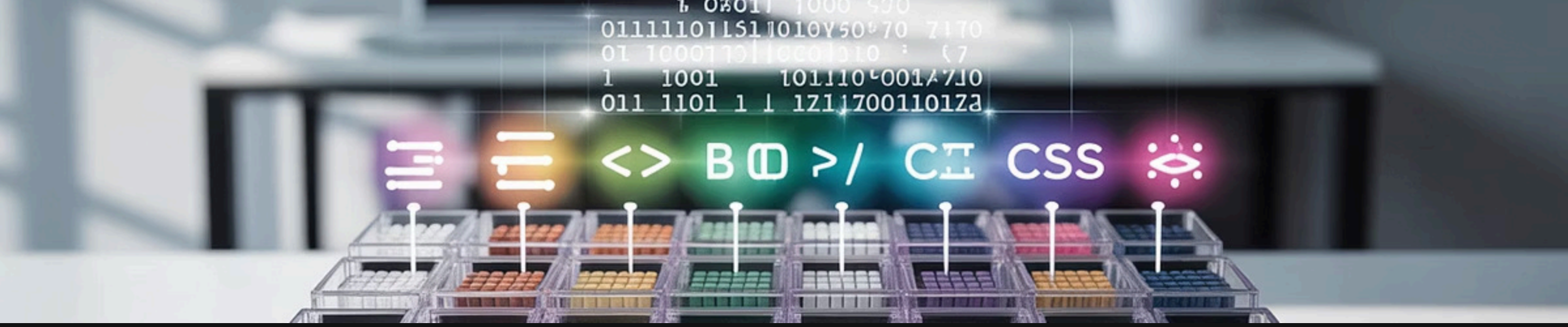
Estructuras de Datos en Python

Bienvenidos a este curso fundamental para todo desarrollador.

Exploraremos cómo Python organiza la información y cómo aprovechar estas estructuras para crear código eficiente.

UTN - Tenicatura Universitaria en Programación



¿Qué es una estructura de datos?

Definición básica

Son formas especiales de organizar datos para usarlos eficientemente.

Analogía práctica

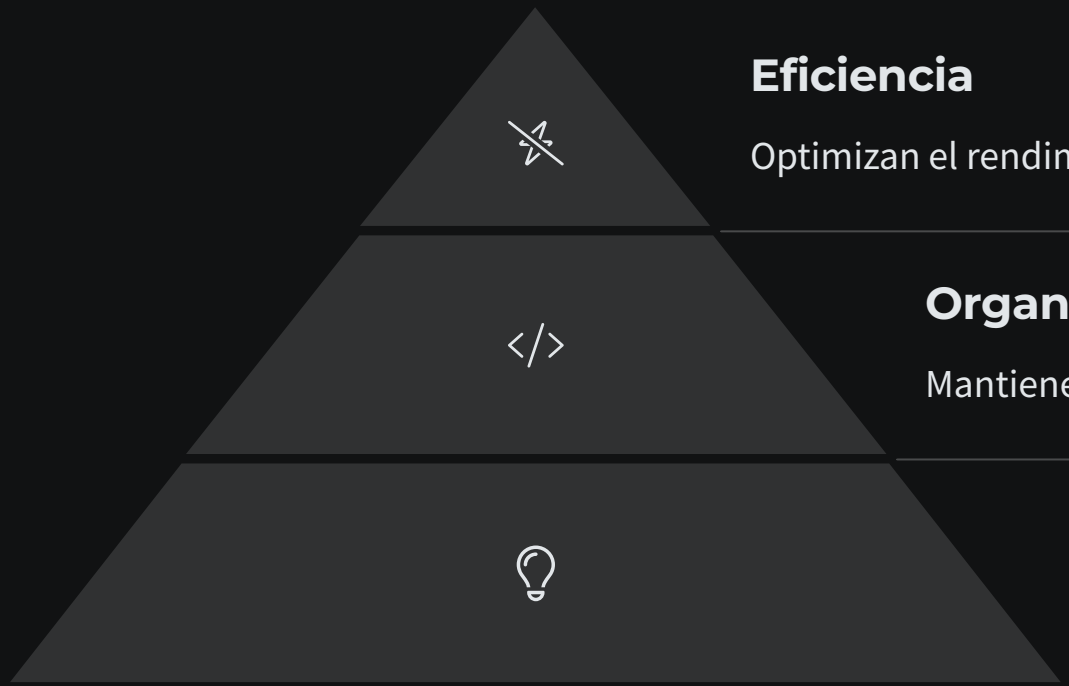
Como una caja de herramientas donde cada una sirve para un problema específico.

Pilar fundamental

Sin ellas, programar sería como construir sin planos.

