

# UNIVERSIDAD PRIVADA DOMINGO SAVIO



## Actividad 02

**DOCENTE:** Jimmy N. Requena Llorentty

**TURNO:** Mañana

**CARRERA:** Ing. En Sistemas

**MATERIA:** Programación II

**ESTUDIANTE:** Franz Almanza Galindo

**Fecha y hora actual:** 2025-06-24 18:35:42

**Santa Cruz - Bolivia**

Evidencia visual de códigos correctamente implementados y breve descripción.

### Captura #1 Ordenamiento Burbuja.

```
1 #Codigo Ordenamiento Burbuja
2 def ordenamiento_burbuja(lista):
3
4     n = len(lista) # Cantidad de elementos en la lista
5
6     for i in range(n - 1): # Bucle exterior para las pasadas
7         hubo_intercambio = False # Marca si hubo un intercambio en esta pasada
8
9         # Bucle interior para las comparaciones e intercambios
10        for j in range(n - 1 - i): # Cada pasada evita revisar los últimos ya
ordenados
11            if lista[j] > lista[j + 1]:
12                # ¡Intercambio!
13                lista[j], lista[j + 1] = lista[j + 1], lista[j]
14                hubo_intercambio = True
15
16        if not hubo_intercambio: # Si no hubo ningún intercambio, la lista ya está
ordenada
17            break
18
19    return lista # Opcional: también se puede omitir
20
```

Antes: [6, 3, 8, 2, 5]  
Después Ordenamiento Burbuja: [2, 3, 5, 6, 8]

--- Ejecutando pruebas con asserts ---  
Caso 1 (Lista desordenada): Exitoso  
Caso 2 (Lista ya ordenada): Exitoso  
Caso 3 (Lista ordenada a la inversa): Exitoso  
Caso 4 (Lista con elementos duplicados): Exitoso  
Caso borde (Lista vacía): Exitoso  
Caso borde (Lista con un solo elemento): Exitoso

Programa realizado por Franz Almanza

#### Utilidad

La característica principal del ordenamiento burbuja es su simplicidad. Es fácil de entender e implementar, lo que lo convierte en una excelente herramienta para iniciarse en los algoritmos de ordenamiento. Sin embargo, para listas grandes, es ineficiente comparado con otros algoritmos más avanzados debido a el tiempo que le toma completar el ordenamiento afectando así el rendimiento.

## Captura #2 Ordenamiento por Inserción.

```
ain.py > | ordenamiento_insercion > ...
#Codigo ordenamiento por insercion
def ordenamiento_insercion(lista):
    for i in range(1, len(lista)):
        valor_actual = lista[i] # La "carta" que vamos a insertar
        posicion_actual = i

        # Desplazar elementos mayores hacia la derecha
        while posicion_actual > 0 and lista[posicion_actual - 1] > valor_actual:
            lista[posicion_actual] = lista[posicion_actual - 1]
            posicion_actual -= 1

        # Insertar la "carta" en su hueco correcto
        lista[posicion_actual] = valor_actual

    return lista

if __name__ == "__main__":
    numeros = [6, 3, 8, 2, 5]
    print("Antes:", numeros)
    ordenamiento_insercion(numeros)
    print("Después Ordenamiento Inserción:", numeros)

    print("\n--- Ejecutando pruebas con asserts ---")

Antes: [6, 3, 8, 2, 5]
Después Ordenamiento Inserción: [2, 3, 5, 6, 8]

--- Ejecutando pruebas con asserts ---
Caso 1 (Lista desordenada): SUCCESS
Caso 2 (Lista ya ordenada): SUCCESS
Caso 3 (Lista ordenada a la inversa): SUCCESS
Caso 4 (Lista con duplicados): SUCCESS
Caso borde (Lista vacía): SUCCESS
Caso borde (Lista con un solo elemento): SUCCESS

Programa realizado por Franz Almanza
```

### Utilidad

El ordenamiento por inserción en Python es útil para listas pequeñas o casi ordenadas. Funciona insertando cada elemento en su posición correcta dentro de la parte ya ordenada del arreglo. Su principal ventaja es su simplicidad y eficiencia para entradas pequeñas, lo que lo hace adecuado para tareas donde el tamaño de los datos es limitado o cuando los datos se reciben de forma incremental. También es estable, lo que significa que el orden relativo de los elementos iguales se mantiene. Sin embargo, su complejidad de tiempo de  $O(n^2)$  lo hace ineficiente para conjuntos de datos grandes.

