

# UNIVERSIDAD PRIVADA DOMINGO SAVIO



## “PRÁCTICO 2 PROGRAMACIÓN”

**MATERIA:** PROGRAMACIÓN 2

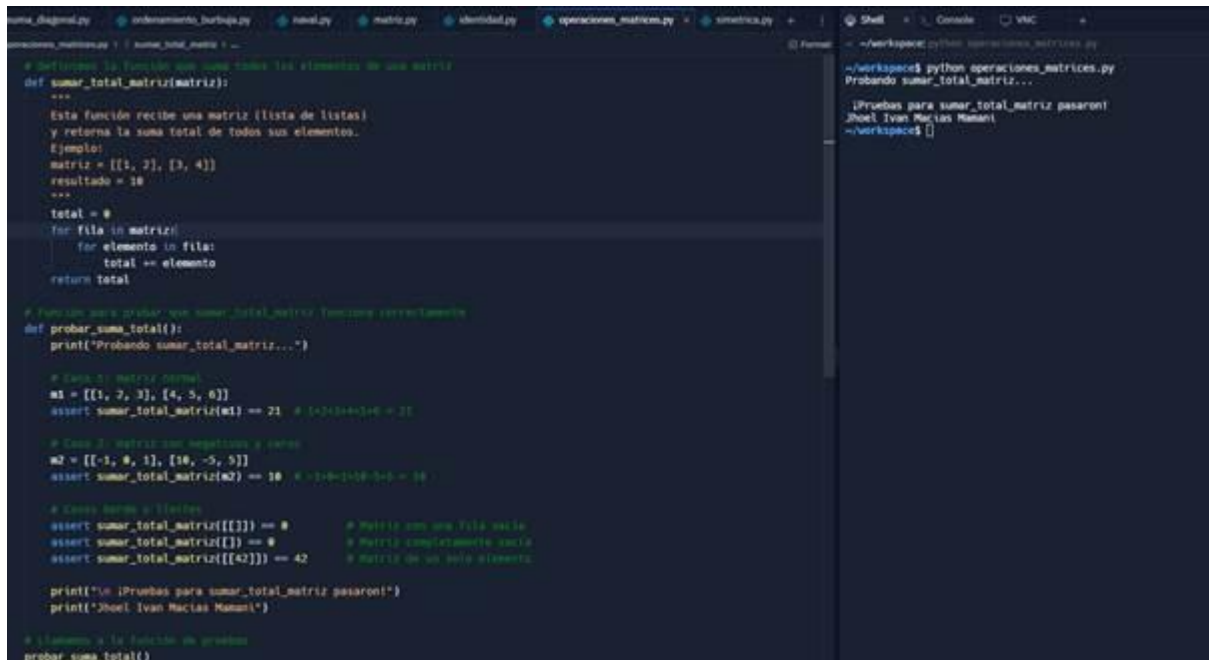
**DOCENTE:** ING. JIMMY NATANIEL REQUENA  
LLORENTTY

**ALUMNOS:**

FLAVIO CESAR ROJAS VARGAS

JHOEL IVAN MACIAS MAMANI

## EJERCICIO 1: suma de valores



```
# Definimos la función que suma todos los elementos de una matriz
def sumar_total_matriz(matriz):
    """
    Esta función recibe una matriz (lista de listas)
    y retorna la suma total de todos sus elementos.
    Ejemplo:
    matriz = [[1, 2], [3, 4]]
    resultado = 10
    """
    total = 0
    for fila in matriz:
        for elemento in fila:
            total += elemento
    return total

# Función para probar que sumar_total_matriz funciona correctamente
def probar_suma_total():
    print("Probando sumar_total_matriz...")

    # Caso 1: matriz normal
    m1 = [[1, 2, 3], [4, 5, 6]]
    assert sumar_total_matriz(m1) == 21 # 1+2+3+4+5+6 = 21

    # Caso 2: matriz con negativos y ceros
    m2 = [[-1, 0, 1], [10, -5, 5]]
    assert sumar_total_matriz(m2) == 10 # -1+0+1+10-5+5 = 10

    # Casos borde o límites
    assert sumar_total_matriz([[]]) == 0 # Matriz con una fila vacía
    assert sumar_total_matriz([]) == 0 # Matriz completamente vacía
    assert sumar_total_matriz([[42]]) == 42 # Matriz de un solo elemento

    print("\n ¡Pruebas para sumar_total_matriz pasaron! ")
    print("Joel Ivan Macias Mamani")

# llamamos a la función de pruebas
probar_suma_total()
```

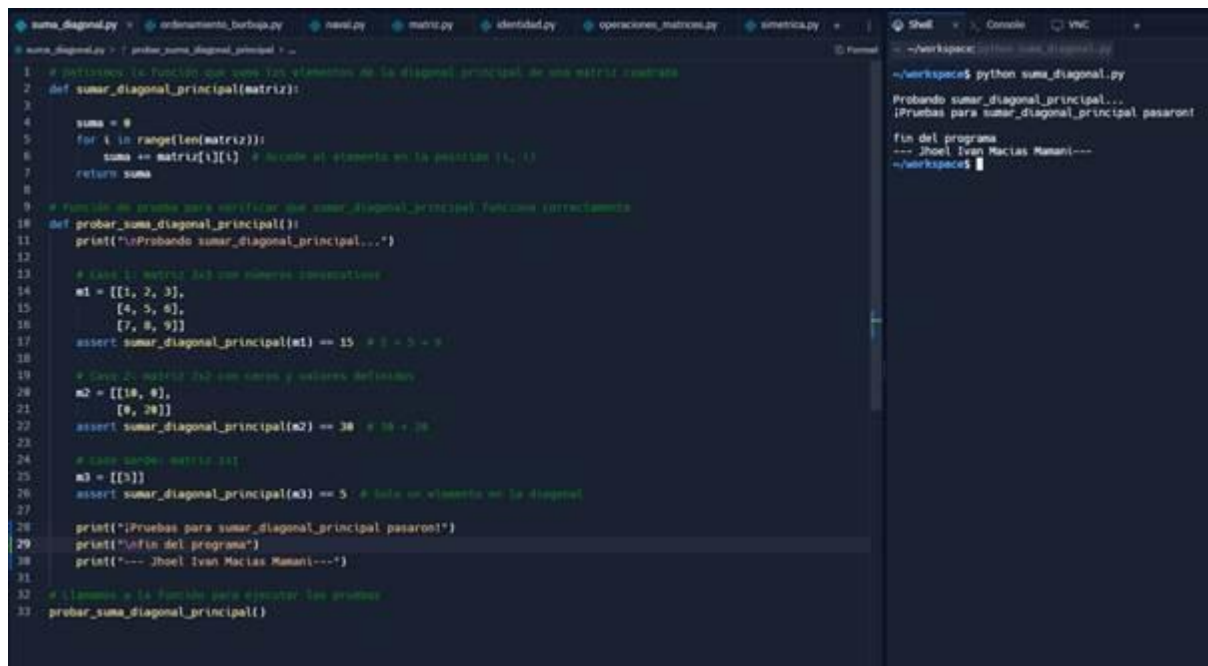
```
~/workspace: python operaciones_matrices.py
~/workspace$ python operaciones_matrices.py
Probando sumar_total_matriz...

¡Pruebas para sumar_total_matriz pasaron!
Joel Ivan Macias Mamani
~/workspace$
```

