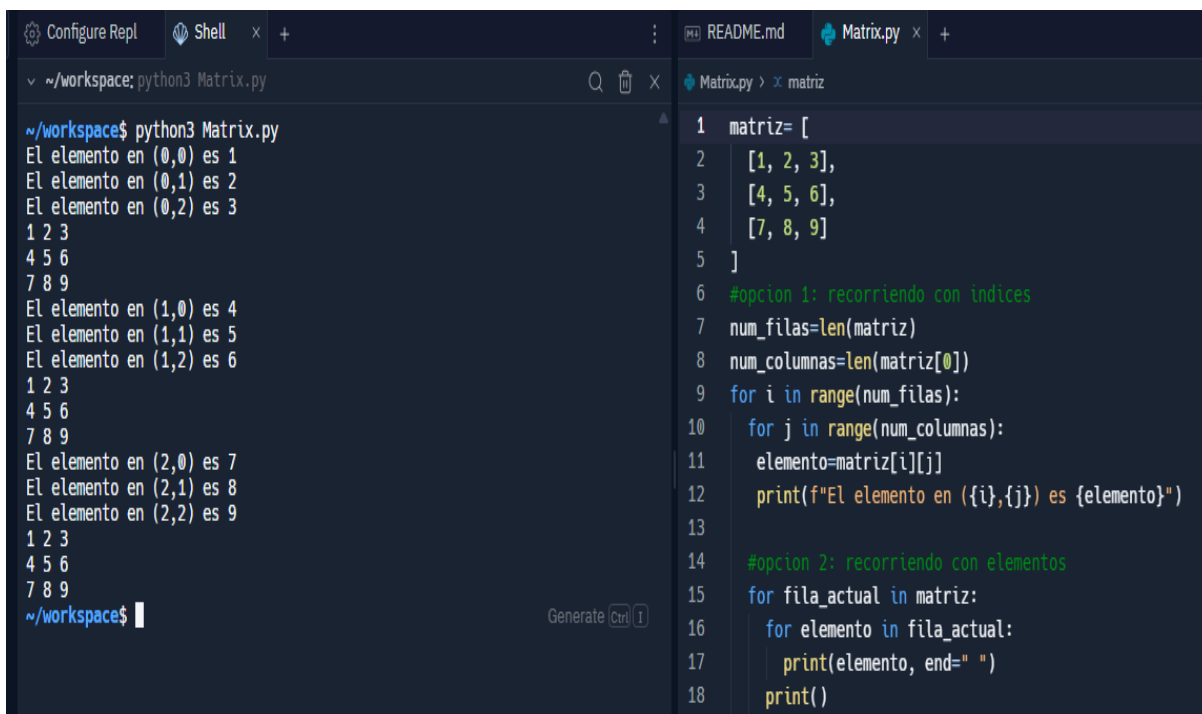
```
~/workspace: python3 inventario.py
Número de tipos de producto en el inventario: 3

--- Inventario Actual ---
- Short 'Guess': 15 unidades en stock.
- Pantalón jean: 10 unidades en stock.
- Crop Top 'Strapless': 12 unidades en stock.
~/workspace$
```

```
1 # 1. Crear una lista vacía llamada inventario
2 inventario = []
3
4 # 2. Crear tres diccionarios con productos diferentes
5 producto1 = {"nombre": "Short 'Guess'", "stock": 15}
6 producto2 = {"nombre": "Pantalón jean", "stock": 10}
7 producto3 = {"nombre": "Crop Top 'Strapless'", "stock": 12}
8
9 # 3. Añadir los diccionarios a la lista inventario
10 inventario.append(producto1)
11 inventario.append(producto2)
12 inventario.append(producto3)
13
14 # 4. Imprimir la cantidad de tipos de producto en el inventario
15 print(f"Número de tipos de producto en el inventario: {len(inventario)}\n")
16
17 # 5. Recorrer la lista inventario e imprimir el resumen de cada producto
18 print("--- Inventario Actual ---")
19 for producto in inventario:
20     print(f"- {producto['nombre']}: {producto['stock']} unidades en stock.")
```

Este código crea y gestiona un inventario básico en Python utilizando una lista de diccionarios, donde cada diccionario representa un producto con su nombre y cantidad en stock. Es un ejemplo simple y efectivo del uso combinado de listas y diccionarios para representar colecciones de datos estructurados.

6. MATRIZ 1

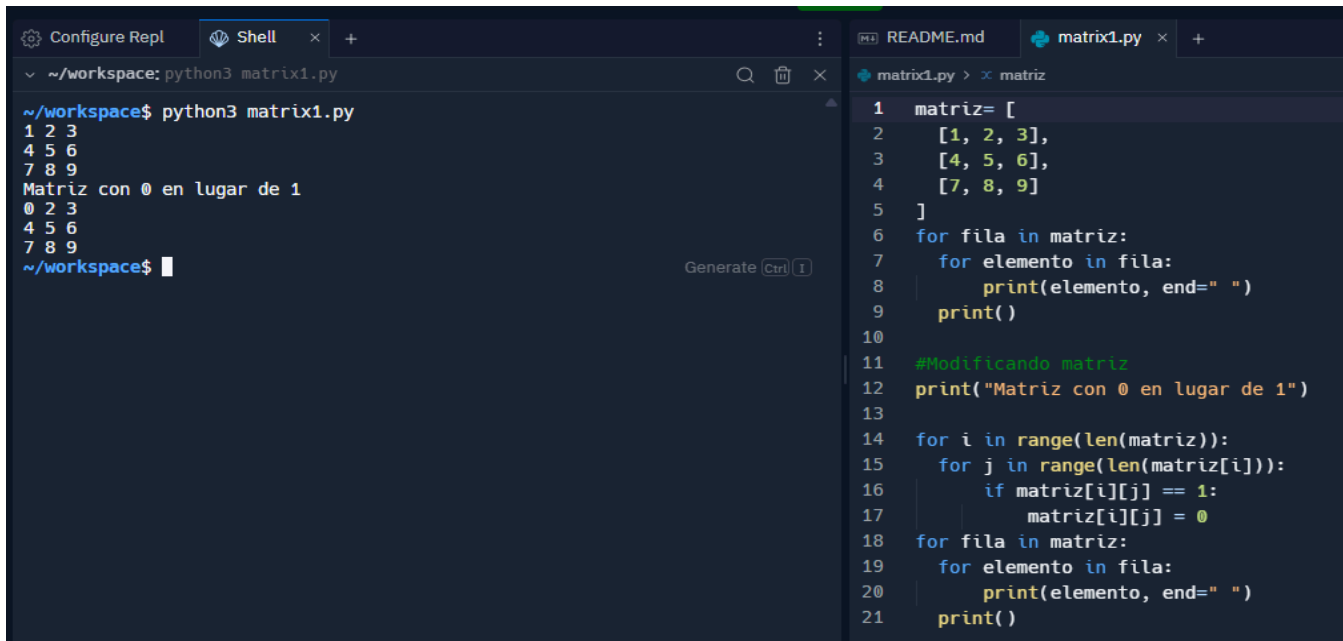


```
~/workspace: python3 Matrix.py
El elemento en (0,0) es 1
El elemento en (0,1) es 2
El elemento en (0,2) es 3
1 2 3
4 5 6
7 8 9
El elemento en (1,0) es 4
El elemento en (1,1) es 5
El elemento en (1,2) es 6
1 2 3
4 5 6
7 8 9
El elemento en (2,0) es 7
El elemento en (2,1) es 8
El elemento en (2,2) es 9
1 2 3
4 5 6
7 8 9
~/workspace$
```

```
1 matriz= [
2     [1, 2, 3],
3     [4, 5, 6],
4     [7, 8, 9]
5 ]
6 #opcion 1: recorriendo con indices
7 num_filas=len(matriz)
8 num_columnas=len(matriz[0])
9 for i in range(num_filas):
10     for j in range(num_columnas):
11         elemento=matriz[i][j]
12         print(f"El elemento en ({i},{j}) es {elemento}")
13
14 #opcion 2: recorriendo con elementos
15 for fila_actual in matriz:
16     for elemento in fila_actual:
17         print(elemento, end=" ")
18     print()
```

