Programación II

Ricardo Maldonado Suarez

CAPTURAS DE EJERCISIO

Ordenamiento burbuja

```
  Pasada 3:  [3, 2, 1, 4, 5]
  Pasada 4:  [2, 1, 3, 4, 5]
Ordenado: [1, 2, 3, 4, 5]

Resultado: [1, 2, 3, 4, 5]
==================================================
🔸 Caso 4: Lista con duplicados
Estado inicial: [5, 1, 4, 2, 8, 5, 0]
  Pasada 1:  [5, 1, 4, 2, 8, 5, 0]
  Pasada 2:  [1, 4, 2, 5, 5, 0, 8]
  Pasada 3:  [1, 2, 4, 5, 0, 5, 8]
  Pasada 4:  [1, 2, 4, 0, 5, 5, 8]
  Pasada 5:  [1, 2, 0, 4, 5, 5, 8]
  Pasada 6:  [1, 0, 2, 4, 5, 5, 8]
Ordenado: [0, 1, 2, 4, 5, 5, 8]

Resultado: [0, 1, 2, 4, 5, 5, 8]
==================================================
🔸 Caso 5: Lista vacía
Estado inicial: []
Ordenado: []

Resultado: []
==================================================
🔸 Caso 6: Lista con un solo elemento
Estado inicial: [42]
Ordenado: [42]

Resultado: [42]
==================================================
✅ RICARDO MALDONADO
~/workspace$
```

```python
def ordenamiento_de_burbuja(lista):
    n = len(lista)
    print("Estado inicial:", lista)
    for i in range(n - 1):
        hubo_intercambio = False
        print(f"  Pasada {i + 1}: ", lista)
        for j in range(n - 1 - i):
            if lista[j] > lista[j + 1]:
                lista[j], lista[j + 1] = lista[j + 1], lista[j]
                hubo_intercambio = True
        if not hubo_intercambio:
            break
    print("Ordenado:", lista, "\n")
    return lista


# 🔸 Prueba simple de impresión
print("🔸 Caso Base")
lista_a_ordenar = [64, 34, 25, 12, 22, 11, 90]
print("Lista original:", lista_a_ordenar)
ordenamiento_de_burbuja(lista_a_ordenar)
print("Resultado:", lista_a_ordenar)
print("=" * 50)

```

Ordenamiento inserccion

```python

# Función para probar la ordenación por inserción
def probar_insercion():
    print("Probando ordenamiento_insercion...\n")

    casos = [
        ([5, 3, 1, 4, 2], [1, 2, 3, 4, 5]),
        ([1, 2, 3, 4, 5], [1, 2, 3, 4, 5]),
        ([3, 1, 2, 3, 1], [1, 1, 2, 3, 3]),
        ([42], [42]),
        ([], []),
    ]

    for i, (entrada, esperado) in enumerate(casos, start=1):
        original = entrada.copy()  # Copia para mostrar el original sin modificar
        resultado = ordenamiento_insercion(entrada)
        print(f"Caso {i}:")
        print(f"  Lista original: {original}")
        print(f"  Lista ordenada: {resultado}")
        print(f"  Resultado {'correcto ✅' if resultado == esperado else 'incorrecto ❌'}\n")

# Llamamos a la función para probar
probar_insercion()
```

```
~/workspace$ python insercion.py
Probando ordenamiento_insercion...

Caso 1:
  Lista original: [5, 3, 1, 4, 2]
  Lista ordenada: [1, 2, 3, 4, 5]
  Resultado correcto ✅

Caso 2:
  Lista original: [1, 2, 3, 4, 5]
  Lista ordenada: [1, 2, 3, 4, 5]
  Resultado correcto ✅

Caso 3:
  Lista original: [3, 1, 2, 3, 1]
  Lista ordenada: [1, 1, 2, 3, 3]
  Resultado correcto ✅

Caso 4:
  Lista original: [42]
  Lista ordenada: [42]
  Resultado correcto ✅

Caso 5:
  Lista original: []
  Lista ordenada: []
  Resultado correcto ✅

estudiante: Ricardo maldonado
~/workspace$
```