Este código suma todos los números dentro de una matriz (una lista de listas) y verifica que funcione bien usando ejemplos.

Fecha y hora actual: 2025-06-24 16:20:48

## EJERCICIO 2: diagonal principal



```
suma_diagonal.py | ordenamiento_burbuja.py | naval.py | matriz.py | identidad.py | operaciones_matrices.py | simetrica.py | Shell | Console | VNC |
suma_diagonal.py > probar_suma_diagonal_principal > ...

1 # Definir la función que suma los elementos de la diagonal principal de una matriz cuadrada
2 def sumar_diagonal_principal(matriz):
3
4     suma = 0
5     for i in range(len(matriz)):
6         suma += matriz[i][i] # Accede al elemento en la posición (i, i)
7     return suma
8
9 # Función de prueba para verificar que sumar_diagonal_principal funciona correctamente
10 def probar_suma_diagonal_principal():
11     print("\nProbando sumar_diagonal_principal...")
12
13     # Caso 1: Matriz 3x3 con números consecutivos
14     m1 = [[1, 2, 3],
15           [4, 5, 6],
16           [7, 8, 9]]
17     assert sumar_diagonal_principal(m1) == 15 # 1 + 5 + 9 = 15
18
19     # Caso 2: Matriz 2x2 con ceros y valores definidos
20     m2 = [[10, 4],
21           [0, 20]]
22     assert sumar_diagonal_principal(m2) == 30 # 10 + 20 = 30
23
24     # Caso simple: Matriz 1x1
25     m3 = [[3]]
26     assert sumar_diagonal_principal(m3) == 3 # Solo un elemento en la diagonal
27
28     print("\nPruebas para sumar_diagonal_principal pasaron!")
29     print("\nfin del programa")
30     print("---- Jhoel Ivan Macias Mamani ----")
31
32 # Llamamos a la función para ejecutar las pruebas
33 probar_suma_diagonal_principal()

~/workspace$ python suma_diagonal.py
Probando sumar_diagonal_principal...
¡Pruebas para sumar_diagonal_principal pasaron!
fin del programa
---- Jhoel Ivan Macias Mamani ----
~/workspace$
```

Esta función recibe una matriz cuadrada (misma cantidad de filas y columnas) y retorna la suma de los elementos en su diagonal principal.

Fecha y hora actual: 2025-06-24 16:41:53

## EJERCICIO 3: diagonal secundaria

```
ejercicio-1 100% used
diagonal_secundaria.py 1 ...
diagonal_secundaria.py
def sumar_diagonal_secundaria(matriz):
    """
    Recibe una matriz cuadrada (misma cantidad de filas y columnas)
    y devuelve la suma de los elementos en la diagonal secundaria.
    La diagonal secundaria va desde la esquina superior derecha
    hasta la esquina inferior izquierda.
    """
    n = len(matriz) # Numero de filas (y columnas)
    suma = 0
    for i in range(n):
        suma += matriz[i][n - 1 - i] # Elemento en la diagonal secundaria
    return suma

# Función de pruebas para verificar que sumar_diagonal_secundaria funcione correctamente
def probar_suma_diagonal_secundaria():
    print("Probando sumar_diagonal_secundaria...")

    # Caso 1: Matriz 3x3
    m1 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
    esperado1 = 15 # 3 + 5 + 7
    resultado1 = sumar_diagonal_secundaria(m1)
    print("Matriz m1:", m1)
    print("Esperado:", esperado1, "Obtenido:", resultado1)
    print("PASO" if resultado1 == esperado1 else "X NO PASO", "\n")

    # Caso 2: Matriz 2x2
    m2 = [[10, 1], [2, 20]]
    esperado2 = 3 # 1 + 2
    resultado2 = sumar_diagonal_secundaria(m2)
    print("Matriz m2:", m2)
    print("Esperado:", esperado2, "Obtenido:", resultado2)
    print("PASO" if resultado2 == esperado2 else "X NO PASO", "\n")

    # Caso 3: Matriz 1x1
    m3 = [[42]]
    esperado3 = 42

~/workspace$ python diagonal_secundaria.py
Probando sumar_diagonal_secundaria...
Matriz m1: [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
Esperado: 15 | Obtenido: 15
PASO
Matriz m2: [[10, 1], [2, 20]]
Esperado: 3 | Obtenido: 3
PASO
Matriz m3: [[42]]
Esperado: 42 | Obtenido: 42
PASO
✓ ¡Todas las pruebas finalizadas!
~/workspace$
```

Este código suma los elementos de la diagonal secundaria de una matriz cuadrada (la que va de la esquina superior derecha a la inferior izquierda). También incluye una función para probar que la suma se hace correctamente con diferentes matrices.

