Colección de Nuevos Ejercicios de Python: Manejo de Listas de Listas

A continuación, se presenta una colección de 10 ejercicios diseñados para practicar la manipulación de estructuras de datos anidadas en Python, centrándose exclusivamente en listas, tuplas y tipos de datos básicos.

Ejercicio 10: Filtrar Tareas por Prioridad

Objetivo: Escribir una función que filtre una lista de tareas y devuelva solo aquellas que tienen una prioridad específica.

La Función a Implementar:

filtrar_tareas_por_prioridad(tareas_por_proyecto, prioridad_buscada)

Parámetros:

- tareas_por_proyecto: Una lista de listas. Cada lista interna representa un proyecto y contiene tuplas (nombre_tarea, responsable, prioridad), donde prioridad es un número entero.
- prioridad_buscada: Un número entero que representa la prioridad de las tareas a buscar.

Valor a devolver: Una tupla con dos matrices (listas de listas), donde la primera matriz contiene los nombres de las tareas por filas según están en las tareas_por_proyecto y la segunda matriz la lista de los responsables por filas según están en las tareas_por_proyecto, pero en ambos casos, sólo cuya prioridad sea la buscada.

Ejemplo de Uso:

```
# --- Datos de entrada ---
```

```
tareas = [
    [('Diseñar UI', 'Ana', 1), ('API', 'Juan', 2), ('Lib', 'Yoel', 1)],
    [('Testear UI', 'Eva', 1), ('Desplegar en servidor', 'Juan', 3)]
]

# --- Implementación del código solicitado ---
# TO DO

# --- Llamada y muestra del resultado ---
resultado = filtrar_tareas_por_prioridad(tareas, 1)
```

```
print("Tareas con prioridad 1:")
```

```
print(resultado)
```

Resultado esperado:

```
Tareas con prioridad 1:
```

```
([
   [('Diseñar UI', 'Lib')],
   [('Testear UI')]
 ],
   [('Ana', 'Yoel')],
   [('Eva')]
])
```

Ejercicio 11: Calcular Ventas Totales por Producto

Objetivo: Crear una función que calcule las ventas totales para una lista de productos a partir de una lista de transacciones de varias tiendas.

La Función a Implementar:

calcular_ventas_totales(ventas_por_tienda, lista_de_productos)

Parámetros:

- ventas_por_tienda: Una lista de listas. Cada lista interna representa las ventas de una tienda y contiene tuplas (id_producto, cantidad_vendida, precio_unitario).
- lista_de_productos: Lista de id_productos de los cuales hay que calcular las ventas totales.

Valor a devolver: Una lista de tuplas con el id_producto y el total de ingresos generados para ese producto en todas las tiendas.

Ejemplo de Uso:

```
# --- Datos de entrada ---
```

```
ventas = [
    [("P001", 10, 5.0), ("P002", 5, 12.0)],
    [("P001", 8, 5.0), ("P003", 3, 25.0)],
    [("P002", 12, 12.0), ("P003", 6, 25.0)]
]
```

```
# --- Implementación del código solicitado ---
# TO DO

# --- Llamada y muestra del resultado ---
ventas_totales = calcular_ventas_totales(ventas)
print("Ventas totales por producto:")
```

```
print(ventas_totales)
```

Resultado esperado:

```
Ventas totales por producto:
```

```
[('P001', 90.0), ('P002', 204.0), ('P003', 225.0)]
```

Ejercicio 12: Encontrar el Camino de Mínimo Coste

Objetivo: Escribir una función que encuentre el camino de mínimo coste en una matriz que representa un mapa de costes. Solo se puede mover hacia abajo o hacia la derecha.

La Función a Implementar:

encontrar_camino_minimo(mapa_costes)

Parámetros:

- mapa_costes: Una lista de listas de números enteros, donde cada número representa el coste de pasar por esa celda.

Valor a devolver: El coste total del camino de mínimo coste desde la celda (0, 0) hasta la celda inferior derecha.

Ejemplo de Uso:

```
# --- Datos de entrada ---
```

```
mapa = [
    [1, 3, 1],
    [1, 5, 1],
    [4, 2, 1]
]

# --- Implementación del código solicitado ---
```

```
# TO DO

# --- Llamada y muestra del resultado ---
coste_minimo = encontrar_camino_minimo(mapa)
```

```
print("El coste mínimo del camino es:", coste_minimo)
```

Resultado esperado:

```
El coste mínimo del camino es: 7
```

Ejercicio 13: Identificar Empleado con Mejor Rendimiento

Objetivo: Escribir una función que identifique al empleado con el mejor rendimiento promedio en base a una lista de evaluaciones trimestrales.