Este código muestra dos formas de recorrer una matriz en Python: una usando índices para acceder y mostrar cada elemento junto con su posición, y otra recorriendo directamente los elementos fila por fila para imprimirlos, demostrando diferentes métodos para manipular listas anidadas.

7. MATRIZ 2



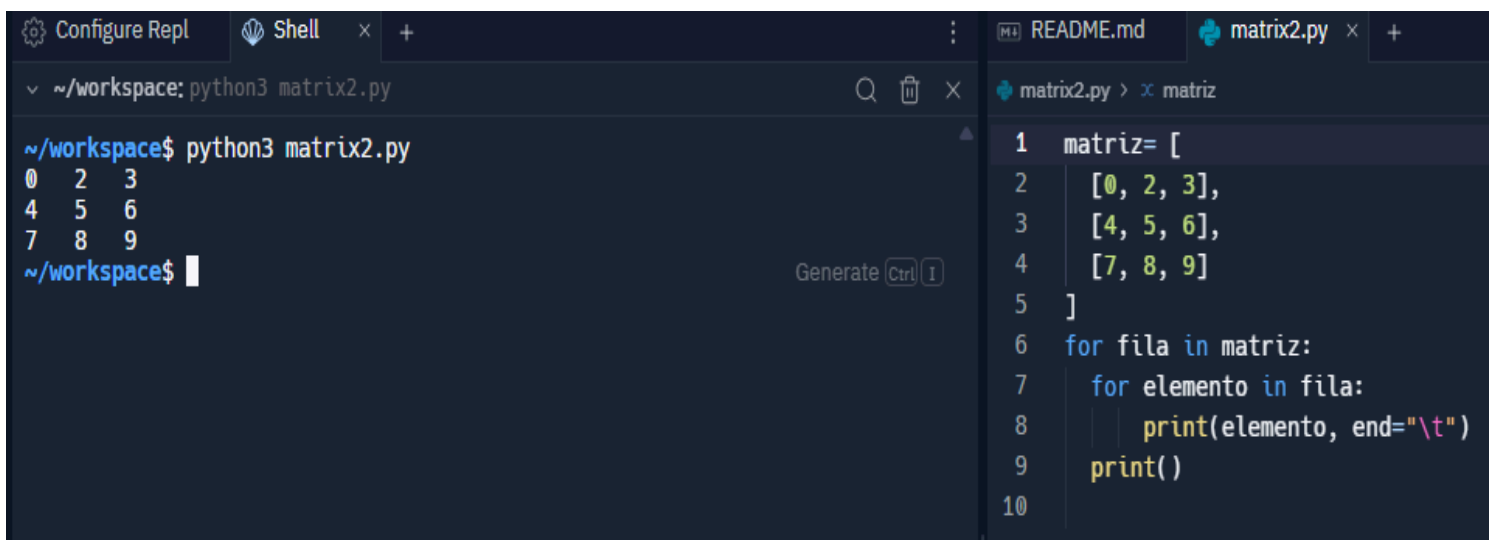
The screenshot shows a code editor with two panels. The left panel displays the terminal output of running `python3 matrix1.py`. The output shows a 3x3 matrix of numbers (1, 2, 3; 4, 5, 6; 7, 8, 9), followed by a message "Matriz con 0 en lugar de 1" and the same matrix with the value 1 replaced by 0. The right panel shows the source code of `matrix1.py`, which defines a 3x3 matrix, prints it, and then iterates through it to replace the value 1 with 0.

```
~/workspace$ python3 matrix1.py
1 2 3
4 5 6
7 8 9
Matriz con 0 en lugar de 1
0 2 3
4 5 6
7 8 9
~/workspace$
```

```
1 matriz= [
2     [1, 2, 3],
3     [4, 5, 6],
4     [7, 8, 9]
5 ]
6 for fila in matriz:
7     for elemento in fila:
8         print(elemento, end=" ")
9     print()
10
11 #Modificando matriz
12 print("Matriz con 0 en lugar de 1")
13
14 for i in range(len(matriz)):
15     for j in range(len(matriz[i])):
16         if matriz[i][j] == 1:
17             matriz[i][j] = 0
18 for fila in matriz:
19     for elemento in fila:
20         print(elemento, end=" ")
21     print()
```

Este código recorre e imprime una matriz de números, y luego modifica el valor 1 por 0 dentro de la matriz, mostrando el resultado actualizado. Así demuestra cómo recorrer y modificar elementos en listas anidadas (matrices) en Python.