Fecha y hora actual: 2025-06-24 16:51:19

## EJERCICIO 4: teclado numérico

```
1 # Simulamos una matriz que simula un teclado
2 teclado = [
3     [1, 2, 3],
4     [4, 5, 6],
5     [7, 8, 9],
6     [ "*", 0, "# "]
7 ]
8
9 # Simulamos la matriz como cuadrada
10 print("\n MATRIZ DEL TECLADO:\n")
11 for fila in teclado:
12     for elemento in fila:
13         # Simulamos cada elemento con tabulaciones, sin salto de línea
14         print(elemento, end="\t")
15     # Al final de cada fila, hacemos un salto de línea
16     print()
17
18 # Creamos una matriz 5x5 de ceros usando bucles anidados
19 print("\n MATRIZ 5x5 CON CEROS (usando bucles):\n")
20 matriz_5x5 = [] # Creamos una lista para la matriz
21
22 for i in range(5): # Recorremos 5 filas
23     fila = [] # Creamos una nueva fila vacía
24     for j in range(5): # Recorremos 5 columnas
25         fila.append(0) # Agregamos un cero a la fila
26     matriz_5x5.append(fila) # Agregamos la fila completa a la matriz
27
28 # Simulamos la matriz como cuadrada
29 for fila in matriz_5x5:
30     for elemento in fila:
31         print(elemento, end="\t")
32     print()
33
34 # Creamos la misma matriz usando comprensión de listas
35 print("\n MATRIZ 5x5 CON CEROS (usando comprensión de listas):\n")
36
37 # Creamos una matriz 5x5 con ceros de forma compacta
38 matriz_compacta = [[0 for j in range(5)] for i in range(5)]
```

```
~/workspace$ python teclado.py
\n MATRIZ DEL TECLADO:
1 2 3
4 5 6
7 8 9
* 0 #
\n
\n MATRIZ 5x5 CON CEROS (usando bucles):
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
\n
\n MATRIZ 5x5 CON CEROS (usando comprensión de listas):
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
\n
EXPLICACIÓN:
matriz_compacta = [[0 for j in range(5)] for i in range(5)]
↳ Parte interna: [0 for j in range(5)] - crea una fila con cinco ceros
↳ Parte externa: for i in range(5) - repite esa fila cinco veces
Resultado: Una matriz de 5x5 completamente llena de ceros.
\n
fin del Programa
---Shael Ivan Paeles Mamari---
~/workspace$
```

Este código crea e imprime matrices en Python. Primero, simula un teclado con una matriz de 4 filas y 3 columnas. Luego, genera una matriz de 5x5 llena de ceros usando bucles anidados y también usando comprensión de listas de forma más compacta. Finalmente, explica cómo funciona la comprensión de listas para crear la matriz.

Fecha y hora actual: 2025-06-24 16:54:29

## EJERCICIO 5: batalla naval

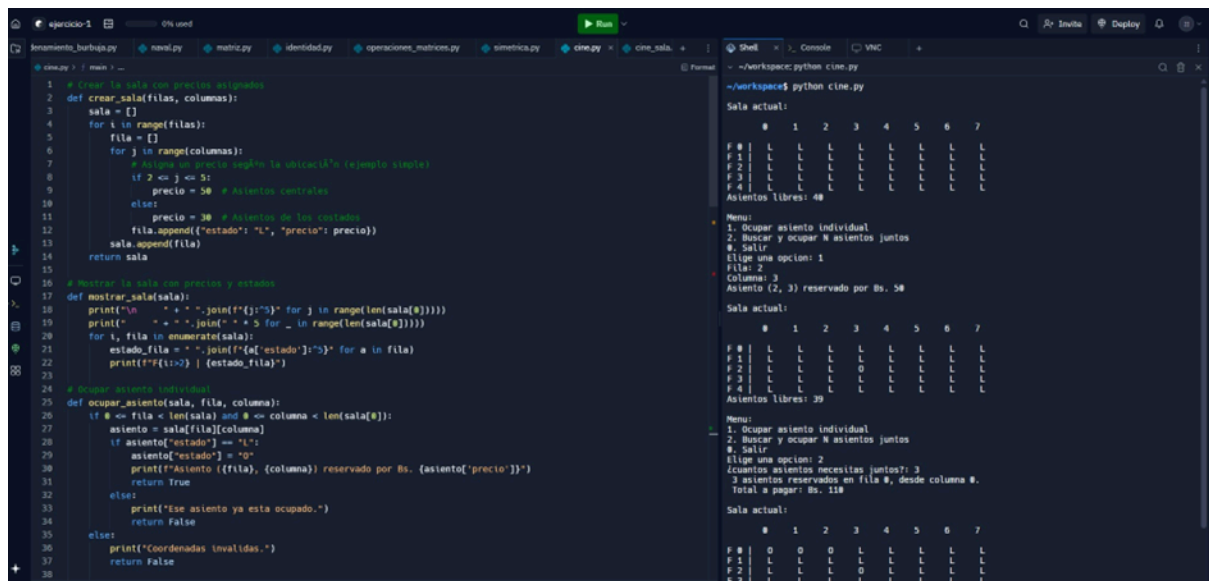
```
1 import random
2 import json
3
4 TAM, NUM_BARCOS, INTENTOS = 5, 3, 10
5 LETRAS = ['A', 'B', 'C', 'D', 'E']
6
7 # Función para crear un tablero vacío
8 def crear_tablero(): return [['-'] * TAM for _ in range(TAM)]
9
10 # Función para mostrar el tablero
11 def mostrar(tablero):
12     print("  " + " ".join(str(i+1) for i in range(TAM)))
13     for i, fila in enumerate(tablero):
14         print(f"{LETRAS[i]} " + " ".join(fila))
15
16 # Función para colocar los barcos en el tablero
17 def colocar_barcos(tablero):
18     for _ in range(NUM_BARCOS):
19         while True:
20             f, c = random.randint(0, TAM-1), random.randint(0, TAM-1)
21             if tablero[f][c] != '-':
22                 tablero[f][c] = 'B'
23                 break
24
25 # Función para pedir la coordenada al jugador
26 def pedir_coord(nombre):
27     while True:
28         coord = input(f"{nombre}, coordenada (A1-E5): ").upper()
29         if len(coord) != 2 and coord[0] in LETRAS and coord[1:].isdigit():
30             f, c = LETRAS.index(coord[0]), int(coord[1:]) - 1
31             if 0 <= c < TAM: return f, c
32             print("X Coordenada inválida.")
33
34 # Función para realizar un disparo
35 def disparo(nombre, visible, oculto):
36     mostrar(visible)
37     f, c = pedir_coord(nombre)
38     if visible[f][c] != '-':
```

```
91
92     print(f"🟢 (n1): {pts[n1]} | 🟡 (n2): {pts[n2]}")
93     if pts[n1] > pts[n2]: print(f"🏆 ¡Gana (n1)!")
94     elif pts[n1] > pts[n2]: print(f"🏆 ¡Gana (n2)!")
95     else: print(f"🤝 ¡Empate!")
96     for i in range(2):
97         print(f"🚢 Barcos de {(n2, n1)[i]}:")
98         revelar(vis[i], occ[i-1])
99         mostrar(vis[i])
100
101     # Guardar los puntajes
102     guardar_puntajes(n1, n2, pts)
103
104 else:
105     nombre = input("Tu nombre: ")
106     vis, occ = crear_tablero(), crear_tablero()
107     colocar_barcos(occ)
108     aciertos = 0
109
110     for _ in range(INTENTOS):
111         aciertos += disparo(nombre, vis, occ)
112         if aciertos == NUM_BARCOS: break
113
114     if aciertos == NUM_BARCOS:
115         print(f"🏆 ¡Hundiste todos los barcos!")
116     else:
117         print(f"🔴 Sin intentos. Perdiste.")
118         print(f"📍 Posiciones reales:")
119         revelar(vis, occ)
120         mostrar(vis)
121
122     # Guardar el puntaje
123     guardar_puntajes(nombre, [aciertos])
124
125 if __name__ == "__main__":
126     jugar()
```

