Módulo 0. Ejemplos 1

Filosofía: Este fichero no explica. Demuestra. Asume que conoces la sintaxis del Módulo 0. Aquí combinamos slicing, comprehensions y lambda para resolver problemas más realistas.

1. Slicing Aplicado (Parsing de Logs)

El slicing no es solo para listas, es fundamental para extraer datos de strings con formato fijo o semi-fijo.

```
# Escenario: Tenemos líneas de log con un formato fijo:
# [TIMESTAMP][NIVEL] MENSAJE
log_line_1 = "[2024-10-25T10:30:01Z][INFO] Sensor T-01 operativo."
log_line_2 = "[2024-10-25T10:31:05Z][ERROR] Fallo en sensor H-02 (Timeout)."

log_completo = [log_line_1, log_line_2]

# --- Ejemplo 1.1: Extraer partes usando slicing ---
# Queremos (timestamp, nivel, mensaje)

print("--- Ejemplo 1.1: Slicing en Logs ---")
for line in log_completo:
    # El timestamp siempre tiene 22 caracteres (incluyendo corchetes)
    timestamp = line[1:21] # Slicing de 1 (evita '[') a 21 (evita ']')

    # El nivel está justo después
    nivel_start = 22 # ] + [
    nivel_end = line.find(']', nivel_start) # Buscar el ']' desde la pos 22
    nivel = line[nivel_start:nivel_end]

    # El mensaje es el resto
    mensaje = line[nivel_end + 2:] # +2 para saltar '] '

    print(f"  Timestamp: {timestamp}\n  Nivel: {nivel}\n  Mensaje: {mensaje}\n--")


# --- Ejemplo 1.2: Slicing + List Comprehension ---
# Crear una lista de tuplas (nivel, mensaje) solo para logs de 'ERROR'

print("\n--- Ejemplo 1.2: Slicing + Comprehension ---")
errores = [
    (line[22:line.find(']', 22)], line[line.find(']', 22) + 2:])
```

```
    for line in log_completo if line[22:27] == 'ERROR'
]

print(f"Errores encontrados: {errores}")
```

2. Comprehensions Complejas (Transformación de Datos)

Aquí es donde se ve el poder real para reestructurar datos.

```
# --- Ejemplo 2.1: Aplanar una Matriz (Nested Comprehension) ---
# Escenario: Datos de sensores que vienen en lotes (listas de listas)
lotes_datos = [
    [10.2, 10.5, 10.1],
    [9.8, 9.9],
    [11.1, 10.9, 11.0, 11.2]
]

# Queremos una única lista con todas las lecturas
print("\n--- Ejemplo 2.1: Aplanar Matriz ---")
lecturas_planas = [lectura for lote in lotes_datos for lectura in lote]
print(f"Lecturas aplanadas: {lecturas_planas}")

# --- Ejemplo 2.2: Aplanar y Filtrar ---
# Queremos lo mismo, pero solo lecturas "calientes" (> 10.0)
print("\n--- Ejemplo 2.2: Aplanar y Filtrar ---")
lecturas_calientes = [
    lectura for lote in lotes_datos
            for lectura in lote
            if lectura > 10.0
]
print(f"Lecturas calientes: {lecturas_calientes}")

# --- Ejemplo 2.3: Dictionary Comprehension (Invertir un Diccionario) ---
# Escenario: Mapeo de ID de sensor a nombre legible
mapa_sensores = {
    'T-01': 'Temperatura_Exterior_Norte',
    'H-02': 'Humedad_Interior',
    'P-01': 'Presion_Atmosferica'
}

# Queremos el mapa inverso (Nombre -> ID)
# Nota: ¡Esto solo funciona si los valores son únicos!
```

```
print("\n--- Ejemplo 2.3: Invertir Diccionario ---")
mapa_inverso = {
    nombre: id_sensor
    for id_sensor, nombre in mapa_sensores.items()
}
print(f"Mapa inverso: {mapa_inverso}")

# --- Ejemplo 2.4: Dictionary Comprehension + Lógica ---
# Escenario: Crear un diccionario de promedios desde 'lotes_datos'
print("\n--- Ejemplo 2.4: Dict Comprehension + Lógica ---")
promedios = {
    f"lote_{i+1}": sum(lote) / len(lote)
    for i, lote in enumerate(lotes_datos)
    if len(lote) > 0 # Evitar divisiones por cero
}
print(f"Promedios por lote: {promedios}")
```