8. MATRIZ 3



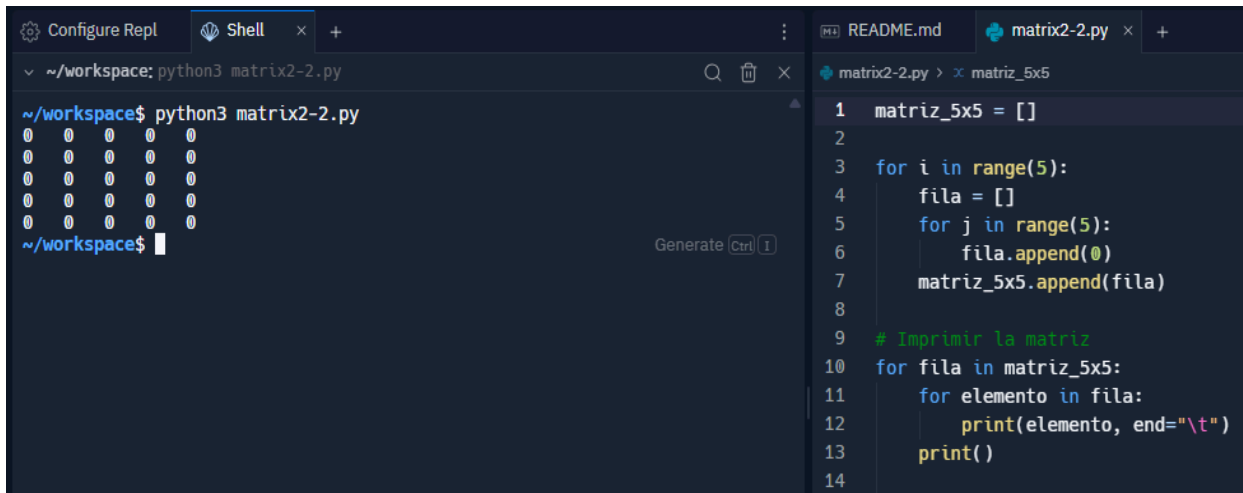
The screenshot shows a code editor with two panels. The left panel displays the terminal output of running `python3 matrix2.py`. The output shows a 3x3 matrix of numbers (0, 2, 3; 4, 5, 6; 7, 8, 9) with elements separated by tabs. The right panel shows the source code of `matrix2.py`, which defines a 3x3 matrix and prints it using `end="\t"` to format the output as a table.

```
~/workspace$ python3 matrix2.py
0 2 3
4 5 6
7 8 9
~/workspace$
```

```
1 matriz= [
2     [0, 2, 3],
3     [4, 5, 6],
4     [7, 8, 9]
5 ]
6 for fila in matriz:
7     for elemento in fila:
8         print(elemento, end="\t")
9     print()
10
```

Este código imprime una matriz de números en formato tabulado, recorriendo cada fila y mostrando sus elementos separados por tabulaciones para facilitar su lectura en forma de tabla.

9. MATRIZ 4

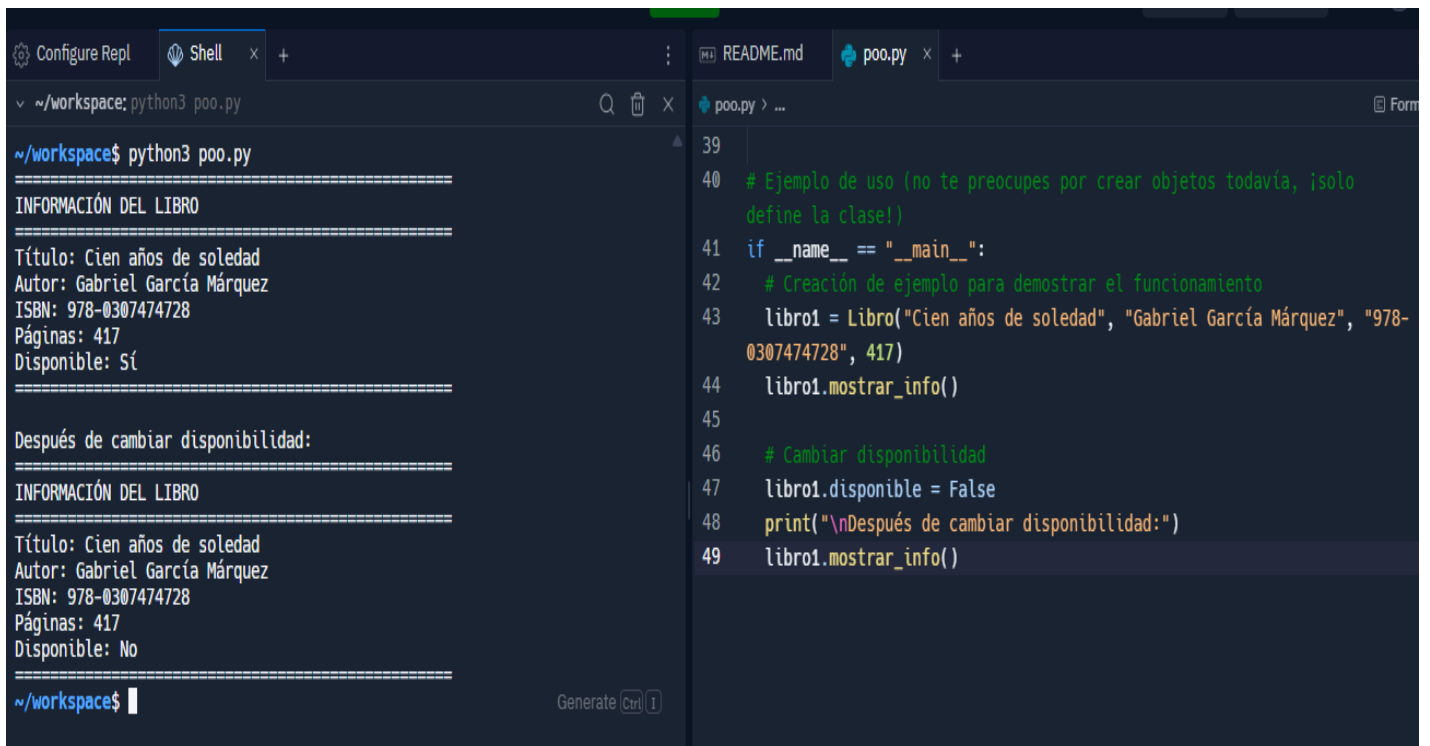


```
~/workspace: python3 matrix2-2.py
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
~/workspace$
```

```
1 matriz_5x5 = []
2
3 for i in range(5):
4     fila = []
5     for j in range(5):
6         fila.append(0)
7     matriz_5x5.append(fila)
8
9 # Imprimir la matriz
10 for fila in matriz_5x5:
11     for elemento in fila:
12         print(elemento, end="\t")
13     print()
14
```

Este código crea una matriz 5x5 llena de ceros usando bucles anidados para construir cada fila y luego la imprime en formato tabulado, mostrando claramente su estructura como una tabla de 5 filas y 5 columnas.

