

LABORATORIOS 2

Docente: Ing.Jimmy Nataniel Requena Llorentty

Materia: Programación 2

Estudiante: Lora Colodro Fabrizzio

Sarai Alejandra Vidaurre

Yoel Escalante Escobar

Flavia Gutiérrez Soliz

Rodrigo Andrés Miranda

Borda

Ejercicio N°12 – Búsqueda Lineal y Binaria. –	2
Ejercicio N°13 – Ordenamiento Burbuja. –	3
Ejercicio N°14 – Ordenamiento Inserción. –	4
Ejercicio N°15 – Ordenamiento Avanzado. –	5
Ejercicio N°16 – Recorriendo una matriz. –	6
Ejercicio N°17 – Simulación con matrices Teclado numérico. –	7
Ejercicio N°18 – Suma total de una matriz. –	8
Ejercicio N°19 - Suma por filas de una matriz. –	9
Ejercicios N°20 – suma diagonal principal de una matriz. –	10
Ejercicio N°21 – Suma diagonal secundaria de una matriz. –	10
Ejercicio N°22 – Suma de una columna. –	12
Ejercicio N°23 – Matriz identidad. –	12
Ejercicio N°24 – Matriz simétrica. –	13
Ejercicio N°25 – Sala de cine usando matrices. –	13
Ejercicio N°26 – juego de Batalla naval. –	14
Ejercicio N°27 – Diccionarios en Python de un producto. -	15
Ejercicio N°28 – Diccionario de Python con varios productos sistema de inventario pequeño. -	16
Ejercicio N°29 – Diccionario de Python con una canción, coche y una red social. –	17
Ejercicio N°30 – Gestión de Tareas. –	17
Ejercicio N°31 – Agenda Contactos. –	18

Ejercicio N°12 – Búsqueda Lineal y Binaria. –

Descripción:

El código define dos funciones de búsqueda: `busqueda_lineal` y `busqueda_binaria`. Luego, realiza pruebas sobre estas funciones usando `assert` para verificar que funcionan. Finalmente, realiza un experimento llamado "experimento del caos" que intenta usar la búsqueda binaria en una lista desordenada. Muestra dos algoritmos de búsqueda fundamentales. La búsqueda lineal y La búsqueda binaria

The screenshot shows a code editor interface with two tabs open: `busqueda.py` and `busqueda_lineal.py`. The `busqueda.py` tab contains the following code:

```
def busqueda_lineal(lista, clave):
    for i in range(len(lista)):
        if lista[i] == clave:
            return i
    return -1

# 1. Prueba tu función con assert
mi_lista_desordenada = [10, 5, 42, 8, 17, 30, 25]
print("Probando busqueda_lineal...")

assert busqueda_lineal(mi_lista_desordenada, 42) == 2
assert busqueda_lineal(mi_lista_desordenada, 10) == 0 # Al inicio
assert busqueda_lineal(mi_lista_desordenada, 25) == 6 # Al final
assert busqueda_lineal(mi_lista_desordenada, 99) == -1 # No existe
assert busqueda_lineal([], 5) == -1 # Lista vacía
print("Pruebas para busqueda_lineal pasaron! ✅")
```

The `busqueda_lineal.py` tab contains the following code:

```
# clase00_busquedas.py (continuación)
# 2. Definir la función busqueda_binaria
def busqueda_binaria(lista_ordenada, clave):
    izquierda = 0
    derecha = len(lista_ordenada) - 1
    while izquierda <= derecha:
        medio = (izquierda + derecha) // 2
        if lista_ordenada[medio] == clave:
            return medio
        elif clave > lista_ordenada[medio]:
            izquierda = medio + 1
        else:
            derecha = medio - 1
    return -1

# 3. Prueba de la función
lista_ordenada = [2, 5, 8, 12, 16, 23, 38, 56, 72, 91]
print("\nProbando busqueda_binaria...")
```

To the right of the code editor, there is a terminal window titled "Shell" showing the output of running the script:

```
~/workspace$ python busqueda.py
Probando busqueda_lineal...
Pruebas para busqueda_lineal pasaron! ✅

Probando busqueda_binaria...
Pruebas para busqueda_binaria pasaron! ✅
fin del programa

Realizando el experimento del caos...
Búsqueda binaria de '5' en lista desordenada devolvió: 1
Fin del programa.
Fabrizio Lora (FaxX@)
~/workspace$
```

Ejercicio N°13 – Ordenamiento Burbuja. –

Descripción:

Este código implementa el algoritmo de ordenamiento burbuja (bubble sort), una técnica sencilla y clásica utilizada para organizar listas de números en orden ascendente.

En este código, además de la función principal, se incluye una función de prueba que verifica distintos casos (listas desordenadas, ordenadas, con repetidos, negativas o vacías) para asegurar que el algoritmo funcione correctamente en diversas situaciones. También se muestra una ejecución visual del algoritmo con una lista de ejemplo para observar el antes y el después del ordenamiento.

The screenshot shows a code editor with two tabs: 'Code' and 'Blame'. The 'Code' tab displays the following Python code:

```
1 def ordenamiento_burbuja(lista):
2     n = len(lista)
3     for i in range(n - 1):
4         hubo_intercambio = False
5         for j in range(n - 1 - i):
6             if lista[j] > lista[j + 1]:
7                 #intercambio
8                 lista[j], lista[j + 1] = lista[j + 1], lista[j]
9                 hubo_intercambio = True
10            if not hubo_intercambio:
11                break
12        return lista
13 if __name__ == "__main__":
14     numeros = [6,3,8,2,5]
15     print("antes", numeros)
16     ordenamiento_burbuja(numeros)
17     print("despues", numeros)
18
19     lista_a_ordenar = [64,34,25,12,22,11,90]
20     print("lista original: (lista_a_ordenar)")
21
22     ordenamiento_burbuja(lista_a_ordenar) #llamamos la funcion
23     print("lista ordenada: (lista_a_ordenar)")
24
25     #caso 1: lista desordenada
26     lista1 = [6,3,8,2,5]
27     ordenamiento_burbuja(lista1)
28     assert lista1 == [2,3,5,6,8], "Falló en caso 1"
29
30     #caso 2: lista ya ordenada
31     lista2 = [1,2,3,4,5]
32     ordenamiento_burbuja(lista2)
33     assert lista2 == [1,2,3,4,5], "Falló en caso 2"
34
35     #caso 3: lista ordenada a la inversa(por caso)
36     lista3 = [5,4,3,2,1]
37     ordenamiento_burbuja(lista3)
38     assert lista3 == [1,2,3,4,5], "Falló en caso 3"
39
40     #caso 4: lista con elementos duplicados
41     lista4 = [5,1,4,2,8,5,2]
42     ordenamiento_burbuja(lista4)
43     assert lista4 == [1,2,4,5,5,8], "falló en caso 4"
44
45     #caso borde
46     assert ordenamiento_burbuja([]) == []
47     assert ordenamiento_burbuja([42]) == [42], "Falló en caso borde"
48
49     print ("todas las pruebas pasaron")
```

The screenshot shows a second instance of a code editor with the same code as the first one. It includes the same imports, functions, and test cases for bubble sort.

Ejercicio N°14 – Ordenamiento Inserción. –

Descripción:

Este código implementa el algoritmo de ordenamiento por inserción (insertion sort), una técnica eficiente para ordenar listas pequeñas o parcialmente ordenadas.

El código incluye una serie de pruebas automáticas (assert) que validan su comportamiento con diferentes tipos de listas: desordenadas, ya ordenadas, inversamente ordenadas, con elementos duplicados, vacías y con un solo elemento. Estas pruebas aseguran que la función se comporte correctamente en diversos escenarios y resalten errores si ocurren. Al finalizar, se imprime un mensaje confirmando que todas las pruebas fueron exitosas.