¿Por qué son esenciales y de gran ayuda?

Listas - Tuplas - Sets - Diccionarios



Eficiencia

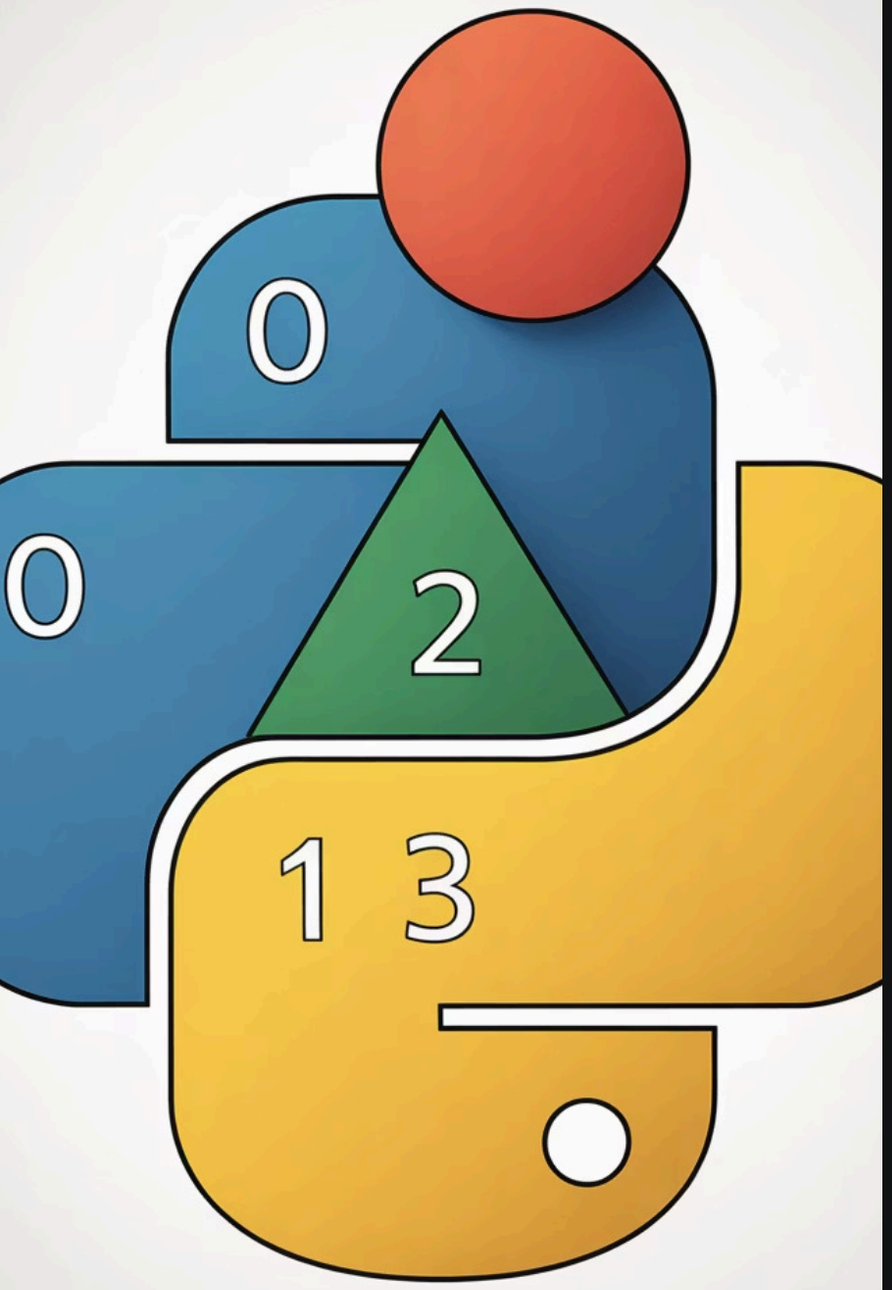
Optimizan el rendimiento del programa

Organización

Mantienen el código limpio y estructurado

Resolución de problemas

Facilitan soluciones elegantes a problemas complejos



Listas:

La estructura más versátil

Ordenadas

Cada elemento tiene una posición específica accesible por índice.