10. POO 1



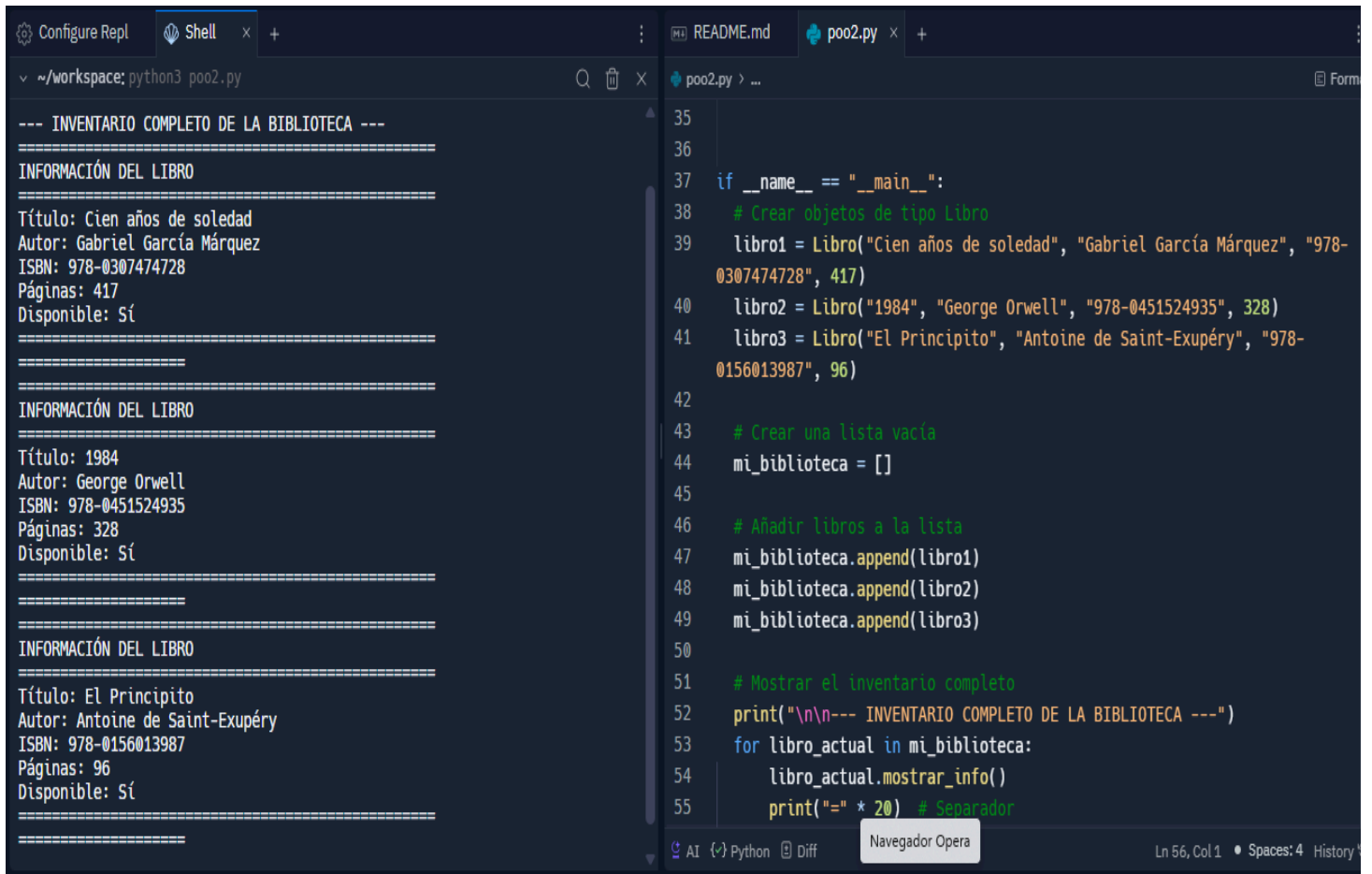
```
~/workspace: python3 poo.py
=====
INFORMACIÓN DEL LIBRO
=====
Título: Cien años de soledad
Autor: Gabriel García Márquez
ISBN: 978-0307474728
Páginas: 417
Disponible: Sí
=====

Después de cambiar disponibilidad:
=====
INFORMACIÓN DEL LIBRO
=====
Título: Cien años de soledad
Autor: Gabriel García Márquez
ISBN: 978-0307474728
Páginas: 417
Disponible: No
=====
~/workspace$
```

```
39
40 # Ejemplo de uso (no te preocupes por crear objetos todavía, solo
41   define la clase!)
42
43 if __name__ == "__main__":
44     # Creación de ejemplo para demostrar el funcionamiento
45     libro1 = Libro("Cien años de soledad", "Gabriel García Márquez", "978-
46               0307474728", 417)
47     libro1.mostrar_info()
48
49     # Cambiar disponibilidad
50     libro1.disponible = False
51     print("\nDespués de cambiar disponibilidad:")
52     libro1.mostrar_info()
```

Este código define una clase Libro que modela un libro con atributos como título, autor, ISBN, número de páginas y disponibilidad. Incluye un constructor para inicializar estos datos y un método mostrar_info que imprime toda la información del libro de forma clara y ordenada. Además, muestra un ejemplo práctico creando un libro, mostrando su información, cambiando su estado de disponibilidad y volviendo a mostrar los datos actualizados.

11. POO 2



The screenshot shows a code editor with two panels. The left panel displays the output of a Python script in a terminal window, showing a library inventory. The right panel shows the source code of the script, `poo2.py`.

Terminal Output (Left Panel):

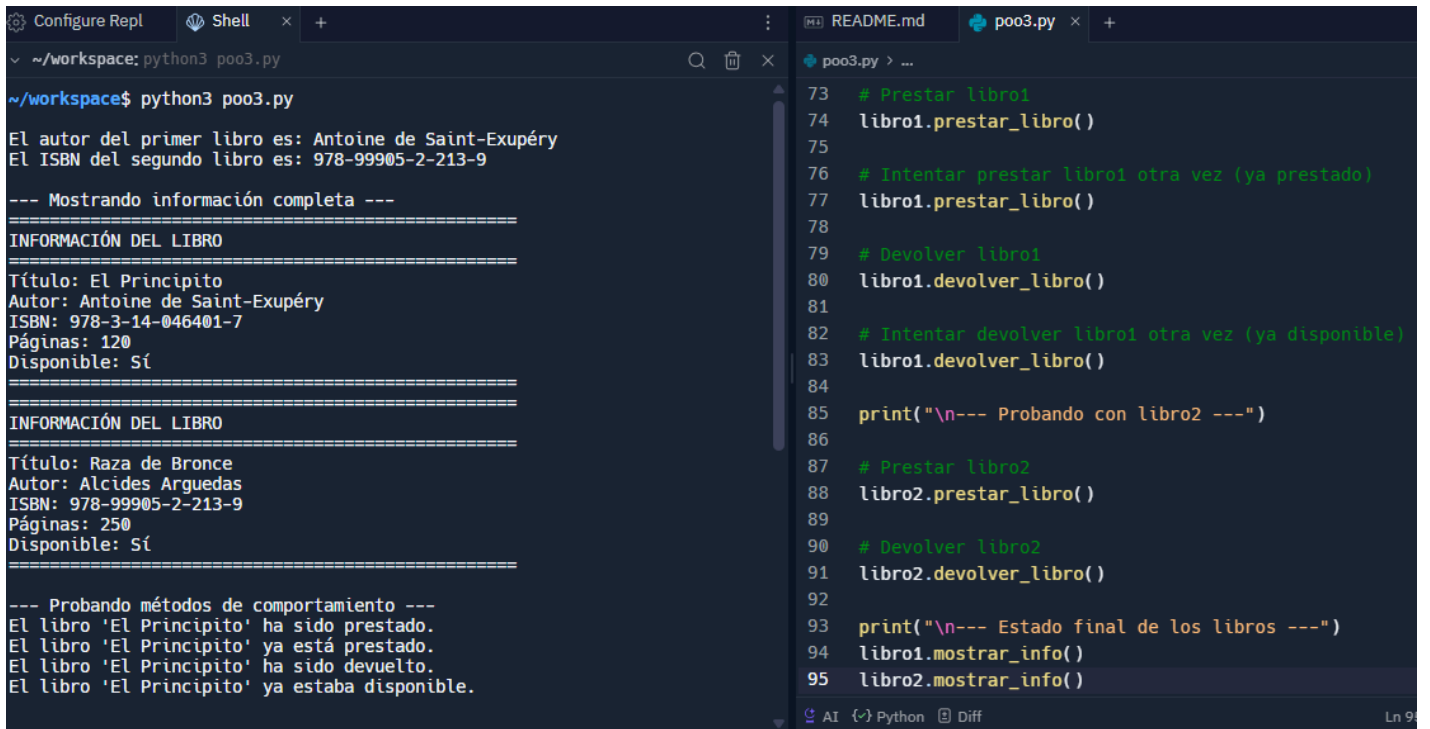
```
--- INVENTARIO COMPLETO DE LA BIBLIOTECA ---  
=====  
INFORMACIÓN DEL LIBRO  
=====  
Título: Cien años de soledad  
Autor: Gabriel García Márquez  
ISBN: 978-0307474728  
Páginas: 417  
Disponible: Sí  
=====  
INFORMACIÓN DEL LIBRO  
=====  
Título: 1984  
Autor: George Orwell  
ISBN: 978-0451524935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

Source Code (Right Panel):