Mergesort

```python
def merge_sort(lista):
    # Paso 1: DIVIDIR
    if len(lista) <= 1:  # Condición base
        return lista

    medio = len(lista) // 2  # Encontrar el punto medio
    mitad_izquierda = lista[:medio]  # Primera mitad
    mitad_derecha = lista[medio:]  # Segunda mitad

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

    # Paso 3: COMBINAR
    return merge(izquierda_ordenada, derecha_ordenada)  #

def merge(izquierda, derecha):
    resultado = []  # Lista para almacenar el resultado
    i = j = 0  # Índices para recorrer las listas

    # Mezclar las listas de manera ordenada
    while i < len(izquierda) and j < len(derecha):
        if izquierda[i] < derecha[j]:
            resultado.append(izquierda[i])
```

```
~/workspace$ python mergesort.py
Lista original: [38, 27, 43, 3, 9, 82, 10]
Lista ordenada: [3, 9, 10, 27, 38, 43, 82]
estudiante: Ricardo maldonado
~/workspace$
```

matrices

```python
# Función que suma los elementos por cada fila de la matriz
def sumar_filas_matriz(matriz):
    """
    Recibe una matriz (lista de listas) y retorna una lista con
    la suma de cada fila.
    """
    suma_filas = []
    for fila in matriz:
        total = 0
        for elemento in fila:
            total += elemento
        suma_filas.append(total)
    return suma_filas


# Función de prueba con salida clara de cada caso
def probar_suma_filas():
    print("🔍 Probando sumar_filas_matriz...\n")

    # Caso 1: matriz normal
    m1 = [[1, 2, 3], [4, 5, 6]]
    resultado1 = sumar_filas_matriz(m1)
    print("Caso 1: matriz normal:", m1)
    print("Resultado esperado: [6, 15]")
```

```
~/workspace$ python matrices.py
🔍 Probando sumar_filas_matriz...

Caso 1: matriz normal: [[1, 2, 3], [4, 5, 6]]
Resultado esperado: [6, 15]
Resultado obtenido: [6, 15]
✅ Prueba 1 pasada

Caso 2: matriz con negativos y ceros: [[-1, 0, 1], [10, -5, 5]]
Resultado esperado: [0, 10]
Resultado obtenido: [0, 10]
✅ Prueba 2 pasada

Caso 3: una fila vacía: [[]]
Resultado esperado: [0]
Resultado obtenido: [0]
✅ Prueba 3 pasada

Caso 4: matriz vacía: []
Resultado esperado: []
Resultado obtenido: []
✅ Prueba 4 pasada

Caso 5: un solo elemento: [[42]]
Resultado esperado: [42]
Resultado obtenido: [42]
✅ Prueba 5 pasada

🎉 ¡Todas las pruebas de suma por filas pasaron correctamente!
RICARDO MALDONADO
~/workspace$
```

Matriz cuadricula

```python
print("=== MATRIZ 5x5 USANDO BUCLES ===")
# Paso 1: Crear la matriz 5x5 con ceros usando bucles
teclado_bucle = []
for i in range(5):
    fila = []
    for j in range(5):
        fila.append(0)
    teclado_bucle.append(fila)

# Paso 2: Imprimir la matriz completa para verificar
print("Matriz original:")
for fila in teclado_bucle:
    print(fila)

# Paso 3: Mostrar valores por índice doble
print("\nAccediendo a valores específicos:")
print("Número en el centro:", teclado_bucle[2][2])              # Cen
print("Número en la esquina inferior derecha:", teclado_bucle[4][4])

# Paso 4: Modificar el número en la esquina superior izquierda (0 → 1
teclado_bucle[0][0] = 1

# Paso 5: Imprimir la matriz modificada
print("\nMatriz después del cambio:")
```

```
Accediendo a valores específicos:
Número en el centro: 0
Número en la esquina inferior derecha: 0

Matriz después del cambio:
[1, 0, 0, 0, 0]
[0, 0, 0, 0, 0]
[0, 0, 0, 0, 0]
[0, 0, 0, 0, 0]
[0, 0, 0, 0, 0]

=== MATRIZ 5x5 USANDO COMPRENSIÓN DE LISTAS ===
Matriz original:
[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]

Accediendo a valores específicos:
Número en el centro: 0
Número en la esquina inferior derecha: 0

Matriz después del cambio:
[1, 0, 0, 0, 0]
[0, 0, 0, 0, 0]
[0, 0, 0, 0, 0]
[0, 0, 0, 0, 0]
[0, 0, 0, 0, 0]
RICARDO MALDONADO - FIN DEL PROGRAMA DE MATRICES)
~/workspace$
```