### Captura #3 Merge Sort.

```
#Codigo para el merge sort
def merge_sort(lista):
    # Paso Vencer (Condición Base de la Recursividad):
    if len(lista) <= 1:
        return lista

    # Paso 1: DIVIDIR
    medio = len(lista) // 2
    mitad_izquierda = lista[:medio]
    mitad_derecha = lista[medio:]

    # Paso 2: VENCER (Recursión)
    izquierda_ordenada = merge_sort(mitad_izquierda)
    derecha_ordenada = merge_sort(mitad_derecha)

    # Paso 3: COMBINAR
    # Comentamos la línea de impresión de mezcla para no saturar la salida con los
    asserts
    # print(f"Mezclaría {izquierda_ordenada} y {derecha_ordenada}")
    return merge(izquierda_ordenada, derecha_ordenada)
```

python3 mai... Ask Assistant ✓

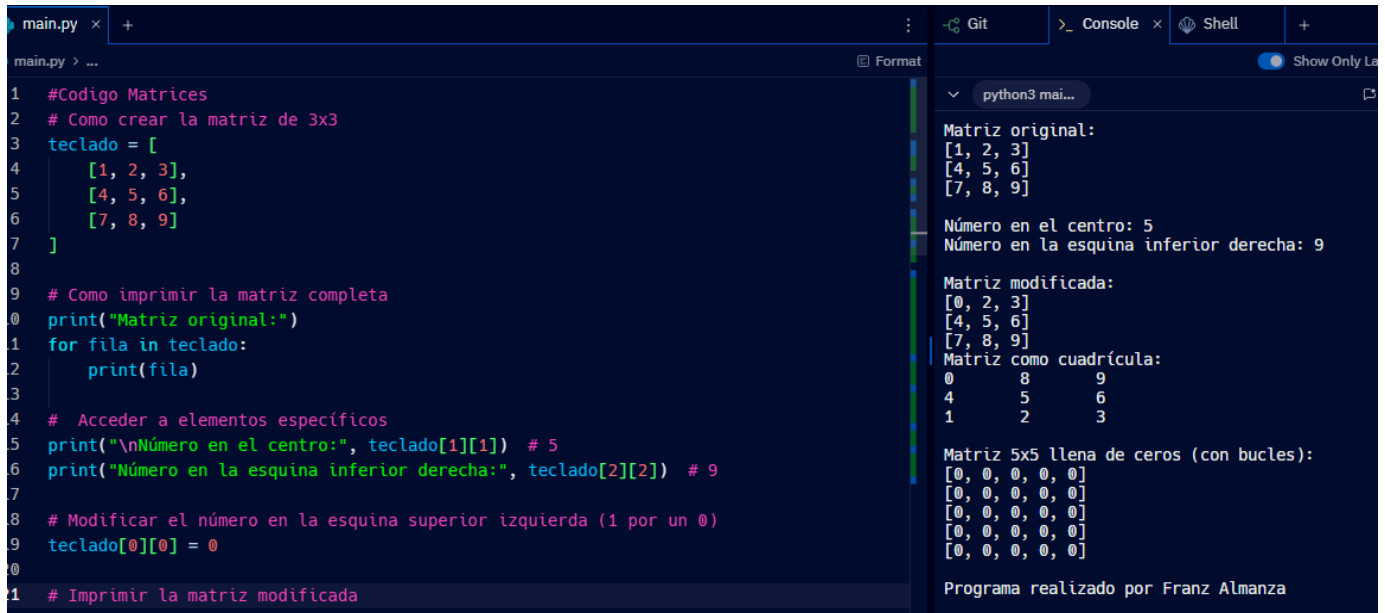
Lista original: [8, 3, 5, 1]  
Lista ordenada: [1, 3, 5, 8]

--- Ejecutando pruebas automatizadas con asserts ---  
Caso 1 (Lista vacía): succes  
Caso 2 (Lista con un solo elemento): succes  
Caso 3 (Lista con dos elementos): succes  
Caso 4 (Lista con tres elementos desordenados): succes  
Caso 5 (Lista par): succes  
Caso 6 (Lista en orden descendente): succes  
Caso 7 (Lista ya ordenada): succes  
Caso 8 (Lista con elementos repetidos): succes  
Caso 9 (Lista con enteros negativos y positivos): succes  
Caso 10 (Lista con flotantes): succes