```
35  
36  
37 if __name__ == "__main__":  
38     # Crear objetos de tipo Libro  
39     libro1 = Libro("Cien años de soledad", "Gabriel García Márquez", "978-  
0307474728", 417)  
40     libro2 = Libro("1984", "George Orwell", "978-0451524935", 328)  
41     libro3 = Libro("El Principito", "Antoine de Saint-Exupéry", "978-  
0156013987", 96)  
42  
43     # Crear una lista vacía  
44     mi_biblioteca = []  
45  
46     # Añadir libros a la lista  
47     mi_biblioteca.append(libro1)  
48     mi_biblioteca.append(libro2)  
49     mi_biblioteca.append(libro3)  
50  
51     # Mostrar el inventario completo  
52     print("\n\n--- INVENTARIO COMPLETO DE LA BIBLIOTECA ---")  
53     for libro_actual in mi_biblioteca:  
54         libro_actual.mostrar_info()  
55         print("=" * 20) # Separador
```

Este código define la clase `Libro` para representar libros con atributos como título, autor, ISBN, número de páginas y disponibilidad. Luego crea tres objetos `Libro`, los almacena en una lista llamada `mi_biblioteca` y finalmente recorre esa lista para mostrar la información detallada de cada libro usando el método `mostrar_info`. Así se simula un inventario sencillo de una biblioteca.

12. POO 3-1



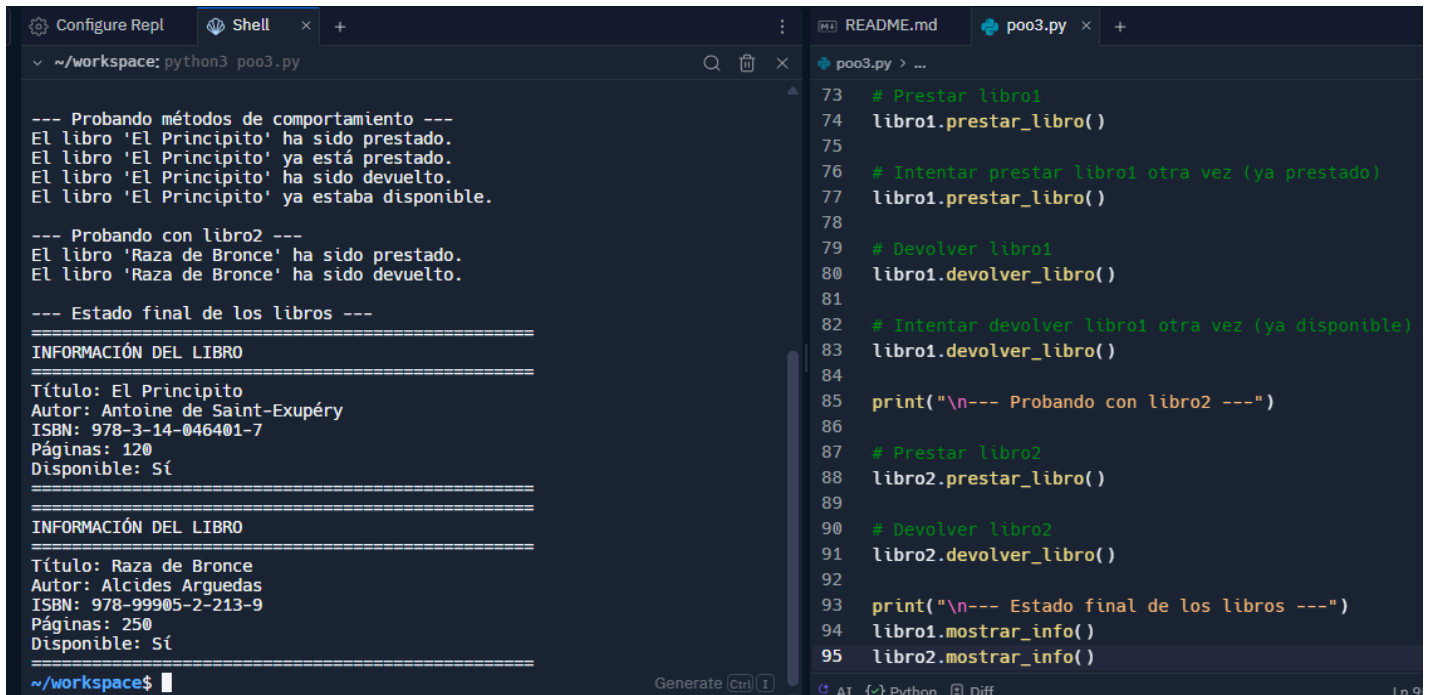
```
~/workspace$ python3 poo3.py
El autor del primer libro es: Antoine de Saint-Exupéry
El ISBN del segundo libro es: 978-99905-2-213-9

--- 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.