Matricial

```python
    """
    total = 0

    for fila in matriz:
        for elemento in fila:
            total += elemento
    return total


def probar_suma_total():
    print("Probando sumar_total_matriz...")


    m1 = [[1, 2, 3], [4, 5, 6]]
    assert sumar_total_matriz(m1) ==   10


    m2 = [[-1, 0, 1], [10, -5, 5]]
    assert sumar_total_matriz(m2) == 10


    assert sumar_total_matriz([[]]) == 0
    assert sumar_total_matriz([]) == 0
    assert sumar_total_matriz([[42]]) == 10
```

```
~/workspace$ python matricial.py
Probando sumar_total_matriz...
Traceback (most recent call last):
  File "/home/runner/workspace/matricial.py", line 36, in <module>
    probar_suma_total()
    ^^^^^^^^^^^^^^^^^^^
  File "/home/runner/workspace/matricial.py", line 22, in probar_suma_total
    assert sumar_total_matriz(m1) ==   10
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
AssertionError
~/workspace$ python matricial.py
Probando sumar_total_matriz...
¡Pruebas para sumar_total_matriz pasaron! ✅
RICARDO MALDONADO
~/workspace$
```

Función por filas

```python
    Ejemplo:
    matriz = [[1, 2], [3, 4], [5, 6]]
    resultado = [3, 7, 11]
    """
    resultado = []
    for fila in matriz:
        suma_fila_actual = sum(fila)  # Suma todos los el
        resultado.append(suma_fila_actual)
    return resultado

# Función de prueba
def probar_sumar_por_filas():
    print("\nProbando sumar_por_filas...")
    # Caso 1: matriz con 3 filas y 3 columnas
    m1 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
    assert sumar_por_filas(m1) == [6, 15, 24]  # 1+2+3, 4

    # Caso 2: matriz con pares repetidos
    m2 = [[20, 40], [60, 20], [30, 30]]
    assert sumar_por_filas(m2) == [60, 80, 60]

    # Caso borde: matriz vacía
    assert sumar_por_filas([]) == []  # No hay filas que
    print("¡Pruebas para sumar_por_filas pasaron! [✓]")
```

```
~/workspace$ python funcion_por_filas.py

Probando sumar_por_filas...
Traceback (most recent call last):
  File "/home/runner/workspace/funcion_por_filas.py", line 32, in <module>
    probar_sumar_por_filas()
  File "/home/runner/workspace/funcion_por_filas.py", line 25, in probar_sumar_por_filas
    assert sumar_por_filas(m2) == [60,  60]
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

AssertionError
~/workspace$ python funcion_por_filas.py

Probando sumar_por_filas...
¡Pruebas para sumar_por_filas pasaron! [✓]
estudiante: Ricardo Maldonado
```

Suma diagonal

```python
    """
    suma = 0
    for i in range(len(matriz)):
        suma += matriz[i][i]
    return suma


def probar_suma_diagonal_principal():
    print("\nProbando sumar_diagonal_principal...")


    m1 = [[1, 2, 3],
          [4, 5, 6],
          [7, 8, 9]]
    assert sumar_diagonal_principal(m1) == 15
    m2 = [[10, 0],
          [0, 20]]
    assert sumar_diagonal_principal(m2) == 30


    m3 = [[5]]
    assert sumar_diagonal_principal(m3) == 5

    print("Pruebas para sumar_diagonal_principal pasaron! ✅")
```

```
~/workspace$ python suma_diagonal.py

Probando sumar_diagonal_principal...
Pruebas para sumar_diagonal_principal pasaron! ✅
~/workspace$
```