The screenshot shows a code editor interface with two panes. The left pane displays the source code for '07Lab_ordenamientos_B_Insercion.py'. The right pane shows the terminal output of running the script.

```
#!/usr/bin/python
# Ordenamiento por insercion
# Desplazar elementos mayores hacia la derecha
# mientras la posicion sea valida y el elemento de la izquierda sea mayor que el actual
# Insertar el valor actual en su posicion correcta
# Caso 1: Lista desordenada
# Caso 2: Lista ya ordenada
# Caso 3: Lista ordenada a la inversa (peor caso)
# Caso 4: Lista con duplicados

def ordenamiento_por_insercion(lista):
    for i in range(1, len(lista)):
        valor_actual = lista[i]
        posicion_actual = i
        while posicion_actual > 0 and lista[posicion_actual - 1] > valor_actual:
            lista[posicion_actual] = lista[posicion_actual - 1]
            posicion_actual -= 1
        lista[posicion_actual] = valor_actual
    return lista

# Caso 1: Lista desordenada
lista1 = [6, 3, 8, 2, 5]
ordenamiento_por_insercion(lista1)
assert lista1 == [2, 3, 5, 6, 8], "Fallo en Caso 1"

# Caso 2: Lista ya ordenada
lista2 = [1, 2, 3, 4, 5]
ordenamiento_por_insercion(lista2)
assert lista2 == [1, 2, 3, 4, 5], "Fallo en Caso 2"

# Caso 3: Lista ordenada a la inversa (peor caso)
lista3 = [5, 4, 3, 2, 1]
ordenamiento_por_insercion(lista3)
assert lista3 == [1, 2, 3, 4, 5], "Fallo en Caso 3"

# Caso 4: Lista con duplicados
lista4 = [1, 2, 3, 4, 5, 1]
ordenamiento_por_insercion(lista4)
assert lista4 == [1, 2, 3, 4, 5, 1], "Fallo en Caso 4"
```

Terminal Output:

```
~/workspace$ python 07Lab_ordenamientos_B_Insercion.py
Antes: [6, 3, 8, 2, 5]
Despues: [2, 3, 5, 6, 8]
Todas las pruebas pasaron correctamente.
Todas las pruebas del ordenamiento por insercion pasaron!
```

Ejercicio N°15 – Ordenamiento Avanzado. –

Descripción:

Este código implementa el algoritmo de ordenamiento Merge Sort (ordenamiento por mezcla), una técnica eficiente basada en el paradigma “divide y vencerás”.

La función `merge_sort` se encarga de dividir la lista y aplicar recursivamente el ordenamiento, mientras que la función auxiliar `merge` une dos sublistas ya ordenadas en una sola lista ordenada. Este algoritmo tiene una eficiencia de tiempo $O(n \log n)$, por lo que es ideal para ordenar grandes volúmenes de datos. Además, el código incluye una serie de pruebas automáticas (`assert`) que validan su funcionamiento correcto en diferentes casos: listas vacías, con un solo elemento, ordenadas, desordenadas, con duplicados, números negativos y flotantes. También imprime los pasos intermedios de combinación para visualizar cómo se arma la lista final.

The screenshot shows a split-screen view of the Visual Studio Code interface. The left pane displays a Python script named `merge_sort.py`. The code implements the merge sort algorithm, which follows three main steps: dividing the list into halves, sorting each half, and then merging them back together. The right pane shows the terminal window with the command `python clase08_ordenamientoavanzado.py` run, resulting in the output `Lista ordenada: [3, 9, 10, 27, 38, 43, 82]`.

```
def merge_sort(lista):
    # Paso 1: Vencer (Condición base de la recursividad)
    if len(lista) <= 1:
        return lista
    else:
        # Paso 1: DIVIDIR
        medio = len(lista) // 2
        mitad_izquierda = lista[:medio]
        mitad_derecha = lista[medio:]

        # Paso 2: VENCER (ordenar recursivamente cada mitad)
        izquierda_ordenada = merge_sort(mitad_izquierda)
        derecha_ordenada = merge_sort(mitad_derecha)

        # Paso 3: COMBINAR (mezclar ambas mitades ordenadas)
        return merge(izquierda_ordenada, derecha_ordenada)

def merge(izquierda, derecha):
    resultado = []
    i = j = 0

    while i < len(izquierda) and j < len(derecha):
        if izquierda[i] < derecha[j]:
            resultado.append(izquierda[i])
            i += 1
        else:
            resultado.append(derecha[j])
            j += 1

    # Agregar los elementos restantes (si quedan)
    resultado.extend(izquierda[i:])
    resultado.extend(derecha[j:])

    return resultado
```

```
12 izquierda_ordenada = merge_sort(izquierda)
13 derecha_ordenada = merge_sort(derecha)
14
15 # Paso 3: COMBINAR (mezclar ambas mitades ordenadas)
16 return merge(izquierda_ordenada, derecha_ordenada)
17
18 v 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     resultado.extend(derecha[j:])
34
35     return resultado
36
37 # Ejemplo de uso
38 mi_lista = [38, 27, 43, 3, 9, 82, 10]
39 ordenada = merge_sort(mi_lista)
40 print("Lista ordenada:", ordenada)
```

Ejercicio N°16 – Recorriendo una matriz. –

Descripción:

Este código muestra dos formas básicas de recorrer una matriz (lista de listas) en En la primera parte, se emplea un recorrido por índices, usando range y la longitud de filas y columnas para acceder a cada elemento individualmente mediante matriz[i][j]. Esto es útil cuando se necesita conocer la posición exacta de cada elemento. En la segunda parte, se utiliza un recorrido directo por filas, donde se itera primero por cada fila y luego por cada elemento de esa fila, lo cual es más simple y legible cuando solo se necesita acceder a los valores sin preocuparse por sus posiciones. Este tipo de recorrido es común en operaciones sobre matrices como sumas, transposiciones, o búsqueda de elementos. El código también utiliza print(end=" ") para mostrar los elementos de cada fila en una sola línea, y print() para saltar a la siguiente línea al final de cada fila. Es una forma clara y eficiente de trabajar con estructuras bidimensionales en Python.

```

ejercicios-de-estructura-de-control
0% used
Shell > Workflows > Console + ...
~/workspace$ python clase09_matrices.py
El valor es: 70
--- Recorrido con indices ---
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 | 0 | X
-----
| X | 0
-----
0 | 1 |
-----
~/workspace$ Generate Ctrl+I

```

```

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("El valor es: " + str(valor)) # Resultado: 70
12
13 # Modificar el valor 90 por un 0
14 matriz[2][0] = 0
15
16 print("\n--- Recorrido con indices ---")
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("Elemento en ({}, {}) es {}".format(i, j, 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 linea entre filas
29
30 # --- TABLERO DE TRES EN RAYA ---
31 print("\n--- Tablero de Tres en Raya ---")
32 tablero = [
33     ['X', '0', 'X'],
34     [' ', 'X', '0'],
35     ['0', ' ', ' ']
36 ]
37
38 for fila in tablero:
39     print(" ".join(fila))
40     print("-" * 9)

```

Ejercicio N°17 – Simulación con matrices Teclado numérico. –

Descripción:

Este código utiliza una matriz 3x3 que simula un teclado numérico. Primero, se crea la matriz con los números del 1 al 9 organizados en tres filas. Luego, se imprime la matriz completa fila por fila. Después, se accede al número central (el 5) ubicado en la posición [1][1] y al número en la esquina inferior derecha (el 9) en la posición [2][2]. A continuación, se modifica el valor en la esquina superior izquierda, cambiando el 1 por un 0. Finalmente, se imprime la matriz nuevamente para mostrar el cambio. Este ejemplo permite comprender cómo acceder, modificar y representar datos en una matriz bidimensional en Python.

