

UNIVERSIDAD PRIVADA DOMINGO SAVIO

FACULTAD DE INGENIERIA



Actividad 2

Título: Ejercicios en Replit

Docente: Ing.Jimmy Nataniel Requena Llorentty

Materia: Programación 2

Estudiante: Yoel Escalante Escobar

Fecha y hora actual: 2025-07-02 09:12:49

Santa Cruz – Bolivia

Julio del 2025

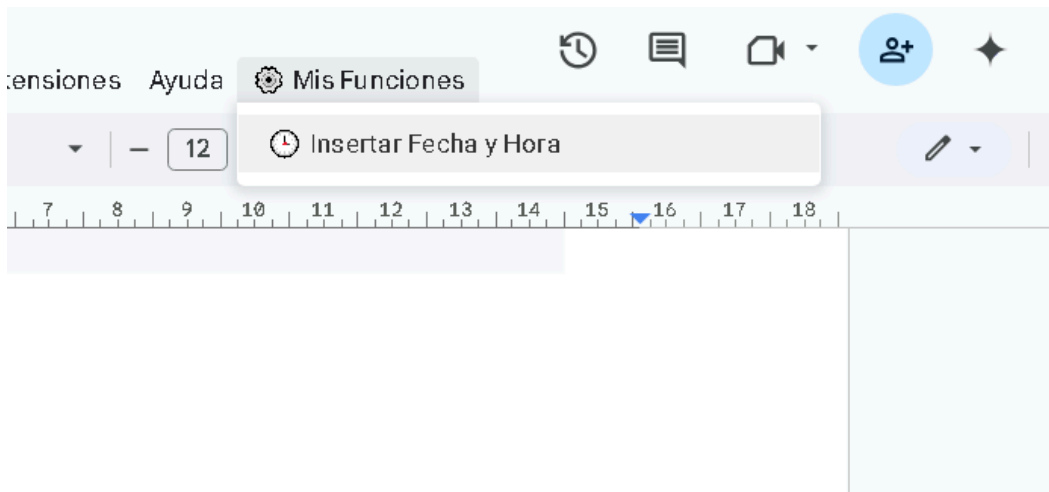
Scrip del documento	3
Actividad 1 Ejercicios	4
Ejercicios de la clase 07	4
Ejercicios de la clase 08 Marge sort	5
Ejercicios de la clase 09	6
Ejercicios de la clase 10	7
Clase 10 ejercicio 2	7
Clase 10 ejercicio 3	8
Ejercicios de la clase 11	8
Clase 11 ejercicio 2	9
Clase 11 ejercicio 3	9
Ejercicios de la clase 12	10
Ejercicios de la clase 13	11
Suma diagonal secundaria	13
Gestión de asientos de cine	14
Batalla Naval con persistencia	15
Agenda	19
Teclado numérico	20
Inventario	21

Scrip del documento

Implementamos el scrip para el google docs

```
function onOpen() {  
  DocumentApp.getUi()  
    .createMenu('⚙ Mis Funciones')  
    .addItem('🕒 Insertar Fecha y Hora', 'insertarFechaYHora')  
    .addToUi();  
}  
  
function insertarFechaYHora() {  
  var body = DocumentApp.getActiveDocument().getBody();  
  var fecha = Utilities.formatDate(new Date(), Session.getScriptTimeZone(),  
    "yyyy-MM-dd HH:mm:ss");  
  var parrafo = body.appendParagraph("Fecha y hora actual: " + fecha);  
  
  // Aplicar formato Arial tamaño 16  
  parrafo.setFontFamily("Arial");  
  parrafo.setFontSize(16);  
}
```

Con la implementación ahora aparece el botón para añadir fecha y hora



Actividad 1 Ejercicios

Ejercicios de la clase 07

- ordenamiento burbuja
- caso#1 lista desordenada
- caso#2 lista ya ordenada
- caso#3 lista ordenada a la inversa
- caso#4 lista con elementos duplicados
- caso borde

```
~/workspace$ python clase07_ordenamiento.py
antes [6, 3, 8, 2, 5]
despues [2, 3, 5, 6, 8]
lista original: [64, 34, 25, 12, 22, 11, 90]
lista ordenada: [11, 12, 22, 25, 34, 64, 90]
todas las pruebas pasaron
~/workspace$
```

```
def ordenamiento_burbuja(lista):
    n = len(lista)
    for i in range(n - 1):
        hubo_intercambio = False
        for j in range(n - 1 - i):
            if lista[j] > lista[j + 1]:
                #intercambio
                lista[j], lista[j + 1] = lista[j + 1], lista[j]
                hubo_intercambio = True
        if not hubo_intercambio:
            break
    return lista

if __name__ == "__main__":
    numeros = [6, 3, 8, 2, 5]
    print("antes", numeros)
    ordenamiento_burbuja(numeros)
    print("despues", numeros)

lista_a_ordenar = [64, 34, 25, 12, 22, 11, 90]
print(f"lista original: {lista_a_ordenar}")

ordenamiento_burbuja(lista_a_ordenar) #llamamos la funcion
print(f"lista ordenada: {lista_a_ordenar}")

#caso 1: lista desordenada
lista1 = [6, 3, 8, 2, 5]
ordenamiento_burbuja(lista1)
assert lista1 == [2, 3, 5, 6, 8], "Fallo en caso 1"

#caso 2: lista ya ordenada
lista2 = [1, 2, 3, 4, 5]
ordenamiento_burbuja(lista2)
assert lista2 == [1, 2, 3, 4, 5], "Fallo en caso 2"

#caso 3: lista ordenada a la inversa(por caso)
lista3 = [5, 4, 3, 2, 1]
ordenamiento_burbuja(lista3)
assert lista3 == [1, 2, 3, 4, 5], "Fallo en caso 3"

#caso 4: lista con elementos duplicados
lista4 = [5, 1, 4, 2, 8, 5, 2]
ordenamiento_burbuja(lista4)
assert lista4 == [1, 2, 2, 4, 5, 5, 8], "fallo en caso 4"

#caso borde
assert ordenamiento_burbuja([]) == []
assert ordenamiento_burbuja([42]) == [42], "fallo en caso borde"

print("todas las pruebas pasaron")
```

Ejercicios de la clase 08 Merge sort

```
1 def merge_sort(lista):
2     # Paso Vencer (Condición base de la recursividad)
3     if len(lista) <= 1:
4         return lista
5
6     # Paso 1: DIVIDIR
7     medio = len(lista) // 2
8     mitad_izquierda = lista[:medio]
9     mitad_derecha = lista[medio:]
10
11     # Paso 2: VENCER (ordenar recursivamente cada mitad)
12     izquierda_ordenada = merge_sort(mitad_izquierda)
13     derecha_ordenada = merge_sort(mitad_derecha)
14
15     # Paso 3: COMBINAR (mezclar ambas mitades ordenadas)
16     return merge(izquierda_ordenada, derecha_ordenada)
17
18 def merge(izquierda, derecha):
19     resultado = []
20     i = j = 0
21
22     # Comparar elementos y mezclar en orden
23     while i < len(izquierda) and j < len(derecha):
24         if izquierda[i] < derecha[j]:
25             resultado.append(izquierda[i])
26             i += 1
27         else:
28             resultado.append(derecha[j])
29             j += 1
30
31     # Agregar los elementos restantes (si quedan)
32     resultado.extend(izquierda[i:])
33
34 izquierda_ordenada = merge_sort(mitad_izquierda)
35 derecha_ordenada = merge_sort(mitad_derecha)
36
37 # Paso 3: COMBINAR (mezclar ambas mitades ordenadas)
38 return merge(izquierda_ordenada, derecha_ordenada)
39
40 def merge(izquierda, derecha):
41     resultado = []
42     i = j = 0
43
44     # Comparar elementos y mezclar en orden
45     while i < len(izquierda) and j < len(derecha):
46         if izquierda[i] < derecha[j]:
47             resultado.append(izquierda[i])
48             i += 1
49         else:
50             resultado.append(derecha[j])
51             j += 1
52
53     # Agregar los elementos restantes (si quedan)
54     resultado.extend(izquierda[i:])
55     resultado.extend(derecha[j:])
56
57 return resultado
58
59 # Ejemplo de uso
60 mi_lista = [38, 27, 43, 3, 9, 82, 10]
61 ordenada = merge_sort(mi_lista)
62 print("Lista ordenada:", ordenada)
```

```
~/workspace$ python clase08_ordenamientoavanzado.py
Lista ordenada: [3, 9, 10, 27, 38, 43, 82]
```