Mutables

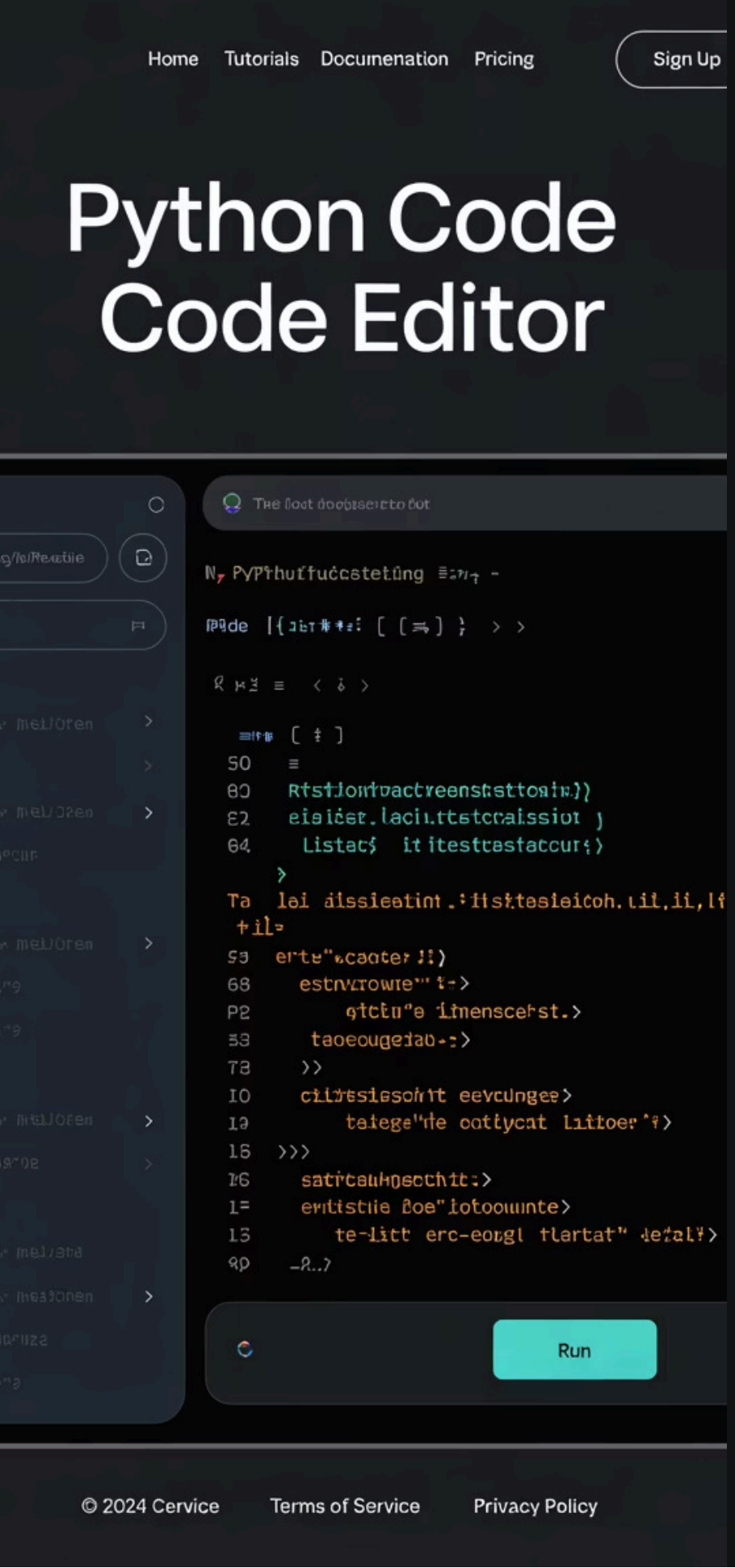
Podemos modificar, añadir o eliminar elementos después de crearlas.

Flexibles

Permiten almacenar cualquier tipo de dato, incluso mezclarlos.

Duplicados

Pueden contener elementos repetidos sin restricción.



Creación de listas en Python

```
# Crear lista de alumnos
alumnos = ["Ana", "Luis", "María"]

# Agregar un nuevo alumno
alumnos.append("Pedro")

# Ver alumno en posición 1
print(alumnos[1])  # Luis

# Ordenar alfabéticamente
alumnos.sort()
```

Cómo definir una lista:

Usa corchetes y elementos separados por comas.

Cómo manipularla

Añade, elimina o modifica elementos con métodos integrados.

Cómo acceder a sus

Utiliza índices para obtener elementos específicos.

Actividad práctica: Listas

Crear una lista

Define 5 materias en una lista.

Agregar elemento

Añade una nueva materia con el método append().

Eliminar elemento

Quita una materia con remove() o pop().

Ordenar y mostrar

Usa sort() y luego imprime la lista.

```
# 1. Crear una lista de materias
materias = ["Matemáticas", "Física", "Química",
"Historia", "Programación"]

# 2. Agregar una nueva materia
materias.append("Literatura")

# 3. Eliminar una materia
materias.remove("Física")
# Alternativa: materias.pop(1)

# 4. Ordenar y mostrar la lista
materias.sort()  # ordenar
print(materias)
# ['Historia', 'Literatura', 'Matemáticas',
#  'Programación', 'Química']
```

Tuplas: para datos fijos y seguros

Inmutables

Una vez creadas, **no** pueden modificarse.

