Glosario

U2. Manejo avanzado de estructura de datos

Python Intermedio

Glosario

« 1.

ChainMap

Herramienta que permite agrupar múltiples diccionarios en una sola vista lógica, facilitando búsquedas en cascada sin mezclar físicamente los datos.

Este ejemplo sirve para **combinar múltiples diccionarios en una sola vista, sin fusionarlos físicamente:**

```python
from collections import ChainMap

dict1 = {"a": 1, "b": 2}
dict2 = {"b": 3, "c": 4}
cm = ChainMap(dict1, dict2)
print(cm["b"])  # 2 (lo toma del primer diccionario)
```

« 2.

Conjuntos Disjuntos

Dos conjuntos se consideran disjuntos cuando no comparten ningún elemento. Esta propiedad puede evaluarse con `isdisjoint()` en Python.

Este ejemplo sirve para **verificar si dos conjuntos no comparten ningún elemento:**

```python
A = {1, 2, 3}
B = {4, 5, 6}
print(A.isdisjoint(B))  # True
```

« 3.

Counter

Estructura especializada, que cuenta la frecuencia de elementos en una colección. Ideal para análisis estadístico o procesamiento de texto.

Este ejemplo sirve para **contar cuántas veces aparece cada elemento en una colección:**

```python
from collections import Counter

frutas = ["manzana", "pera", "manzana", "uva"]
conteo = Counter(frutas)
print(conteo)  # Counter({'manzana': 2, 'pera': 1, 'uva': 1})
```

« 4.

DefaultDict

Variante de diccionario, que asigna automáticamente un valor por defecto a las claves no definidas, evitando errores comunes como KeyError.

Este ejemplo sirve para **evitar errores cuando se accede a una clave inexistente, asignando un valor por defecto:**

```python
from collections import defaultdict

d = defaultdict(int)
d["a"] += 1
print(d)  # defaultdict(<class 'int'>, {'a': 1})
```

« 5.

Deque

Estructura tipo cola de doble extremo, optimizada para inserciones y eliminaciones, desde ambos extremos, con complejidad constante.

Este ejemplo sirve para **manejar una cola de doble extremo con inserciones rápidas en ambos lados:**

```python
from collections import deque

# Crear un deque con algunos elementos
cola = deque(["a", "b", "c"])

# Agregar elementos a la derecha e izquierda
cola.append("d")      # ['a', 'b', 'c', 'd']
cola.appendleft("z")  # ['z', 'a', 'b', 'c', 'd']

# Quitar elementos de la derecha e izquierda
cola.pop()       # quita 'd' → ['z', 'a', 'b', 'c']
cola.popleft()   # quita 'z' → ['a', 'b', 'c']

print(cola)
# Resultado final: deque(['a', 'b', 'c'])
```

« 6.

Dict Comprehension

Construcción que permite crear diccionarios en una sola línea, derivando claves y valores a partir de un iterable. Facilita el procesamiento de estructuras clave-valor de manera declarativa.

Este ejemplo sirve para **construir diccionarios rápidamente, derivando claves y valores de un iterable:**

```python
# Crear un diccionario con números y su cuadrado
cuadrados_dict = {x: x**2 for x in range(1, 6)}
print(cuadrados_dict)  # {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

« 7.

Diferencia

Resultado de extraer del primer conjunto todos los elementos que también están en el segundo.

Este ejemplo sirve para **obtener elementos que están en el primer conjunto, pero no en el segundo:**

```
A = {1, 2, 3}
B = {2, 3}
print(A - B)  # {1}
```

« 8.

Diferencia simétrica

Conjunto resultante de tomar **los elementos que están en uno u otro conjunto, pero no en ambos.**

Este ejemplo lo explica:

```
A = {1, 2}
B = {2, 3}
print(A ^ B)  # {1, 3}
```

« 9.

Intersección

Operación que genera un conjunto con los elementos que se repiten en ambos conjuntos originales.

Este ejemplo sirve **para obtener los elementos que aparecen en ambos conjuntos:**