La Función a Implementar:

identificar_mejor_empleado(evaluaciones)

Parámetros:

- evaluaciones: Una lista de listas. Cada lista interna representa un trimestre y contiene tuplas (nombre_empleado, ventas_realizadas, satisfaccion_cliente), donde satisfaccion_cliente es una puntuación de 1 a 5.

Valor a devolver: El nombre del empleado con el mayor "score" de rendimiento. El score se calcula como (ventas_realizadas * 0.7) + (satisfaccion_cliente * 0.3). Se debe devolver el nombre del empleado con el score promedio más alto en todos los trimestres.

Ejemplo de Uso:

```
# --- Datos de entrada ---
```

```
evaluaciones = [
    [('Ana', 100, 4), ('Juan', 120, 3)],
    [('Ana', 110, 5), ('Juan', 100, 4)]
]

# --- Implementación del código solicitado ---
# TO DO

# --- Llamada y muestra del resultado ---
```

```
mejor_empleado = identificar_mejor_empleado(evaluaciones)
```

```
print("El empleado con mejor rendimiento es:", mejor_empleado)
```

Resultado esperado:

```
El empleado con mejor rendimiento es: Juan
```

Ejercicio 14: Rotar Matriz 90 Grados

Objetivo: Implementar una función que rote una matriz (lista de listas) 90 grados en el sentido de las agujas del reloj.

La Función a Implementar:

rotar_matriz(matriz)

Parámetros:

- matriz: Una lista de listas de cualquier tipo de dato básico (números, cadenas, etc.). Se debe asumir que la matriz de entrada no es vacía y es cuadrada (mismo número de filas y columnas).

Valor a devolver: Una nueva lista de listas que es la matriz rotada 90 grados en el sentido de las agujas del reloj.

Ejemplo de Uso:

```
# --- Datos de entrada ---
```

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

# --- Implementación del código solicitado ---
# TO DO

# --- Llamada y muestra del resultado ---
matriz_rotada = rotar_matriz(matriz_original)
print("Matriz original:")
for fila in matriz_original:
    print(fila)
```

```
print("\nMatriz rotada:")
for fila in matriz_rotada:
    print(fila)
```

Resultado esperado:

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

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

Ejercicio 15: Consolidar Registros de Pacientes

Objetivo: Consolidar los registros de una lista de pacientes de diferentes hospitales en un único registro maestro.

La Función a Implementar:

consolidar_registros_pacientes(registros_por_hospital, lista_pacientes)

Parámetros:

- registros_por_hospital: Una lista de listas. Cada lista interna representa los registros de un hospital y contiene tuplas (id_paciente, diagnostico, fecha_admision).
- lista_pacientes: Una lista de id_pacientes a consolidar.

Valor a devolver: Una lista de listas con los valores id_paciente y tuplas (diagnostico, fecha_admision) de cada paciente.

Ejemplo de Uso:

```
# --- Datos de entrada ---
registros = [
    [("P01", "Gripe", "2023-01-10"), ("P02", "Fractura", "2023-01-12")],
```

```
    [("P01", "Neumonía", "2023-03-15"), ("P03", "Apendicitis", "2023-03-
20")],
    [("P02", "Revisión", "2023-02-20")]
]

# --- Implementación del código solicitado ---
# TO DO

# --- Llamada y muestra del resultado ---
registros_maestros = consolidar_registros_pacientes(registros)
print("Registros maestros de pacientes:")
```

```
print(registros_maestros)
```

Resultado esperado:

```
Registros maestros de pacientes:
```

```
[
   ['P01', ('Gripe', '2023-01-10'), ('Neumonía', '2023-03-15')],
   ['P02', ('Fractura', '2023-01-12'), ('Revisión', '2023-02-20')],
   ['P03', ('Apendicitis', '2023-03-20')]
]
```

Ejercicio 16: Generar Reporte de Vuelos por Aerolínea

Objetivo: Crear una función que genere un reporte del total de pasajeros por cada aerolínea a partir de una lista de vuelos.

La Función a Implementar:

generar_reporte_vuelos(vuelos_por_aerolinea)

Parámetros:

- vuelos_por_aerolinea: Una lista de listas. Cada lista interna representa los vuelos de una aerolínea y contiene tuplas (id_vuelo, destino, num_pasajeros).

Valor a devolver: Una lista de listas de nombres de las aerolíneas (se pueden generar como "Aerolinea 0", "Aerolinea 1", etc.) y los valores con el número total de pasajeros de cada aerolínea.

Ejemplo de Uso:

```
# --- Datos de entrada ---
```

```
vuelos = [
    [("V001", "JFK", 250), ("V002", "LAX", 300)],
    [("V003", "LHR", 200), ("V004", "CDG", 220)],
    [("V005", "JFK", 260)]
]

# --- Implementación del código solicitado ---
# TO DO

# --- Llamada y muestra del resultado ---
reporte = generar_reporte_vuelos(vuelos)
print("Reporte de vuelos por aerolínea:")
```

```
print(reporte)
```

Resultado esperado:

```
Reporte de vuelos por aerolínea:
```

```
[['Aerolinea 0', 550], ['Aerolinea 1', 420], ['Aerolinea 2', 260]]
```

Ejercicio 17: Encontrar la Palabra más Frecuente en Documentos

Objetivo: Escribir una función que encuentre la palabra más frecuente en una colección de documentos.

La Función a Implementar:

palabra_mas_frecuente(documentos)

Parámetros:

- documentos: Una lista de listas de strings, donde cada lista interna representa un documento y contiene palabras.

Valor a devolver: La palabra que aparece con mayor frecuencia en todos los documentos. Si hay un empate, se puede devolver cualquiera de las palabras empatadas.

Ejemplo de Uso:

```
# --- Datos de entrada ---
```

```
documentos = [
    ["hola", "mundo", "hola"],
    ["python", "es", "genial"],
    ["hola", "python"]
]

# --- Implementación del código solicitado ---
# TO DO

# --- Llamada y muestra del resultado ---
mas_frecuente = palabra_mas_frecuente(documentos)
```

```
print("La palabra más frecuente es:", mas_frecuente)
```

Resultado esperado:

```
La palabra más frecuente es: hola
```

Ejercicio 18: Validar Sudoku

Objetivo: Escribir una función que valide si un tablero de Sudoku de 9x9 está correctamente resuelto.

La Función a Implementar:

validar_sudoku(tablero)

Parámetros:

- tablero: Una lista de listas de 9x9 que representa el tablero de Sudoku. Los números van del 1 al 9, y los espacios vacíos se pueden representar con 0.

Valor a devolver: True si el tablero es válido, False en caso contrario. Un tablero es válido si cada fila, cada columna y cada subcuadrícula de 3x3 contiene los números del 1 al 9 sin repetición.

Ejemplo de Uso:

```
# --- Datos de entrada ---
```

```
tablero_valido = [
```

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

tablero_invalido = [
    [5, 3, 4, 6, 7, 8, 9, 1, 2],
    [6, 7, 2, 1, 9, 5, 3, 4, 8],
    [1, 9, 8, 3, 4, 2, 5, 6, 7],
    [8, 5, 9, 7, 6, 1, 4, 2, 3],
    [4, 2, 6, 8, 5, 3, 7, 9, 1],
    [7, 1, 3, 9, 2, 4, 8, 5, 6],
    [9, 6, 1, 5, 3, 7, 2, 8, 4],
    [2, 8, 7, 4, 1, 9, 6, 3, 5],
    [3, 4, 5, 2, 8, 6, 1, 7, 7]  # Error aquí
]

# --- Implementación del código solicitado ---
# TO DO

# --- Llamada y muestra del resultado ---
print("¿El tablero válido es válido?", validar_sudoku(tablero_valido))
```

```
print("¿El tablero inválido es válido?",
validar_sudoku(tablero_invalido))
```

Resultado esperado:

```
¿El tablero válido es válido? True
```

```
¿El tablero inválido es válido? False
```

Ejercicio 19: Generar Calendario de Eventos

Objetivo: Escribir una función que genere un calendario de eventos a partir de una lista de eventos programados.

La Función a Implementar:

generar_calendario(eventos)

Parámetros:

- eventos: Una lista de listas. Cada lista interna representa un día y contiene tuplas (nombre_evento, hora_inicio, duracion_minutos).

Valor a devolver: Una lista de listas con los días (se pueden generar como "Día 0", "Día 1", etc.) y una lista de tuplas (nombre_evento, hora_inicio, hora_fin).

Ejemplo de Uso:

```
# --- Datos de entrada ---
```

```
eventos = [
    [("Reunión A", "09:00", 60), ("Reunión B", "11:00", 30)],
    [("Taller", "14:00", 120)]
]

# --- Implementación del código solicitado ---
# TO DO

# --- Llamada y muestra del resultado ---
calendario = generar_calendario(eventos)
print("Calendario de eventos:")
```

```
print(calendario)
```

Resultado esperado:

```
Calendario de eventos:
```

```
[['Día 0', [('Reunión A', '09:00', '10:00'), ('Reunión B', '11:00',
'11:30')]], ['Día 1', [('Taller', '14:00', '16:00')]]]
```