Ordenadas

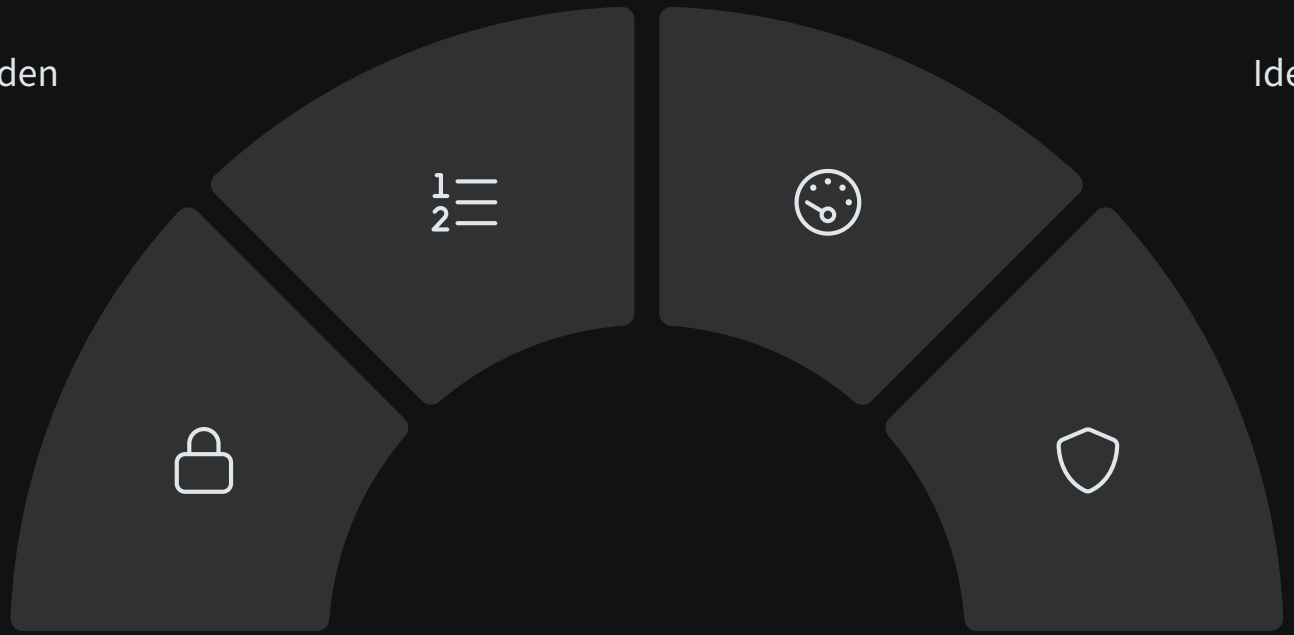
Mantienen el orden de inserción de elementos.

Eficientes

Ocupan menos memoria que las listas.

Seguras

Ideales para datos que **no** deben cambiar.



Ejemplos de tuplas en Python

Definición básica

```
# Coordenadas fijas de GPS
ubicacion = (34.56, -58.47)

# Acceder a un valor
print("Latitud:", ubicacion[0])
```

Operaciones comunes

```
# Contar ocurrencias
colores = ("rojo", "azul", "rojo")
print(colores.count("rojo"))  # 2

# Índice de un elemento
print(colores.index("azul"))  # 1
```

Actividad práctica: Tuplas

Crear tuplas de productos

Modela productos como (nombre, categoría, precio)

Filtrar por categoría

Usa un bucle para filtrar productos

Mostrar resultados

Imprime solo los productos de una categoría

```
# 1. Crear tuplas de productos
productos = [
    ("Laptop", "Electrónica", 1200),
    ("Camisa", "Ropa", 25),
    ("Smartphone", "Electrónica", 800),
    ("Zapatos", "Ropa", 60),
    ("Tablet", "Electrónica", 300)
]

# 2. Filtrar por categoría
categoria_buscada = "Electrónica"
productos_filtrados = []

for producto in productos:
    if producto[1] == categoria_buscada:
        productos_filtrados.append(producto)

# 3. Mostrar resultados
print(f"Productos de {categoria_buscada}:")
for nombre, categoria, precio in productos_filtrados:
    print(f"- {nombre}: ${precio}")
```