Programa realizado por Franz Almanza

Utilidad
<p><b>Merge Sort es un algoritmo de ordenamiento altamente eficiente basado en la estrategia "Divide y Vencerás". Su principal utilidad radica en su consistente complejidad temporal de <math>O(n \log n)</math> en el mejor, promedio y peor de los casos, lo que lo hace ideal para ordenar grandes conjuntos de datos. Es un algoritmo estable, manteniendo el orden relativo de elementos iguales.</b></p>

## Captura #4 Matrices, matriz cuadrícula.



```
1 #Codigo Matrices
2 # Como crear la matriz de 3x3
3 teclado = [
4     [1, 2, 3],
5     [4, 5, 6],
6     [7, 8, 9]
7 ]
8
9 # Como imprimir la matriz completa
10 print("Matriz original:")
11 for fila in teclado:
12     print(fila)
13
14 # Acceder a elementos específicos
15 print("\nNúmero en el centro:", teclado[1][1]) # 5
16 print("Número en la esquina inferior derecha:", teclado[2][2]) # 9
17
18 # Modificar el número en la esquina superior izquierda (1 por un 0)
19 teclado[0][0] = 0
20
21 # Imprimir la matriz modificada
22 print("Matriz modificada:")
23 for fila in teclado:
24     print(fila)
```

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

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

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

Matriz como cuadrícula:

0	8	9
4	5	6
1	2	3

Matriz 5x5 llena de ceros (con 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]

Programa realizado por Franz Almanza

### Utilidad

Las matrices en Python son herramientas versátiles que facilitan el manejo de grandes volúmenes de datos numéricos y la ejecución de cálculos matemáticos complejos con alta performance.

**Cálculos científicos y de ingeniería:** Permiten realizar operaciones de álgebra lineal complejas (multiplicación de matrices, inversión, determinantes) de manera rápida y sencilla, crucial para física, química, y otras disciplinas.

**Machine Learning y Data Science:** Son la base para representar conjuntos de datos (filas = ejemplos, columnas = características), pesos de redes neuronales, y transformaciones de datos.

**Procesamiento de imágenes y gráficos:** Las imágenes se representan como matrices de píxeles, y las transformaciones (rotación, escala) se realizan mediante operaciones matriciales.

## Captura #5 Suma total de una matriz.

```
ain.py > f probar_suma_total > ...
# Como sumar todos los elementos de una matriz
def sumar_total_matriz(matriz):

    # 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]]
    try:
        assert sumar_total_matriz(m1) == 21, "Fallo en Caso 1: Matriz normal"
        print("Caso 1 (Matriz normal): PASSED")
    except AssertionError as e:
        print(f"Error: {e}")
```

--- Probando sumar\_total\_matriz ---  
Caso 1 (Matriz normal): PASSED  
Caso 2 (Matriz con negativos y ceros): PASSED  
Caso 3 (Matriz con una fila vacía): PASSED  
Caso 4 (Matriz completamente vacía): PASSED  
Caso 5 (Matriz de un solo elemento): PASSED  
Todas las pruebas para sumar\_total\_matriz han finalizado  
Programa realizado por Franz Almanza

### Utilidad

**Esta función permite calcular la suma de todos los elementos dentro de una matriz o array, o bien, realizar sumas a lo largo de ejes específicos (filas o columnas).**

**Análisis de datos:** Permite obtener rápidamente resúmenes agregados de grandes conjuntos de datos representados como matrices. Por ejemplo, en un dataset donde las filas son observaciones y las columnas son características, se puede calcular la suma total de una característica específica, o la suma de todas las características para una observación particular.

**Cálculos estadísticos:** Es fundamental para calcular métricas como la suma de cuadrados, la suma de productos, y otras operaciones que forman la base de la estadística descriptiva e inferencial.

**Machine Learning y Deep Learning:** En el entrenamiento de modelos, a menudo se necesita sumar los valores de los gradientes, los errores, o los pesos de las capas. La función de suma total facilita estas operaciones de manera eficiente.

**Procesamiento de imágenes:** Las imágenes se representan como matrices de píxeles. Sumar los valores de los píxeles puede ser útil para calcular el brillo total de una imagen o para operaciones de convolución.

**Optimización numérica:** En algoritmos de optimización, la suma de funciones de costo o penalización es una operación común, y la función de suma total proporciona una forma eficiente de realizarla.