Ejercicios de la clase 09

- Matriz de números
- acceder a una matriz
- modificar una matriz
- tres en raya

```
ejercicios-de-estructura-de-control 0% used
Shell x Workflows Console +
~/workspace:python clase09_matrices.py
El valor es: 70

--- Recorrido con índices ---
Elemento en (0,0) es 10
Elemento en (0,1) es 20
Elemento en (0,2) es 30
Elemento en (0,3) es 40
Elemento en (1,0) es 50
Elemento en (1,1) es 60
Elemento en (1,2) es 70
Elemento en (1,3) es 80
Elemento en (2,0) es 0
Elemento en (2,1) es 91
Elemento en (2,2) es 92
Elemento en (2,3) es 93

--- Recorrido Pythonico ---
10 20 30 40
50 60 70 80
0 91 92 93

--- Tablero de Tres en Raya ---
X | O | X
---
| X | O
---
O | | 
---
~/workspace$
```

```
Code Blame
1 # --- MATRIZ DE NÚMEROS ---
2 # Definimos una matriz de 3 filas y 4 columnas
3 matriz = [
4     [10, 20, 30, 40], # Fila 0
5     [50, 60, 70, 80], # Fila 1
6     [90, 91, 92, 93] # Fila 2
7 ]
8
9 # Acceder al número 70
10 valor = matriz[1][2]
11 print(f"El valor es: {valor}") # Resultado: 70
12
13 # Modificar el valor 90 por un 0
14 matriz[2][0] = 0
15
16 print("\n--- Recorrido con índices ---")
17 num_filas = len(matriz)
18 num_columnas = len(matriz[0])
19 for i in range(num_filas):
20     for j in range(num_columnas):
21         elemento = matriz[i][j]
22         print(f"Elemento en ({i},{j}) es {elemento}")
23
24 print("\n--- Recorrido Pythonico ---")
25 for fila_actual in matriz:
26     for elemento in fila_actual:
27         print(elemento, end=" ")
28     print() # Salto de línea entre filas
29
30 # --- TABLERO DE TRES EN RAYA ---
31 print("\n--- Tablero de Tres en Raya ---")
32 tablero = [
33     ['X', 'O', 'X'],
34     [' ', 'X', 'O'],
35     ['O', ' ', ' ']
```

```
Shell x Workflows Console +
~/workspace:python clase09_matrices.py
El valor es: 70

--- Recorrido con índices ---
Elemento en (0,0) es 10
Elemento en (0,1) es 20
Elemento en (0,2) es 30
Elemento en (0,3) es 40
Elemento en (1,0) es 50
Elemento en (1,1) es 60
Elemento en (1,2) es 70
Elemento en (1,3) es 80
Elemento en (2,0) es 0
Elemento en (2,1) es 91
Elemento en (2,2) es 92
Elemento en (2,3) es 93

--- Recorrido Pythonico ---
10 20 30 40
50 60 70 80
0 91 92 93

--- Tablero de Tres en Raya ---
X | O | X
---
| X | O
---
O | | 
---
~/workspace$
```

```
Code Blame
12
13 # Modificar el valor 90 por un 0
14 matriz[2][0] = 0
15
16 print("\n--- Recorrido con índices ---")
17 num_filas = len(matriz)
18 num_columnas = len(matriz[0])
19 for i in range(num_filas):
20     for j in range(num_columnas):
21         elemento = matriz[i][j]
22         print(f"Elemento en ({i},{j}) es {elemento}")
23
24 print("\n--- Recorrido Pythonico ---")
25 for fila_actual in matriz:
26     for elemento in fila_actual:
27         print(elemento, end=" ")
28     print() # Salto de línea entre filas
29
30 # --- TABLERO DE TRES EN RAYA ---
31 print("\n--- Tablero de Tres en Raya ---")
32 tablero = [
33     ['X', 'O', 'X'],
34     [' ', 'X', 'O'],
35     ['O', ' ', ' ']]
36
37
38 for fila in tablero:
39     print(" | ".join(fila))
40     print("-" * 9)
```

Ejercicios de la clase 10

- suma total de matrices
- prueba de suma total
- caso#1 matriz normal
- caso#2 matriz con negativos y ceros

```
Shell  x  Workflows  > Console  +

~/workspace: python clase10_operacionesconmatrices.py

~/workspace$ python clase10_operacionesconmatrices.py
Probando sumar_total_matriz...
¡Pruebas para sumar_total_matriz pasaron!

Probando sumar_por_filas...
¡Pruebas para sumar_por_filas pasaron!
ejercicio 2: ¡Todo está correcto!

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

```
1  # Definimos la función que suma todos los elementos de una matriz
2  def sumar_total_matriz(matriz):
3      """
4      Esta función recibe una matriz (lista de listas)
5      y retorna la suma total de todos sus elementos.
6      Ejemplo:
7      matriz = [[1, 2], [3, 4]]
8      resultado = 10
9      """
10     total = 0
11     for fila in matriz:
12         for elemento in fila:
13             total += elemento
14     return total
15
16 # Función para probar que sumar_total_matriz funciona correctamente
17 def probar_suma_total():
18     print("Probando sumar_total_matriz...")
19
20     # Caso 1: matriz normal
21     m1 = [[1, 2, 3], [4, 5, 6]]
22     assert sumar_total_matriz(m1) == 21 # 1+2+3+4+5+6 = 21
23
24     # Caso 2: matriz con negativos y ceros
25     m2 = [[-1, 0, 1], [10, -5, 5]]
26     assert sumar_total_matriz(m2) == 10 # -1+0+1+10-5+5 = 10
27
28     # Casos borde o límites
29     assert sumar_total_matriz([]) == 0 # Matriz con una fila vacía
30     assert sumar_total_matriz([[]]) == 0 # Matriz completamente vacía
31     assert sumar_total_matriz([[42]]) == 42 # Matriz de un solo elemento
32
33     print("¡Pruebas para sumar_total_matriz pasaron!")
```

Clase 10 ejercicio 2

- suma de elementos por fila
- matriz con 3 filas y 3 columnas
- matriz con pares repetidos
- caso borde: matriz vacía

```
Shell  x  Workflows  > Console  +

~/workspace: python clase10_operacionesconmatrices.py

~/workspace$ python clase10_operacionesconmatrices.py
Probando sumar_total_matriz...
¡Pruebas para sumar_total_matriz pasaron!

Probando sumar_por_filas...
¡Pruebas para sumar_por_filas pasaron!
ejercicio 2: ¡Todo está correcto!

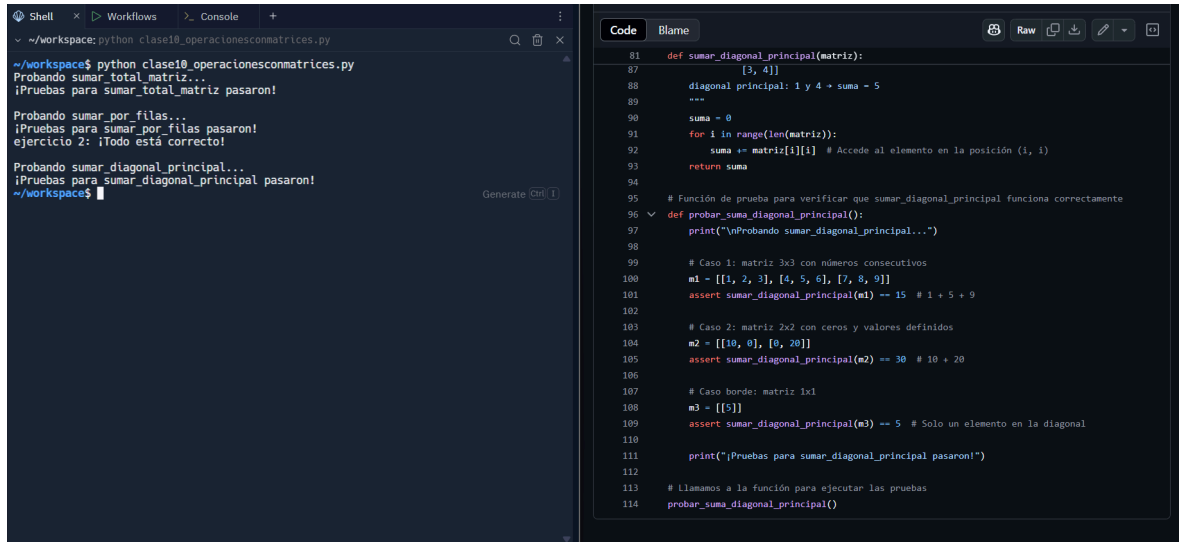
Probando sumar_diagonal_principal...
¡Pruebas para sumar_diagonal_principal pasaron!
~/workspace$
```

```
Code  Blame

17  def probar_suma_total():
18
19  # Llamamos a la función de pruebas
20  probar_suma_total()
21
22  #Ejercicio 2
23
24  # Definimos la función que suma los elementos por cada fila de la matriz
25  def sumar_por_filas(matriz):
26      """
27      Esta función recibe una matriz (lista de listas)
28      y devuelve una lista con la suma de cada fila.
29      Ejemplo:
30      matriz = [[1, 2, 3], [4, 5, 6]]
31      resultado = [6, 15]
32      """
33      resultado = []
34      for fila in matriz:
35          suma_fila = sum(fila) # Suma todos los elementos de la fila
36          resultado.append(suma_fila)
37      return resultado
38
39  # Función de prueba para verificar que sumar_por_filas funciona correctamente
40  def probar_suma_por_filas():
41      print("\nProbando sumar_por_filas...")
42
43      # Caso 1: matriz con 3 filas y 3 columnas
44      m1 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
45      assert sumar_por_filas(m1) == [6, 15, 24] # 1+2+3, 4+5+6, 7+8+9
```

Clase 10 ejercicio 3

suma de la diagonal principal



```
~/workspace$ python clase10_operacionesconmatrices.py
Probando sumar_total_matriz...
¡Pruebas para sumar_total_matriz pasaron!

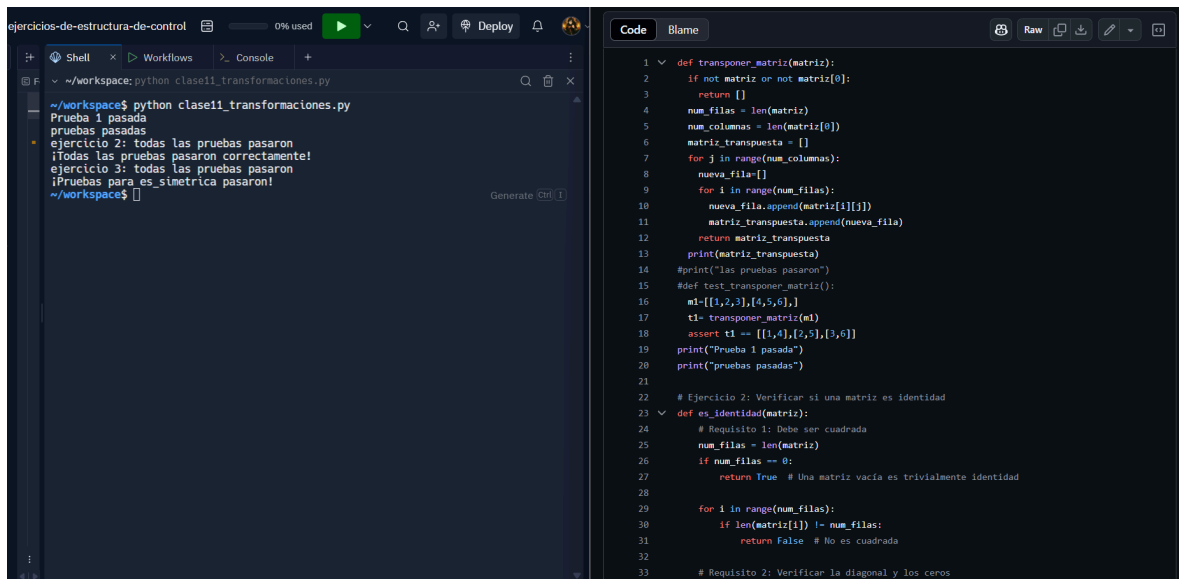
Probando sumar_por_filas...
¡Pruebas para sumar_por_filas pasaron!
ejercicio 2: ¡Todo está correcto!