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="\t")
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="\t")
```

ejercicios-de-estructura-de-control Shell Workflows Console +

Code Blame

```
6     teclado = [
7         for j in range(5): # Recorremos 5 columnas
8             fila.append(0) # Añadimos un cero a la fila
9         matriz_5x5.append(fila) # Añadimos la fila completa a la matriz
10
11    # Imprimimos la matriz como cuadrícula
12    for fila in matriz_5x5:
13        for elemento in fila:
14            print(elemento, end="\t")
15        print()
16
17    # 4. Crear la misma matriz usando comprensión de listas
18    print("\n► MATRIZ 5x5 CON CEROS (usando comprensión de listas):\n")
19
20    # Esta línea crea una matriz 5x5 con ceros de forma compacta
21    matriz_comprehension = [[0 for j in range(5)] for i in range(5)]
22
23    # Imprimimos la matriz generada
24    for fila in matriz_comprehension:
25        for elemento in fila:
26            print(elemento, end="\t")
27        print()
28
29    # 5. Explicación visual para programadores novatos
30    print("\nEXPLICACIÓN:")
31    print("• matriz_comprehension = [[0 for j in range(5)] for i in range(5)]\n")
32    # Parte interna: [0 for j in range(5)] - crea una fila con cinco ceros
33    # Parte externa: for i in range(5) - repite esa fila cinco veces
34    Resultado: Una matriz de 5x5 completamente llena de ceros.
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")
```

Ejercicio N°18 – Suma total de una matriz. –

Descripción:

Este código define una función que suma todos los valores dentro de una matriz en Python recorriendo fila por fila. También incluye una función de prueba que verifica su correcto funcionamiento en distintos casos, incluyendo matrices con negativos, vacías o con un solo valor. Al final, se confirma que todas las pruebas se realizaron con éxito, mostrando buenas prácticas de validación en programación.

Ejercicio N°19 - Suma por filas de una matriz. –

Descripción:

Este código define la función `sumar_por_filas`, que toma como entrada una matriz y devuelve una lista con la suma de los elementos de cada fila. Para cada fila en la matriz, se utiliza la función incorporada `sum()` para calcular su total, y ese resultado se agrega a una lista llamada `resultado`.

Además, se incluye una función de prueba llamada `probar_suma_por_filas`, la cual valida el funcionamiento correcto de la función principal mediante casos de prueba. Se verifican: una matriz 3x3 con valores secuenciales, una matriz con pares repetidos, y un caso borde con una matriz vacía (sin filas). Las afirmaciones con `assert` garantizan que la función devuelva los resultados esperados, y si todas las pruebas pasan, se muestra un mensaje de éxito.

Ejercicios N°20 – suma diagonal principal de una matriz. –

Descripción:

Este código define la función `sumar_diagonal_principal`, que suma los elementos de la diagonal principal de una matriz cuadrada. Usa un bucle para acumular los valores en `matriz[i][i]`. También incluye la función `probar_suma_diagonal_principal`, que prueba la función con matrices de diferentes tamaños, asegurando que funcione correctamente. Si todas las pruebas pasan, se imprime un mensaje de éxito. Este ejercicio ayuda a entender cómo trabajar con índices en matrices.

Shell > Workflows > Console

```
~/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$
```

Generate Ctrl + I

```

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!")

```

Shell > Workflows > Console

```
~/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$
```

Generate Ctrl + I

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
34             resultado = []
35             for fila in matriz:
36                 suma_fila = sum(fila) # Suma todos los elementos de la fila
37                 resultado.append(suma_fila)
38             return resultado
39
40     # Función de prueba para verificar que sumar_por_filas funciona correctamente
41     def probar_suma_por_filas():
42         print("\nProbando sumar_por_filas...")

43
44     # Caso 1: matriz con 3 filas y 3 columnas
45     m1 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
46     assert sumar_por_filas(m1) == [6, 15, 24] # 1+2+3, 4+5+6, 7+8+9
47
48     # Caso 2: matriz con pares repetidos
49     m2 = [[10, 10], [20, 20], [30, 30]]
50     assert sumar_por_filas(m2) == [20, 40, 60]
51
52     # Caso borde: matriz vacía
53     assert sumar_por_filas([]) == [] # No hay filas que sumar
54
55     print("Pruebas para sumar_por_filas pasaron!")
56     print("ejercicio 2: ¡Todo está correcto!")

57     # Llamamos a la función para ejecutar las pruebas
58     probar_suma_por_filas()
59
60
61 #EJERCICIO 3
62
63     # Definimos la función que suma los elementos de la diagonal principal de una matriz cuadrada
64     def sumar_diagonal_principal(matriz):
65         """
66             Esta función recibe una matriz cuadrada (misma cantidad de filas y columnas)
67             y retorna la suma de los elementos en su diagonal principal.
68             Ejemplo:
69                 matriz = [[1, 2],
70                           [3, 4]]
71                 diagonal_principal: 1 y 4 = suma = 5
72             ...
73
74             suma = 0
75             for i in range(len(matriz)):
76                 suma += matriz[i][i] # Accede al elemento en la posición (i, i)

```

Shell > Workflows > Console

```
~/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$
```

Generate Ctrl + I

Code Blame

```

57 def probar_suma_por_filas():
58
59     # Caso 2: matriz con pares repetidos
60     m2 = [[10, 10], [20, 20], [30, 30]]
61     assert sumar_por_filas(m2) == [20, 40, 60]
62
63     # Caso borde: matriz vacía
64     assert sumar_por_filas([]) == [] # No hay filas que sumar
65
66     print("Pruebas para sumar_por_filas pasaron!")
67     print("ejercicio 2: ¡Todo está correcto!")

68     # Llamamos a la función para ejecutar las pruebas
69     probar_suma_por_filas()
70
71
72 #EJERCICIO 3
73
74     # Definimos la función que suma los elementos de la diagonal principal de una matriz cuadrada
75     def sumar_diagonal_principal(matriz):
76         """
77             Esta función recibe una matriz cuadrada (misma cantidad de filas y columnas)
78             y retorna la suma de los elementos en su diagonal principal.
79             Ejemplo:
80                 matriz = [[1, 2],
81                           [3, 4]]
82                 diagonal_principal: 1 y 4 = suma = 5
83             ...
84
85             suma = 0
86             for i in range(len(matriz)):
87                 suma += matriz[i][i] # Accede al elemento en la posición (i, i)

```

Ejercicio N°21 – Suma diagonal secundaria de una matriz. –

Descripción:

Este código define una función que suma los elementos de la diagonal secundaria de una matriz cuadrada. La función usa un ciclo para acumular los valores. También hay una función de prueba que verifica su funcionamiento con distintas matrices. Se usan aserciones para confirmar que los resultados son correctos y se imprime un mensaje si todas las pruebas pasan.

The screenshot shows a Python code editor with the following details:

- File:** matrix.py
- Content:** The code defines a function `sumar_diagonal_secundaria` that calculates the sum of the secondary diagonal of a square matrix. It also includes a test function `probar_suma_diagonal_secundaria` with several assertions for different matrix cases.
- Terminal:** The terminal shows the command `python matrix.py` being run, followed by the output: "Fabrizio Lora (Fxx8)", "Pruebas para sumar_diagonal_secundaria pasaron!", and "Suma de la diagonal secundaria2: 15".

Opción 2 - de suma diagonal secundaria de una matriz. –

Descripción:

Este código define la función `sumar_diagonal_secundaria2`, que suma los elementos de la diagonal secundaria de una matriz cuadrada. Verifica que la matriz sea cuadrada y lanza un `ValueError` si no lo es. Luego, recorre la matriz y suma los elementos de la diagonal. Al final, se muestra un ejemplo con una matriz 3x3, esperando una suma de 15. Este código incluye buenas prácticas como validación de entrada y documentación clara.

```

145 < def sumar_diagonal_secundaria2(matriz_cuadrada):
146     # Calcula la suma de los elementos en la diagonal secundaria de una matriz cuadrada: la diagonal secundaria es la que va desde la esquina superior derecha hasta la esquina inferior izquierda. #Parámetros: #matriz_cuadrada (list[list]): Matriz cuadrada (N x N) de números
147     #Retorna#int/float: Suma de los elementos en la diagonal secundaria
148     #Lanza#ValueError: si la matriz no es cuadrada #Verificar si la matriz es cuadrada
149     n = len(matriz_cuadrada)
150     < for fila in matriz_cuadrada:
151         < if len(fila) != n:
152             raise ValueError(
153                 "La matriz debe ser cuadrada (mismo número de filas y columnas)"
154             )
155         # Calcular suma de diagonal secundaria
156         suma = 0
157         < for i in range(n):
158             j = n - 1 - i # Índice de columna para la diagonal secundaria
159             suma += matriz_cuadrada[i][j]
160         return suma
161     # Ejemplo de uso
162     < if __name__ == "__main__":
163         matriz = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
164
165         resultado = sumar_diagonal_secundaria2(matriz)
166         print(
167             f"Suma de la diagonal secundaria2: {resultado}"
168         ) # Salida: 35 (1 + 5 + 7) # Llamamos a la función para ejecutar las pruebas y ejecutamos las pruebas de la función sumar_diagonal_secundaria2 con la matriz de ejemplo
169         print("Fabrizio Lora (FaXx8)")

```

Shell

```

~/workspace/python matriz.py
Fabrizio Lora (FaXx8)
Suma de la diagonal secundaria2: 15
Fabrizio Lora (FaXx8)
¡Pruebas para transponer_matriz pasaron!