Matriz transponer

```python
# Pruebas rigurosas (incluyendo matrice
if __name__ == "__main__":
    m1 = [[1, 2, 3], [4, 5, 6]]  # 2x3
    t1 = transponer_matriz(m1)
    assert t1 == [[1, 4], [2, 5], [3, 6]]
    print("Prueba 1 (2x3) pasada!")

    m2 = [[1, 2], [3, 4], [5, 6]]  # 3x2
    t2 = transponer_matriz(m2)
    assert t2 == [[1, 3, 5], [2, 4, 6]]
    print("Prueba 2 (3x2) pasada!")

    m3 = [[1]]  # 1x1
    t3 = transponer_matriz(m3)
    assert t3 == [[1]]
    print("Prueba 3 (1x1) pasada!")

    m4 = []  # matriz vacía
    t4 = transponer_matriz(m4)
    assert t4 == [4,5,8]
    print("Prueba 4 (vacía) pasada!")

    m5 = [[1, 2, 3]]  # 1x3
    t5 = transponer_matriz(m5)
```

```
~/workspace$ python transponer.py
Prueba 1 (2x3) pasada!
Prueba 2 (3x2) pasada!
Prueba 3 (1x1) pasada!
Traceback (most recent call last):
  File "/home/runner/workspace/transponer.py", line 34, in <module>
    assert t4 == [4,5,8]
           ^^^^^^^^^^^^^
AssertionError
~/workspace$ python transponer.py
Prueba 1 (2x3) pasada!
Prueba 2 (3x2) pasada!
Prueba 3 (1x1) pasada!
Prueba 4 (vacía) pasada!
Prueba 5 (1x3) pasada!
Todas las pruebas pasaron correctamente.
estudiante: Ricardo Maldonado
~/workspace$
```

Función de identidad

```python
    if num_filas == 0:
        return True  # Una matriz vacía es trivialmen

    for fila in matriz:
        if len(fila) != 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 t
            else:
                if matriz[i][j] != 0:
                    return False  # Elemento fuera de

    return True  # Cumple con todas las condiciones

# Pruebas
identidad = [[1, 0, 0], [0, 1, 0], [0, 0, 1]]
no_identidad = [[1, 0, 0], [0, 0, 0], [0, 0, 1]]
no_cuadrada = [[1, 0, 1], [0, 0, 0]]
tambien_identidad = [[1, 0], [0, 1]]
```

```
~/workspace$ python funcion_identidad.py
Todas las pruebas pasaron correctamente.
estudiante: Ricardo Maldonado
~/workspace$
```

Funcion simétrica

```python
def es_simetrica(matriz):
    # Requisito 1: Debe ser cuadrada
    num_filas = len(matriz)
    if num_filas == 0:
        return True  # Una matriz vacía es trivialm

    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
    for i in range(num_filas):
        for j in range(i + 1, num_filas):  # Solo r
            if matriz[i][j] != matriz[j][i]:
                return False  # ¡Con una diferencia

    return True  # Si nunca encontramos diferencias

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

assert es_simetrica(sim) == True
```

```
~/workspace$ python funcion_simetrica.py
¡Pruebas para es_simetrica pasaron! ✅
estudiante: Ricardo Maldonado
~/workspace$
```