Sets: cuando importa la unicidad

Elementos únicos

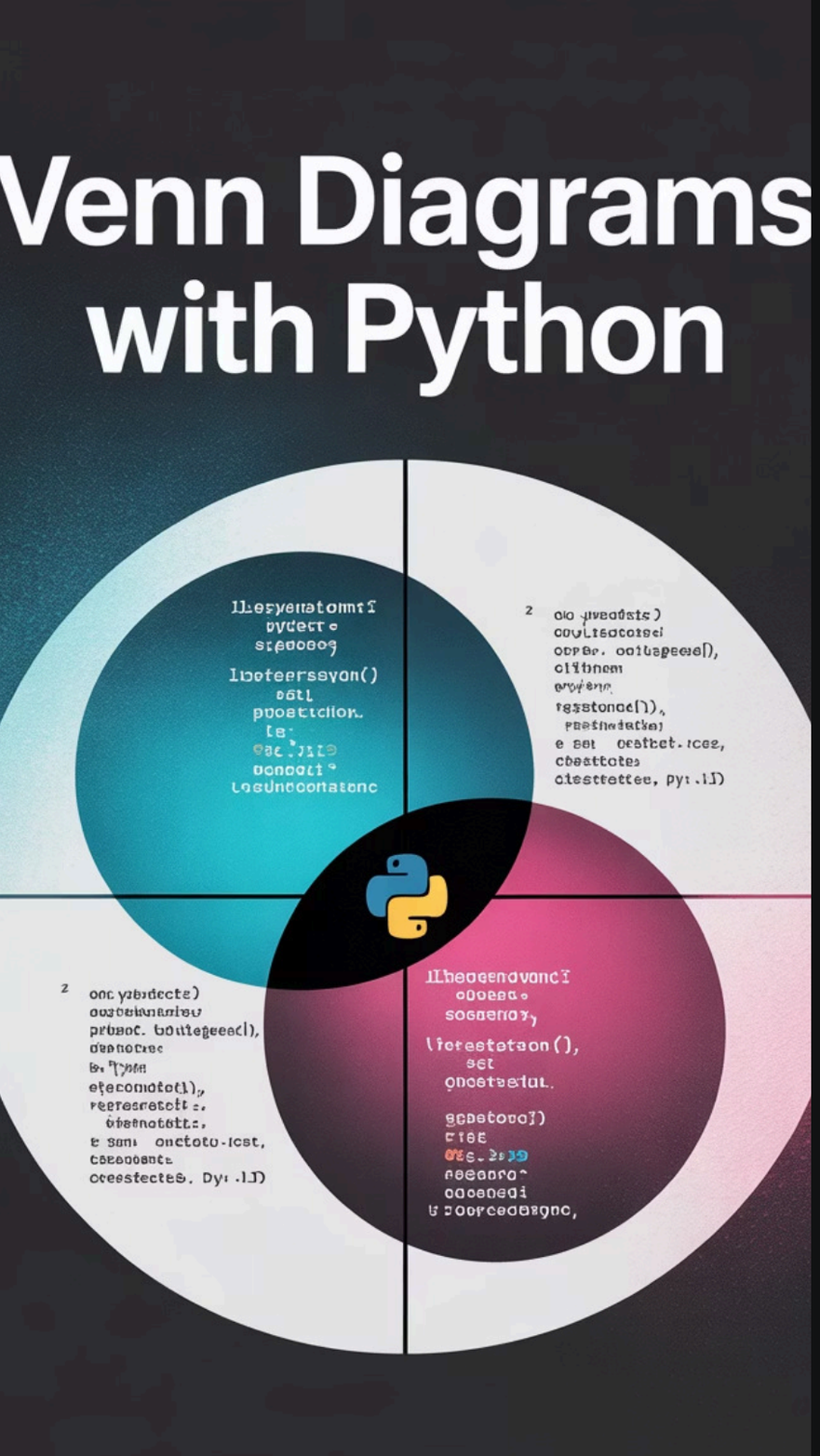
No permiten duplicados, perfectos para eliminar repetidos.

Sin orden definido

Los elementos no tienen posición fija ni acceso por índice.

Operaciones matemáticas

Ideales para uniones, intersecciones y diferencias entre conjuntos.