```

Ejercicio N°22 – Suma de una columna. –

Descripción:

Este código implementa la función `transponer_matriz`, que cambia filas por columnas en una matriz. Primero verifica si la matriz está vacía, luego calcula el tamaño y crea la matriz transpuesta. También hay una función de prueba que verifica su funcionamiento con diferentes tipos de matrices y muestra mensajes si pasa. Se imprime que todas las pruebas fueron exitosas, ayudando a entender el manejo de matrices en Python.

```

172 < def transponer_matriz(matriz):
173     < if not matriz or not matriz[0]:
174         return []
175     num_filas = len(matriz)
176     num_columnas = len(matriz[0])
177     #Calculando el transponer con la estructura correcta (la construimos dinámicamente)
178     matriz_transpuesta = []
179     < for j in range(num_columnas):
180         nueva_fila = []
181         < for i in range(num_filas):
182             < nueva_fila.append(matriz[i][j])
183         matriz_transpuesta.append(nueva_fila)
184     return matriz_transpuesta
185     #Prueba
186     < def probar_transponer_matriz():
187         print("\nProbando transponer_matriz...")
188         #Case 1: Matriz 3x3 con números consecutivos
189         m1 = [[1, 2, 3], [4, 5, 6]] #2x3
190         t1 = transponer_matriz(m1)
191         assert t1 == [[1, 4], [2, 5], [3, 6]] #debe ser 3x2
192         print("Prueba 1 pasada!")
193         #Case 2: matriz 2x2 con ceros y valores definidos
194         m2 = [[10, 0], [0, 20]] #2x2
195         t2 = transponer_matriz(m2)
196         assert t2 == [[10, 0], [0, 20]] #debe ser 2x2
197         print("Prueba 2 pasada!")
198         #Case 3: Matriz 1x1

```

Shell

```

~/workspace/python matriz.py
¡Pruebas para transponer_matriz pasaron!
Fabrizio Lora (FaXx8)
¡Pruebas para es_identidad pasaron!
Fabrizio Lora (FaXx8)

```

Ejercicio N°23 – Matriz identidad. –

Descripción:

Este código define la función `es_identidad`, que verifica si una matriz cuadrada es una matriz identidad, donde la diagonal principal tiene 1s y los demás elementos son 0s. Comprueba primero si la matriz es cuadrada. Luego, revisa cada elemento de la matriz con bucles anidados. Si está en la diagonal, debe ser 1; si está fuera, debe ser 0. Si no, devuelve False. La función `probar_es_identidad` ejecuta pruebas con matrices de diferentes tamaños. Se imprime un mensaje si las pruebas son exitosas.

Ejercicio N°24 – Matriz simétrica. –

Descripción:

Este código define la función `es_simetrica`, que verifica si una matriz cuadrada es simétrica. Una matriz es simétrica si es igual a su transpuesta. Primero, se comprueba si la matriz es cuadrada. Luego, se comparan elementos simétricos. Si algún par no coincide, retorna False; si todos coinciden, retorna True. La función `probar_es_simetrica` realiza pruebas automáticas para verificar distintas matrices, imprimiendo mensajes de éxito y confirmando que todas las pruebas pasaron.

The screenshot shows a Jupyter Notebook interface with two panes. The left pane is a terminal window displaying command-line output for a Python script named `clase11_transformaciones.py`. The output shows several test cases being run, all of which pass successfully. The right pane is a code editor window containing the source code for the exercises. The code includes functions for transposing a matrix and checking if it is an identity matrix or symmetric. It also contains a test function for the transpose operation and a main function for testing identity/symmetry across different matrix sizes.

```
ejercicios-de-estructura-de-control 0% used Shell Workflows Console ~ workspace: python clase11_transformaciones.py
~/workspace$ python clase11_transformaciones.py
Prueba 1 pasada
pruebas pasadas
ejercicio 2: todas las pruebas pasaron
iTodas las pruebas pasaron correctamente!
ejercicio 3: todas las pruebas pasaron
iPruebas para es_simetrica pasaron!
~/workspace$ 

Code Blame Raw Deploy
def transponer_matriz(matriz):
    if not matriz or not matriz[0]:
        return []
    num_filas = len(matriz)
    num_columnas = len(matriz[0])
    matriz_transpuesta = []
    for j in range(num_columnas):
        nueva_fila = []
        for i in range(num_filas):
            nueva_fila.append(matriz[i][j])
        matriz_transpuesta.append(nueva_fila)
    return matriz_transpuesta
print(matriz_transpuesta)
#print("las pruebas pasaron")
#def test_transponer_matriz():
#    m1=[[1,2,3],[4,5,6]]
#    t1= transponer_matriz(m1)
#    assert t1 == [[1,4],[2,5],[3,6]]
#    print("Prueba 1 pasada")
#    print("pruebas pasadas")
#
# Ejercicio 2: Verificar si una matriz es 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 and matriz[i][j] != 1:
                return False
            if i != j and matriz[i][j] != 0:
                return False
    return True
```

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

Generate [Ctrl + I](#)

```
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_identidad = [[1, 0, 0], [0, 2, 0], [0, 0, 1]]
48     no_cuadrada = [[1, 0], [0, 1], [0, 0]]
49
50     assert es_identidad(identidad) == True
51     assert es_identidad(no_identidad) == False
52     assert es_identidad(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:
```

The screenshot shows a Jupyter Notebook interface with two panes. The left pane is a terminal window titled 'Shell' where the command `python clase11_transformaciones.py` is run, resulting in the output: 'Prueba 1 pasada', 'pruebas pasadas', 'ejercicio 2: todas las pruebas pasaron', '¡todas las pruebas pasaron correctamente!', 'ejercicio 3: todas las pruebas pasaron', and '¡Pruebas para es_simetrica pasaron!'. The right pane is a code editor titled 'Code' containing Python code for matrix operations. The code includes a function `es_idiabilidad` and a function `es_simetrica`. It also contains several test cases for the `es_simetrica` function using lists of lists to represent matrices.

```
def es_idiabilidad(matriz):
    # Ejercicio 3: Verificar si una matriz es simétrica

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 verificar la parte superior
            if matriz[i][j] != matriz[j][i]:
                return False # Con una diferencia basta

    return True # Si todo coincide, es simétrica

# Pruebas
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)
assert not es_simetrica(no_sim)
assert not es_simetrica(no_cuadrada)

print("ejercicio 3: todas las pruebas pasaron")
print("Pruebas para es_simetrica pasaron!")
```

Ejercicio N°25 – Sala de cine usando matrices. –

Descripción:

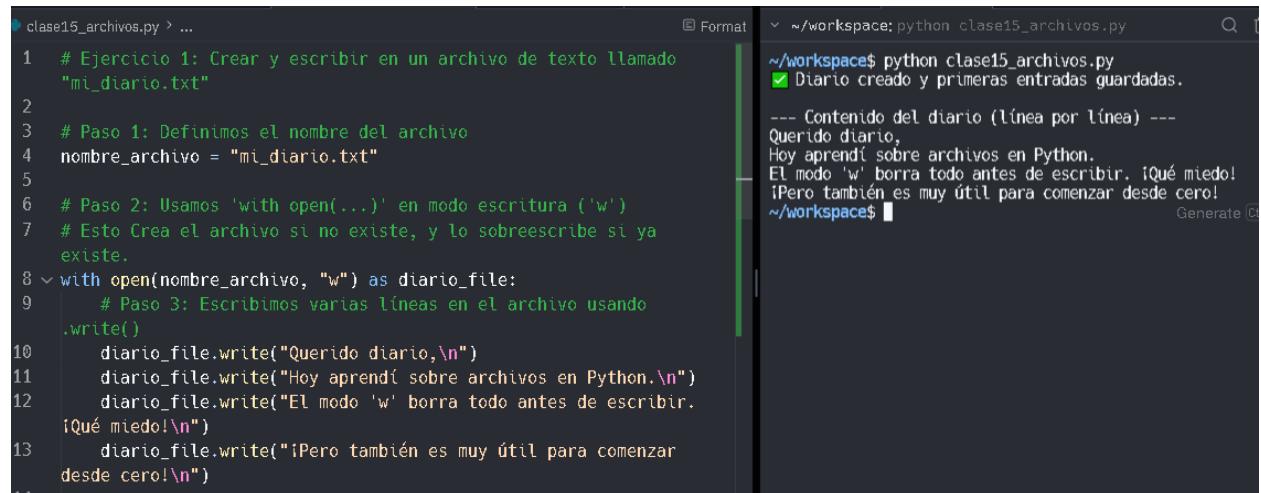
Este código crea un sistema básico de reservas para una sala de cine usando una **matriz bidimensional**, donde 'L' representa un asiento libre y 'O' uno ocupado. El usuario puede ver la sala, seleccionar asientos y saber cuántos quedan disponibles, todo desde la consola.

