Módulo 0: DataScientist Toolbox

Filosofía: Olvida los bucles for lentos y verbosos. Este módulo te enseñará a usar las herramientas nativas de Python para manipular datos de forma rápida, eficiente y legible. Cada línea de código que evitemos escribir es una victoria.

1. Slicing (El Bisturí)

El "slicing" (rebanado) es la sintaxis que nos permite seleccionar subconjuntos de secuencias (listas, tuplas, strings).
Sintaxis Principal: secuencia[start:stop:step]

- start: Índice de inicio (incluido). Si se omite, es 0 (el principio).
- stop: Índice de fin (¡**no** incluido!). Si se omite, es hasta el final.
- step: El "paso" o incremento. Si se omite, es 1.

Ejemplos Básicos

```
# Nuestra lista de datos de ejemplo
sensores = ['TEMP_A', 'TEMP_B', 'HUM_A', 'PENG_C', 'ICE_S', 'WIND_A']

# Coger los elementos del índice 1 al 3 (el 4 no se incluye)
# (TEMP_B, HUM_A, PENG_C)
print(f"Del 1 al 4: {sensores[1:4]}")

# Coger desde el principio hasta el índice 3 (el 3 no se incluye)
# (TEMP_A, TEMP_B, HUM_A)
print(f"Desde el inicio hasta el 3: {sensores[:3]}")

# Coger desde el índice 2 hasta el final
# (HUM_A, PENG_C, ICE_S, WIND_A)
print(f"Desde el 2 hasta el final: {sensores[2:]}")
```

Ejemplos Avanzados (El "truco" de Data Science)

Aquí es donde se pone potente.

```
# --- Indexación Negativa ---
# Significa "empezar a contar desde el final".
# -1 es el último elemento.

# Coger el último elemento
# (WIND_A)
print(f"Último elemento: {sensores[-1]}")
```

```
# Coger los últimos 3 elementos (muy útil para "últimas lecturas")
# (PENG_C, ICE_S, WIND_A)
print(f"Últimos 3 elementos: {sensores[-3:]}")

# --- Uso de 'step' ---
# Coger elementos alternos (de 2 en 2)
# (TEMP_A, HUM_A, ICE_S)
print(f"Elementos alternos (pares): {sensores[::2]}")

# Coger elementos alternos, empezando por el segundo (impares)
# (TEMP_B, PENG_C, WIND_A)
print(f"Elementos alternos (impares): {sensores[1::2]}")

# --- El 'step' negativo: INVERTIR ---
# Esto es un clásico. step = -1 invierte la lista.
# (WIND_A, ICE_S, PENG_C, HUM_A, TEMP_B, TEMP_A)
print(f"Lista invertida: {sensores[::-1]}")
```

Práctica de Slicing

```
# Tenemos 20 lecturas de temperatura
lecturas = [21, 22, 23, 20, 19, 18, 22, 24, 25, 23, 21, 20, 19, 22, 23, 25, 26, 27, 28, 29]

# 1. Coge las últimas 5 lecturas
ultimas_5 = lecturas[-5:]
print(f"Práctica 1 (Últimas 5): {ultimas_5}")

# 2. Coge una muestra de cada 3 lecturas
cada_3 = lecturas[::3]
print(f"Práctica 2 (Cada 3): {cada_3}")

# 3. Coge las 10 primeras lecturas e inviértelas
primeras_10_invertidas = lecturas[:10][::-1]
print(f"Práctica 3 (10 primeras invertidas): {primeras_10_invertidas}")
```

2. List Comprehensions (El Nuevo for)

Las "list comprehensions" (comprensiones de listas) son la forma más *pythónica* de crear listas. Son más rápidas y legibles que un bucle for para crear una lista.