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

```
81 def sumar_diagonal_principal(matriz):
82     [3, 4]
83     diagonal principal: 1 y 4 + suma = 5
84     ---
85     suma = 0
86     for i in range(len(matriz)):
87         suma += matriz[i][i] # Accede al elemento en la posición (i, i)
88     return suma
89
90 # Función de prueba para verificar que sumar_diagonal_principal funciona correctamente
91 def probar_suma_diagonal_principal():
92     print("\nProbando sumar_diagonal_principal...")
93
94     # Caso 1: matriz 3x3 con números consecutivos
95     m1 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
96     assert sumar_diagonal_principal(m1) == 15 # 1 + 5 + 9
97
98     # Caso 2: matriz 2x2 con ceros y valores definidos
99     m2 = [[10, 0], [0, 20]]
100    assert sumar_diagonal_principal(m2) == 30 # 10 + 20
101
102    # Caso borde: matriz 1x1
103    m3 = [[5]]
104    assert sumar_diagonal_principal(m3) == 5 # Solo un elemento en la diagonal
105
106    print("¡Pruebas para sumar_diagonal_principal pasaron!")
107
108    # Llamamos a la función para ejecutar las pruebas
109    probar_suma_diagonal_principal()
```

Ejercicios de la clase 11

- transpuesta matriz



```
~/workspace$ python clase11_transformaciones.py
Prueba 1 pasada
pruebas pasadas
ejercicio 2: todas las pruebas pasaron
¡Todas las pruebas pasaron correctamente!
ejercicio 3: todas las pruebas pasaron
¡Pruebas para es_simetrica pasaron!
~/workspace$
```

```
1 def transponer_matriz(matriz):
2     if not matriz or not matriz[0]:
3         return []
4     num_filas = len(matriz)
5     num_columnas = len(matriz[0])
6     matriz_transpuesta = []
7     for j in range(num_columnas):
8         nueva_fila = []
9         for i in range(num_filas):
10            nueva_fila.append(matriz[i][j])
11            matriz_transpuesta.append(nueva_fila)
12    return matriz_transpuesta
13    print(matriz_transpuesta)
14    #print("las pruebas pasaron")
15    #def test_transponer_matriz():
16    m1 = [[1,2,3],[4,5,6],[]]
17    t1 = transponer_matriz(m1)
18    assert t1 == [[1,4],[2,5],[3,6]]
19    print("Prueba 1 pasada")
20    print("pruebas pasadas")
21
22    # Ejercicio 2: Verificar si una matriz es identidad
23    def es_identidad(matriz):
24        # Requisito 1: Debe ser cuadrada
25        num_filas = len(matriz)
26        if num_filas == 0:
27            return True # Una matriz vacía es trivialmente identidad
28
29        for i in range(num_filas):
30            if len(matriz[i]) != num_filas:
31                return False # No es cuadrada
32
33        # Requisito 2: Verificar la diagonal y los ceros
```


Clase 11 ejercicio 2

- verificación si una matriz es identidad

```
~/workspace$ python clase11_transformaciones.py
Prueba 1 pasada
pruebas pasadas
ejercicio 2: todas las pruebas pasaron
¡Todas las pruebas pasaron correctamente!
ejercicio 3: todas las pruebas pasaron
¡Pruebas para es_simetrica pasaron!
~/workspace$
```

```
33 # Requisito 2: Verificar la diagonal y los ceros
34 for i in range(num_filas):
35     for j in range(num_filas):
36         if i == j:
37             if matriz[i][j] != 1:
38                 return False # La diagonal no tiene 1
39         else:
40             if matriz[i][j] != 0:
41                 return False # Elemento fuera de la diagonal no es 0
42
43     return True # Cumple con todas las condiciones de identidad
44
45 # Pruebas
46 identidad = [[1, 0, 0], [0, 1, 0], [0, 0, 1]]
47 no_identities = [[1, 0, 0], [0, 2, 0], [0, 0, 1]]
48 no_cuadrada = [[1, 0], [0, 1], [0, 0]]
49
50 assert es_identities(identidad) == True
51 assert es_identities(no_identities) == False
52 assert es_identities(no_cuadrada) == False
53
54 print("ejercicio 2: todas las pruebas pasaron")
55 print("¡Todas las pruebas pasaron correctamente!")
56
57
58 # Ejercicio 3: Verificar si una matriz es simétrica
59 def es_simetrica(matriz):
60     # Requisito 1: Debe ser cuadrada
61     num_filas = len(matriz)
62     if num_filas == 0:
```

Clase 11 ejercicio 3

- identificar si una matriz es simétrica

```
~/workspace$ python clase11_transformaciones.py
Prueba 1 pasada
pruebas pasadas
ejercicio 2: todas las pruebas pasaron
¡Todas las pruebas pasaron correctamente!
ejercicio 3: todas las pruebas pasaron
¡Pruebas para es_simetrica pasaron!
~/workspace$
```

```
23 def es_identities(matriz):
24     # Ejercicio 3: Verificar si una matriz es simétrica
25     def es_simetrica(matriz):
26         # Requisito 1: Debe ser cuadrada
27         num_filas = len(matriz)
28         if num_filas == 0:
29             return True # Una matriz vacía es trivialmente simétrica
30
31         for i in range(num_filas):
32             if len(matriz[i]) != num_filas:
33                 return False # No es cuadrada
34
35         # Requisito 2: Comparar matriz[i][j] con matriz[j][i]
36         for i in range(num_filas):
37             for j in range(i + 1, num_filas): # Solo verificar la parte superior
38                 if matriz[i][j] != matriz[j][i]:
39                     return False # Con una diferencia basta
40
41         return True # Si todo coincide, es simétrica
42
43 # Pruebas
44 sim = [[1, 7, 3], [7, 4, -5], [3, -5, 6]]
45 no_sim = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
46 no_cuadrada = [[1, 2], [3, 4], [5, 6]]
47
48 assert es_simetrica(sim)
49 assert not es_simetrica(no_sim)
50 assert not es_simetrica(no_cuadrada)
51
52 print("ejercicio 3: todas las pruebas pasaron")
53 print("¡Pruebas para es_simetrica pasaron!")
```

