PYTHON NIVEL INTERMEDIO

INFOTEC

Centro de investigación e innovación en tecnologías de la información y comunicación

TÍTULO DEL PROJECTO

Lista, diccionario, operadores, y collections

PRESENTA

Armando Gabriel Jacinto López

DOCENTE

Vladimir Morales Pérez

Fecha: 12/11/2025

Introducción

Este informe presenta la aplicación de estructuras de datos fundamentales de Python, como Listas, Conjuntos (Sets) y Diccionarios, junto con el uso de módulos avanzados del paquete collections para la manipulación y el análisis de datos. El objetivo del proyecto es transformar datos brutos de registros de clientes y compras para extraer información relevante, como la identificación de clientes nuevos, la eliminación de duplicados y el conteo de frecuencias.

El desarrollo se enfoca en tres áreas clave:

1. Transformación de Listas a Conjuntos: Utilización del operador de diferencia de conjuntos (-) para realizar comparaciones eficientes y filtrar datos.
2. Mantenimiento del Orden y Unicidad: Uso de la clase especializada OrderedDict del módulo collections para eliminar elementos duplicados de una lista mientras se conserva el orden original de inserción.
3. Conteo de Frecuencias y Resumen: Implementación de Counter para un conteo rápido de elementos y la creación de un nuevo Diccionario mediante una comprensión de diccionario (Dictionary Comprehension) para generar resúmenes condicionales.

Desarrollo

El programa procesa dos listas iniciales: compras (que contiene nombres con repeticiones) y registrados (una lista de clientes conocidos).

```
# Datos sugeridos
compras = [
 "Luis", "Ana", "Luis", "Carlos", "Marta", "Ana", "Sofía",
"Elena", "Luis", "Carlos"
]

registrados = [
 "Ana", "Carlos", "Marta", "Elena"
]
```

Ilustración 1 Datos sugeridos

1. Transformación de Datos y Operadores de Conjunto

Los datos iniciales se transforman de listas a Conjuntos (compras_set y registrados_set). Esta conversión es crucial porque los conjuntos ofrecen:

- Unicidad: Eliminan automáticamente los duplicados.
- Operadores Eficientes: Permiten usar operadores matemáticos para la comparación lógica.

```
# Tranformar a lista a sets
compras_set = set(compras)
registrados_set = set(registrados)
```

Ilustración 2 transformación a Conjuntos

La identificación de clientes_nuevos se realiza mediante el Operador de Diferencia (-):

```
# 1. Filtrar clientes nuevos (Diferencia)
clientes_nuevos = compras_set - registrados_set
```

Ilustración 3 Filtrar clientes nuevos (Diferencia)

Esta operación devuelve todos los elementos presentes en compras_set que no están en registrados_set, identificando eficazmente a los clientes que han comprado pero que no estaban registrados previamente.

2. Conservación del Orden con OrderedDict

La lista original compras contenía duplicados, y el requisito es obtener una lista de clientes únicos manteniendo el orden en que aparecieron por primera vez.

Uso de OrderedDict: Aunque las versiones modernas de Python garantizan que los diccionarios estándar recuerden el orden de inserción, se utiliza OrderedDict (una colección especializada) para asegurar la compatibilidad con este requisito en entornos de Python más antiguos.

Proceso: Se itera sobre la lista compras, utilizando cada nombre como clave en el OrderedDict. Debido a que las claves de los diccionarios deben ser únicas, se eliminan los duplicados, y el diccionario preserva el orden de la primera aparición.

Resultado: Finalmente, se extraen las claves del diccionario a una lista (compras_orden) para obtener el listado de clientes únicos y ordenados.

```
# 2. Eliminar duplicados y mantener el orden
# OrderedDict, mantiene valores unicos y el orden de insercion
compras_orden_dict = OrderedDict()

for compra in compras:
    compras_orden_dict[compra] = None

compras_orden = list(compras_orden_dict.keys())

print(compras_orden)
```

Ilustración 4 Implementación de OrderedDict

3. Conteo de Frecuencias con Counter

Para obtener un resumen de cuántas veces compró cada cliente, se utiliza la clase Counter del módulo collections.

Eficiencia: Counter(compras) crea un diccionario que mapea automáticamente los nombres a sus recuentos en una sola línea de código, lo cual es mucho más eficiente que un bucle manual.

```
# 3. Contar cuántas veces se repite cada nombre
contador = Counter(compras)
```

Ilustración 5 Contador de frecuencia de clientes

4. Generación de Diccionarios Condicionales (Dictionary Comprehension)

Se utiliza una Comprensión de Diccionario (Dictionary Comprehension) para crear diccionarios de resumen basados en condiciones específicas.

- Clientes Frecuentes: El diccionario clientes_dict y clientes_frecuentes se generan filtrando el objeto contador para incluir solo aquellos clientes cuyo número de compras (num) es mayor que 1.

```
# 4. Crear un resumen perzonalizado
clientes_dict = {cliente: f'Ha comprado {num} veces' for cliente, num in contador.items() if num > 1}
```

Ilustración 6 Obtención de clientes frecuentes usando Diccionarios Condicionales

- Clientes Únicos no Registrados: Se genera un diccionario que combina las condiciones de los dos pasos anteriores: el cliente debe estar en la lista de clientes_no_registrados Y su recuento de compras debe ser exactamente 1.

```
clientes_unicos = {cliente: num for cliente, num in contador.items() if cliente in clientes_no_registrados and num == 1}
print(clientes_unicos)
```

Ilustración 7 Obtención de clientes no registrados usando Diccionarios Condicionales

Este método resulta en un código extremadamente compacto y legible para la construcción de nuevas estructuras de datos a partir de colecciones existentes.

5. Resultados

```python
# 5. Formato final, Imprime tres bloques
clientes_no_registrados = compras_set - registrados_set

print('Clientes no registrados')
print(clientes_no_registrados)

print('\nClientes unicos')
clientes_unicos = {cliente: num for cliente, num in contador.items() if cliente in clientes_no_registrados and num == 1}
print(clientes_unicos)

print('\nResumen por cliente frecuente (más de 1 compra)')
clientes_frecuentes = {cliente: num for cliente, num in contador.items() if num > 1}
print(clientes_frecuentes)
```

Ilustración 8 Código de las operaciones para el formato final

```
Clientes no registrados
{'Sofía', 'Luis'}

Clientes unicos
{'Sofía': 1}

Resumen por cliente frecuente (más de 1 compra)
{'Luis': 3, 'Ana': 2, 'Carlos': 2}
```

Ilustración 9 Resultados del formato final

Conclusión

El proyecto ha demostrado cómo el manejo inteligente de las colecciones de datos de Python, listas, conjuntos y el módulo collections, permite resolver problemas comunes de procesamiento y análisis de datos de manera elegante y eficiente.

- Eficiencia con Sets: La conversión a Conjuntos y el uso del operador de diferencia (-) simplificaron drásticamente la lógica para identificar nuevos clientes.
- Módulos Especializados: El uso de Counter eliminó la necesidad de construir lógica de conteo manual, mientras que OrderedDict resolvió el requisito de mantener el orden de inserción.
- Sintaxis Concisa: La aplicación de Comprensiones de Diccionario permitió generar resúmenes personalizados con filtros condicionales en una sintaxis limpia y compacta.

En resumen, la combinación de estas estructuras y herramientas avanzadas de Python es fundamental para la manipulación eficiente de colecciones de datos en cualquier aplicación del mundo real.