## Captura #6 Suma de fila en matriz.

```
main.py > f probar_suma_por_filas > ...  
  
#Codigo para sumar por fila en matriz  
# Definimos la función que suma los elementos por cada fila de la matriz  
def sumar_por_filas(matriz):  
    """  
    Esta función recibe una matriz (lista de listas)  
    y devuelve una lista con la suma de cada fila.  
  
    Ejemplo:  
    matriz = [[1, 2, 3], [4, 5, 6]]  
    resultado = [6, 15]  
    """  
    resultado = []  
    for fila in matriz:  
        suma_fila = sum(fila) # Suma todos los elementos de la fila  
        resultado.append(suma_fila)  
    return resultado  
  
# Función de prueba para verificar que sumar_por_filas funciona correctamente  
def probar_suma_por_filas():  
    print("\n Probando sumar_por_filas")  
  
    # Caso 1: matriz con 3 filas y 3 columnas  
    m1 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

Probando sumar\_por\_filas  
Caso 1 (Matriz 3x3): success  
Caso 2 (Matriz con pares repetidos): success  
Caso 3 (Matriz con números negativos): success  
Caso 4 (Matriz con filas de diferente longitud): success  
Caso 5 (Matriz vacía): success  
Caso 6 (Matriz con filas vacías): success  
Todas las pruebas para sumar\_por\_filas han finalizado  
Programa realizado por Franz Almanza

### Utilidad

Es una operación fundamental que permite calcular la suma de todos los elementos a lo largo de una fila específica (o todas las filas) de una matriz o array multidimensional. **Análisis de Datos y Estadísticas:** En conjuntos de datos tabulares (donde las filas representan registros y las columnas características), sumar las filas puede significar calcular el total de ciertos atributos para cada registro. Por ejemplo, si tienes una matriz de ventas diarias por producto, sumar cada fila te daría el total de ventas para cada día.

#### Machine Learning y Deep Learning:

**Normalización y Scaling:** A menudo se necesita calcular la suma de los valores en una fila para normalizar los datos, asegurando que cada observación (fila) tenga una escala comparable.

**Procesamiento de características:** En el pre procesamiento de datos, puedes sumar ciertos atributos dentro de una fila para crear una nueva característica compuesta.

## Captura #7 Suma diagonal matriz.

```
#Codigo para sumar diagonal en matriz
# Definimos la función que suma los elementos de la diagonal principal de una matriz
cuadrada
def sumar_diagonal_principal(matriz):
    suma = 0
    for i in range(len(matriz)):
        suma += matriz[i][i] # Accede al elemento en la posición (i, i)
    return suma

# Función de prueba para verificar que sumar_diagonal_principal funciona
correctamente
def probar_suma_diagonal_principal():
    print("\nPrueba de sumar diagonal")

    # Caso 1: matriz 3x3 con números consecutivos
    m1 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
    try:
        assert sumar_diagonal_principal(m1) == 15, "Fallo en Caso 1: Matriz 3x3 con
números consecutivos"
        print("Caso 1 (Matriz 3x3): PASSED")
    except AssertionError as e:
        print(f"Error: {e}")
```

Packager Ask Assistant ✓

python3 mai... Ask Assistant ✓

Prueba de sumar diagonal  
Caso 1 (Matriz 3x3): PASSED  
Caso 2 (Matriz 2x2): PASSED  
Caso 3 (Matriz 1x1): PASSED  
Caso 4 (Matriz con números negativos): PASSED  
Caso 5 (Matriz 4x4): PASSED  
Todas las pruebas para sumar\_diagonal\_principal han finaliz  
ado

Programa realizado por Franz Almanza

Utilidad
Es una operación especializada que calcula la suma de los elementos ubicados en la diagonal principal de una matriz cuadrada, o de las diagonales secundarias si se especifica.



## Captura #8 Función transponer matriz.

```
main.py > ...
1 #Codigo Transponer Matriz
2 def transponer_matriz(matriz):
3     if not matriz or not matriz[0]:
4         return []
5
6     num_filas = len(matriz)
7     num_columnas = len(matriz[0])
8
9     # Inicializamos la transpuesta con la estructura correcta
10    matriz_transpuesta = []
11
12    for j in range(num_columnas): # Itera sobre las COLUMNAS originales
13        nueva_fila = []
14        for i in range(num_filas): # Itera sobre las FILAS originales
15            nueva_fila.append(matriz[i][j])
16            matriz_transpuesta.append(nueva_fila)
17
18    return matriz_transpuesta
19
20 # Prueba tu función rigurosamente (incluye matrices no cuadradas):
21 m1 = [[1, 2, 3], [4, 5, 6]] # 2x3
22 t1 = transponer_matriz(m1)
23 assert t1 == [[1, 4], [2, 5], [3, 6]] # Debe ser 3x2
```

Format

Package

python3 mai...

iPrueba 1 (2x3) pasada! ✓  
iPrueba 2 (matriz 1x1) pasada! ✓  
iPrueba 3 (matriz 1x2) pasada! ✓  
iPrueba 4 (matriz 2x1) pasada! ✓  
iPrueba 5 (matriz vacía) pasada! ✓  
iPrueba 6 (matriz con fila vacía) pasada! ✓  
iPrueba 7 (matriz cuadrada 3x3) pasada! ✓  
Todas las pruebas han sido ejecutadas.

Codigo realizado por Franz Almanza

Utilidad
Es una operación fundamental en álgebra lineal que intercambia las filas por las columnas de una matriz. Esto significa que el elemento en la posición (i, j) se mueve a la posición (j, i).

## Captura #9 Función identidad matriz.