3. Lambdas en Acción (Ordenación y Mapeo Avanzado)

Las lambdas son el "pegamento" que nos permite personalizar el comportamiento de funciones de orden superior.

```
# --- Ejemplo 3.1: Ordenar por Múltiples Criterios ---
# Escenario: Lista de registros (sensor, estado, timestamp)
registros = [
    ('T-01', 'ERROR', '2024-10-25T12:00:00Z'),
    ('H-02', 'OK', '2024-10-25T11:00:00Z'),
    ('T-01', 'OK', '2024-10-25T10:00:00Z'),
    ('H-02', 'ERROR', '2024-10-25T13:00:00Z')
]

# Queremos ordenar por sensor (alfabético) y LUEGO por timestamp (reciente primero)
# La clave (key) debe devolver una tupla. Python ordena por el primer
# elemento de la tupla, luego el segundo, etc.
print("\n--- Ejemplo 3.1: Ordenación Múltiple ---")

# Nota: No podemos ordenar 'reverse=True' directamente en el timestamp.
# Una forma es ordenar por timestamp primero, y luego por sensor.
# La forma más robusta es usar la tupla.
# Para ordenar el timestamp (string) de forma inversa, no podemos usar
# reverse=True en la lambda. Si fueran números, usaríamos -x.
# Como son strings, lo dejamos en orden natural (ascendente).
```

```
orden_multiple = sorted(
    registros,
    key=lambda r: (r[0], r[2]) # (sensor, timestamp)
)
print(f"Ordenado (Sensor asc, Timestamp asc):\n{orden_multiple}")

# --- Ejemplo 3.2: Ordenar por Slicing + Lambda ---
# Escenario: IDs de sensor con prefijo numérico "prioritario"
# Formato: "P<prioridad>-<nombre>"
ids = ["P3-TEMP_A", "P1-WIND_B", "P5-HUM_A", "P2-TEMP_B"]

# Queremos ordenar por la prioridad (el número después de 'P')
print("\n--- Ejemplo 3.2: Ordenar por Slicing + Lambda ---")
orden_prioridad = sorted(
    ids,
    key=lambda id_str: int(id_str[1:id_str.find('-')]) # Slicing para coger el número
)
print(f"Ordenado por prioridad: {orden_prioridad}")

# --- Ejemplo 3.3: map() y filter() con Lambdas ---
# (Alternativa a las comprehensions, útil en ciertos contextos)
print("\n--- Ejemplo 3.3: map y filter ---")
numeros = [1, 2, 3, 4, 5, 6, 7, 8]

# Queremos el cuadrado de los números pares
# 1. Filtrar los pares
pares = list(filter(lambda x: x % 2 == 0, numeros))
print(f"Pares: {pares}")

# 2. Mapear al cuadrado
cuadrados_pares = list(map(lambda x: x**2, pares))
print(f"Cuadrados de pares: {cuadrados_pares}")

# La versión en comprehension (generalmente preferida por legibilidad):
# cuadrados_pares_comp = [x**2 for x in numeros if x % 2 == 0]
# print(f"Comp: {cuadrados_pares_comp}")
```

4. JSON + Comprehensions + Lambdas

Escenario: Recibimos un JSON (como string) de una API. Contiene datos de múltiples estaciones, pero está anidado y "sucio".
Objetivo:

1. Parsear el JSON.
2. "Aplanar" la estructura: queremos una lista única de lecturas.
3. Filtrar: solo nos interesan lecturas "válidas" (estado 'OK' y valor no nulo).
4. Transformar: convertir la temperatura de Celsius a Fahrenheit.
5. Ordenar: la lista final debe estar ordenada por valor (descendente).

```
import json

# --- El JSON de entrada (como string) ---
json_data = """
{
  "metadata": {"api_version": "1.2", "timestamp": "2024-10-25T15:00:00Z"},
  "estaciones": [
    {
      "id": "Estacion_Norte",
      "sensores": [
        {"tipo": "TEMP", "estado": "OK", "valor_c": 12.5},
        {"tipo": "HUM", "estado": "ERROR", "valor_c": null},
        {"tipo": "TEMP", "estado": "OK", "valor_c": 13.1}
      ]
    },
    {
      "id": "Estacion_Sur",
      "sensores": [
        {"tipo": "TEMP", "estado": "MAINT", "valor_c": -10.0},
        {"tipo": "TEMP", "estado": "OK", "valor_c": -5.5},
        {"tipo": "HUM", "estado": "OK", "valor_c": 80.2}
      ]
    },
    {
      "id": "Estacion_Oeste",
      "sensores": [
        {"tipo": "TEMP", "estado": "OK", "valor_c": null}
      ]
    }
  ]
}
"""

print("\n--- Ejemplo 4: Desafío BOSS ---")

# --- 1. Parsear el JSON ---
datos = json.loads(json_data)
```

```
# --- 2, 3 y 4. Aplanar, Filtrar y Transformar (Todo en una comprehension) ---
# Usamos una nested comprehension para iterar por estaciones, y luego por sensores
lecturas_procesadas = [
    {
        "estacion": estacion['id'],
        "temp_f": (sensor['valor_c'] * 9/5) + 32  # 4. Transformar
    }
    for estacion in datos['estaciones']         # Nivel 1 (estación)
    for sensor in estacion['sensores']          # Nivel 2 (sensor)
    if sensor['tipo'] == 'TEMP'                 # 3. Filtrar por tipo
    and sensor['estado'] == 'OK'                # 3. Filtrar por estado
    and sensor['valor_c'] is not None           # 3. Filtrar por valor no nulo
]

print(f"Datos procesados y aplanados:\n{lecturas_procesadas}")

# --- 5. Ordenar usando Lambda ---
# Ordenamos la lista final por la nueva clave 'temp_f' (descendente)
lecturas_ordenadas = sorted(
    lecturas_procesadas,
    key=lambda x: x['temp_f'],
    reverse=True
)

print(f"\nDatos ordenados (Fahrenheit desc):\n{lecturas_ordenadas}")

# --- Bonus: Guardar el resultado limpio ---
json_limpio_string = json.dumps(lecturas_ordenadas, indent=2)
# print(f"\nJSON de salida:\n{json_limpio_string}")
# (Se podría guardar en un fichero con json.dump)
```