Ejercicios de la clase 12

Diccionario

```
1 # Modelo 1: Canción
2 cancion = {
3     "titulo": "Caminando bajo la lluvia",
4     "artista": "Luna Morada",
5     "album": "Sueños Acústicos",
6     "duracion_segundos": 245,
7     "genero": "Indie Pop",
8     "colaboradores": ["Nico Beat", "Valeria Flow"],
9     "fecha_lanzamiento": "2023-11-15",
10    "reproducciones": 358726
11 }
12
13 # Modelo 2: Coche
14 coche = {
15     "marca": "Toyota",
16     "modelo": "Corolla Hybrid",
17     "año": 2021,
18     "color": "Azul Medianoche",
19     "placa": "CBZ-854",
20     "kilometraje": 32500,
21     "dueños_previos": 1,
22     "servicios_regulares": ["2022-06-01", "2023-01-15", "2023-12-05"],
23     "electricidad": True
24 }
25
26 # Modelo 3: Post de Red Social
27 post = {
28     "id_post": 304,
29     "autor": "Harumi",
30     "contenido_texto": "¡Hoy perdí el miedo a postear mi primer libro! #book #feliz",
31     "lista_de_likes": ["lucas_dev", "ani.lu", "mariox23", "sofia_dsg"],
32     "fecha_publicacion": "2025-06-24",
33     "comentarios": [
34         {"usuario": "lucas_dev", "comentario": "¡Esooo! ¡Felicidades!"},
35         {"usuario": "ani.lu", "comentario": "Qué crack, Harumi!"}
36     ],
37     "publicado_desde": "Web"
38 }
39
40 # Función para imprimir diccionarios con formato
41 def imprimir_diccionario(titulo, diccionario):
42     print(f"\n--- {titulo} ---")
43     for clave, valor in diccionario.items():
44         print(f"{clave}: {valor}")
45
46 # Mostrar cada modelo
47 imprimir_diccionario("Canción", cancion)
48 imprimir_diccionario("Coche", coche)
49 imprimir_diccionario("Post de Red Social", post)
```

```
ejercicios-de-estructura-de-control / clase12_diccionarios.py
Code Blame Raw
14 coche = {
22     "servicios_regulares": ["2022-06-01", "2023-01-15", "2023-12-05"],
23     "electricidad": True
24 }
25
26 # Modelo 3: Post de Red Social
27 post = {
28     "id_post": 304,
29     "autor": "Harumi",
30     "contenido_texto": "¡Hoy perdí el miedo a postear mi primer libro! #book #feliz",
31     "lista_de_likes": ["lucas_dev", "ani.lu", "mariox23", "sofia_dsg"],
32     "fecha_publicacion": "2025-06-24",
33     "comentarios": [
34         {"usuario": "lucas_dev", "comentario": "¡Esooo! ¡Felicidades!"},
35         {"usuario": "ani.lu", "comentario": "Qué crack, Harumi!"}
36     ],
37     "publicado_desde": "Web"
38 }
39
40 # Función para imprimir diccionarios con formato
41 def imprimir_diccionario(titulo, diccionario):
42     print(f"\n--- {titulo} ---")
43     for clave, valor in diccionario.items():
44         print(f"{clave}: {valor}")
45
46 # Mostrar cada modelo
47 imprimir_diccionario("Canción", cancion)
48 imprimir_diccionario("Coche", coche)
49 imprimir_diccionario("Post de Red Social", post)
```

```
Shell x Workflows _ Console +
~/workspace:python clase12_diccionarios.py
~/workspace$ python clase12_diccionarios.py

--- Canción ---
titulo: Caminando bajo la lluvia
artista: Luna Morada
album: Sueños Acústicos
duracion_segundos: 245
genero: Indie Pop
colaboradores: ['Nico Beat', 'Valeria Flow']
fecha_lanzamiento: 2023-11-15
reproducciones: 358726

--- Coche ---
marca: Toyota
modelo: Corolla Hybrid
año: 2021
color: Azul Medianoche
placa: CBZ-854
kilometraje: 32500
dueños_previos: 1
servicios_regulares: ['2022-06-01', '2023-01-15', '2023-12-05']
electricidad: True

--- Post de Red Social ---
id_post: 304
autor: Harumi
contenido_texto: ¡Hoy perdí el miedo a postear mi primer libro! #book #feliz
lista_de_likes: ['lucas_dev', 'ani.lu', 'mariox23', 'sofia_dsg']
fecha_publicacion: 2025-06-24
comentarios: [{'usuario': 'lucas_dev', 'comentario': '¡Esooo! ¡Felicidades!'}, {'usua
rio': 'ani.lu', 'comentario': 'Qué crack, Harumi!'}]
publicado_desde: Web
~/workspace$
```

```
ejercicios-de-estructura-de-control 0% used
Shell x Workflows _ Console +
~/workspace:python clase12_diccionarios.py
~/workspace$ python clase12_diccionarios.py

--- Canción ---
titulo: Caminando bajo la lluvia
artista: Luna Morada
album: Sueños Acústicos
duracion_segundos: 245
genero: Indie Pop
colaboradores: ['Nico Beat', 'Valeria Flow']
fecha_lanzamiento: 2023-11-15
reproducciones: 358726

--- Coche ---
marca: Toyota
modelo: Corolla Hybrid
año: 2021
color: Azul Medianoche
placa: CBZ-854
kilometraje: 32500
dueños_previos: 1
servicios_regulares: ['2022-06-01', '2023-01-15', '2023-12-05']
electricidad: True

--- Post de Red Social ---
id_post: 304
autor: Harumi
contenido_texto: ¡Hoy perdí el miedo a postear mi primer libro! #book #feliz
lista_de_likes: ['lucas_dev', 'ani.lu', 'mariox23', 'sofia_dsg']
fecha_publicacion: 2025-06-24
comentarios: [{'usuario': 'lucas_dev', 'comentario': '¡Esooo! ¡Felicidades!'}, {'usua
rio': 'ani.lu', 'comentario': 'Qué crack, Harumi!'}]
publicado_desde: Web
~/workspace$
```

Ejercicios de la clase 13

Gestor de Tareas

- Agregar Tarea
- Mostrar tarea
- Buscar tarea por id

```
1 lista_de_tareas = []
2 proximo_id_tarea = 1 # Para generar IDs únicos
3
4 def agregar_tarea(descripcion, prioridad="media"):
5     global proximo_id_tarea # Necesario para modificar la variable global
6     nueva_tarea = {
7         "id": proximo_id_tarea,
8         "descripcion": descripcion,
9         "completada": False,
10        "prioridad": prioridad
11    }
12    lista_de_tareas.append(nueva_tarea)
13    proximo_id_tarea += 1
14    print(f" Tarea '{descripcion}' añadida con éxito.")
15
16 def mostrar_tareas():
17     print("\n--- LISTA DE TAREAS ---")
18     if not lista_de_tareas:
19         print("No hay tareas pendientes! ¡A disfrutar!")
20         return
21
22     for tarea in lista_de_tareas:
23         estado = "✓" if tarea["completada"] else "✗"
24         print(f"{estado} ID: {tarea['id']} | {tarea['descripcion']} (Prioridad: {tarea['prioridad']})")
25
26 def buscar_tarea_por_id(id_buscado):
27     """Devuelve la tarea con el ID buscado o None si no existe"""
28     for tarea in lista_de_tareas:
29         if tarea["id"] == id_buscado:
30             return tarea
31     return None
32
33
34 def marcar_tarea_completada(id_tarea):
35     tarea = buscar_tarea_por_id(id_tarea)
36     if tarea:
37         tarea["completada"] = True
38         print(f" Tarea '{tarea['descripcion']}' marcada como completada.")
39     else:
40         print(f" Error: No se encontró la tarea con ID {id_tarea}.")
41
42 def eliminar_tarea(id_tarea):
43     tarea = buscar_tarea_por_id(id_tarea)
44     if tarea:
45         lista_de_tareas.remove(tarea)
46         print(f" Tarea '{tarea['descripcion']}' eliminada.")
47     else:
48         print(f" Error: No se encontró la tarea con ID {id_tarea}.")
49
50 # Pruebas iniciales
51 agregar_tarea("Estudiar para el examen de Cálculo")
52 agregar_tarea("Hacer las compras", prioridad="alta")
53 mostrar_tareas()
54
55 tarea_encontrada = buscar_tarea_por_id(1)
56 if tarea_encontrada:
57     print(f"\nBúsqueda exitosa: {tarea_encontrada['descripcion']}")
58 else:
59     print("\nBúsqueda fallida: Tarea no encontrada.")
60
61 tarea_fantasma = buscar_tarea_por_id(99)
62 if not tarea_fantasma:
```

```
~/workspace$ python clase13_registros.py
Tarea 'Estudiar para el examen de Cálculo' añadida con éxito.
Tarea 'Hacer las compras' añadida con éxito.

--- LISTA DE TAREAS ---
✗ ID: 1 | Estudiar para el examen de Cálculo (Prioridad: media)
✗ ID: 2 | Hacer las compras (Prioridad: alta)

Búsqueda exitosa: Estudiar para el examen de Cálculo
Búsqueda de tarea inexistente funcionó correctamente.
Tarea 'Estudiar para el examen de Cálculo' marcada como completada.

--- LISTA DE TAREAS ---
✓ ID: 1 | Estudiar para el examen de Cálculo (Prioridad: media)
✗ ID: 2 | Hacer las compras (Prioridad: alta)

Tarea 'Hacer las compras' eliminada.

--- LISTA DE TAREAS ---
✓ ID: 1 | Estudiar para el examen de Cálculo (Prioridad: media)

Error: No se encontró la tarea con ID 99.

===== MENU 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: pdf
Prioridad (alta, media, baja): alta
Tarea 'pdf' añadida con éxito.

===== MENU 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
```

CodeBlame

42def eliminar_tarea(id_tarea):

43

44 if not tarea_fantasma:

45 print("Búsqueda de tarea inexistente funcionó correctamente.")

46

47 marcar_tarea_completada(1)

48 mostrar_tareas() # Tarea 1 como completada

49 eliminar_tarea(2)

50 mostrar_tareas() # Tarea 2 eliminada

51 marcar_tarea_completada(99) # Intentar marcar tarea inexistente

52

53 # Menú interactivo

54 while True:

55 print("\n===== MENÚ TO-DO LIST =====")

56 print("1. Agregar nueva tarea")

57 print("2. Mostrar todas las tareas")

58 print("3. Marcar tarea como completada")

59 print("4. Eliminar tarea")

60 print("0. Salir")

61 opcion = input("Elige una opción: ")

62 if opcion == '1':

63 desc = input("Descripción de la nueva tarea: ")

64 prio = input("Prioridad (alta, media, baja): ").lower()

65 if prio not in ["alta", "media", "baja"]:

66 prio = "media"

67 agregar_tarea(desc, prio)

68 elif opcion == '2':

69 mostrar_tareas()

70 elif opcion == '3':

71 try:

72 id_t = int(input("ID de la tarea a completar: "))

73 marcar_tarea_completada(id_t)

74 except ValueError:

75 print("Por favor ingresa un número válido para el ID.")

76 elif opcion == '4':

77 try:

78 id_t = int(input("ID de la tarea a eliminar: "))

79 eliminar_tarea(id_t)

80 except ValueError:

81 print("Por favor ingresa un número válido para el ID.")

82 elif opcion == '0':

83 print("¡Hasta pronto!")

84 break

85 else:

86 print("Opción no válida. Intentalo de nuevo.")

87

Shell

Workflows

Console

+

For

~/workspace:python clase13_registros.py

LISTA DE TAREAS ---

✓ ID: 1 | Estudiar para el examen de Cálculo (Prioridad: media)

Error: No se encontró la tarea con ID 99.

=====

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: pdf

Prioridad (alta, media, baja): alta

Tarea 'pdf' 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 | Estudiar para el examen de Cálculo (Prioridad: media)

✗ ID: 3 | pdf (Prioridad: alta)

=====

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

Files

main

ejercicios-de-estructura-de-control / clase13_registros.py

↑ Top

CodeBlame

42def eliminar_tarea(id_tarea):

43

44 if not tarea_fantasma:

45 print("Búsqueda de tarea inexistente funcionó correctamente.")