```
#Codigo Matriz Identidad
def es_identidad(matriz):
    # Requisito 1: Debe ser cuadrada
    num_filas = len(matriz)
    if num_filas == 0:
        return True # Una matriz vacía es trivialmente identidad

    for i in range(num_filas):
        if len(matriz[i]) != num_filas:
            return False # No es cuadrada

    # Requisito 2: Verificar la diagonal y los ceros
    for i in range(num_filas):
        for j in range(num_filas):
            if i == j:
                if matriz[i][j] != 1:
                    return False # La diagonal no tiene 1
            else:
                if matriz[i][j] != 0:
                    return False # Elemento fuera de la diagonal no es 0

    return True # Cumple con todas las condiciones de matriz identidad
```

Packager

python3 mai...

¡Prueba 1 (matriz identidad 3x3) pasada! ✓  
¡Prueba 2 (matriz 3x3 no identidad por diagonal) pasada! ✓  
¡Prueba 3 (matriz no cuadrada) pasada! ✓  
¡Prueba 4 (matriz vacía) pasada! ✓  
¡Prueba 5 (matriz 1x1 identidad) pasada! ✓  
¡Prueba 6 (matriz 1x1 no identidad) pasada! ✓  
¡Prueba 7 (matriz con fila vacía) pasada! ✓  
¡Prueba 8 (matriz 2x2 con elemento fuera de diagonal no cero) pasada! ✓

---

Todas las pruebas han sido ejecutadas.

Codigo realizado por Franz Almanza

### Utilidad

Es fundamental por varias razones en el ámbito de la manipulación matricial y el álgebra lineal. Una matriz identidad es una matriz cuadrada donde todos los elementos de la diagonal principal son uno, y todos los demás elementos son cero.

Su utilidad principal radica en ser el elemento neutro para la multiplicación de matrices. Al igual que el número 1 no altera un número al multiplicarlo ( $x \cdot 1 = x$ ), la matriz identidad no altera una matriz cuando se multiplica por ella ( $A \cdot I = A$  y  $I \cdot A = A$ ).

## Captura #10 Función simétrica matriz.

```
#Codigo Matriz Simetrica
def es_simetrica(matriz):
    # Requisito 1: Debe ser cuadrada
    num_filas = len(matriz)
    if num_filas == 0:
        return True # Una matriz vacía es trivialmente simétrica

    for i in range(num_filas):
        if len(matriz[i]) != num_filas:
            return False # No es cuadrada

    # Requisito 2: Comparar matriz[i][j] con matriz[j][i]
    for i in range(num_filas):
        for j in range(i + 1, num_filas): # Solo necesitamos chequear la
            triangular superior
            if matriz[i][j] != matriz[j][i]:
                return False # ¡Con una diferencia es suficiente!

    return True # Si nunca encontramos diferencias, es simétrica

# Prueba tu función:
sim = [[1, 7, 3], [7, 4, -5], [3, -5, 6]]
no_sim = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

python3 mai... 200ms • 14 minutes ✓

¡Prueba 1 (matriz simétrica 3x3) pasada! ✓  
 ¡Prueba 2 (matriz 3x3 no simétrica) pasada! ✓  
 ¡Prueba 3 (matriz no cuadrada) pasada! ✓  
 ¡Prueba 4 (matriz vacía) pasada! ✓  
 ¡Prueba 5 (matriz 1x1) pasada! ✓  
 ¡Prueba 6 (matriz 2x2 no simétrica) pasada! ✓  
 ¡Prueba 7 (matriz con fila vacía que no es cuadrada) pasada! ✓

Codigo realizado por Franz Almanza

Utilidad
Tiene una utilidad inmensa porque sus propiedades algebraicas y geométricas simplifican el análisis y la resolución de problemas complejos en una amplia gama de disciplinas científicas y de ingeniería. Permite la diagonalización, la interpretación de valores propios como cantidades físicas reales y la simplificación de formas cuadráticas, lo que se traduce en herramientas poderosas para el análisis de datos, el modelado físico y la optimización.