Las funciones principales permiten **crear la sala, mostrarla visualmente con colores, ocupar asientos y contar los libres**.

El programa valida que las posiciones sean válidas y que el asiento esté disponible antes de marcarlo como ocupado.

Es un ejemplo claro y práctico para aprender a manejar **matrices**, **validación de entradas**, y **visualización en consola** en Python, ideal para estudiantes o proyectos educativos.

tivos.



The screenshot shows a terminal window with two panes. The left pane displays a Python script named 'clase15_archivos.py' with the following code:

```
1 # Ejercicio 1: Crear y escribir en un archivo de texto llamado "mi_diario.txt"
2
3 # Paso 1: Definimos el nombre del archivo
4 nombre_archivo = "mi_diario.txt"
5
6 # Paso 2: Usamos 'with open(...)' en modo escritura ('w')
7 # Esto Crea el archivo si no existe, y lo sobreescribe si ya existe.
8 with open(nombre_archivo, "w") as diario_file:
9     # Paso 3: Escribimos varias líneas en el archivo usando .write()
10    diario_file.write("Querido diario,\n")
11    diario_file.write("Hoy aprendí sobre archivos en Python.\n")
12    diario_file.write("El modo 'w' borra todo antes de escribir.\n")
13    diario_file.write("¡Qué miedo!\n")
14    diario_file.write("¡Pero también es muy útil para comenzar desde cero!\n")
```

The right pane shows the command being run and its output:

```
~/workspace$ python clase15_archivos.py
✓ Diario creado y primeras entradas guardadas.
--- Contenido del diario (línea por línea) ---
Querido diario,
Hoy aprendí sobre archivos en Python.
El modo 'w' borra todo antes de escribir. ¡Qué miedo!
¡Pero también es muy útil para comenzar desde cero!
~/workspace$
```

```
1 # Crear la sala con precios fijos
2 def crear_sala(filas, columnas):
3     sala = []
4     for i in range(filas):
5         fila = []
6         for j in range(columnas):
7             fila.append({"estado": "L", "precio": 25})
8         sala.append(fila)
9     return sala
10
11 # Mostrar la sala con precios y estados
12 def mostrar_sala(sala):
13     print("\n" + ".join(f"{j:^5}" for j in range(len(sala[0]))))
14     print(" " + "- * 5 for _ in range(len(sala[0]))")
15     for i, fila in enumerate(sala):
16         estado_fila = " ".join(f"{a['estado']:^5}" for a in fila)
17         print(f" {i:>2} | {estado_fila}")
18
19 # Método de pago simulado
20 def metodo_de_pago(total):
21     print(f"\n Total a pagar: Bs. {total}")
22     print("Métodos de pago:")
23     print("1. Efectivo")
24     print("2. QR (simulado)")
25     opcion = input("Selecciona método de pago: ")
26     if opcion in ['1', '2']:
27         print(" ✅ Pago procesado con éxito.")
28     else:
29         print(" ❌ Método no reconocido. Se asume pago en efectivo.")
30     print(" 🎉 ¡Gracias por tu compra!\n")
```

```
# Ocupar asiento individual
def ocupar_asiento(sala, fila, columna):
    if 0 <= fila < len(sala) and 0 <= columna < len(sala[0]):
        asiento = sala[fila][columna]
        if asiento["estado"] == "L":
            asiento["estado"] = "O"
            total = asiento["precio"]
            print(f"Asiento ({fila}, {columna}) reservado por Bs. {total}")
            metodo_de_pago(total)
            return True
        else:
            print("X Ese asiento ya está ocupado.")
            return False
    else:
        print("X Coordenadas inválidas.")
        return False

# Buscar N asientos juntos en una fila
def buscar_asientos_juntos(sala, cantidad):
    for i, fila in enumerate(sala):
        consecutivos = 0
        for j, asiento in enumerate(fila):
            if asiento["estado"] == "L":
                consecutivos += 1
                if consecutivos == cantidad:
                    return i, j - cantidad + 1
            else:
                consecutivos = 0
    return None, None
```

```
# Ocupar N asientos juntos
def ocupar_asientos_juntos(sala, cantidad):
    fila, inicio = buscar_asientos_juntos(sala, cantidad)
    if fila is not None:
        total = 0
        for j in range(inicio, inicio + cantidad):
            sala[fila][j]["estado"] = "O"
            total += sala[fila][j]["precio"]
        print(f"🕒 {cantidad} asientos reservados en fila {fila}, desde columna {inicio}.")
        metodo_de_pago(total)
        return True
    else:
        print("🔴 No hay suficientes asientos contiguos disponibles.")
        return False

# Contar asientos libres
def contar_asientos_libres(sala):
    return sum(asiento["estado"] == "L" for fila in sala for asiento in fila)
```

```
# Programa principal
def main():
    filas, columnas = 5, 8
    sala = crear_sala(filas, columnas)

    while True:
        print("\n■■■ Sala actual:")
        mostrar_sala(sala)
        print(f"Asientos libres: {contar_asientos_libres(sala)}")
        print("\nMenú:")
        print("1. Ocupar asiento individual")
        print("2. Buscar y ocupar N asientos juntos")
        print("0. Salir")

        opcion = input("Elige una opción: ")

        if opcion == '1':
            try:
                fila = int(input("Fila: "))
                columna = int(input("Columna: "))
                ocupar_asiento(sala, fila, columna)
            except ValueError:
                print("X Entrada inválida.")
        elif opcion == '2':
            try:
                n = int(input("¿Cuántos asientos necesitas juntos?: "))
                ocupar_asientos_juntos(sala, n)
            except ValueError:
                print("X Entrada inválida.")
        elif opcion == '0':
            print("Gracias por usar el sistema de reserva de cine. 🎉")
            break
        else:
            print("X Opción no válida.")

# Ejecutar
if __name__ == "__main__":
    main()
```

Resulado:

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

```
■ Sala actual:
```

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

```
Asientos libres: 40
```

```
Menú:
```

1. Ocupar asiento individual
2. Buscar y ocupar N asientos juntos
0. Salir

```
Elige una opción: 1
```

```
Fila: 2
```

```
Columna: 4
```

```
Asiento (2, 4) reservado por Bs. 25
```

```
■ Total a pagar: Bs. 25
```

```
Métodos de pago:
```

1. Efectivo
 2. QR (simulado)
- Selecciona método de pago: 1
- Pago procesado con éxito.
- ¡Gracias por tu compra!

Ejercicio N°26 – juego de Batalla naval. –

Descripción:

Este programa en Python crea el juego Batalla Naval, permitiendo la colocación de barcos, disparos por turnos, y juegos contra la CPU o entre dos jugadores, con opciones para guardar partidas. Usa listas de listas y funciones para organizar el código. Aunque no utiliza el ordenamiento burbuja, puede ser útil para agregar funciones como clasificaciones. Este algoritmo es sencillo y enseña conceptos básicos de programación como ciclos y condicionales.