En el código se implementó un módulo **JSON** (JavaScript Object Notation) que nos permite almacenar datos en este caso de los jugadores y puntajes. Otro módulo también **random** que nos genera valores aleatorios en este caso los barcos.

Fecha y hora actual: 2025-06-24 17:07:45

## EJERCICIO 6: gestión de sala de cine

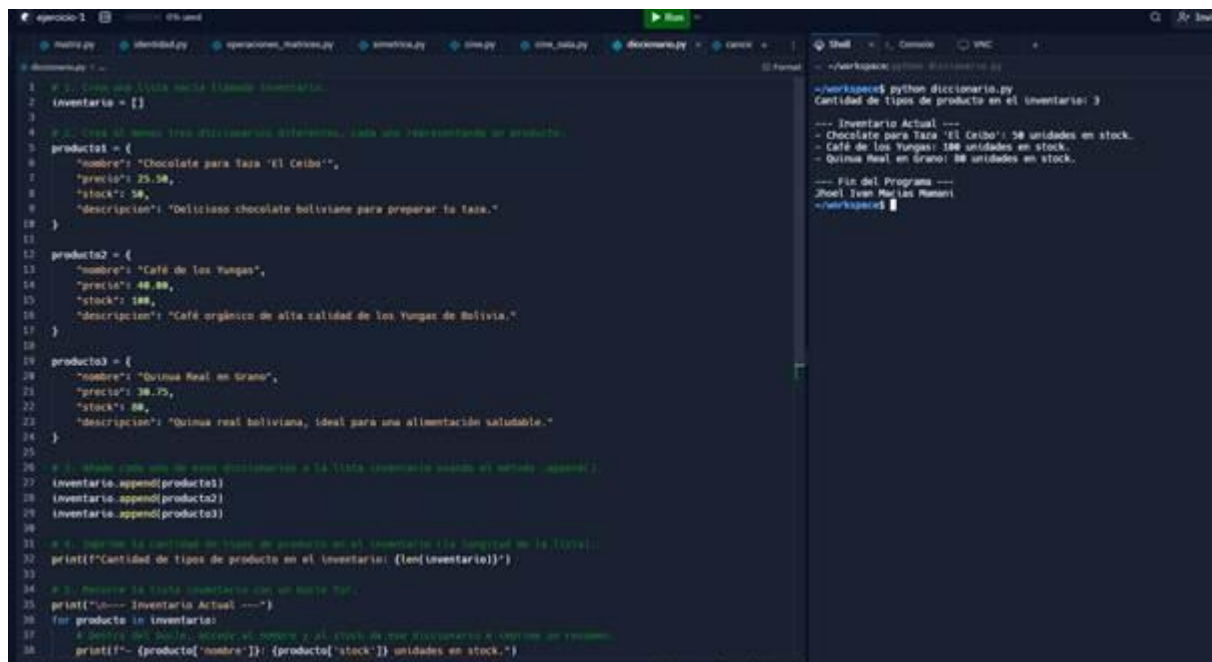


Se crea una matriz (lista de listas) que representa los asientos de una sala de cine. Cada asiento es un diccionario que guarda dos datos: "estado", que puede ser "L"(asientos) si está libre o "O" si está ocupado, y "precio", que puede ser 30 o 50 Bs dependiendo de su ubicación (costados o centro). El programa muestra la sala en forma de tabla y nos permite interactuar con un menú donde puede reservar un asiento individual, buscar y reservar varios asientos juntos en una fila, ver cuántos asientos libres quedan o salir del sistema.