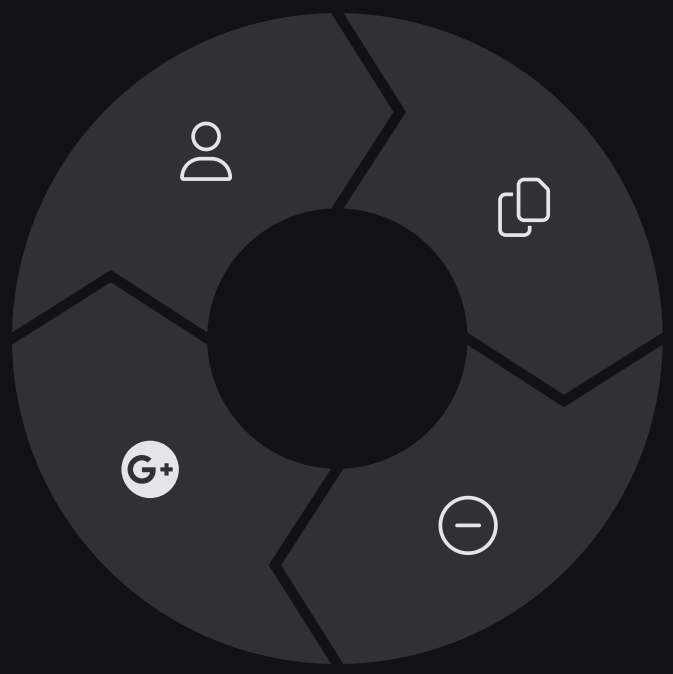


Trabajando con sets en Python

```
lenguajes_A = {"Python", "Java", "C++"}
# los sets usan llaves en lugar de corchetes

lenguajes_B = {"Python", "JavaScript"}

# Intersección (elementos comunes)
print(lenguajes_A & lenguajes_B)  # {'Python'}

# Diferencia (solo en A)
print(lenguajes_A - lenguajes_B)  # {'Java', 'C++'}

# Unión (todos los elementos)
print(lenguajes_A | lenguajes_B)  # {'Python', 'Java', 'C++',
'JavaScript'}  # sin repetidos
```

Estas operaciones hacen que los sets sean perfectos para comparar colecciones de datos sin preocuparse por duplicados.

Actividad práctica: Sets



Crear sets de materias

Define las materias de dos estudiantes

Encontrar materias comunes

Usa el operador & o intersection()

Encontrar materias exclusivas

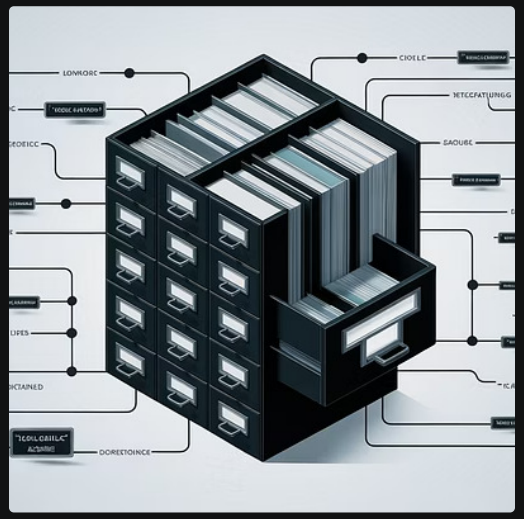
Utiliza el operador - o difference()

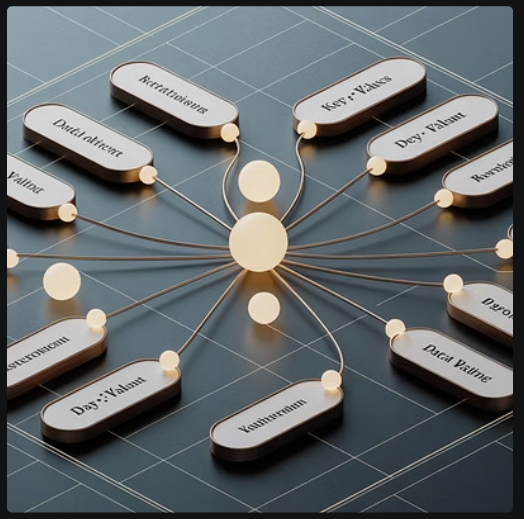
Mostrar todas las materias

Emplea el operador | o union()

```
# 1. Crear sets de materias
materias_ana = {"Matemáticas", "Física", "Programación",
"Literatura"}
materias_luis = {"Historia", "Programación", "Física", "Arte"}

# 2. Encontrar materias comunes
materias_comunes = materias_ana & materias_luis
print("Materias comunes:")
print(materias_comunes)  # {'Programación', 'Física'}

# 3. Encontrar materias exclusivas
solo_ana = materias_ana - materias_luis
print("\nMaterias exclusivas de Ana:")
print(solo_ana)  # {'Matemáticas', 'Literatura'}

solo_luis = materias_luis - materias_ana
print("\nMaterias exclusivas de Luis:")
print(solo_luis)  # {'Historia', 'Arte'}

# 4. Mostrar todas las materias
todas_materias = materias_ana | materias_luis
print("\nTodas las materias:")
print(todas_materias)  # {'Matemáticas', 'Física',
'Programación', 'Literatura', 'Historia', 'Arte'}
```

Diccionarios: estructura clave-valor





Los diccionarios son la estructura perfecta cuando necesitas asociar valores con identificadores únicos.