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows files like 'main.py' and 'batalla Naval.py'.
- Terminal:** Shows the command `git push` and the output of running the script: "BATALLA NAVAL", options 1-4, player names, and a grid for placing ships.
- Code Editor:** Displays the 'batalla Naval.py' code, which defines functions for placing ships and handling player shots.
- Status Bar:** Shows '0% used' and other status indicators.

```
ejercicios-de-estructura-de-control 0% used Deploy
Shell > Workflows > Console + ...
~/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 🌊
```

The screenshot shows a terminal window with the Naval Battle game running. The game interface includes a grid for placing ships and attacking. The player names are sarai and haru. The terminal shows the game's logic for ship placement and turn-based attacks.

The terminal shows the command `git push` followed by the execution of `batalla_naval.py`. The game starts with the menu:

```
== 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
```

The player chooses to play against the CPU. The game then asks for the placement of ships:

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

The player chooses manual mode. The board state is shown:

```
[sarai ATACA]
 1 2 3 4
A ● ● ● ●
B ● ● ● ●
C ● ● ● ●
D ● ● ● ●
```

The player places ships at positions A1, B1, C1, and D1. The game then asks for the next move:

```
sarai, disparo (A1) o escribe GUARDAR: A1
Agua ☀
```

The code editor shows the implementation of the `realizar_disparo` function:

```
56 def realizar_disparo(tablero, jugador):
57     ...
58
59     def cargar_partida():
60         if not os.path.exists(ARCHIVO_PARTIDA):
61             return None
62
63         try:
64             with open(ARCHIVO_PARTIDA, 'r') as f:
65                 return json.load(f)
66         except:
67             return None
68
69     def jugar_vs_cpu():
70         nombre = input("Tu nombre: ")
71         jugador = nombre
72
73         tablero_jugador = crear_tablero()
74         tablero_cpu = crear_tablero()
75         disparos_cpu = []
76
77         print("\n[COLOCACIÓN DE BARCOS]")
78         modo = input("Manual (M) o Automático (A) ").upper()
79         tablero_jugador = colocar_barcos(tablero_jugador, "manual" if modo == "M" else "auto", jugar)
80         tablero_cpu = colocar_barcos(tablero_cpu, "auto")
81
82         while True:
83             print("\n[TU TABLERO]")
84             mostrar_tablero(tablero_jugador)
85             print("\n[TABLERO CPU]")
86             mostrar_tablero(tablero_cpu, ocultar_barcos=True)
87
88             resultado = realizar_disparo(tablero_cpu, jugador)
89             ...
90
91             def jugar_2_jugadores():
92                 ...
93
94             jugador1 = input("Nombre Jugador 1: ")
95             jugador2 = input("Nombre Jugador 2: ")
```

The terminal shows the command `git push` followed by the execution of `batalla_naval.py`. The game starts with the menu:

```
== 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
```

The player chooses to play against the CPU. The game then asks for the placement of ships:

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

The player chooses manual mode. The board state is shown:

```
[sarai ATACA]
 1 2 3 4
A ● ● ● ●
B ● ● ● ●
C ● ● ● ●
D ● ● ● ●
```

The player places ships at positions A1, B1, C1, and D1. The game then asks for the next move:

```
sarai, disparo (A1) o escribe GUARDAR: A1
Agua ☀
```

The code editor shows the implementation of the `jugar_vs_cpu` function:

```
98 def jugar_vs_cpu():
99     ...
100
101     def jugar_2_jugadores():
102         ...
103
104         if not quedan_barcos(tablero_cpu):
105             print(f"\n¡{jugador} GANASTE! 🎉")
106             return
107
108         print("\nTurno de la CPU...")
109         while True:
110             fila = random.randint(0, FILAS-1)
111             col = random.randint(0, COLUMNAS-1)
112             if (fila, col) not in disparos_cpu:
113                 disparos_cpu.append((fila, col))
114                 break
115
116             if tablero_jugador[fila][col] == "▲":
117                 tablero_jugador[fila][col] = "✖"
118                 print(f"CPU disparó en ({chr(65+fila)},{col+1}): Impacto!")
119             else:
120                 tablero_jugador[fila][col] = "✖"
121                 print(f"CPU disparó en ({chr(65+fila)},{col+1}): Agua ☀")
122
123             if not quedan_barcos(tablero_jugador):
124                 print("\n¡LA CPU GANA! 😞")
125                 return
126
127             def jugar_2_jugadores():
128                 ...
129
130             jugador1 = input("Nombre Jugador 1: ")
131             jugador2 = input("Nombre Jugador 2: ")
```

ejercicios-de-estructura-de-control

~/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 ⚡ ⚡ ⚡ ⚡
sarah, 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 🌊
```

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

```
98     def jugar_vs_cpu():
99
100    def jugar_2_jugadores():
101        jugador1 = input("Nombre Jugador 1: ")
102        jugador2 = input("Nombre Jugador 2: ")
103
104        tablero1 = crear_tablero()
105        tablero2 = crear_tablero()
106
107        print(f"\n{jugador1} COLOCA TUS BARCOS")
108        modo = input("Manual (M) o Automático (A)?").upper()
109        tablero1 = colocar_barcos(tablero1, "manual" if modo == "M" else "auto", jugador1)
110
111        print(f"\n{jugador2} COLOCA TUS BARCOS")
112        modo = input("Manual (M) o Automático (A)?").upper()
113        tablero2 = colocar_barcos(tablero2, "manual" if modo == "M" else "auto", jugador2)
114
115        while True:
116            print(f"\n[{jugador1}] ATACA")
117            mostrar_tablero(tablero2, ocultar_barcos=True)
118            if realizar_disparo(tablero2, jugador1) == "guardar":
119                guardar_partida({
120                    "modo": "2jugadores",
121                    "jugador1": jugador1,
122                    "jugador2": jugador2,
123                    "tablero1": tablero1,
124                    "tablero2": tablero2
125                })
126            return
127
128            if not quedan_barcos(tablero2):
129                print(f"\n{jugador1} GANÓ! 🎉")
130                break
131
132    def realizar_disparo(tablero, jugador):
133        fila = int(input("Fila (1-4): ")) - 1
134        columna = int(input("Columna (1-4): ")) - 1
135
136        if tablero[fila][columna] == "H":
137            print("Agua 🌊")
138        elif tablero[fila][columna] == "B":
139            print("Barco!")
140            tablero[fila][columna] = "X"
141        else:
142            print("No hay barco en esa posición")
143
144        return tablero
145
146    def mostrar_tablero(tablero, ocultar_barcos=False):
147        for fila in range(4):
148            print(f" {fila+1} ", end="")
149            for columna in range(4):
150                if ocultar_barcos and tablero[fila][columna] == "B":
151                    print(" "), end=" "
152                else:
153                    print(tablero[fila][columna], end=" ")
154            print()
155
156    def guardar_partida(partida):
157        with open("partida.json", "w") as f:
158            json.dump(partida, f)
```

ejercicios-de-estructura-de-control

~/workspace\$ python batalla_naival.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 ⚡ ⚡ ⚡ ⚡
sarah, 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 🌊
```

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

```
151    def jugar_2_jugadores():
152        if not quedan_barcos(tablero2):
153            print(f"\n{jugador1} GANÓ! 🎉")
154            return
155
156        print(f"\n{jugador2} ATACA")
157        mostrar_tablero(tablero1, ocultar_barcos=True)
158        if realizar_disparo(tablero1, jugador2) == "guardar":
159            guardar_partida({
160                "modo": "2jugadores",
161                "jugador1": jugador1,
162                "jugador2": jugador2,
163                "tablero1": tablero1,
164                "tablero2": tablero2
165            })
166        return
167
168        if not quedan_barcos(tablero2):
169            print(f"\n{jugador2} GANÓ! 🎉")
170            return
171
172    def realizar_disparo(tablero, jugador):
173        fila = int(input("Fila (1-4): ")) - 1
174        columna = int(input("Columna (1-4): ")) - 1
175
176        if tablero[fila][columna] == "H":
177            print("Agua 🌊")
178        elif tablero[fila][columna] == "B":
179            print("Barco!")
180            tablero[fila][columna] = "X"
181        else:
182            print("No hay barco en esa posición")
183
184        return tablero
185
186    def continuar_partida():
187        partida = cargar_partida()
188        if not partida:
189            print("No hay partida guardada.")
190            return
191
192        if partida["modo"] == "cpu":
193            partida["tablero_jugador"] = [[celda for celda in fila] for fila in partida["tablero_jugador"]]
194            partida["tablero_puerto"] = [[celda for celda in fila] for fila in partida["tablero_puerto"]]
195            jugar_vs_cpu()
196        elif partida["modo"] == "2jugadores":
197            pass
```