Sala de cine

```python
def crear_sala(filas, columnas):
    return [['L' for _ in range(columnas)] for _ in range(filas)]

def mostrar_sala(sala):
    for fila in sala:
        print(" | ".join(fila))
        print("-" * (len(sala[0]) * 4 - 1))

def ocupar_asiento(sala, fila, columna):
    if fila < 0 or fila >= len(sala) or columna < 0 or columna >= len(sala[0]):
        print("Error: Coordenadas fuera de rango.")
        return False
    if sala[fila][columna] == 'L':
        sala[fila][columna] = 'O'
        return True
    else:
        print("Error: El asiento ya está ocupado.")
        return False

def contar_asientos_libres(sala):
    return sum(fila.count('L') for fila in sala)

def guardar_sala(sala, archivo="sala_guardada.txt"):
    with open(archivo, "w") as f:
```

```
~/workspace: python sala_cine.py

L | L | L | L | L | L | L | L
-------------------------------
L | L | L | L | L | L | L | L
-------------------------------
L | L | L | L | L | L | L | L
-------------------------------
L | L | L | L | L | L | L | O
-------------------------------
L | L | L | L | L | L | L | L
-------------------------------
¿Quieres ocupar un asiento? (1: Sí, 0: Salir): 1
Introduce la fila (0 a 4): 0
Introduce la columna (0 a 7): 0
Asiento ocupado exitosamente.

Sala de Cine:
O | L | L | L | L | L | L | L
-------------------------------
L | L | L | L | L | L | L | L
-------------------------------
L | L | L | L | L | L | L | L
-------------------------------
L | L | L | L | L | L | L | O
-------------------------------
L | L | L | L | L | L | L | L
-------------------------------
¿Quieres ocupar un asiento? (1: Sí, 0: Salir): 0

Total de asientos libres: 38
✅ Sala guardada exitosamente en sala_guardada.txt
estudiante: Ricardo Maldonado
~/workspace$
```

Diccionario

```python
producto = {
    'codigo': 'P001',
    'nombre': 'Café',
    'precio': 38.0,
    'stock': 100
}

# Iterar sobre claves
print("\n--- Claves del producto ---")
for clave in producto:
    print(clave)

# Iterar sobre claves y valores
print("\n--- Clave y Valor ---")
for clave in producto:
    valor = producto[clave]
    print(f"{clave.capitalize()}: {valor}")

# Otras formas de acceder a claves, valores e ítems
print("\n--- Usando keys(), values(), items() ---")
print("Claves:", list(producto.keys()))
print("Valores:", list(producto.values()))
print("Items:", list(producto.items()))
```

```
~/workspace$ pyhton diccionario.py
bash: pyhton: command not found
~/workspace$ python diccionario.py

--- Claves del producto ---
codigo
nombre
precio
stock

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

--- Usando keys(), values(), items() ---
Claves: ['codigo', 'nombre', 'precio', 'stock']
Valores: ['P001', 'Café', 38.0, 100]
Items: [('codigo', 'P001'), ('nombre', 'Café'), ('precio', 38.0), ('stock', 100)]

Descuento aplicable: 0.0
estudiante: Ricardo Maldonado
~/workspace$
```

Teclado numérico

```python
teclado = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]

print("Matriz original:")
for fila in teclado:
    print(fila)


print(f"\nNúmero en el centro: {teclado[1][1]}")
print(f"Número en la esquina inferior derecha: {teclado[2][2]}")

teclado[0][0] = 0

print("\nMatriz modificada:")
for fila in teclado:
    print(fila)

print("RICARDO MALDONADO - FIN DEL PROGRAMA DE MATRICES)")
```

```
~/workspace$ python teclado_numerico.py
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]
RICARDO MALDONADO - FIN DEL PROGRAMA DE MATRICES)
~/workspace$
```

Inventario

```python
}

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

producto3 = {
    'nombre': 'Quinoa Real en Grano',
    'stock': 80
}

# Añadir cada uno de esos diccionarios a la lista inventario
inventario.append(producto1)
inventario.append(producto2)
inventario.append(producto3)

# Imprimir la cantidad de tipos de producto en el inventario
print(f'Cantidad de tipos de productos en el inventario: {len(inventario)}')

# Recorrer la lista inventario con un bucle for
print('--- Inventario Actual ---')
for producto in inventario:
    print(f"- {producto['nombre']}: {producto['stock']} unidades en stock.")
```

```
~/workspace$ python creando_inventario.py
Cantidad de tipos de productos 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.
estudiante: Ricardo Maldonado
~/workspace$
```