Fecha y hora actual: 2025-06-24 17:26:34



## EJERCICIO 7: diccionarios



```
ejercicio7.py | 1 | # 1. Creamos una lista vacía llamada inventario.
2 | inventario = []
3 |
4 | # 2. Creamos tres diccionarios diferentes, cada uno representando un producto.
5 | producto1 = {
6 |     "nombre": "Chocolate para Taza "El Ceibo",
7 |     "precio": 25.50,
8 |     "stock": 50,
9 |     "descripcion": "Delicioso chocolate boliviano para preparar la taza."
10 | }
11 |
12 | producto2 = {
13 |     "nombre": "Café de los Yungas",
14 |     "precio": 40.00,
15 |     "stock": 100,
16 |     "descripcion": "Café orgánico de alta calidad de los Yungas de Bolivia."
17 | }
18 |
19 | producto3 = {
20 |     "nombre": "Quinoa Real en Grano",
21 |     "precio": 30.75,
22 |     "stock": 80,
23 |     "descripcion": "Quinoa real boliviana, ideal para una alimentación saludable."
24 | }
25 |
26 | # 3. Agregamos cada uno de esos diccionarios a la lista inventario usando el método .append().
27 | inventario.append(producto1)
28 | inventario.append(producto2)
29 | inventario.append(producto3)
30 |
31 | # 4. Imprimos la cantidad de tipos de producto en el inventario (la longitud de la lista).
32 | print(f"Cantidad de tipos de producto en el inventario: {len(inventario)}")
33 |
34 | # 5. Recorremos la lista inventario con un bucle for.
35 | print("\n--- Inventario Actual ---")
36 | for producto in inventario:
37 |     # Dentro del bucle, accedemos al nombre y al stock de cada diccionario e imprimos su resumen.
38 |     print(f"~ {producto['nombre']}: {producto['stock']} unidades en stock.")
39 |
40 | # Fin del Programa
41 | Jhoel Ivan Morales Mamani
42 | ~~~~~
```

```
<~/workspace> python diccionario.py
Cantidad de tipos de producto en el inventario: 3

--- Inventario Actual ---
- Chocolate para Taza "El Ceibo": 50 unidades en stock.
- Café de los Yungas: 100 unidades en stock.
- Quinoa Real en Grano: 80 unidades en stock.

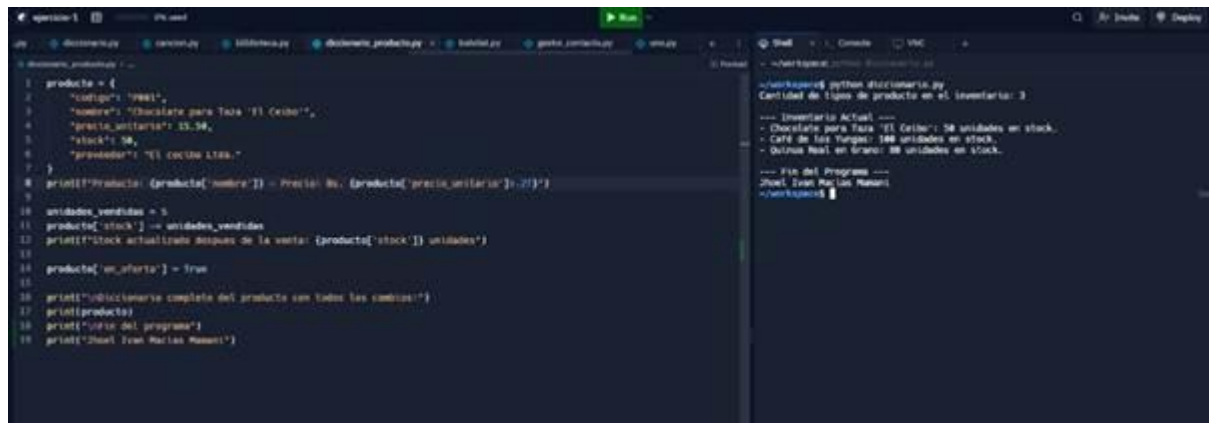
--- Fin del Programa ---
Jhoel Ivan Morales Mamani
~/workspace
```

Los productos se agregan a la lista usando `.append()`. Luego, se imprime la cantidad total de productos y se recorre la lista con un `for` para mostrar el stock disponible de cada uno.

Fecha y hora actual: 2025-06-24 17:58:51



## EJERCICIO 8: inventario



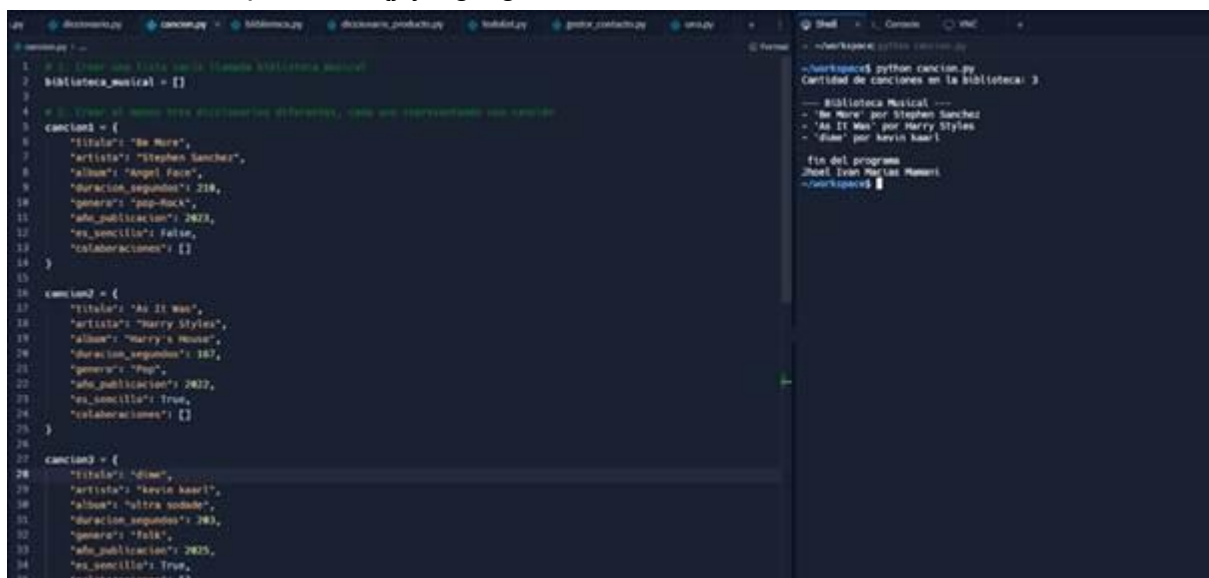
```
1 producto = {
2     "codigo": "9881",
3     "nombre": "Chocolate para Taza 'El Ceibo'",
4     "precio_unitario": 15.58,
5     "stock": 58,
6     "proveedor": "El Cuchito Ltda."
7 }
8 print(f"Producto: {producto['nombre']} - Precio: Bs. {producto['precio_unitario']*1.2}")
9
10 unidades_vendidas = 5
11 producto['stock'] -= unidades_vendidas
12 print(f"Stock actualizado despues de la venta: {producto['stock']} unidades")
13
14 producto['en_oferta'] = True
15
16 print(f"Diccionario completo del producto con todos los cambios:")
17 print(producto)
18 print("Fin del programa")
19 print("Jhon Ivan Nicolas Mamani")
```

```
~/workspace python diccionario_producto.py
Cantidad de tipo de producto en el inventario: 3

--- Inventario Actual ---
- Chocolate para Taza 'El Ceibo': 58 unidades en stock.
- Café de las Yungas: 100 unidades en stock.
- Quesos Real en Grano: 20 unidades en stock.

--- Fin del Programa ---
Jhon Ivan Nicolas Mamani
~/workspace
```