73 # Prestar libro1
74 libro1.prestar_libro()
75
76 # Intentar prestar libro1 otra vez (ya prestado)
77 libro1.prestar_libro()
78
79 # Devolver libro1
80 libro1.devolver_libro()
81
82 # Intentar devolver libro1 otra vez (ya disponible)
83 libro1.devolver_libro()
84
85 print("\n--- Probando con libro2 ---")
86
87 # Prestar libro2
88 libro2.prestar_libro()
89
90 # Devolver libro2
91 libro2.devolver_libro()
92
93 print("\n--- Estado final de los libros ---")
94 libro1.mostrar_info()
95 libro2.mostrar_info()
```

POO 3-2



```
--- 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
=====

73 # Prestar libro1
74 libro1.prestar_libro()
75
76 # Intentar prestar libro1 otra vez (ya prestado)
77 libro1.prestar_libro()
78
79 # Devolver libro1
80 libro1.devolver_libro()
81
82 # Intentar devolver libro1 otra vez (ya disponible)
83 libro1.devolver_libro()
84
85 print("\n--- Probando con libro2 ---")
86
87 # Prestar libro2
88 libro2.prestar_libro()
89
90 # Devolver libro2
91 libro2.devolver_libro()
92
93 print("\n--- Estado final de los libros ---")
94 libro1.mostrar_info()
95 libro2.mostrar_info()
```

Este código amplía la clase `Libro` con métodos para prestar y devolver libros, cambiando el estado de disponibilidad. Luego crea dos libros, muestra sus atributos, y prueba los métodos para cambiar y mostrar su estado de préstamo, incluyendo mensajes claros para casos de libros ya prestados o disponibles. Así se simula un sistema básico de gestión de préstamos en una biblioteca. ¿Quieres que te ayude a agregar algo más?

13. SALA CINE

```

Sala actual:
  0  1  2  3  4  5  6  7
F 0 | L  L  L  L  L  L  L  L
F 1 | L  L  L  L  L  L  L  L
F 2 | L  L  L  L  L  L  L  L
F 3 | L  L  L  L  L  L  L  L
F 4 | L  L  L  L  L  L  L  L
Asientos libres: 40

Menú:
1. Ocupar asiento individual
2. Buscar y ocupar N asientos juntos
0. Salir
Elige una opción: 1
Fila: 3
Columna: 5
Asiento (3, 5) reservado por Bs. 50

Sala actual:
  0  1  2  3  4  5  6  7
F 0 | L  L  L  L  L  L  L  L
F 1 | L  L  L  L  L  L  L  L
F 2 | L  L  L  L  L  L  L  L
F 3 | L  L  L  L  L  0  L  L
F 4 | L  L  L  L  L  L  L  L
Asientos libres: 39

```

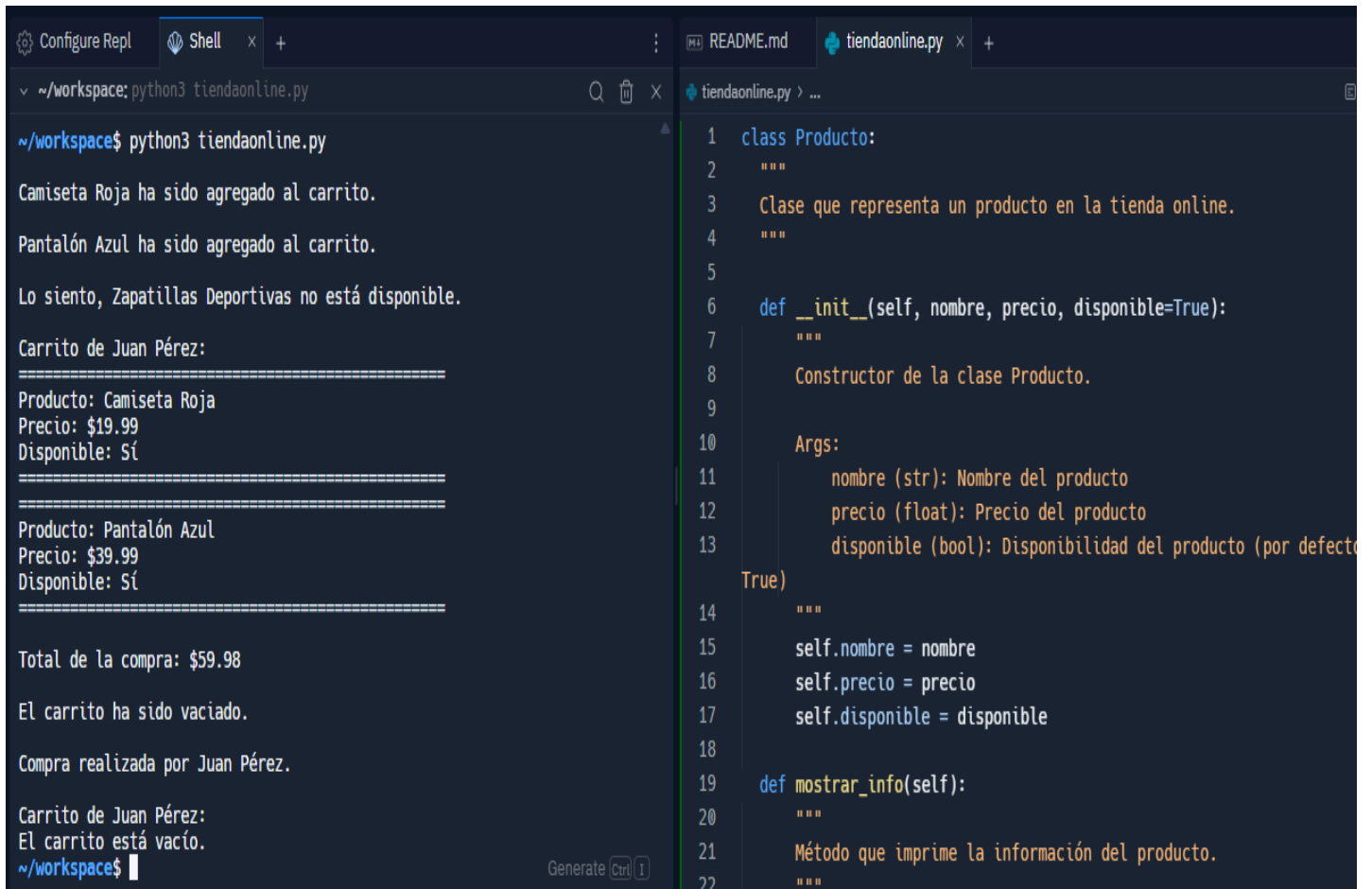
```