Lista de tareas

```python
def agregar_tarea(descripcion, prioridad="media"):
    global proximo_id_tarea
    nueva_tarea = {
        "id": proximo_id_tarea,
        "descripcion": descripcion,
        "completada": False,
        "prioridad": prioridad.lower()
    }
    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!")
        return

    for tarea in lista_de_tareas:
        estado = "✅" if tarea["completada"] else "⬜"
        print(f"{estado} ID: {tarea['id']} | {tarea['descripcion']} (Prioridad: {tarea['prioridad']})")
    print("-------------------------")
```



```
~/workspace: python todo_list.py

~/workspace$ python todo_list.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: 2

--- 📝 LISTA DE TAREAS ---
⬜ ID: 1 | levantarme temprano a las 6:00 AM (Prioridad: alta)
⬜ ID: 2 | ir al gym lunes , miercoles. viernes (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: 0
¡Hasta pronto!
estudiante: Ricardo Maldonado
~/workspace$
```

Invertir lista

```python
def invertir_lista(lista_original):
    # Crea una lista vacía para la lista invertida
    lista_invertida = []

    # Recorre la lista original de atrás hacia adelante
    for i in range(len(lista_original) - 1, -1, -1):
        lista_invertida.append(lista_original[i])

    return lista_invertida

# Pruebas
print("\nProbando invertir_lista...")
lista_prueba = [1, 2, 3, 4, 5]
lista_resultante = invertir_lista(lista_prueba)
assert lista_resultante == [5, 4, 3, 2, 1]
assert lista_prueba == [1, 2, 3, 4, 5]  # Verifica que la
assert invertir_lista([]) == []  # Prueba con lista vacía
assert invertir_lista(["a", "b", "c"]) == ["c", "b", "a"]
print("Pruebas para invertir_lista pasaron! ✅")

print("estudiante: Ricardo maldonado")
```

```
~/workspace$ python invertir_lista.py

Probando invertir_lista...
Pruebas para invertir_lista pasaron! ✅
estudiante: Ricardo maldonado
~/workspace$
```

Modelando productos

```python
# crear un diccionario llamado 'producto'
producto = {
    "codigo": "P001",
    "nombre": "Chocolate para Taza 'El Ceibo'",
    "precio_unitario": 15.50,
    "stock": 50,
    "proveedor": "El Ceibo Ltda."
}

# 2. Imprimir un mensaje que muestre el nombre y el precio
print(f"Producto: {producto['nombre']}")
print(f"Precio unitario: ${producto['precio_unitario']}")

# Simula una venta actualizando el stock
producto['stock'] -= 5  # Restando 5 unidades

# 4. El producto entra en promoción
producto["en_oferta"] = True

# 5. Imprimir el diccionario completo
print("\nDetalles del producto después de la venta:")
print(producto)

print("estudiante: Ricardo Maldonado")
```

```
~/workspace$ python modelando_producto.py
Producto: Chocolate para Taza 'El Ceibo'
Precio unitario: $15.5

Detalles del producto después de la venta:
{'codigo': 'P001', 'nombre': "Chocolate para Taza 'El Ceibo'", 'precio_unitario': 15.5, 'stock': 45, 'proveedor': 'El Ceibo Ltda.', 'e
oferta': True}
estudiante: Ricardo Maldonado
```

Gestor de contactos

```python
import json
import os

ARCHIVO = "contactos.json"
contactos = []

# --- CARGA y GUARDA ---
def cargar_contactos():
    global contactos
    if os.path.exists(ARCHIVO):
        with open(ARCHIVO, "r", encoding="utf-8") as f:
            contactos = json.load(f)

def guardar_contactos():
    with open(ARCHIVO, "w", encoding="utf-8") as f:
        json.dump(contactos, f, ensure_ascii=False, indent=2)

# --- Agrega un nuevo contacto ---
def agregar_contacto(nombre, telefonos, email):
    nuevo = {
        "nombre": nombre,
        "telefonos": telefonos,
        "email": email
    }
```

```
===== MENÚ GESTOR DE CONTACTOS =====
1. Agregar nuevo contacto
2. Mostrar todos los contactos
3. Buscar contacto por nombre
4. Editar contacto
0. Salir
Elige una opción: 2

📒 Lista de Contactos:
👤 flavio | 📞 74001914 | ✉️ flavio20@gamil.com
👤 richi | 📞 62112170 | ✉️ maldonadosuarezricardo988@gmail.com
------------------------------

===== MENÚ GESTOR DE CONTACTOS =====
1. Agregar nuevo contacto
2. Mostrar todos los contactos
3. Buscar contacto por nombre
4. Editar contacto
0. Salir
Elige una opción: 0
¡Hasta pronto!
estudiante: Ricardo Maldonado
~/workspace$
```