Características de los diccionarios

Acceso por clave

Cada valor se recupera mediante su clave única, no por posición.

2

Claves únicas

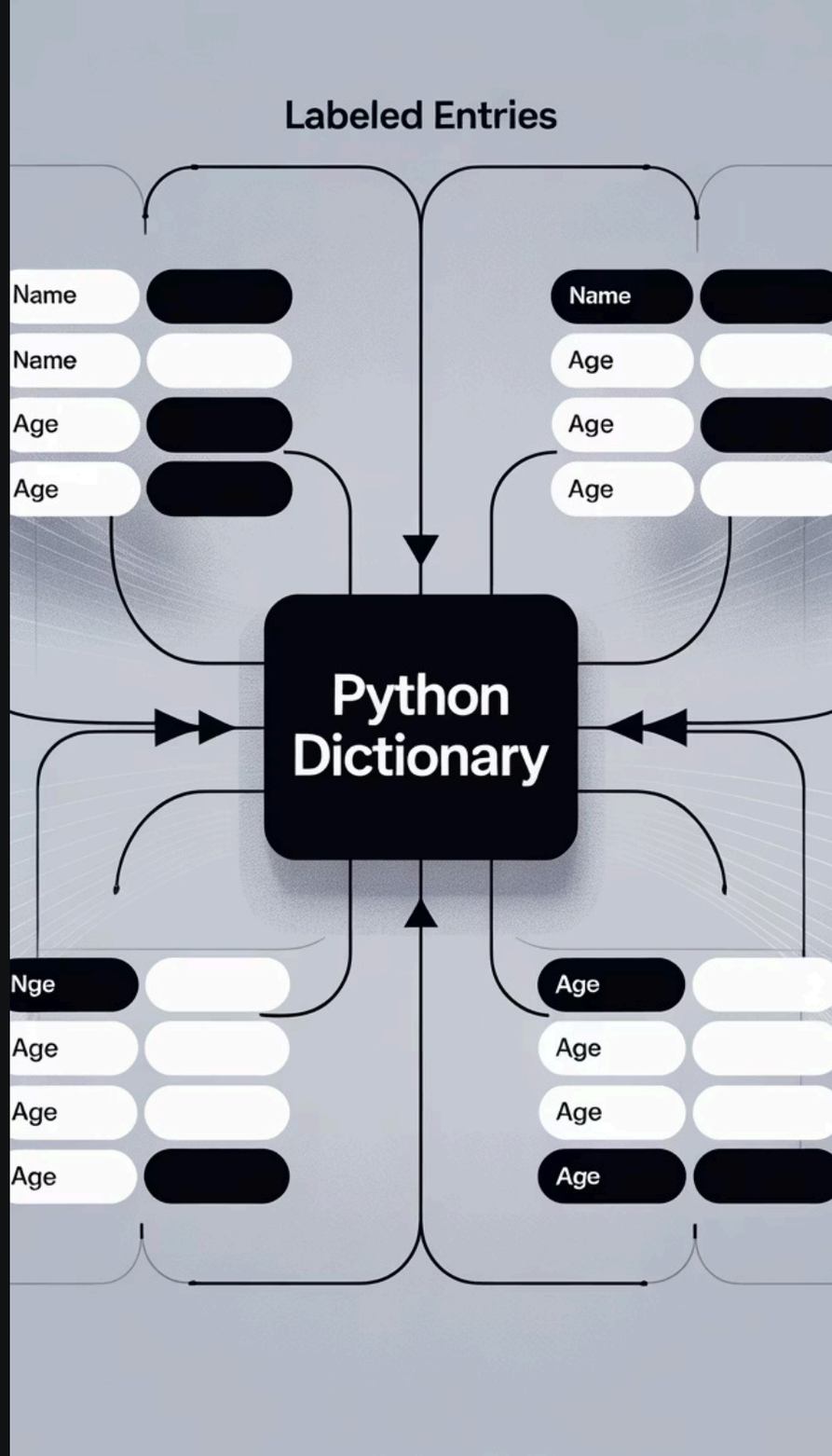
Cada clave debe ser única y de tipo inmutable (strings, números, tuplas).

Valores flexibles

Los valores pueden ser de cualquier tipo: listas, números, otros diccionarios.

Mutabilidad

Puedes añadir, modificar o eliminar pares clave-valor después de crear el diccionario.