46

47 marcar_tarea_completada(1)

48 mostrar_tareas() # Tarea 1 como completada

49 eliminar_tarea(2)

50 mostrar_tareas() # Tarea 2 eliminada

51 marcar_tarea_completada(99) # Intentar marcar tarea inexistente

52

53 # Menú interactivo

54 while True:

55 print("\n===== MENÚ TO-DO LIST =====")

56 print("1. Agregar nueva tarea")

57 print("2. Mostrar todas las tareas")

58 print("3. Marcar tarea como completada")

59 print("4. Eliminar tarea")

60 print("0. Salir")

61 opcion = input("Elige una opción: ")

62 if opcion == '1':

63 desc = input("Descripción de la nueva tarea: ")

64 prio = input("Prioridad (alta, media, baja): ").lower()

65 if prio not in ["alta", "media", "baja"]:

66 prio = "media"

67 agregar_tarea(desc, prio)

68 elif opcion == '2':

69 mostrar_tareas()

70 elif opcion == '3':

71 try:

72 id_t = int(input("ID de la tarea a completar: "))

73 marcar_tarea_completada(id_t)

74 except ValueError:

75 print("Por favor ingresa un número válido para el ID.")

76 elif opcion == '4':

77 try:

78 id_t = int(input("ID de la tarea a eliminar: "))

79 eliminar_tarea(id_t)

80 except ValueError:

81 print("Por favor ingresa un número válido para el ID.")

82 elif opcion == '0':

83 print("¡Hasta pronto!")

84 break

85 else:

86 print("Opción no válida. Intentalo de nuevo.")

87

ejercicios-de-estructura-de-control

0% used

Shell

Workflows

Console

+

For

~/workspace:python clase13_registros.py

Elige una opción: 1

Descripción de la nueva tarea: pdf

Prioridad (alta, media, baja): alta

Tarea 'pdf' 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 | Estudiar para el examen de Cálculo (Prioridad: media)

✗ ID: 3 | pdf (Prioridad: alta)

=====

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: 3

ID de la tarea a completar: 3

Tarea 'pdf' marcada como completada.

=====

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!

~/workspace\$

Suma diagonal secundaria

The screenshot shows a code editor with a terminal on the left and a code editor on the right. The terminal displays the command `python sumadiagonalsecundaria.py` and the output: `Probando sumar_diagonal_secundaria... ¡Pruebas para sumar_diagonal_secundaria pasaron!`. The code editor shows the following Python code:

```
1 # Función que suma los elementos de la diagonal secundaria de una matriz cuadrada
2 def sumar_diagonal_secundaria(matriz):
3     """
4     Recibe una matriz cuadrada (misma cantidad de filas y columnas)
5     y devuelve la suma de los elementos en la diagonal secundaria.
6     La diagonal secundaria va desde la esquina superior derecha
7     hasta la esquina inferior izquierda.
8
9     Por ejemplo, en una matriz 3x3:
10    [[a, b, c],
11     [d, e, f],
12     [g, h, i]]
13    La diagonal secundaria está en las posiciones: (0,2), (1,1), (2,0)
14    y su suma sería: c + e + g
15    """
16    n = len(matriz) # Número de filas (y columnas, ya que es cuadrada)
17    suma = 0
18    for i in range(n):
19        suma += matriz[i][n - 1 - i] # Accede al elemento en la posición (i, n-1-i)
20    return suma
21
22 # Función de pruebas para validar que sumar_diagonal_secundaria funciona correctamente
23 def probar_suma_diagonal_secundaria():
24     print("\nProbando sumar_diagonal_secundaria...")
25
26     # Caso 1: matriz 3x3 normal
27     m1 = [
28         [1, 2, 3],
29         [4, 5, 6],
30         [7, 8, 9]
31     ]
32     # Diagonal secundaria: 3 + 5 + 7 = 15
33     assert sumar_diagonal_secundaria(m1) == 15
```

The screenshot shows a code editor with a terminal on the left and a code editor on the right. The terminal displays the command `python sumadiagonalsecundaria.py` and the output: `Probando sumar_diagonal_secundaria... ¡Pruebas para sumar_diagonal_secundaria pasaron!`. The code editor shows the following Python code:

```
23 def probar_suma_diagonal_secundaria():
24     print("\nProbando sumar_diagonal_secundaria...")
25
26     # Caso 1: matriz 3x3 normal
27     m1 = [
28         [1, 2, 3],
29         [4, 5, 6],
30         [7, 8, 9]
31     ]
32     # Diagonal secundaria: 3 + 5 + 7 = 15
33     assert sumar_diagonal_secundaria(m1) == 15
34
35     # Caso 2: matriz 2x2
36     m2 = [
37         [10, 1],
38         [2, 20]
39     ]
40     # Diagonal secundaria: 1 + 20 = 21
41     assert sumar_diagonal_secundaria(m2) == 21
42
43     # Caso 3: matriz 1x1
44     m3 = [[42]]
45     # Solo hay un elemento: 42
46     assert sumar_diagonal_secundaria(m3) == 42
47
48     print("\n¡Pruebas para sumar_diagonal_secundaria pasaron!")
49
50 # Llamamos a la función de prueba
51 probar_suma_diagonal_secundaria()
```

Gestión de asientos de cine

```
~/workspace$ python asientos_decine.py
C0 C1 C2 C3 C4 C5 C6 C7
F0 | L L L L L L L L
F1 | L L L L L L L L
F2 | L L L L L L L L
F3 | L L L L L L L L
F4 | L L L L L L L L

MENÚ DE OPCIONES
1. Ocupar un asiento
0. Salir
Selecciona una opción: 0
```

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

def mostrar_sala(sala):
    print("\n   " + " ".join(f"C{j}" for j in range(len(sala[0])))
    print(" " + "-" * len(sala[0]))
    for i, fila in enumerate(sala):
        print(f"F{i} | " + " ".join(fila))
    print()

def ocupar_asiento(sala, fila, columna):
    filas = len(sala)
    columnas = len(sala[0])
    if fila < 0 or fila >= filas or columna < 0 or columna >= columnas:
        print("⚠️ Coordinadas fuera de rango.")
        return False
    if sala[fila][columna] == 'O':
        print("❌ Ese asiento ya está ocupado.")
        return False
    sala[fila][columna] = 'O'
    print(f"✅ Asiento F{fila} C{columna} ocupado correctamente.")
    return True

# --- Programa Principal ---
sala = crear_sala(5, 8) # Sala de 5 filas x 8 columnas

while True:
    mostrar_sala(sala)
    print("📌 MENÚ DE OPCIONES")
    print("1. Ocupar un asiento")
    print("0. Salir")
```

```
~/workspace$ python asientos_decine.py
C0 C1 C2 C3 C4 C5 C6 C7
F0 | L L L L L L L L
F1 | L L L L L L L L
F2 | L L L L L L L L
F3 | L L L L L L L L
F4 | L L L L L L L L

MENÚ DE OPCIONES
1. Ocupar un asiento
0. Salir
Selecciona una opción: 1
Ingresa la fila del asiento (ej. 0 a 4): 0
Ingresa la columna del asiento (ej. 0 a 7): 5
✅ Asiento F0 C5 ocupado correctamente.

C0 C1 C2 C3 C4 C5 C6 C7
F0 | L L L L L O L L
F1 | L L L L L L L L
F2 | L L L L L L L L
F3 | L L L L L L L L
F4 | L L L L L L L L

MENÚ DE OPCIONES
1. Ocupar un asiento
0. Salir
Selecciona una opción: 0
👋 ¡Gracias por usar el sistema de cine! Hasta luego.
~/workspace$
```

```
def ocupar_asiento(sala, fila, columna):
    ...
    sala[fila][columna] = 'O'
    print(f"✅ Asiento F{fila} C{columna} ocupado correctamente.")
    return True

# --- Programa Principal ---
sala = crear_sala(5, 8) # Sala de 5 filas x 8 columnas

while True:
    mostrar_sala(sala)
    print("📌 MENÚ DE OPCIONES")
    print("1. Ocupar un asiento")
    print("0. Salir")
    opcion = input("Selecciona una opción: ")

    if opcion == '1':
        try:
            fila = int(input("Ingresa la fila del asiento (ej. 0 a 4): "))
            columna = int(input("Ingresa la columna del asiento (ej. 0 a 7): "))
            ocupar_asiento(sala, fila, columna)
        except ValueError:
            print("❗ Por favor, ingresa números válidos.")
    elif opcion == '0':
        print("👋 ¡Gracias por usar el sistema de cine! Hasta luego.")
        break
    else:
        print("❗ Opción inválida. Intenta de nuevo.")
```

Batalla Naval con persistencia

The image displays two screenshots of a code editor, likely VS Code, showing the development of a Battleship game in Python. The left screenshot shows the game's main menu and the initial board setup. The right screenshot shows the logic for placing ships and the game loop.

Left Screenshot:

```
ejercicios-de-estructura-de-control 0% used
```

```
~/workspace: git push
```

```
~/workspace$ python batalla_naval.py
```

```
== BATALLA NAVAL ==
1. Jugar vs CPU
2. Jugar 2 Jugadores
3. Continuar Partida
4. Salir
Elige una opción: 2
Nombre Jugador 1: sarai
Nombre Jugador 2: haru

[sarai COLOCA TUS BARCOS]
¿Manual (M) o Automático (A)? a

[haru COLOCA TUS BARCOS]
¿Manual (M) o Automático (A)? a

[sarai ATACA]

  1 2 3 4
A  ██████████
B  ██████████
C  ██████████
D  ██████████
sarai, disparo (A1) o escribe GUARDAR: A1
Agua 🌊

[haru ATACA]

  1 2 3 4
A  ██████████
B  ██████████
C  ██████████
D  ██████████
haru, disparo (A1) o escribe GUARDAR: b2
Agua 🌊
```

Right Screenshot:

```
Code Blame
```

```
1 import random
2 import json
3 import os
4
5 # Configuración
6 FILAS = 4
7 COLUMNAS = 4
8 NUM_BARCOS = 3
9 ARCHIVO_PARTIDA = "batalla_naval_save.json"
10
11 def crear_tablero():
12     return [{" " for _ in range(COLUMNAS)] for _ in range(FILAS)]
13
14 def mostrar_tablero(tablero, ocultar_barcos=False):
15     print("\n " + " ".join(str(i+1) for i in range(COLUMNAS)))
16     for i, fila in enumerate(tablero):
17         print(chr(65+i) + " ", end=" ")
18         for celda in fila:
19             if ocultar_barcos and celda == "▲":
20                 print(" ", end=" ")
21             else:
22                 print(celda, end=" ")
23         print()
24
25 def colocar_barcos(tablero, modo, jugador=""):
26     if modo == "auto":
27         for _ in range(NUM_BARCOS):
28             while True:
29                 fila = random.randint(0, FILAS-1)
30                 col = random.randint(0, COLUMNAS-1)
31                 if tablero[fila][col] == " " or "▲":
32                     tablero[fila][col] = "▲"
33                     break
```