este código nos imprime el inventario usando el método .append() creamos un módulo llamado producto {} y agregamos los datos



```
1 # 1. Crear una lista con la llamada Biblioteca Musical
2 biblioteca_musical = []
3
4 # 2. Crear el nombre de las canciones diferentes, como una representación con canciones
5 canciones = [
6     "titulo": "Be More",
7     "artista": "Stephen Sanchez",
8     "album": "Angel Face",
9     "duracion_segundos": 218,
10    "genero": "Pop-Rock",
11    "año_publicacion": 2023,
12    "es_sencillo": False,
13    "colaboraciones": []
14 ]
15
16 cancion2 = {
17     "titulo": "As It Was",
18     "artista": "Harry Styles",
19     "album": "Harry's House",
20     "duracion_segundos": 187,
21     "genero": "Pop",
22     "año_publicacion": 2022,
23     "es_sencillo": True,
24     "colaboraciones": []
25 }
26
27 cancion3 = {
28     "titulo": "Dime",
29     "artista": "Kevin Kool",
30     "album": "Altra sodade",
31     "duracion_segundos": 203,
32     "genero": "Folk",
33     "año_publicacion": 2025,
34     "es_sencillo": True,
35     "colaboraciones": []
36 }
```

```
~/workspace python cancion.py
Cantidad de canciones en la biblioteca: 3

--- Biblioteca Musical ---
- 'Be More' por Stephen Sanchez
- 'As It Was' por Harry Styles
- 'Dime' por Kevin Kool

fin del programa
Jhon Ivan Nicolas Mamani
~/workspace
```

Este programa es una lista de canciones realizadas en clases creamos módulos llamado cancion1, cancion2, cancion3, donde agregamos los datos de los valores de cada canción.

Fecha y hora actual: 2025-06-24 18:04:02

## EJERCICIO 9: kanban tareas

```
import tkinter as tk
from tkinter import simpledialog, messagebox
import json
import os
import sys
import pandas as pd

# Detectar carpeta del ejecutable o script
BASE_DIR = os.path.dirname(sys.executable if getattr(sys, 'frozen', False) else __file__)
ARCHIVO_TAREAS = os.path.join(BASE_DIR, "tareas_guardadas.json")
ARCHIVO_EXCEL = os.path.join(BASE_DIR, "tareas_exportadas.xlsx")

class KanbanApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Kanban To-Do List")
        self.root.geometry("850x540")
        self.tareas = {"Por Hacer": [], "En Progreso": [], "Completado": []}
        self.columnas = {}
        self._crear_columnas()
        self._crear_botones_superiores()
        self._cargar_tareas()

    def _crear_columnas(self):
        contenedor = tk.Frame(self.root)
        contenedor.pack(fill='both', expand=True)
        for nombre in ["Por Hacer", "En Progreso", "Completado"]:
            frame = tk.Frame(contenedor, bg="#eeeeee", width=250, height=450, bd=2,
                             relief='groove')
            frame.pack(side='left', padx=10, pady=10, fill='both', expand=True)
            titulo = tk.Label(frame, text=nombre, bg="#dddddd", font=('Arial', 12,
                             'bold'))
            titulo.pack(fill='x')
            self.columnas[nombre] = frame

    def _crear_botones_superiores(self):
        barra = tk.Frame(self.root)
        barra.pack(pady=5)
```