The terminal window shows a session of the 'batalla Naval' game. The user has chosen to play against the CPU. The game menu lists options: 1. Jugar vs CPU, 2. Jugar 2 Jugadores, 3. Continuar Partida, and 4. Salir. The user selects option 1. The game then asks for player names and starts the game. The code editor displays the source code for the game:

```

197 def continuar_partida():
198     jugar_vs_cpu()
199
200     elif partida['modo'] == "2jugadores":
201         jugar_2_jugadores()
202
203     def menu_principal():
204         while True:
205             print("\n==== BATALLA NAVAL ===")
206             print("1. Jugar vs CPU")
207             print("2. Jugar 2 Jugadores")
208             print("3. Continuar Partida")
209             print("4. Salir")
210             opcion = input("Elige una opción: ")
211
212             if opcion == "1":
213                 jugar_vs_cpu()
214             elif opcion == "2":
215                 jugar_2_jugadores()
216             elif opcion == "3":
217                 continuar_partida()
218             elif opcion == "4":
219                 print("¡Gracias por jugar!")
220                 break
221             else:
222                 print("Opción inválida. Intenta otra vez.")
223
224     # █ Corrección final: esta es la forma correcta
225     if __name__ == "__main__":
226         menu_principal()

```

Ejercicio N°27 – Diccionarios en Python de un producto. -

Description:

Este código enseña a recorrer un diccionario en Python sobre un producto. Se imprimen las claves y se muestran los valores con un formato claro. Las funciones mejoran la presentación. Se procesan estructuras de datos útiles como inventarios o catálogos. Esto ayuda a entender el acceso y manipulación de datos.

Ejercicio N°28 – Diccionario de Python con varios productos sistema de inventario pequeño. -

Descripción:

Este código en Python simula un sistema de inventario con listas y diccionarios. Cada producto tiene un diccionario con su información. Los productos se guardan en una lista llamada inventario, y el programa muestra sus detalles en un formato ordenado. Algunas variables se sobrescriben, conservando solo el último valor.

Ejercicio N°29 – Diccionario de Python con una canción, coche y una red social. –

Descripción:

Este código en Python muestra tres ejemplos de diccionarios para representar una canción, un coche y una publicación en redes sociales. Cada diccionario tiene información clave-valor. Se imprime cada uno para visualizarlos. Estas estructuras ayudan a organizar datos complejos de manera clara y reutilizable.

The screenshot shows a code editor interface with several tabs and panes. The main pane displays Python code for creating dictionaries and printing them. The code defines three dictionaries: 'producto', 'inventario', and 'comida'. It then prints each dictionary's keys and values. The right side of the interface shows the terminal output of running the script, which prints the dictionaries' contents. The terminal also shows the path to the workspace and the command used.

```
class12_Diccionarios.py
1 producto = {'codigo' : 100, 'nombre' : 'Iphone', 'precio' : 4200.00, 'cantidad' : 10} # Diccionario de producto
2 print ("--- Claves del producto ---")
3 for clave in producto: # Iteración sobre las claves del diccionario
4     print(clave)
5 print(" --- Clave y valor del producto ---")
6 for clave, valor in producto.items(): # Iteración sobre las claves y valores del diccionario
7     valor = producto [clave] # Obtención del valor de la clave
8     print(f'{clave.capitalize()}: {valor}')
9 # Fin del programa
10 print ("Fabrizio Lora (Fxx0*)")
11 # -----
12 # -----
13 # -----
14 # Diccionario de productos
15 inventario=[]
16 comida = {'codigo' : 123, 'nombre' : 'frutas', 'precio' : 15.00, 'cantidad' : 10}
17 comida = {'codigo' : 124, 'nombre' : 'Pan', 'precio' : 5.00, 'cantidad' : 40}
18 tecnologia = {'codigo' : 456, 'nombre' : 'Laptop', 'precio' : 6500.00, 'cantidad' : 10}
19 celulares = {'codigo' : 457, 'nombre' : 'Iphone', 'precio' : 4200.00, 'cantidad' : 5}
20 limpieza = {'codigo' : 789, 'nombre' : 'jabon', 'precio' : 18.00, 'cantidad' : 10}
21 limpieza = {'codigo' : 790, 'nombre' : 'detergente', 'precio' : 30.00, 'cantidad' : 10}
22 ropa = {'codigo' : 101112, 'nombre' : 'camisa', 'precio' : 150.00, 'cantidad' : 10}
23 ropa = {'codigo' : 101113, 'nombre' : 'pantalon', 'precio' : 150.00, 'cantidad' : 10}
24 ropa = {'codigo' : 101114, 'nombre' : 'zapatos', 'precio' : 270.00, 'cantidad' : 10}
25 # Diccionario de producto
26 inventario.append(comida) # Agrega el diccionario de comida al inventario
27 inventario.append(producto) # Agrega el diccionario de producto al inventario
28 inventario.append(limpieza) # Agrega el diccionario de limpieza al inventario
29 inventario.append(ropa) # Agrega el diccionario de ropa al inventario
30 for producto in inventario: # Iteración sobre los productos del inventario
31     print(" --- Clave y valor del producto ---")
32     for clave, valor in producto.items(): # Iteración sobre las claves y valores del diccionario
33         valor = producto [clave] # Obtención del valor de la clave
34         print(f'{clave.capitalize()}: {valor}')


~/workspace$ python class12_Diccionarios.py
--- Claves del producto ---
codigo
nombre
precio
cantidad
--- Clave y valor del producto ---
Codigo: 100
Nombre: Iphone
Precio: 4200.0
Cantidad: 10
Fabrizio Lora (Fxx0*)

~/workspace$ bash
```

```

10 print("Fabrizio Lora (FaX@*)")
11 #-----#
12 #-----#
13 #
14 # Diccionario de productos
15 inventario = []
16 comida = {"codigo": 123, "nombre": "frutas", "precio": 15.00, "cantidad": 10}
17 comida = {"codigo": 124, "nombre": "Pan", "precio": 5.00, "cantidad": 40}
18 tecnologia = {"codigo": 456, "nombre": "Laptop", "precio": 6500.00, "cantidad": 10}
19 celulares = {"codigo": 457, "nombre": "Iphone", "precio": 4200.00, "cantidad": 5}
20 limpieza = {"codigo": 789, "nombre": "jabon", "precio": 18.00, "cantidad": 10}
21 limpieza = {"codigo": 790, "nombre": "detergente", "precio": 30.00, "cantidad": 10}
22 ropa = {"codigo": 101112, "nombre": "camisa", "precio": 150.00, "cantidad": 10}
23 ropa = {"codigo": 101113, "nombre": "pantalon", "precio": 150.00, "cantidad": 10}
24 ropa = {"codigo": 101114, "nombre": "zapatos", "precio": 270.00, "cantidad": 10}
25 # Diccionario de productos
26 inventario.append(comida) # Agrega el diccionario de comida al inventario
27 inventario.append(producto) # Agrega el diccionario de producto al inventario
28 inventario.append(limpieza) # Agrega el diccionario de limpieza al inventario
29 inventario.append(ropa) # Agrega el diccionario de ropa al inventario
30 for producto in inventario: # Iteración sobre los productos del inventario
31     print("\n--- Clave y valor del producto ---")
32     for clave, valor in producto.items(): # Iteración sobre las claves y valores del diccionario
33         valor = producto [clave] # Obtenido del valor de la clave
34         print(f'{clave.capitalize()}: {valor}')
35     # Fin del programa
36 print ("Fabrizio Lora (FaX@*)")
37
38 #-----#
39 #-----#
40
41 # Diccionario para una Canción
42 cancion = {
43     "titulo": "Shape of You",
44 }

```

Output from the Shell tab:

```

--- Clave y valor del producto ---
Codigo: 124
Nombre: Pan
Precio: 5.0
Cantidad: 40

--- Clave y valor del producto ---
Codigo: 100
Nombre: Iphone
Precio: 1200.0
Cantidad: 10

--- Clave y valor del producto ---
Codigo: 790
Nombre: detergente
Precio: 30.0
Cantidad: 10

--- Clave y valor del producto ---
Codigo: 101114
Nombre: zapatos
Precio: 270.0
Cantidad: 10
Fabrizio Lora (FaX@*)