```
ejercicios-de-estructura-de-control 0% used
~/workspace: git push
~/workspace$ python batalla_naval.py
=== BATALLA NAVAL ===
1. Jugar vs CPU
2. Jugar 2 Jugadores
3. Continuar Partida
4. Salir
Elige una opción: 2
Nombre Jugador 1: sarai
Nombre Jugador 2: haru

[sarai COLOCA TUS BARCOS]
¿Manual (M) o Automático (A)? a

[haru COLOCA TUS BARCOS]
¿Manual (M) o Automático (A)? a

[sarai ATACA]

  1 2 3 4
A  ██████
B  ██████
C  ██████
D  ██████
sarai, disparo (A1) o escribe GUARDAR: A1
Agua 🌊

[haru ATACA]

  1 2 3 4
A  ██████
B  ██████
C  ██████
D  ██████
haru, disparo (A1) o escribe GUARDAR: b2
Agua 🌊

def realizar_disparo(tablero, jugador):
    fila = ord(coord[0]) - 65
    col = int(coord[1]) - 1
    if not (0 <= fila < FILAS and 0 <= col < COLUMNAS):
        print("¡Coordenadas inválidas!")
        continue
    if tablero[fila][col] in ["*", "🚢"]:
        print("¡Ya disparaste ahí!")
        continue
    if tablero[fila][col] == "A":
        tablero[fila][col] = "🚢"
        print("¡Impacto! 🚢")
        return "Impacto"
    else:
        tablero[fila][col] = "🌊"
        print("Agua 🌊")
        return "agua"
    except:
        print("Formato inválido.")

def quedan_barcos(tablero):
    return any("A" in fila for fila in tablero)

def guardar_partida(datos):
    with open(ARCHIVO_PARTIDA, "w") as f:
        json.dump(datos, f)
    print("Partida guardada correctamente.")

def cargar_partida():
    if not os.path.exists(ARCHIVO_PARTIDA):
        return None
```

```
~/workspace$ python batalla_naval.py
=== BATALLA NAVAL ===
1. Jugar vs CPU
2. Jugar 2 Jugadores
3. Continuar Partida
4. Salir
Elige una opción: 2
Nombre Jugador 1: sarai
Nombre Jugador 2: haru

[sarai COLOCA TUS BARCOS]
¿Manual (M) o Automático (A)? a

[haru COLOCA TUS BARCOS]
¿Manual (M) o Automático (A)? a

[sarai ATACA]

  1 2 3 4
A  ██████
B  ██████
C  ██████
D  ██████
sarai, disparo (A1) o escribe GUARDAR: A1
Agua 🌊

[haru ATACA]

  1 2 3 4
A  ██████
B  ██████
C  ██████
D  ██████
haru, disparo (A1) o escribe GUARDAR: b2
Agua 🌊

def realizar_disparo(tablero, jugador):
    ...
def cargar_partida():
    if not os.path.exists(ARCHIVO_PARTIDA):
        return None
    try:
        with open(ARCHIVO_PARTIDA, "r") as f:
            return json.load(f)
    except:
        return None
def jugar_vs_cpu():
    nombre = input("Tu nombre: ")
    jugador = nombre
    tablero_jugador = crear_tablero()
    tablero_cpu = crear_tablero()
    disparos_cpu = []
    print("\nCOLOCACIÓN DE BARCOS")
    modo = input("Manual (M) o Automático (A)? ").upper()
    tablero_jugador = colocar_barcos(tablero_jugador, "manual" if modo == "M" else "auto", jugador)
    tablero_cpu = colocar_barcos(tablero_cpu, "auto")
    while True:
        print("\n¡TU TABLERO!")
        mostrar_tablero(tablero_jugador)
        print("\n¡TABLERO CPU!")
        mostrar_tablero(tablero_cpu, ocultar_barcos=True)
        resultado = realizar_disparo(tablero_cpu, jugador)
        if resultado == "Ganaste":
```

```
~/workspace$ python batalla_naval.py
=== BATALLA NAVAL ===
1. Jugar vs CPU
2. Jugar 2 Jugadores
3. Continuar Partida
4. Salir
Elige una opción: 2
Nombre Jugador 1: sarai
Nombre Jugador 2: haru

[sarai COLOCA TUS BARCOS]
¿Manual (M) o Automático (A)? a

[haru COLOCA TUS BARCOS]
¿Manual (M) o Automático (A)? a

[sarai ATACA]

  1 2 3 4
A  ██████
B  ██████
C  ██████
D  ██████
sarai, disparo (A1) o escribe GUARDAR: A1
Agua 🌊

[haru ATACA]

  1 2 3 4
A  ██████
B  ██████
C  ██████
D  ██████
haru, disparo (A1) o escribe GUARDAR: b2
Agua 🌊

def jugar_vs_cpu():
    ...
    if not quedan_barcos(tablero_cpu):
        print(f"¡{jugador} GANASTE! 🏆")
        return
    print("\nTurno de la CPU...")
    while True:
        fila = random.randint(0, FILAS-1)
        col = random.randint(0, COLUMNAS-1)
        if (fila, col) not in disparos_cpu:
            disparos_cpu.append((fila, col))
            break
        if tablero_jugador[fila][col] == "A":
            tablero_jugador[fila][col] = "🚢"
            print(f"CPU disparó en {chr(65+fila)}{col+1}: ¡Impacto!")
        else:
            tablero_jugador[fila][col] = "🌊"
            print(f"CPU disparó en {chr(65+fila)}{col+1}: Agua 🌊")
        if not quedan_barcos(tablero_jugador):
            print("\n¡LA CPU GANÓ! 🏆")
            return
def jugar_2_jugadores():
    jugador1 = input("Nombre Jugador 1: ")
    jugador2 = input("Nombre Jugador 2: ")
```



```
ejercicios-de-estructura-de-control 0% used
~/workspace: git push

~/workspace$ python batalla_naval.py

=== BATALLA NAVAL ===
1. Jugar vs CPU
2. Jugar 2 Jugadores
3. Continuar Partida
4. Salir
Elige una opción: 2
Nombre Jugador 1: sarai
Nombre Jugador 2: haru

[sarai COLOCA TUS BARCOS]
¿Manual (M) o Automático (A)? a

[haru COLOCA TUS BARCOS]
¿Manual (M) o Automático (A)? a

[sarai ATACA]

  1 2 3 4
A  ██████
B  ██████
C  ██████
D  ██████
sarai, disparo (A1) o escribe GUARDAR: A1
Agua :

[haru ATACA]

  1 2 3 4
A  ██████
B  ██████
C  ██████
D  ██████
haru, disparo (A1) o escribe GUARDAR: b2
Agua :

def jugar_vs_cpu():
def jugar_2_jugadores():
    jugador1 = input("Nombre Jugador 1: ")
    jugador2 = input("Nombre Jugador 2: ")

    tablero1 = crear_tablero()
    tablero2 = crear_tablero()

    print(f"\n[{jugador1}] COLOCA TUS BARCOS")
    modo = input("Manual (M) o Automático (A)? ").upper()
    tablero1 = colocar_barcos(tablero1, "manual" if modo == "M" else "auto", jugador1)

    print(f"\n[{jugador2}] COLOCA TUS BARCOS")
    modo = input("Manual (M) o Automático (A)? ").upper()
    tablero2 = colocar_barcos(tablero2, "manual" if modo == "M" else "auto", jugador2)

    while True:
        print(f"\n[{jugador1}] ATACA")
        mostrar_tablero(tablero2, ocultar_barcos=True)
        if realizar_disparo(tablero2, jugador1) == "guardar":
            guardar_partida({
                "modo": "2jugadores",
                "jugador1": jugador1,
                "jugador2": jugador2,
                "tablero1": tablero1,
                "tablero2": tablero2
            })
        return
        if not quedan_barcos(tablero2):
            print(f"\n[{jugador1}] GANÓ! 🏆")
```

```
ejercicios-de-estructura-de-control 0% used
~/workspace: git push

~/workspace$ python batalla_naval.py

=== BATALLA NAVAL ===
1. Jugar vs CPU
2. Jugar 2 Jugadores
3. Continuar Partida
4. Salir
Elige una opción: 2
Nombre Jugador 1: sarai
Nombre Jugador 2: haru

[sarai COLOCA TUS BARCOS]
¿Manual (M) o Automático (A)? a

[haru COLOCA TUS BARCOS]
¿Manual (M) o Automático (A)? a

[sarai ATACA]

  1 2 3 4
A  ██████
B  ██████
C  ██████
D  ██████
sarai, disparo (A1) o escribe GUARDAR: A1
Agua :

[haru ATACA]