Captura #11 Sala de cine.

```
#Codigo Sala de Cine
def crear_sala(filas, columnas):
    sala = []
    for i in range(filas):
        fila = []
        for j in range(columnas):
            # Asigna un precio según la ubicación (ejemplo simple)
            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

# Mostrar la sala con precios y estados
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}")

# Ejemplo de uso
sala = crear_sala(5, 8)
mostrar_sala(sala)
```

python3 mai... Ask Assistant 27s • 14 minutes ag ✓

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

Sala actual:

	0	1	2	3	4	5	6	7
F 0	L	L	0	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

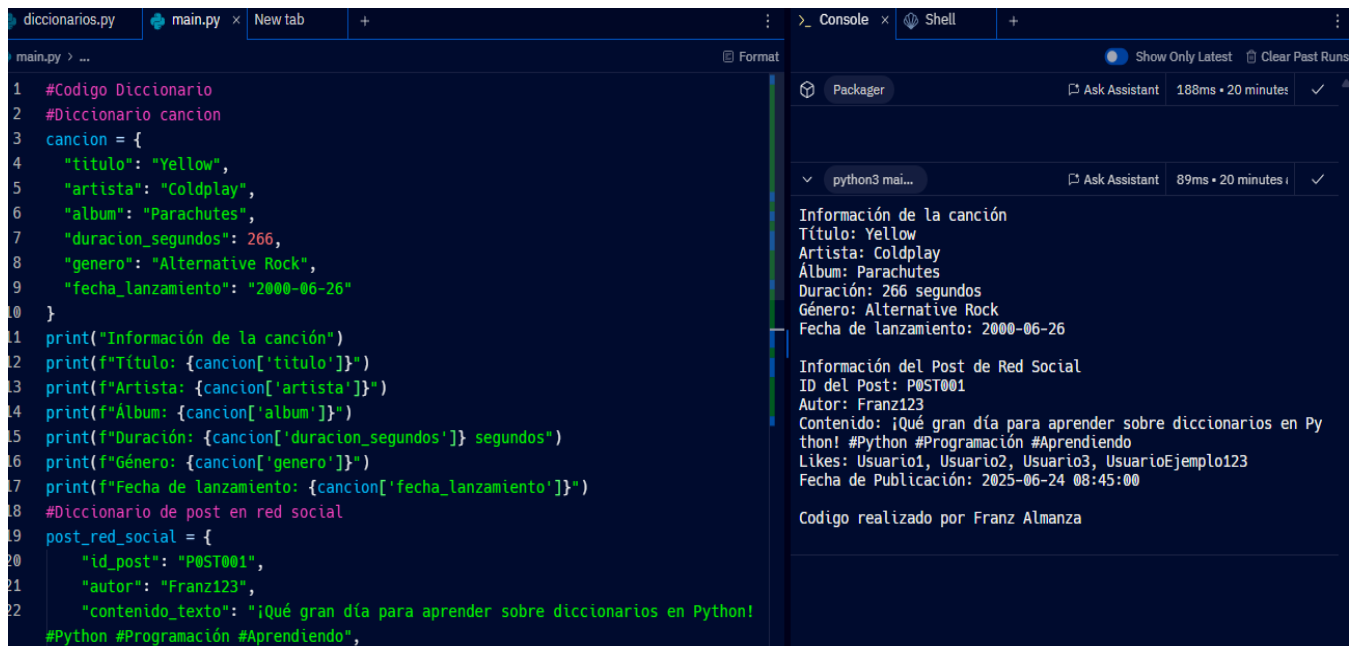
Asientos libres: 39

Menú:  
1. Ocupar asiento individual  
2. Buscar y ocupar N asientos juntos  
0. Salir  
Elige una opción: 0  
Gracias por usar el sistema de reserva de cine. 🍿

Código realizado por Franz Almanza

Utilidad
Funciona como una herramienta estratégica que mejora la eficiencia operativa, reduce costos, optimiza la capacidad de la sala y, lo más importante, eleva significativamente la experiencia del cliente.