Sintaxis 1: Mapeo Simple

Formato: [expresion for item in iterable]

Ejemplo: Convertir temperaturas de Celsius a Fahrenheit.

```
# Lista de temperaturas en Celsius
celsius_temps = [0, 10, 20, 30, -5]

# --- El método LENTO (¡NO HACER!) ---
fahrenheit_temps_loop = []
for temp in celsius_temps:
    fahrenheit_temps_loop.append((temp * 9/5) + 32)
# print(fahrenheit_temps_loop)

# --- El método RÁPIDO (¡HACER SIEMPRE!) ---
fahrenheit_temps_comp = [(temp * 9/5) + 32 for temp in celsius_temps]

print(f"Celsius: {celsius_temps}")
print(f"Fahrenheit (comp): {fahrenheit_temps_comp}")
```

Sintaxis 2: Mapeo con Filtro

Formato: [expresion for item in iterable if condicion]
Ejemplo: Quedarnos solo con las temperaturas positivas Y convertirlas a Fahrenheit.

```
# Solo convertir si la temperatura es > 0
positivas_fahrenheit = [(temp * 9/5) + 32 for temp in celsius_temps if temp > 0]
print(f"Positivas en Fahrenheit: {positivas_fahrenheit}")
```

Sintaxis 3: Mapeo con if-else

Formato: [expresion_si_true if condicion else expresion_si_false for item in iterable]
Ejemplo: Etiquetar temperaturas como "caliente" (>25) o "frio" (<=25).

```
lecturas = [21, 22, 23, 26, 27, 28, 29, 15]

etiquetas = ["caliente" if temp > 25 else "frio" for temp in lecturas]
print(f"Lecturas: {lecturas}")
print(f"Etiquetas: {etiquetas}")
```

Práctica de List Comprehensions

```
# Datos de entrada: Nombres de sensores "sucios"
nombres_sensores = ["  temp_A", "hum_B ", " sensor_TEMP_C", "wind_A"]

# 1. Limpiar la lista:
#    Quita espacios en blanco (método .strip())
#    Conviértelos a mayúsculas (método .upper())
```

```
sensores_limpios = [nombre.strip().upper() for nombre in nombres_sensores]
print(f"Práctica 1 (Limpios): {sensores_limpios}")

# 2. Filtrar la lista limpia:
#    Quédate solo con los sensores que contienen "TEMP"
sensores_temp = [sensor for sensor in sensores_limpios if "TEMP" in sensor]
print(f"Práctica 2 (Solo TEMP): {sensores_temp}")

# 3. (Avanzado) Dictionary Comprehensions
#    Crea un diccionario con el nombre limpio como clave
#    y su longitud como valor.
longitudes = {sensor: len(sensor) for sensor in sensores_limpios}
print(f"Práctica 3 (Diccionario): {longitudes}")
```

3. Funciones Lambda (Funciones de usar y tirar)

Una función lambda es una pequeña función anónima (sin nombre). Se define en una sola línea.
Sintaxis: lambda argumentos: expresion
Son limitadas (solo una expresión), pero su poder no está en usarlas solas, sino en **pasarlas como argumento** a otras funciones.

Uso Principal: key en sorted(), min(), max()

Aquí es donde brillan. La key le dice a la función *con qué criterio* debe ordenar o comparar.
Ejemplo: Ordenar una lista de diccionarios (algo *muy* común).

```
# Lista de registros de pingüinos
conteo_pinguinos = [
   {'estacion': 'Base_A', 'conteo': 150},
   {'estacion': 'Base_C', 'conteo': 80},
   {'estacion': 'Base_B', 'conteo': 210}
]

# --- Ordenar por nombre de estación (alfabético) ---
# La lambda recibe un elemento 'registro' y devuelve 'registro['estacion']'
orden_nombre = sorted(conteo_pinguinos, key=lambda registro: registro['estacion'])
print(f"Ordenado por nombre:\n{orden_nombre}\n")

# --- Ordenar por número de pingüinos (ascendente) ---
# La lambda recibe 'registro' y devuelve 'registro['conteo']'
orden_conteo = sorted(conteo_pinguinos, key=lambda registro: registro['conteo'])
print(f"Ordenado por conteo:\n{orden_conteo}\n")
```

```
# --- Encontrar la estación con MÁS pingüinos ---
max_pinguinos = max(conteo_pinguinos, key=lambda registro: registro['conteo'])
print(f"Máximo conteo:\n{max_pinguinos}\n")
```

Adelanto de Pandas: apply()

En pandas (Módulo 2), usaremos lambda constantemente con .apply() para transformar columnas.

```
# NO EJECUTAR ESTO, ES UN EJEMPLO CONCEPTUAL
# df['temp_fahrenheit'] = df['temp_celsius'].apply(lambda x: (x * 9/5) + 32)
```