  1 2 3 4
A  ██████
B  ██████
C  ██████
D  ██████
haru, disparo (A1) o escribe GUARDAR: b2
Agua :

def jugar_2_jugadores():
    if not quedan_barcos(tablero2):
        print(f"\n[{jugador1}] GANÓ! 🏆")
        return

    print(f"\n[{jugador2}] ATACA")
    mostrar_tablero(tablero1, ocultar_barcos=True)
    if realizar_disparo(tablero1, jugador2) == "guardar":
        guardar_partida({
            "modo": "2jugadores",
            "jugador1": jugador1,
            "jugador2": jugador2,
            "tablero1": tablero1,
            "tablero2": tablero2
        })
    return
    if not quedan_barcos(tablero1):
        print(f"\n[{jugador2}] GANÓ! 🏆")
        return

def continuar_partida():
    partida = cargar_partida()
    if not partida:
        print("No hay partida guardada.")
        return

    if partida["modo"] == "cpu":
        partida["tablero_jugador"] = [[celda for celda in fila] for fila in partida["tablero_cpu"]]
        partida["tablero_cpu"] = [[celda for celda in fila] for fila in partida["tablero_cpu"]]
        jugar_vs_cpu()
    elif partida["modo"] == "2jugadores":
```

ejercicios-de-estructura-de-control0% used

Deploy

ShellWorkflowsConsole

~/workspace: git push

~/workspace\$ python batalla_naval.py

=== BATALLA NAVAL ===

1. Jugar vs CPU

2. Jugar 2 Jugadores

3. Continuar Partida

4. Salir

Elige una opción: 2

Nombre Jugador 1: sarai

Nombre Jugador 2: haru

[sarai COLOCA TUS BARCOS]

¿Manual (M) o Automático (A)? a

[haru COLOCA TUS BARCOS]

¿Manual (M) o Automático (A)? a

[sarai ATACA]

1 2 3 4

A

B

C

D

sarai, disparo (A1) o escribe GUARDAR: A1

Agua

[haru ATACA]

1 2 3 4

A

B

C

D

haru, disparo (A1) o escribe GUARDAR: b2

Agua

?

Filesmainejercicios-de-estructura-de-control / batalla_naval.py

CodeBlame

RawDownloadEdit

197 def continuar_partida():

198 jugar_vs_cpu()

200

207 elif partida["modo"] == "2jugadores":

208 jugar_2_jugadores()

209

210 def menu_principal():

211 while True:

212 print("\n=== BATALLA NAVAL ===")

213 print("1. Jugar vs CPU")

214 print("2. Jugar 2 Jugadores")

215 print("3. Continuar Partida")

216 print("4. Salir")

217 opcion = input("Elige una opción: ")

218

219 if opcion == "1":

220 jugar_vs_cpu()

221

222 elif opcion == "2":

223 jugar_2_jugadores()

224

225 elif opcion == "3":

226 continuar_partida()

227

228 elif opcion == "4":

229 print("¡Gracias por jugar!")

230 break

231

232 else:

233 print("Opción inválida. Intenta otra vez.")

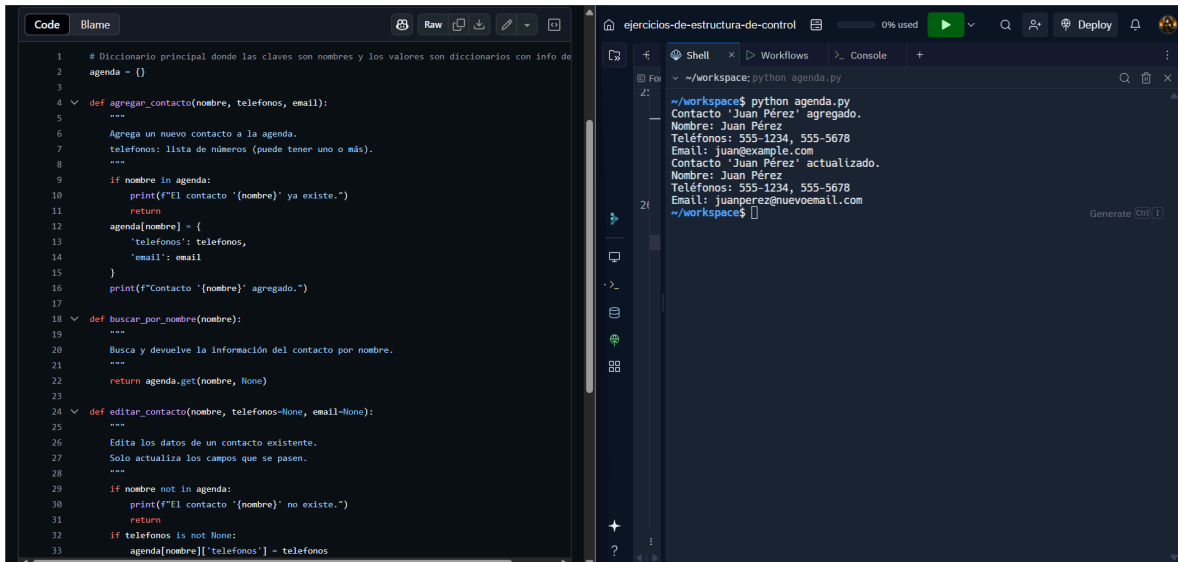
234

235 # Corrección final: esta es la forma correcta

236 if __name__ == "__main__":

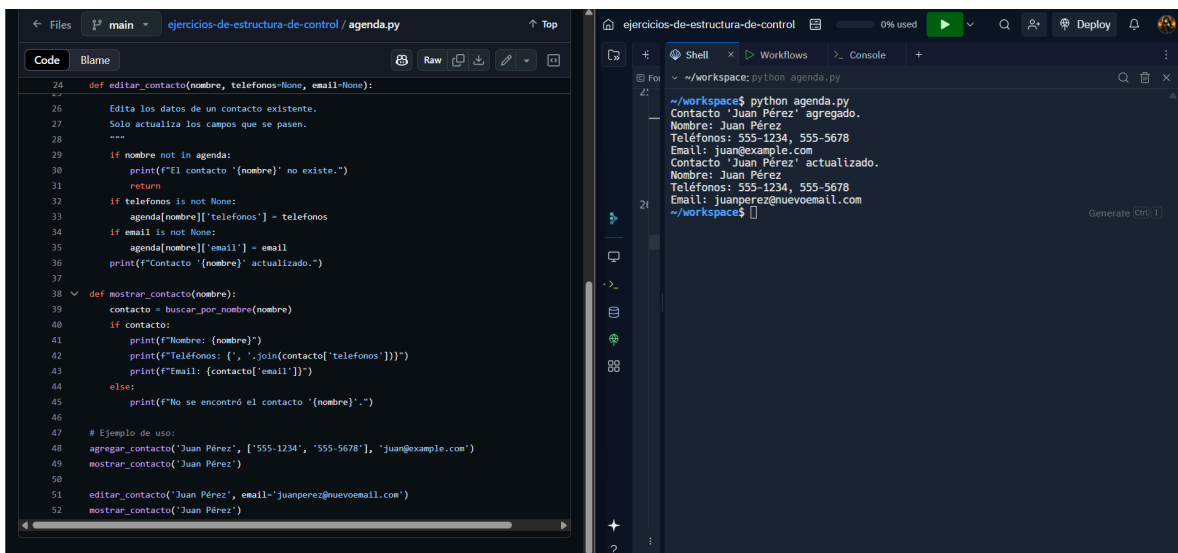
237 menu_principal()

Agenda



```
1 # Diccionario principal donde las claves son nombres y los valores son diccionarios con info de
2 agenda = {}
3
4 def agregar_contacto(nombre, telefonos, email):
5     """
6     Agrega un nuevo contacto a la agenda.
7     telefonos: lista de números (puede tener uno o más).
8     """
9     if nombre in agenda:
10         print(f"El contacto '{nombre}' ya existe.")
11         return
12     agenda[nombre] = {
13         'telefonos': telefonos,
14         'email': email
15     }
16     print(f"Contacto '{nombre}' agregado.")
17
18 def buscar_por_nombre(nombre):
19     """
20     Busca y devuelve la información del contacto por nombre.
21     """
22     return agenda.get(nombre, None)
23
24 def editar_contacto(nombre, telefonos=None, email=None):
25     """
26     Edita los datos de un contacto existente.
27     Solo actualiza los campos que se pasen.
28     """
29     if nombre not in agenda:
30         print(f"El contacto '{nombre}' no existe.")
31         return
32     if telefonos is not None:
33         agenda[nombre]['telefonos'] = telefonos
```

```
~/workspace$ python agenda.py
Contacto 'Juan Pérez' agregado.
Nombre: Juan Pérez
Teléfonos: 555-1234, 555-5678
Email: juan@example.com
Contacto 'Juan Pérez' actualizado.
Nombre: Juan Pérez
Teléfonos: 555-1234, 555-5678
Email: juanperez@nuevoemail.com
~/workspace$
```



```
24 def editar_contacto(nombre, telefonos=None, email=None):
25     """
26     Edita los datos de un contacto existente.
27     Solo actualiza los campos que se pasen.
28     """
29     if nombre not in agenda:
30         print(f"El contacto '{nombre}' no existe.")
31         return
32     if telefonos is not None:
33         agenda[nombre]['telefonos'] = telefonos
34     if email is not None:
35         agenda[nombre]['email'] = email
36     print(f"Contacto '{nombre}' actualizado.")
37
38 def mostrar_contacto(nombre):
39     contacto = buscar_por_nombre(nombre)
40     if contacto:
41         print(f"Nombre: {nombre}")
42         print(f"Teléfonos: {' '.join(contacto['telefonos'])}")
43         print(f"Email: {contacto['email']}")
44     else:
45         print(f"No se encontró el contacto '{nombre}'.")
46
47 # Ejemplo de uso:
48 agregar_contacto('Juan Pérez', ['555-1234', '555-5678'], 'juan@example.com')
49 mostrar_contacto('Juan Pérez')
50
51 editar_contacto('Juan Pérez', email='juanperez@nuevoemail.com')
52 mostrar_contacto('Juan Pérez')
```

```
~/workspace$ python agenda.py
Contacto 'Juan Pérez' agregado.
Nombre: Juan Pérez
Teléfonos: 555-1234, 555-5678
Email: juan@example.com
Contacto 'Juan Pérez' actualizado.
Nombre: Juan Pérez
Teléfonos: 555-1234, 555-5678
Email: juanperez@nuevoemail.com
~/workspace$
```

Teclado numérico

```
Shell x Workflows > Console +
~/workspace: python teclado.py
~/workspace$ python teclado.py
▶ MATRIZ DEL TECLADO:
1 2 3
4 5 6
7 8 9
* 0 #

▶ 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

▶ 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

EXPLICACIÓN:
matriz_compression = [[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.