Instalamos la librería de pandas con el código **pip install pandas** y luego verificamos si se instaló correctamente con el código **pip show pandas**.

```
BASE_DIR = os.path.dirname(sys.executable if getattr(sys, 'frozen', False) else __file__)
ARCHIVO_TAREAS = os.path.join(BASE_DIR, "tareas_guardadas.json")
ARCHIVO_EXCEL = os.path.join(BASE_DIR, "tareas_exportadas.xlsx")

class KanbanApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Kanban To-Do List")
        self.root.geometry("850x540")
        self.tareas = {"Por Hacer": [], "En Progreso": [], "Completado": []}
        self.columnas = {}
        self._crear_columnas()
        self._crear_botones_superiores()
        self._cargar_tareas()

    def _crear_columnas(self):
        contenedor = tk.Frame(self.root)
        contenedor.pack(fill='both', expand=True)
        for nombre in ["Por Hacer", "En Progreso", "Completado"]:
            frame = tk.Frame(contenedor, bg="#eeeeee", width=250, height=450, bd=2,
                             relief='groove')
            frame.pack(side='left', padx=10, pady=10, fill='both', expand=True)
            titulo = tk.Label(frame, text=nombre, bg="#dddddd", font=('Arial', 12,
                             'bold'))
            titulo.pack(fill='x')
            self.columnas[nombre] = frame

    def _crear_botones_superiores(self):
        barra = tk.Frame(self.root)
        barra.pack(pady=5)
```

```
~/workspace$ python kanban.py
Traceback (most recent call last):
  File "/home/runner/workspace/kanban.py", line 6, in <module>
    import pandas as pd
ModuleNotFoundError: No module named 'pandas'

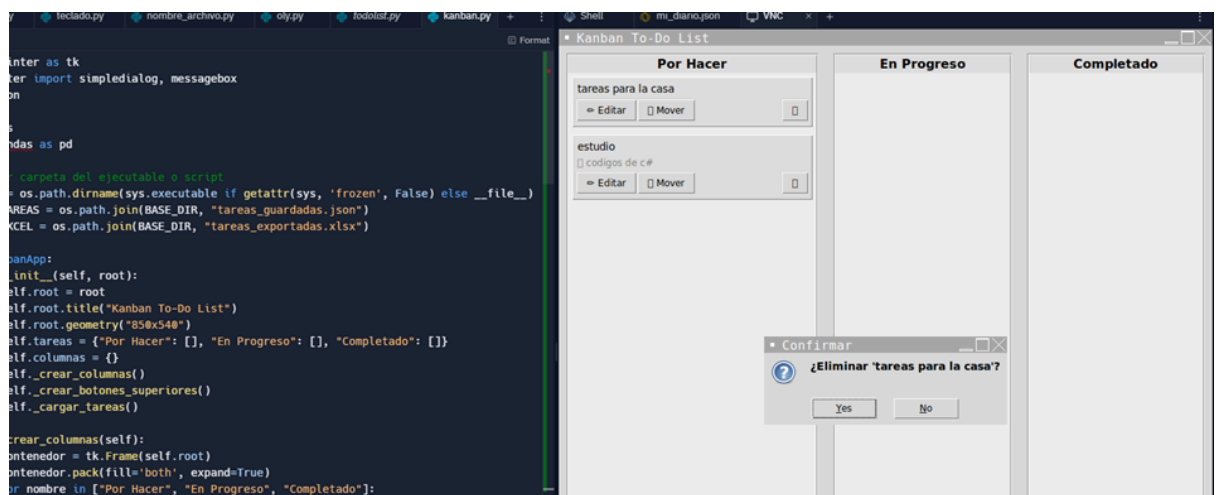
~/workspace$ pip install pandas
Collecting pandas
  Downloading pandas-2.3.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (9
)
Collecting numpy=>1.23.2 (from pandas)
  Downloading numpy-2.3.1-cp311-cp311-manylinux_2_28_x86_64.whl.metadata (62 kB)
Collecting python-dateutil=>2.8.2 (from pandas)
  Downloading python_dateutil-2.9.0.post0-py2.py3-none-any.whl.metadata (8.4 kB)
Collecting pytz=>2020.1 (from pandas)
  Downloading pytz-2025.2-py2.py3-none-any.whl.metadata (22 kB)
Collecting tzdata=>2022.7 (from pandas)
  Downloading tzdata-2025.2-py2.py3-none-any.whl.metadata (1.4 kB)
Collecting six=>1.5 (from python-dateutil->2.8.2->pandas)
  Downloading six-1.17.0-py2.py3-none-any.whl.metadata (1.7 kB)
Downloading pandas-2.3.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (12.4 MB)
  12.4/12.4 MB 75.3 MB/s eta 0:00:00
Downloading numpy-2.3.1-cp311-cp311-manylinux_2_28_x86_64.whl (16.9 MB)
  16.9/16.9 MB 75.5 MB/s eta 0:00:00
Downloading python_dateutil-2.9.0.post0-py2.py3-none-any.whl (229 kB)
  229.9/229.9 kB 13.6 MB/s eta 0:00:00
Downloading pytz-2025.2-py2.py3-none-any.whl (509 kB)
  509.2/509.2 kB 32.6 MB/s eta 0:00:00
Downloading tzdata-2025.2-py2.py3-none-any.whl (347 kB)
  347.8/347.8 kB 21.2 MB/s eta 0:00:00
Downloading six-1.17.0-py2.py3-none-any.whl (11 kB)
Installing collected packages: pytz, tzdata, six, numpy, python-dateutil, pandas
Successfully installed numpy-2.3.1 pandas-2.3.0 python-dateutil-2.9.0.post0 pytz-2025.2

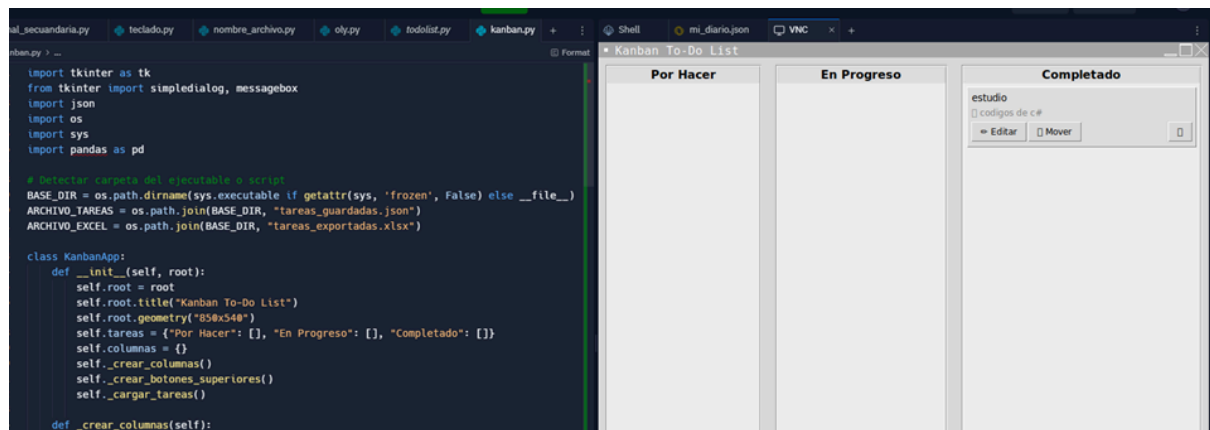
~/workspace$ pip show pandas
Name: pandas
Version: 2.3.0
Summary: Powerful data structures for data analysis, time series, and statistics
Home-page:
Author:
Author-email: The Pandas Development Team <pandas-dev@python.org>
License: BSD 3-Clause License

Copyright (c) 2008-2011, AQR Capital Management, LLC, Lambda Foundry, Inc. and PyDa
All rights reserved.

Copyright (c) 2011-2023, Open source contributors.

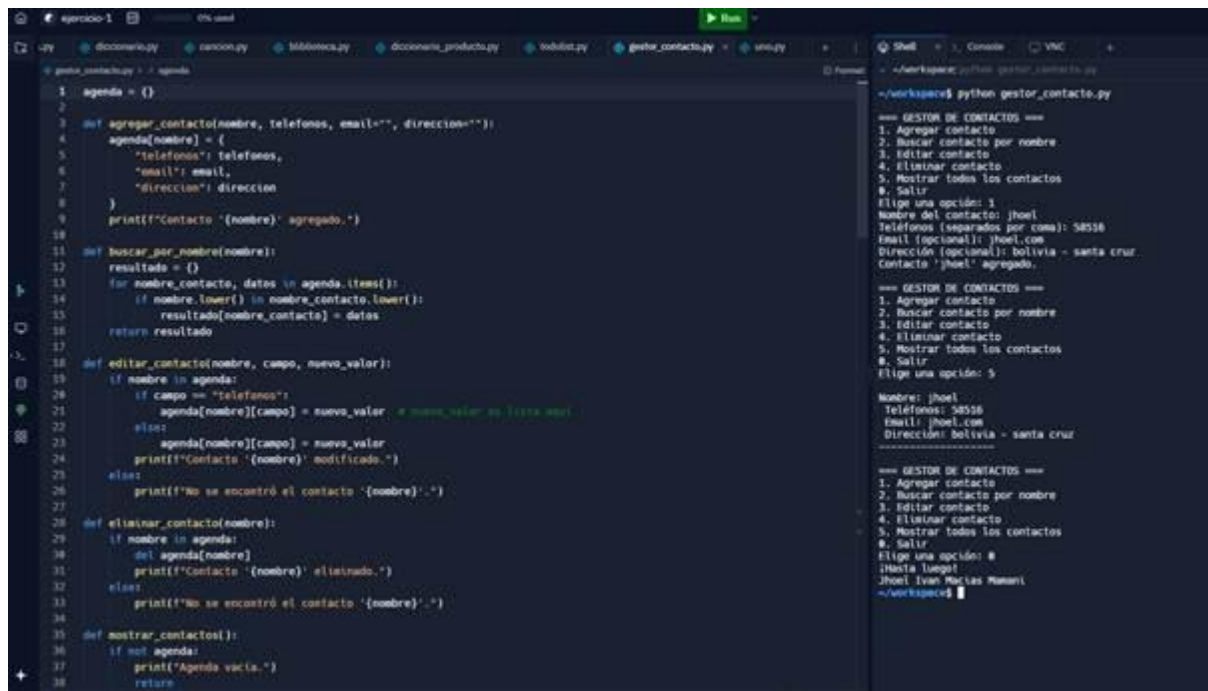
Redistribution and use in source and binary forms, with or without
```