Práctica de Lambdas

```
lista_datos = [
   ('sensor-12', '2023-10-25T10:00:00Z', 5.5),
   ('sensor-03', '2023-10-25T10:01:00Z', 6.2),
   ('sensor-12', '2023-10-25T10:02:00Z', 5.4)
]

# 1. Ordena la lista por el valor de la lectura (tercer elemento, índice 2)
orden_valor = sorted(lista_datos, key=lambda x: x[2])
print(f"Práctica 1 (Ordenado por valor): {orden_valor}")

# 2. Encuentra el registro con la lectura más baja
min_valor = min(lista_datos, key=lambda x: x[2])
print(f"Práctica 2 (Lectura más baja): {min_valor}")
```

4. Manejo de JSON Nativo (El Formato Universal)

JSON (JavaScript Object Notation) es el estándar de facto para intercambiar datos en la web (APIs) y para ficheros de configuración. Python lo mapea directamente a diccionarios (dict) y listas (list).

Usamos la librería import json (ya viene con Python).

Operaciones clave: load/dump vs loads/dumps

- load / dump: Trabajan con **ficheros** (archivos).
- loads / dumps: Trabajan con **strings** (la 's' es de 'string').

Ejemplo 1: De String a Objeto (loads) y de Objeto a String (dumps)

```
import json

# --- De String (JSON) a Objeto (Dict) ---
```

```
# Imaginemos que esto viene de una API
json_string = '{"id": "TEMP_A", "location": "Sector_Norte", "active": true, "last_reading": 22.5}'

# Convertimos el string a un diccionario de Python
datos_dict = json.loads(json_string)

print(f"Tipo de dato: {type(datos_dict)}")
print(f"Valor de 'location': {datos_dict['location']}")

# --- De Objeto (Dict) a String (JSON) ---
# Modificamos el diccionario
datos_dict['active'] = False
datos_dict['new_field'] = "hola"

# Convertimos el diccionario de nuevo a un string
# El 'indent=4' es CLAVE para que sea legible (pretty-print)
string_salida = json.dumps(datos_dict, indent=4)

print("\n--- JSON 'Bonito' (indentado) ---")
print(string_salida)
```

Ejemplo 2: De Fichero (load) y a Fichero (dump)

```
import json

# Nuestros datos a guardar
configuracion = {
    'sensores': ['TEMP_A', 'HUM_A'],
    'parametros': {
        'rate': 300,
        'timeout': 50
    }
}

# --- Escribir (dump) a un fichero ---
try:
    with open('config.json', 'w') as f:
        # Usamos dump (sin 's') para escribir en el fichero 'f'
        json.dump(configuracion, f, indent=4)
    print("\nFichero 'config.json' guardado.")

    # --- Leer (load) desde un fichero ---
    with open('config.json', 'r') as f:
```

```
        # Usamos load (sin 's') para leer desde el fichero 'f'
        datos_leidos = json.load(f)

    print("Datos leídos de 'config.json':")
    print(datos_leidos)
    print(f"Tasa de muestreo leída: {datos_leidos['parametros']['rate']}")

except Exception as e:
    print(f"Ha ocurrido un error: {e}")
```

Práctica de JSON

1. Crea un diccionario que represente la configuración de un sensor (p.ej., id, tipo, latitud, longitud).
2. Conviértelo a un string JSON *indentado* e imprímelo.
3. Guarda ese mismo diccionario en un fichero llamado sensor_1.json.
4. Vuelve a leer el fichero sensor_1.json en una nueva variable.
5. Imprime solo la latitud de la variable que has leído.

```
import json

# 1. Crear diccionario
mi_sensor = {
    "id": "PENG_C_01",
    "tipo": "Conteo",
    "latitud": -77.8463,
    "longitud": 166.6682
}

# 2. Convertir a string indentado
mi_sensor_string = json.dumps(mi_sensor, indent=4)
print("--- Práctica (String) ---")
print(mi_sensor_string)

# 3. Guardar en fichero
try:
    with open('sensor_1.json', 'w') as f:
        json.dump(mi_sensor, f, indent=4)
    print("\n--- Práctica (Fichero) ---")
    print("Fichero 'sensor_1.json' guardado.")

    # 4. Leer fichero
    with open('sensor_1.json', 'r') as f:
        sensor_leido = json.load(f)
```

```
    # 5. Imprimir latitud
    print(f"Latitud leída: {sensor_leido['latitud']}")

except Exception as e:
    print(f"Ha ocurrido un error: {e}")
```