```
A = {1, 2}
B = {2, 3}
print(A & B)  # {2}
```

« 10.

List comprehension

Expresión compacta para crear listas, aplicando una operación sobre los elementos de un iterable, opcionalmente, puede hacer un filtrado con una condición. Mejora la legibilidad del código, cuando se usa de forma clara.

Este ejemplo sirve para **generar listas de manera compacta, aplicando una operación a cada elemento de un iterable:**

```python
# Crear una lista con los cuadrados de los números del 1 al 5
cuadrados = [x**2 for x in range(1, 6)]
print(cuadrados)  # [1, 4, 9, 16, 25]
```

« 11.

NamedTuple

Subclase de tuple que permite acceder a sus elementos por nombre, en lugar del índice. Mejora la claridad en estructuras inmutables.

Este ejemplo sirve para **crear tuplas, a cuyos elementos se pueden acceder por nombre, aumentando la legibilidad:**

```python
from collections import namedtuple

Persona = namedtuple('Persona', ['nombre', 'edad'])
p1 = Persona("Ana", 30)
print(p1.nombre, p1.edad)  # Ana 30
```

« 12.

OrderedDict

Diccionario que conserva el orden en el que se insertaron los elementos. Útil cuando el orden de los datos importa para el resultado.

Este ejemplo sirve para **mantener el orden de inserción de los elementos en un diccionario:**

```
from collections import OrderedDict

od = OrderedDict()
od["uno"] = 1
od["dos"] = 2
print(od)  # OrderedDict([('uno', 1), ('dos', 2)])
```

« 13.

Set

Soporta operaciones matemáticas como unión, intersección y diferencia, lo que la hace útil para el análisis de pertenencia y de agrupamiento.

Este ejemplo sirve para **almacenar elementos únicos y evitar duplicados, de forma automática:**

```
# Conjunto con elementos únicos
numeros = {1, 2, 2, 3}
print(numeros)  # {1, 2, 3}
```

« 14.

Subconjunto

Un conjunto A es subconjunto de B, si todos sus elementos están contenidos en B. Se verifica con `issubset()`.

Este ejemplo sirve para **verificar si todos los elementos de un conjunto están contenidos en otro:**

```
A = {2, 3}
B = {1, 2, 3, 4}
print(A.issubset(B))  # True
```

« 15.

Superconjunto

Un conjunto A es un superconjunto de B, si contiene todos los elementos de B. En Python se verifica con `issuperset()`.

Este ejemplo sirve para **saber si un conjunto contiene todos los elementos de otro:**

```
A = {1, 2, 3, 4}
B = {2, 3}
print(A.issuperset(B))  # True
```

« 16.

Unión

Operación entre dos conjuntos, que produce un nuevo conjunto con todos los elementos únicos, presentes al menos, en uno de ellos.

Este ejemplo sirve para **combinar elementos únicos de dos conjuntos:**

```
A = {1, 2}
B = {2, 3}
print(A | B)  # {1, 2, 3}
```

« 17.

Update()

Método que permite agregar múltiples elementos a un conjunto desde otro iterable, fusionando su contenido sin duplicados.

Este ejemplo sirve para **agregar varios elementos a un conjunto desde otro iterable, sin duplicados:**

```
frutas = {"manzana", "pera"}
frutas.update({"uva", "kiwi"})
print(frutas)  # {'manzana', 'pera', 'uva', 'kiwi'}
```

Elaboró: Enrique Quezada Próspero
Contenido: Enrique Quezada Próspero
DI: Génesis Equihua

« Referencias:

1. Campesato, O. (2023). *Intermediate Python*. Mercury Learning and Information.
2. Lutz, M. (2013). *Learning Python* (5th ed.). O'Reilly Media.
3. Python Software Foundation. (2025a). *Built-in types — set*. Recuperado el 11 de agosto, de: https://docs.python.org/3/library/stdtypes.html#set-types-set-frozenset
4. Python Software Foundation. (2025b). *collections — Container datatypes*. Recuperado el 11 de agosto, de: https://docs.python.org/3/library/collections.html
5. Python Software Foundation. (2025c). *Data Structures — Python 3.11.4 documentation*. Recuperado el 11 de agosto, de: https://docs.python.org/3/tutorial/datastructures.html