Canción

```python
cancion = {
    "titulo": "es un secreto",
    "artista": "plan B",
    "album": "house of pleasure",
    "duracion_segundos": 192,
    "genero": "Reggaeton",
    "anio_publicacion": 2010
}

def mostrar_informacion(cancion):
    print("Título:", cancion["titulo"])
    print("Artista:", cancion["artista"])
    print("Álbum:", cancion["album"])
    print("Duración (segundos):", cancion["duracion_segundos"])
    print("Género:", cancion["genero"])
    print("Año de publicación:", cancion["anio_publicacion"])


mostrar_informacion(cancion)

print("estudiante: Ricardo Maldonado")
```

```
~/workspace$ python cancion.py
Título: es un secreto
Artista: plan B
Álbum: house of pleasure
Duración (segundos): 192
Género: Reggaeton
Año de publicación: 2010
estudiante: Ricardo Maldonado
~/workspace$
```

Batalla naval

```python
import random

class Tablero:
    def __init__(self):
        self.tablero = [['~' for _ in range(5)] for _ in range(5)]
        self.barcos = 0

    def colocar_barco(self, fila, columna):
        if self.tablero[fila][columna] == '~':
            self.tablero[fila][columna] = 'B'
            self.barcos += 1
            return True
        return False

    def disparar(self, fila, columna):
        if self.tablero[fila][columna] == 'B':
            self.tablero[fila][columna] = 'X'
            self.barcos -= 1
            return True
        elif self.tablero[fila][columna] == '~':
            self.tablero[fila][columna] = 'O'
            return False
        return None

```

```
Jugador 2, ingresa la fila (0-4) para colocar tu barco
Jugador 2, ingresa la columna (0-4) para colocar tu ba
Jugador 2, ingresa la fila (0-4) para colocar tu barco
Jugador 2, ingresa la columna (0-4) para colocar tu ba
Turno del Jugador 1
~ ~ ~ ~ B
~ ~ ~ B ~
~ ~ ~ ~ ~
~ ~ B ~ ~
~ ~ ~ ~ ~

Ingresa la fila (0-4) para disparar: 0
Ingresa la columna (0-4) para disparar: 2
Agua!
Turno del Jugador 2
~ B B B ~
~ ~ ~ ~ ~
~ ~ ~ ~ ~
~ ~ ~ ~ ~
~ ~ ~ ~ ~

Ingresa la fila (0-4) para disparar: 0
Ingresa la columna (0-4) para disparar: 4
Agua!
Turno del Jugador 1
~ ~ O ~ B
~ ~ ~ B ~
~ ~ ~ ~ ~
~ ~ B ~ ~
```

Diagonal secundaria

```
    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.
    Por ejemplo, en una matriz 3x3:
    [[a, b, c],
     [d, e, f],
     [g, h, i]]
    La diagonal secundaria está en las posiciones: (0,2), (1,1), (2,0)
    y su suma sería: c + e + g
    """
    n = len(matriz)  # Número de filas (y columnas, ya que es cuadrada)
    suma = 0
    for i in range(n):
        suma += matriz[i][n - 1 - i]  # Accede al elemento en la posición
    return suma


# Función de pruebas para validar que sumar_diagonal_secundaria funciona
def probar_suma_diagonal_secundaria():
    print("\nProbando sumar_diagonal_secundaria...")

    # Caso 1: matriz 3x3 normal
    m1 = [[1, 2, 3],
```

```
~/workspace$ python diagonal_secundaria.py

Probando sumar_diagonal_secundaria...
✔Pruebas para sumar_diagonal_secundaria pasaron!
estudiante: Ricardo Maldonado
~/workspace$
```