## Captura #12 Diccionario.



The screenshot shows a code editor with a file named `diccionarios.py` and a terminal window displaying the output of the script. The script defines two dictionaries: `cancion` and `post_red_social`. The `cancion` dictionary contains information about the song "Yellow" by Coldplay, and the `post_red_social` dictionary contains information about a social media post. The terminal output shows the printed information for both dictionaries.

```
1 #Codigo Diccionario
2 #Diccionario cancion
3 cancion = {
4     "titulo": "Yellow",
5     "artista": "Coldplay",
6     "album": "Parachutes",
7     "duracion_segundos": 266,
8     "genero": "Alternative Rock",
9     "fecha_lanzamiento": "2000-06-26"
10 }
11 print("Información de la canción")
12 print(f"Título: {cancion['titulo']}")
13 print(f"Artista: {cancion['artista']}")
14 print(f"Álbum: {cancion['album']}")
15 print(f"Duración: {cancion['duracion_segundos']} segundos")
16 print(f"Género: {cancion['genero']}")
17 print(f"Fecha de lanzamiento: {cancion['fecha_lanzamiento']}")
18 #Diccionario de post en red social
19 post_red_social = {
20     "id_post": "POST001",
21     "autor": "Franz123",
22     "contenido_texto": "¡Qué gran día para aprender sobre diccionarios en Python!
    #Python #Programación #Aprendiendo",
23 }
```

Información de la canción  
Título: Yellow  
Artista: Coldplay  
Álbum: Parachutes  
Duración: 266 segundos  
Género: Alternative Rock  
Fecha de lanzamiento: 2000-06-26

Información del Post de Red Social  
ID del Post: POST001  
Autor: Franz123  
Contenido: ¡Qué gran día para aprender sobre diccionarios en Python!  
#Python #Programación #Aprendiendo  
Likes: Usuario1, Usuario2, Usuario3, UsuarioEjemplo123  
Fecha de Publicación: 2025-06-24 08:45:00

Código realizado por Franz Almanza

### Utilidad

Los diccionarios son esenciales para organizar y manipular datos de forma eficiente mediante la asociación de claves con valores.

Cada palabra es una clave (única).

La definición de esa palabra es su valor.

## Captura #13 Inventario.

```
#Codigo Inventario
inventario = []

# 2. Crear al menos tres diccionarios de productos diferentes
producto1 = {
    "nombre": "Chocolate para Taza 'El Ceibo'",
    "stock": 50
}

producto2 = {
    "nombre": "Café de los Yungas",
    "stock": 100
}

producto3 = {
    "nombre": "Quinua Real en Grano",
    "stock": 80
}

# 3. Añadir los productos al inventario
inventario.append(producto1)
inventario.append(producto2)
inventario.append(producto3)
```

python3 mai... Ask Assistant 105ms

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 → Quinua Real en Grano  
stock → 80

---

Codigo realizado por Franz Almanza

Utilidad
<p>Puede ser una herramienta poderosa que transforma la gestión manual y propensa a errores en un proceso automatizado, preciso y basado en datos. Esto no solo ahorra tiempo y dinero, sino que también proporciona una visión clara del estado del negocio, permitiendo tomar decisiones más inteligentes y ofrecer un mejor servicio al cliente.</p>

## Captura #14 To Do List.

```
#Codigo To Do List
# Paso 1: Variables Globales
lista_de_tareas = []
proximo_id_tarea = 1 # Para generar IDs únicos

# Paso 2: Implementar agregar_tarea
def agregar_tarea(descripcion, prioridad='media'):
    global proximo_id_tarea
    nueva_tarea = {
        "id": proximo_id_tarea,
        "descripcion": descripcion,
        "completada": False,
        "prioridad": prioridad
    }
    lista_de_tareas.append(nueva_tarea)
    proximo_id_tarea += 1
    print(f"✅ Tarea '{descripcion}' añadida con éxito.")

# Paso 3: Implementar mostrar_tareas
def mostrar_tareas():
    print("\n--- 📌 LISTA DE TAREAS ---")
    if not lista_de_tareas:
        print("¡No hay tareas pendientes! ¡A disfrutar!")

# Ejecución
if __name__ == '__main__':
    # Agregar tarea
    agregar_tarea("Completar proyecto", "alta")
    agregar_tarea("Revisar correo", "media")
    agregar_tarea("Hacer ejercicio", "baja")

    # Mostrar tareas
    mostrar_tareas()

    # Marcar tarea como completada
    # (Este código se ejecutó en la terminal pero no se muestra en la imagen)
```