Fecha y hora actual: 2025-06-24 17:55:29

## EJERCICIO 10: agenda



```
1 agenda = {}
2
3 def agregar_contacto(nombre, telefonos, email="", direccion=""):
4     agenda[nombre] = {
5         "telefonos": telefonos,
6         "email": email,
7         "direccion": direccion
8     }
9     print(f"Contacto '{nombre}' agregado.")
10
11 def buscar_por_nombre(nombre):
12     resultado = {}
13     for nombre_contacto, datos in agenda.items():
14         if nombre.lower() in nombre_contacto.lower():
15             resultado[nombre_contacto] = datos
16     return resultado
17
18 def editar_contacto(nombre, campo, nuevo_valor):
19     if nombre in agenda:
20         if campo == "telefonos":
21             agenda[nombre][campo] = nuevo_valor + ", " + nuevo_valor + ", lista nueva"
22         else:
23             agenda[nombre][campo] = nuevo_valor
24         print(f"Contacto '{nombre}' modificado.")
25     else:
26         print(f"No se encontró el contacto '{nombre}'.")
27
28 def eliminar_contacto(nombre):
29     if nombre in agenda:
30         del agenda[nombre]
31         print(f"Contacto '{nombre}' eliminado.")
32     else:
33         print(f"No se encontró el contacto '{nombre}'.")
34
35 def mostrar_contactos():
36     if not agenda:
37         print("Agenda vacía.")
38     return
```

```
~/workspace$ python gestor_contacto.py
== GESTOR DE CONTACTOS ==
1. Agregar contacto
2. Buscar contacto por nombre
3. Editar contacto
4. Eliminar contacto
5. Mostrar todos los contactos
6. Salir
Elige una opción: 1
Nombre del contacto: jhoel
Teléfonos (separados por coma): 58556
Email (opcional): jhoel.com
Dirección (opcional): bolivia - santa cruz
Contacto 'jhoel' agregado.
== GESTOR DE CONTACTOS ==
1. Agregar contacto
2. Buscar contacto por nombre
3. Editar contacto
4. Eliminar contacto
5. Mostrar todos los contactos
6. Salir
Elige una opción: 5
Nombre: jhoel
Teléfono: 58556
Email: jhoel.com
Dirección: bolivia - santa cruz
=====
== GESTOR DE CONTACTOS ==
1. Agregar contacto
2. Buscar contacto por nombre
3. Editar contacto
4. Eliminar contacto
5. Mostrar todos los contactos
6. Salir
Elige una opción: 6
¡Hasta luego!
jhoel ivan marcos mamani
~/workspace$
```

Este programa permite guardar y gestionar contactos usando un diccionario llamado agenda. Cada contacto tiene nombre, lista de teléfonos, correo y dirección. El usuario puede agregar, buscar, editar, eliminar y ver todos los contactos desde un menú interactivo en consola.

Fecha y hora actual: 2025-06-24 18:08:13

## EJERCICIO 11: Mi diario

```
import json
import os
from datetime import datetime # Para obtener fecha y hora

# Nombre del archivo JSON
nombre_archivo = "mi_diario.json"

# Crear archivo inicial con entradas base (si no existe)
entradas_iniciales = [
    "Querido diario,",
    "Hoy aprendí sobre archivos en Python.",
    "El modo 'w' borra todo antes de escribir. ¡Qué miedo!"
]

# Si el archivo no existe, lo creamos con entradas iniciales
if not os.path.exists(nombre_archivo):
    with open(nombre_archivo, 'w') as f:
        json.dump(entradas_iniciales, f, indent=4, ensure_ascii=False)
    print("✅ Diario creado con entradas iniciales.")
else:
    print("❌ Diario ya existente encontrado.")

# Leer y mostrar contenido actual
print("\n--- Contenido actual del diario ---")
try:
    with open(nombre_archivo, 'r') as f:
        entradas = json.load(f)
        for entrada in entradas:
            print(f"- {entrada}")
except (FileNotFoundError, json.JSONDecodeError):
    print("❌ No se pudo leer el archivo.")

# Permitir añadir nuevas entradas
print("\n🔥 Escribe tus entradas. Escribe 'SALIR' para finalizar.\n")

while True:
    nueva_entrada = input()
    if nueva_entrada.strip().upper() == "SALIR":
```

```
~/workspaces$ python diario.py
✅ Diario creado con entradas iniciales.

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

🔥Escribe tus entradas. Escribe 'SALIR' para finalizar.

hola probando el código de mi diario en replit
salir

--- Contenido final del diario ---
- Querido diario,
- Hoy aprendí sobre archivos en Python.
- El modo 'w' borra todo antes de escribir. ¡Qué miedo!
- [2025-06-27 00:48:06] hola probando el código de mi diario en replit
~/workspaces$
```

Añadimos el formato JSON que nos permitirá guardar los datos escritos y el datetime que nos imprimirá la hora y fecha del mensaje escrito, también añadimos una entrada que al escribir salir nos cierre y guarde los datos.

Fecha y hora actual: 2025-06-26 21:11:51