~/workspace$
```

```
Code Blame
4
5 # 1. Declaramos una matriz que simula un teclado
6 teclado = [
7     [1, 2, 3],
8     [4, 5, 6],
9     [7, 8, 9],
10    ["*", 0, "#"]
11 ]
12
13 # 2. Imprimimos la matriz como cuadrícula
14 print("▶ MATRIZ DEL TECLADO:\n")
15 for fila in teclado:
16     for elemento in fila:
17         # Imprime cada elemento con tabulación, sin salto de línea
18         print(elemento, end=" ")
19     # Al final de cada fila, hacemos un salto de línea
20     print()
21
22 # 3. Crear una matriz 5x5 de ceros usando bucles anidados
23 print("\n▶ MATRIZ 5x5 CON CEROS (usando bucles):\n")
24 matriz_5x5 = [] # Lista vacía para la matriz
25
26 for i in range(5): # Recorremos 5 filas
27     fila = [] # Creamos una nueva fila vacía
28     for j in range(5): # Recorremos 5 columnas
29         fila.append(0) # Añadimos un cero a la fila
30     matriz_5x5.append(fila) # Añadimos la fila completa a la matriz
31
32 # Imprimimos la matriz como cuadrícula
33 for fila in matriz_5x5:
34     for elemento in fila:
35         print(elemento, end=" ")
```

```
ejercicios-de-estructura-de-control 0% used
~/workspace: python teclado.py
~/workspace$ python teclado.py
▶ MATRIZ DEL TECLADO:
1 2 3
4 5 6
7 8 9
* 0 #

▶ 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

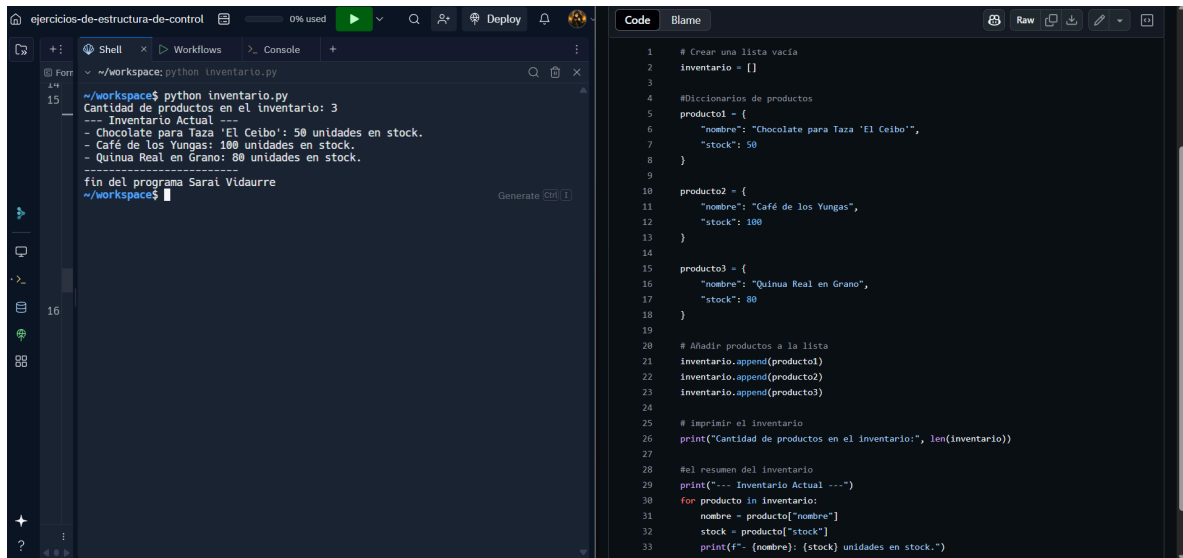
▶ MATRIZ 5x5 CON CEROS (usando comprensión de listas):
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0

EXPLICACIÓN:
matriz_compression = [[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.

~/workspace$
```

```
Files main ejercicios-de-estructura-de-control / teclado.py
Code Blame
6 teclado = [
7     [1, 2, 3],
8     [4, 5, 6],
9     [7, 8, 9],
10    ["*", 0, "#"]
11 ]
12
13 # 2. Imprimimos la matriz como cuadrícula
14 print("▶ MATRIZ DEL TECLADO:\n")
15 for fila in teclado:
16     for elemento in fila:
17         print(elemento, end=" ")
18     print()
19
20 # 3. Crear la misma matriz usando comprensión de listas
21 print("\n▶ MATRIZ 5x5 CON CEROS (usando comprensión de listas):\n")
22
23 # Esta línea crea una matriz 5x5 con ceros de forma compacta
24 matriz_compression = [[0 for j in range(5)] for i in range(5)]
25
26 # Imprimimos la matriz generada
27 for fila in matriz_compression:
28     for elemento in fila:
29         print(elemento, end=" ")
30     print()
31
32 # 5. Explicación visual para programadores novatos
33 print("\nEXPLICACIÓN:")
34 print("\nmatriz_compression = [[0 for j in range(5)] for i in range(5)]\n")
35
36 # Parte interna: [0 for j in range(5)] → crea una fila con cinco ceros
37 # Parte externa: for i in range(5) → repite esa fila cinco veces
38 Resultado: Una matriz de 5x5 completamente llena de ceros.\n")
```

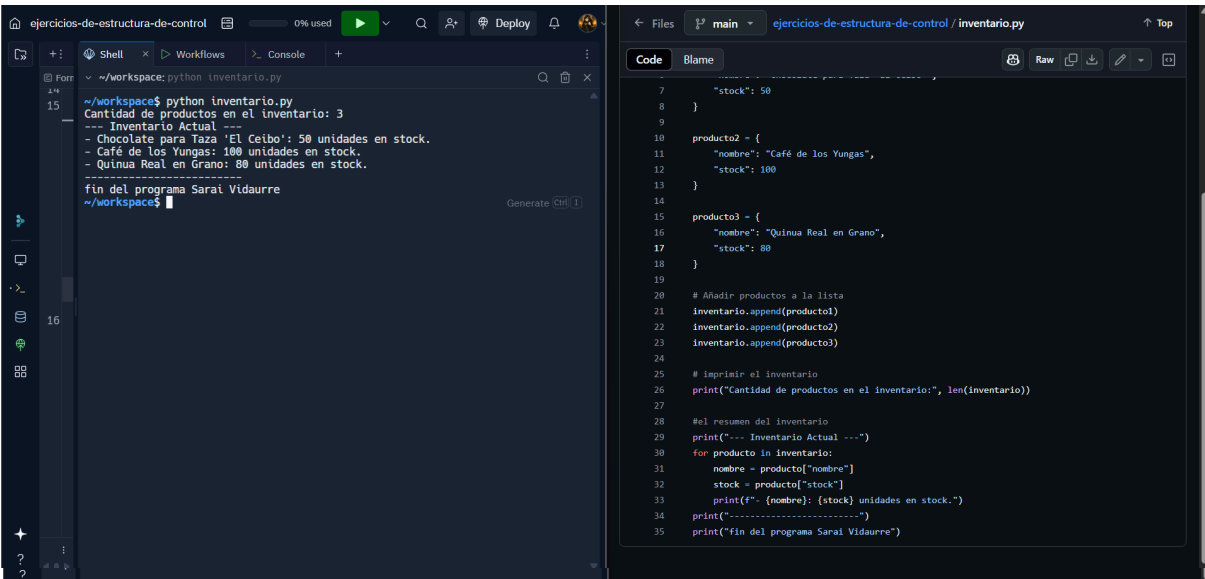
Inventario



The screenshot shows a VS Code editor with a file named `inventario.py` in the workspace. The terminal on the left displays the output of running the script. The code in the editor defines an inventory list and three products, then prints the inventory details.

```
~/workspace$ python inventario.py
Cantidad 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.
- Quinua Real en Grano: 80 unidades en stock.
-----
fin del programa Sarai Vidaurre
~/workspace$
```

```
1 # Crear una lista vacía
2 inventario = []
3
4 #Diccionarios de productos
5 producto1 = {
6     "nombre": "Chocolate para Taza 'El Ceibo'",
7     "stock": 50
8 }
9
10 producto2 = {
11     "nombre": "Café de los Yungas",
12     "stock": 100
13 }
14
15 producto3 = {
16     "nombre": "Quinua Real en Grano",
17     "stock": 80
18 }
19
20 # Añadir productos a la lista
21 inventario.append(producto1)
22 inventario.append(producto2)
23 inventario.append(producto3)
24
25 # Imprimir el inventario
26 print("Cantidad de productos en el inventario:", len(inventario))
27
28 #el resumen del inventario
29 print("--- Inventario Actual ---")
30 for producto in inventario:
31     nombre = producto["nombre"]
32     stock = producto["stock"]
33     print(f"- {nombre}: {stock} unidades en stock.")
34
35
```



This screenshot is identical to the one above, showing the same VS Code editor interface with the `inventario.py` file and its execution output in the terminal.

```
~/workspace$ python inventario.py
Cantidad 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.
- Quinua Real en Grano: 80 unidades en stock.
-----
fin del programa Sarai Vidaurre
~/workspace$
```

```
7     "stock": 50
8 }
9
10 producto2 = {
11     "nombre": "Café de los Yungas",
12     "stock": 100
13 }
14
15 producto3 = {
16     "nombre": "Quinua Real en Grano",
17     "stock": 80
18 }
19
20 # Añadir productos a la lista
21 inventario.append(producto1)
22 inventario.append(producto2)
23 inventario.append(producto3)
24
25 # Imprimir el inventario
26 print("Cantidad de productos en el inventario:", len(inventario))
27
28 #el resumen del inventario
29 print("--- Inventario Actual ---")
30 for producto in inventario:
31     nombre = producto["nombre"]
32     stock = producto["stock"]
33     print(f"- {nombre}: {stock} unidades en stock.")
34     print("-----")
35     print("fin del programa Sarai Vidaurre")
36
```