10         else:
11             precio = 30 # Asientos de los costados
12             fila.append({"estado": "L", "precio": precio})
13             sala.append(fila)
14         return sala
15
16 # Mostrar la sala con precios y estados
17 def mostrar_sala(sala):
18     print("\n      + " ".join(f"{j:^5}" for j in range(len(sala[0])))
19     print("      + " ".join("-" * 5 for _ in range(len(sala[0])))
20     for i, fila in enumerate(sala):
21         estado_fila = " ".join(f"{a['estado']:^5}" for a in fila)
22         print(f"F{i:>2} | {estado_fila}")
23
24 # Ocupar asiento individual
25 def ocupar_asiento(sala, fila, columna):
26     if 0 <= fila < len(sala) and 0 <= columna < len(sala[0]):
27         asiento = sala[fila][columna]
28         if asiento["estado"] == "L":
29             asiento["estado"] = "0"
30             print(f"Asiento ({fila}, {columna}) reservado por Bs.
31                   {asiento['precio']}")
31             return True
32     else:

```

Este programa simula un sistema de reserva para una sala de cine utilizando una matriz que representa los asientos disponibles, donde cada asiento tiene asignado un estado (libre u ocupado) y un precio que varía según su ubicación, otorgando un costo más alto a los asientos centrales y más bajo a los de los costados. El usuario puede visualizar fácilmente el estado actual de todos los asientos, incluyendo su disponibilidad y posición, y tiene la opción de reservar asientos individuales o grupos contiguos de asientos, siempre que estén libres. El sistema realiza una comprobación automática para garantizar que los asientos solicitados estén disponibles antes de confirmar la reserva, e informa al usuario del costo total de los asientos reservados. Además, mantiene actualizado el conteo de asientos libres para proporcionar un control efectivo sobre la ocupación de la sala. Esta solución permite gestionar la asignación de asientos de forma clara y eficiente, facilitando tanto la administración del cine como la experiencia del usuario al momento de seleccionar su lugar para la función.

14. TIENDA ONLINE



```
~/workspace: python3 tiendaonline.py

Camiseta Roja ha sido agregado al carrito.

Pantalón Azul ha sido agregado al carrito.

Lo siento, Zapatillas Deportivas no está disponible.

Carrito de Juan Pérez:
=====
Producto: Camiseta Roja
Precio: $19.99
Disponible: Sí
=====
Producto: Pantalón Azul
Precio: $39.99
Disponible: Sí
=====

Total de la compra: $59.98

El carrito ha sido vaciado.

Compra realizada por Juan Pérez.

Carrito de Juan Pérez:
El carrito está vacío.
~/workspace$
```

```
class Producto:
    """
    Clase que representa un producto en la tienda online.
    """

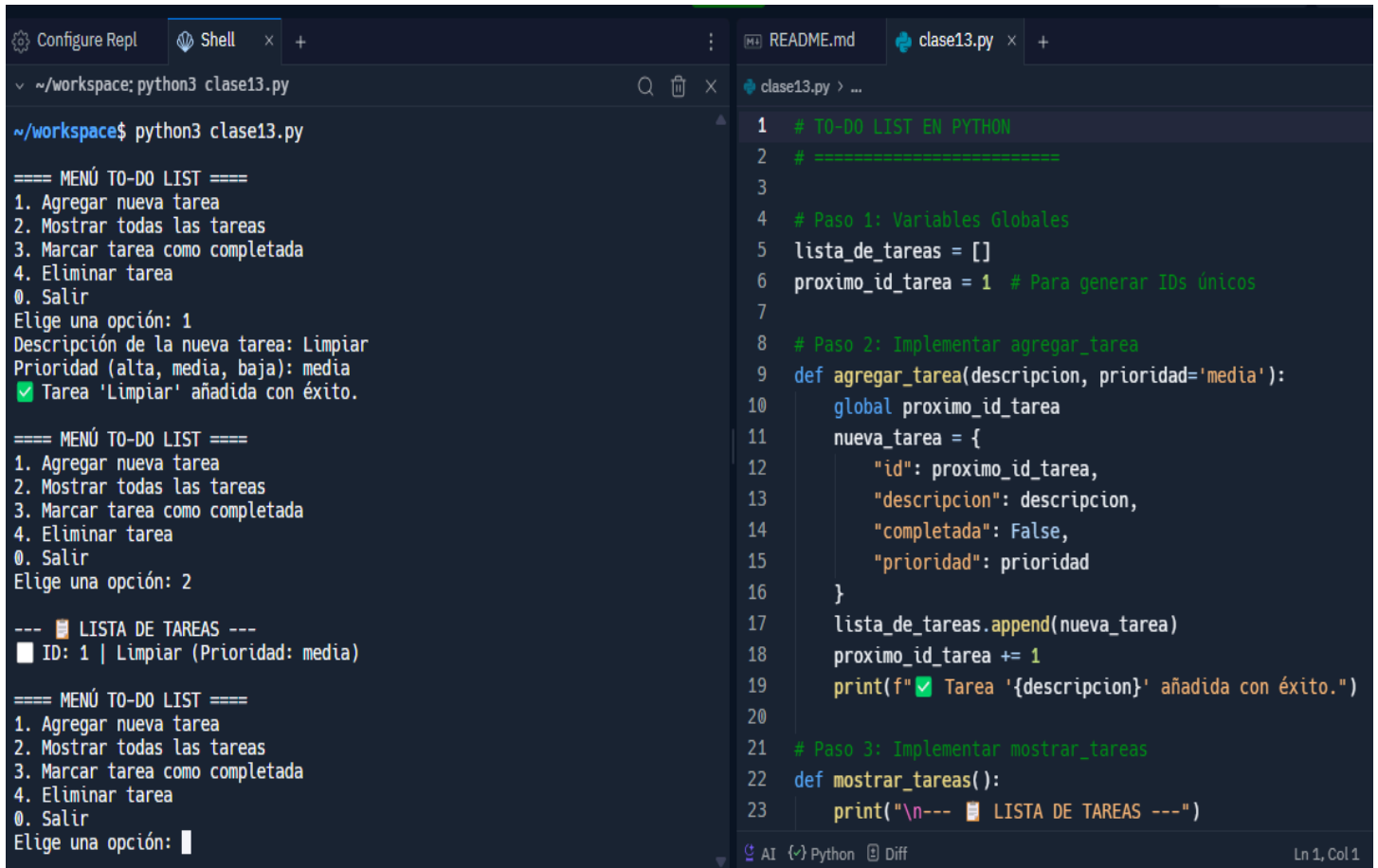
    def __init__(self, nombre, precio, disponible=True):
        """
        Constructor de la clase Producto.