```

```

1 # Modelo 1: Canción
2 canción = {
3     "titulo": "Caminando bajo la lluvia",
4     "artista": "Luna Morada",
5     "album": "Sueños Acuáticos",
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     "duenos_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 perdi el miedo a postear mi primer libro! #book #feliz",
31     "lista_de_likes": ["lucas_dev", "ani.lu", "mariox23", "sofia_dsg"],
32     "fecha_publicacion": "2023-06-24",
33     "comentarios": [

```

Output from the Shell tab:

```

--- Canción ---
titulo: Caminando bajo la lluvia
artista: Luna Morada
album: Sueños Acuáticos
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
duenos_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 perdi el miedo a postear mi primer libro! #book #feliz
lista_de_likes: ['lucas_dev', 'ani.lu', 'mariox23', 'sofia_dsg']
fecha_publicacion: 2023-06-24
comentarios: {'usuario': 'lucas_dev', 'comentario': '¡Esoso! ¡Felicitaciones!', 'usuario': 'ani.lu', 'comentario': 'Qué crack, Harumi!'}


```

Ejercicio N°30 – Gestión de Tareas. –

Descripción:

Este programa en Python crea una aplicación de gestión de tareas usando listas y diccionarios. Cada tarea tiene campos como ID, título y estado.

Incluye funciones para agregar, mostrar, buscar, completar y eliminar tareas.

También tiene un menú interactivo en consola para el usuario.

The image shows a code editor and a terminal window. The code editor displays the Python script for managing tasks, which includes functions for adding, displaying, searching, marking as completed, and deleting tasks. The terminal window shows the execution of the script and its interaction with the user through a command-line menu.

```
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} {tarea['id']} | {tarea['descripcion']} | (Prioridad: {tarea['prioridad']})")
25     print("...") # Separador visual
26
27 def buscar_tarea_por_id(id_buscado):
28     """Devuelve la tarea con el ID buscado o None si no existe"""
29     for tarea in lista_de_tareas:
30         if tarea['id'] == id_buscado:
31             return tarea
32     return None
33
34
35 def marcar_tarea_completada(id_tarea):
36     tarea = buscar_tarea_por_id(id_tarea)
37     if tarea:
38         tarea["completada"] = True
39         print(f" Tarea '{tarea['descripcion']}' marcada como completada.")
40     else:
41         print(f" Error: No se encontró la tarea con ID {id_tarea}.")
42
43 def eliminar_tarea(id_tarea):
44     tarea = buscar_tarea_por_id(id_tarea)
45     if tarea:
46         lista_de_tareas.remove(tarea)
47         print(f" Tarea '{tarea['descripcion']}' eliminada.")
48     else:
49         print(f" Error: No se encontró la tarea con ID {id_tarea}.")
50
51 # Pruebas iniciales
52 agregar_tarea("Estudiar para el examen de Cálculo")
53 agregar_tarea("Hacer las compras", prioridad="alta")
54 mostrar_tareas()
55
56 tarea_encontrada = buscar_tarea_por_id(1)
57 if tarea_encontrada:
58     print("\nBúsqueda exitosa: {tarea_encontrada['descripcion']}")
```

Terminal Output:

```
ejercicios-de-estructura-de-control 0% used ⏪ Deploy 🛡️
For ~/workspace$ python clase13_registros.py
~/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.

==== 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: 1
```

The screenshot shows a code editor with two panes. The left pane displays the Python source code for a todo-list application. The right pane shows the terminal output of running the script.

```

42     def eliminar_tarea(id_tarea):
43         if not tareas_fantasma:
44             print("Búsqueda de tarea inexistente funcionó correctamente.")
45
46         marcar_tarea_completada(1)
47         mostrar_tareas() # Tarea 1 como completada
48         eliminar_tarea()
49         mostrar_tareas() # Tarea 2 eliminada
50         marcar_tarea_completada(99) # intentar marcar tarea inexistente
51
52     # Menú interactivo
53     while True:
54         print("\n----- MENÚ TO-DO LIST -----")
55         print("1. Agregar nueva tarea")
56         print("2. Mostrar todas las tareas")
57         print("3. Marcar tarea como completada")
58         print("4. Eliminar tarea")
59         print("0. Salir")
60         print("Salir")
61
62         opcion = input("Elige una opción: ")
63
64         if opcion == '1':
65             desc = input("Descripción de la nueva tarea: ")
66             prio = input("Prioridad (alta, media, baja): ").lower()
67             if prio not in ["alta", "media", "baja"]:
68                 prio = "media"
69             agregar_tarea(desc, prio)
70
71         elif opcion == '2':
72             mostrar_tareas()
73
74         elif opcion == '3':
75             try:
76                 id_t = int(input("ID de la tarea a completar: "))
77                 marcar_tarea_completada(id_t)
78             except ValueError:
79                 print("Por favor ingresa un número válido para el ID.")
80
81         elif opcion == '4':
82             try:
83                 id_t = int(input("ID de la tarea a eliminar: "))
84                 eliminar_tarea(id_t)
85             except ValueError:
86                 print("Por favor ingresa un número válido para el ID.")
87
88         elif opcion == '0':
89             print("¡Hasta pronto!")
90             break
91
92         else:
93             print("Opción no válida. Intentalo de nuevo.")

```

The terminal output shows the application's menu and user interactions:

```

--- 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: 3

```

This screenshot shows the same code editor and terminal setup as the previous one, but with a different terminal output. The user has chosen option 1 to add a new task.

The terminal output shows:

```

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$ 

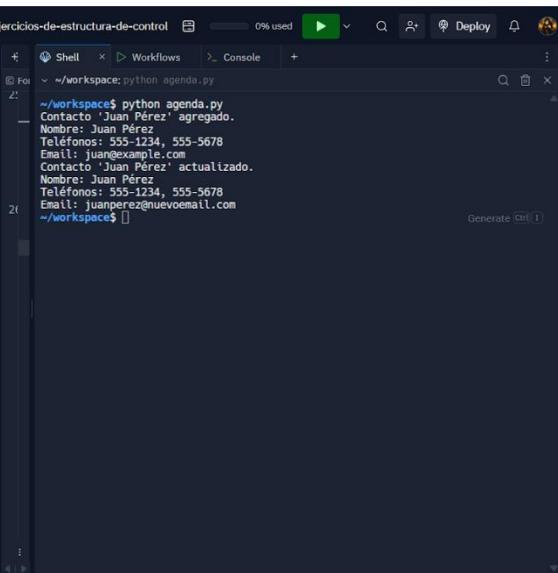
```

Ejercicio N°31 – Agenda Contactos. –

Descripción:

Este programa en Python implementa un **Gestor de Contactos con Interfaz Gráfica (GUI)** utilizando el módulo tkinter, junto con estructuras de datos orientadas a objetos mediante clases. Se definen dos clases principales: Contacto, que representa a cada persona con atributos como ID, nombre, teléfonos y correo electrónico; y Gestor Contactos, que se encarga de administrar la lista de contactos, incluyendo funciones para agregar, editar, eliminar, buscar, guardar y cargar datos en formato JSON. La interfaz gráfica permite al usuario realizar estas acciones de forma intuitiva, ingresando información mediante campos de texto y botones.

Las operaciones se reflejan en una lista visual (Listbox), y se actualizan automáticamente cada vez que el usuario modifica los contactos. Este tipo de aplicación es útil para aprender conceptos como programación orientada a objetos, manejo de archivos JSON y creación de interfaces gráficas básicas en Python.



The screenshot shows a terminal window titled "ejercicios-de-estructura-de-control" running in a workspace. The command "python agenda.py" is executed, and the output is displayed. The output shows the creation of a contact named "Juan Pérez" with phone numbers 555-1234 and 555-5678, and email juanperez@nuevoemail.com. It then updates the contact's email to juanperez@nuevoemail.com.

```
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
```

The screenshot shows a code editor and a terminal window side-by-side.

Code Editor (Left):

```
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"Telefonos: {', '.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')
```

Terminal (Right):

```
~/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$
```