Utilidad
<p>Este código posee una gran utilidad personal y profesional para organizar, priorizar y dar seguimiento a las actividades diarias. Es una herramienta fundamental para la gestión del tiempo y la productividad.</p> <p>Herramienta versátil y poderosa para gestionar la carga de trabajo, mejorar la organización personal y profesional, y aumentar la productividad al proporcionar una estructura clara para el seguimiento de actividades y el cumplimiento de objetivos.</p>

## Captura #15 Batalla Naval.

```
#Codigo Batalla Naval
import random

FILAS = 4
COLUMNAS = 2
BARCOS = 3

# Crear un tablero vacío
def crear_tablero():
    return [[0 for _ in range(COLUMNAS)] for _ in range(FILAS)]

# Mostrar el tablero
def mostrar_tablero(tablero, ocultar_barcos=False):
    print(" " + " ".join(str(i + 1) for i in range(COLUMNAS)))
    for i, fila in enumerate(tablero):
        letra = chr(ord('A') + i)
        fila_mostrar = []
        for celda in fila:
            if ocultar_barcos and celda == 1:
                fila_mostrar.append("0")
            elif celda == 0:
                fila_mostrar.append("0")
            elif celda == 1:
                fila_mostrar.append("1")

        print(letra + " " + " ".join(str(c) for c in fila_mostrar))

--- Turno 7 ---
Tu tablero:
  1 2
A 0 X
B * *
C X *
D 1 *
Tus disparos:
  1 2
A * *
B X X
C * *
D 0 0
Dispara (ej. A3): d1
Franz Almanza disparó al agua.
CPU dispara a D1
CPU hizo ¡Tocado!
¡La CPU gana!

Codigo realizado por Franz Almanza
```

### Utilidad

Es una excelente herramienta educativa para aprender y aplicar conceptos fundamentales de programación, desde estructuras de datos básicas hasta algoritmos más complejos de IA. Al mismo tiempo, ofrece una forma accesible y divertida de disfrutar de un juego clásico que agudiza el pensamiento estratégico y la lógica.



## Captura #16 Gestor de Contactos (agenda).

```
#Codigo Gestor Contactos (agenda)
agenda = {}

def agregar_contacto(nombre, telefonos, email):
    if nombre in agenda:
        print(f"El contacto '{nombre}' ya existe.")
        return
    agenda[nombre] = {
        'telefonos': telefonos,
        'email': email
    }
    print(f"Contacto '{nombre}' agregado.")

def buscar_por_nombre(nombre):
    """
    Busca y devuelve la información del contacto por nombre.
    """
    return agenda.get(nombre, None)

def editar_contacto(nombre, telefonos=None, email=None):
    """
    Edita los datos de un contacto existente.
    """

# Ejecución del programa
print("Menú de la Agenda")
print("1. Añadir contacto")
print("2. Ver contacto")
print("3. Editar contacto")
print("4. Eliminar contacto")
print("5. Listar todos los contactos")
print("6. Salir")
opcion = input("Selecciona una opción: ")

if opcion == "1":
    nombre = input("Introduce el nombre del contacto a agregar: ")
    telefonos = input("Introduce los telefonos (separados por coma): ")
    email = input("Introduce el email: ")
    agregar_contacto(nombre, telefonos, email)

elif opcion == "2":
    nombre = input("Introduce el nombre del contacto a ver: ")
    buscar_por_nombre(nombre)

elif opcion == "3":
    nombre = input("Introduce el nombre del contacto a editar: ")
    telefonos = input("Introduce los telefonos (separados por coma): ")
    email = input("Introduce el email: ")
    editar_contacto(nombre, telefonos, email)

elif opcion == "4":
    nombre = input("Introduce el nombre del contacto a eliminar: ")
    buscar_por_nombre(nombre)

elif opcion == "5":
    listar_contactos()

elif opcion == "6":
    print("Saliendo de la agenda. ¡Hasta pronto!")

else:
    print("Opción no válida. Por favor, selecciona una opción válida.")
```

Utilidad
<p>Posee una utilidad fundamental para organizar y acceder eficientemente a la información de personas, tanto en el ámbito personal como profesional. Es una herramienta básica para la comunicación y la gestión de relaciones. Al ejecutarla llegamos a entender que es una herramienta práctica y poderosa para mantener organizada la red de contactos de una persona o empresa. Mejora la eficiencia en la comunicación, asegura que la información esté siempre actualizada y accesible, y puede ser adaptado con precisión a las necesidades específicas del usuario.</p>