        Args:
            nombre (str): Nombre del producto
            precio (float): Precio del producto
            disponible (bool): Disponibilidad del producto (por defecto
            True)
        """
        self.nombre = nombre
        self.precio = precio
        self.disponible = disponible

    def mostrar_info(self):
        """
        Método que imprime la información del producto.
        """
```

Este programa simula una tienda online con clases para productos, clientes y carritos de compra. Los productos tienen nombre, precio y disponibilidad. Los clientes pueden agregar productos disponibles a su carrito, ver el contenido, calcular el total y realizar compras que vacían el carrito. El sistema controla que no se agreguen productos no disponibles y muestra mensajes claros sobre las acciones realizadas. Así se modela de forma sencilla el proceso básico de compra en línea.

15. TO DO LIST



```
~/workspace$ python3 clase13.py

==== 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: Limpiar
Prioridad (alta, media, baja): media
✅ Tarea 'Limpiar' 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

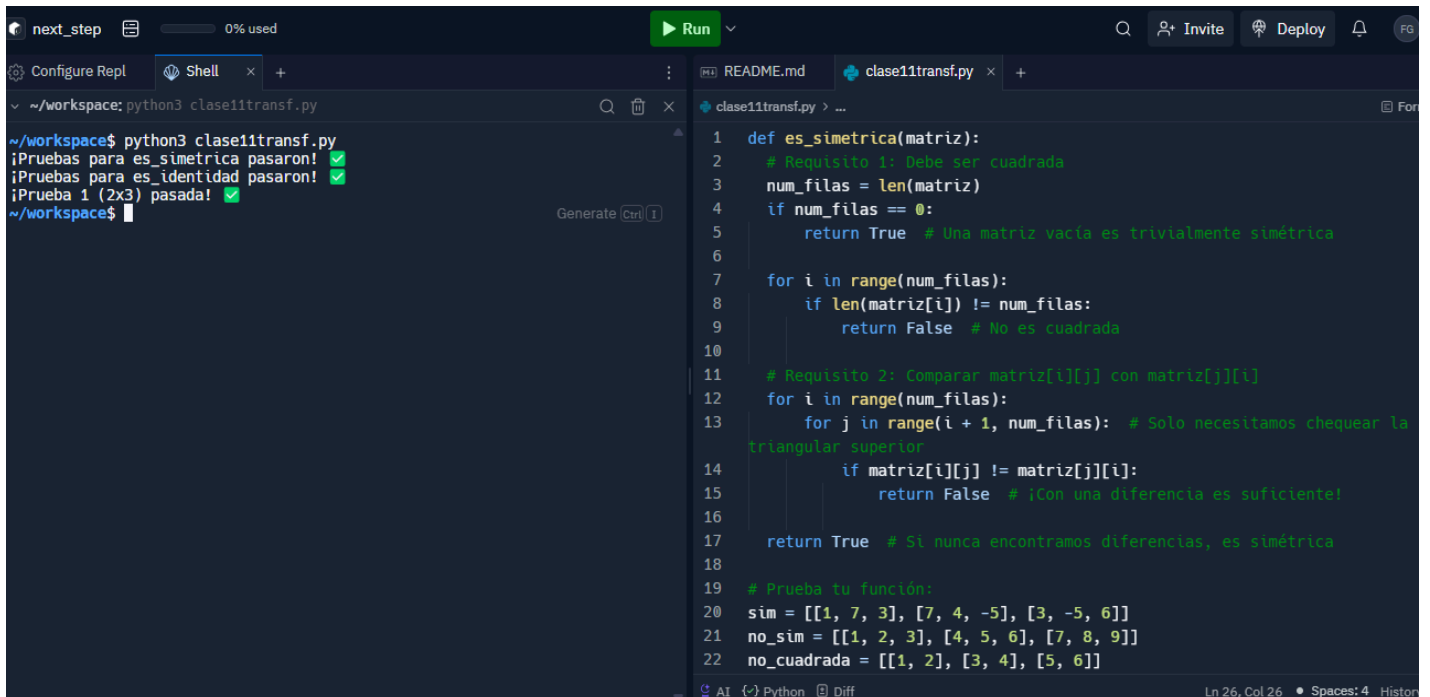
--- 📅 LISTA DE TAREAS ---
■ ID: 1 | Limpiar (Prioridad: media)

==== 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 # TO-DO LIST EN PYTHON
2 # =====
3
4 # Paso 1: Variables Globales
5 lista_de_tareas = []
6 proximo_id_tarea = 1 # Para generar IDs únicos
7
8 # Paso 2: Implementar agregar_tarea
9 def agregar_tarea(descripcion, prioridad='media'):
10     global proximo_id_tarea
11     nueva_tarea = {
12         "id": proximo_id_tarea,
13         "descripcion": descripcion,
14         "completada": False,
15         "prioridad": prioridad
16     }
17     lista_de_tareas.append(nueva_tarea)
18     proximo_id_tarea += 1
19     print(f"✅ Tarea '{descripcion}' añadida con éxito.")
20
21 # Paso 3: Implementar mostrar_tareas
22 def mostrar_tareas():
23     print("\n--- 📅 LISTA DE TAREAS ---")
```

Este programa implementa una lista de tareas (To-Do List) en Python, permitiendo agregar tareas con prioridad, mostrar todas las tareas junto con su estado (completada o pendiente), marcar tareas como completadas y eliminar tareas por su ID. Cada tarea tiene un ID único para facilitar su manejo. El programa funciona con un menú interactivo que permite al usuario elegir qué acción realizar hasta que decida salir.

16. TRANSFORMACIONES



The screenshot displays a code editor with a terminal window on the left and a Python script on the right. The terminal shows the execution of a script named `clase11transf.py`, which performs several tests on a matrix. The script defines a function `es_simetrica` to check if a matrix is symmetric. The tests passed, as indicated by the green checkmarks in the terminal output.

```
~/workspace$ python3 clase11transf.py
¡Pruebas para es_simetrica pasaron! ✓
¡Pruebas para es_identidad pasaron! ✓
¡Prueba 1 (2x3) pasada! ✓
~/workspace$
```

```
1 def es_simetrica(matriz):
2     # Requisito 1: Debe ser cuadrada
3     num_filas = len(matriz)
4     if num_filas == 0:
5         return True # Una matriz vacía es trivialmente simétrica
6
7     for i in range(num_filas):
8         if len(matriz[i]) != num_filas:
9             return False # No es cuadrada
10
11     # Requisito 2: Comparar matriz[i][j] con matriz[j][i]
12     for i in range(num_filas):
13         for j in range(i + 1, num_filas): # Solo necesitamos chequear la
14             triangular superior
15             if matriz[i][j] != matriz[j][i]:
16                 return False # ¡Con una diferencia es suficiente!
17     return True # Si nunca encontramos diferencias, es simétrica
18
19 # Prueba tu función:
20 sim = [[1, 7, 3], [7, 4, -5], [3, -5, 6]]
21 no_sim = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
22 no_cuadrada = [[1, 2], [3, 4], [5, 6]]
```

El código define tres funciones para trabajar con matrices en Python: verificar si una matriz es simétrica, verificar si es una matriz identidad y obtener la matriz transpuesta. Cada función valida primero que la matriz sea cuadrada cuando es necesario, y luego realiza las comprobaciones o transformaciones correspondientes.