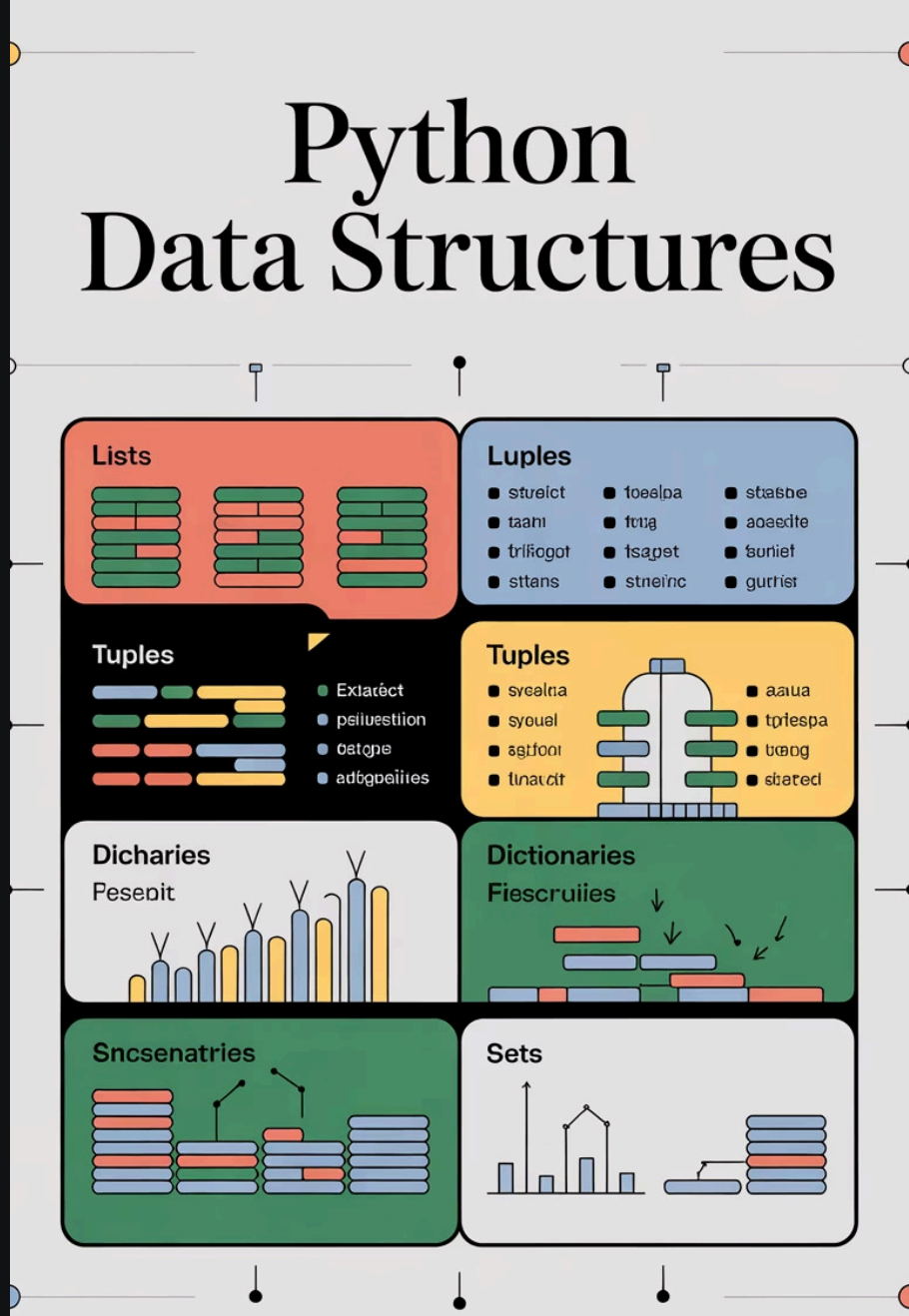


Ejemplos de diccionarios en Python

```
# Datos de un estudiante
estudiante = {
    "nombre": "Lucía",
    "edad": 20,
    "carrera": "Informática"
}

# Acceso
print(estudiante["nombre"])  # Lucía

# Agregar nuevo dato
estudiante["promedio"] = 8.75

# Recorrer con items()
for clave, valor in estudiante.items():
    print(f"{clave} -> {valor}")
```

Comparativa de estructuras de datos

Estructura	Orden	Mutable	Indexable	Único
Lista	✓	✓	✓	☒
Tupla	✓	☒	✓	☒
Set	☒	✓	☒	✓
Diccionario	☒	✓	Clave	✓ clave



¿Cuándo usar cada estructura?

Lista

Cuando necesitas orden y modificarás frecuentemente los elementos.

```
frutas = ['manzana', 'pera', 'uva']
```

Tupla

Para datos fijos que no cambiarán y deben mantener orden.

```
coordenadas = (42.3601, -71.0589)
```

3

Set

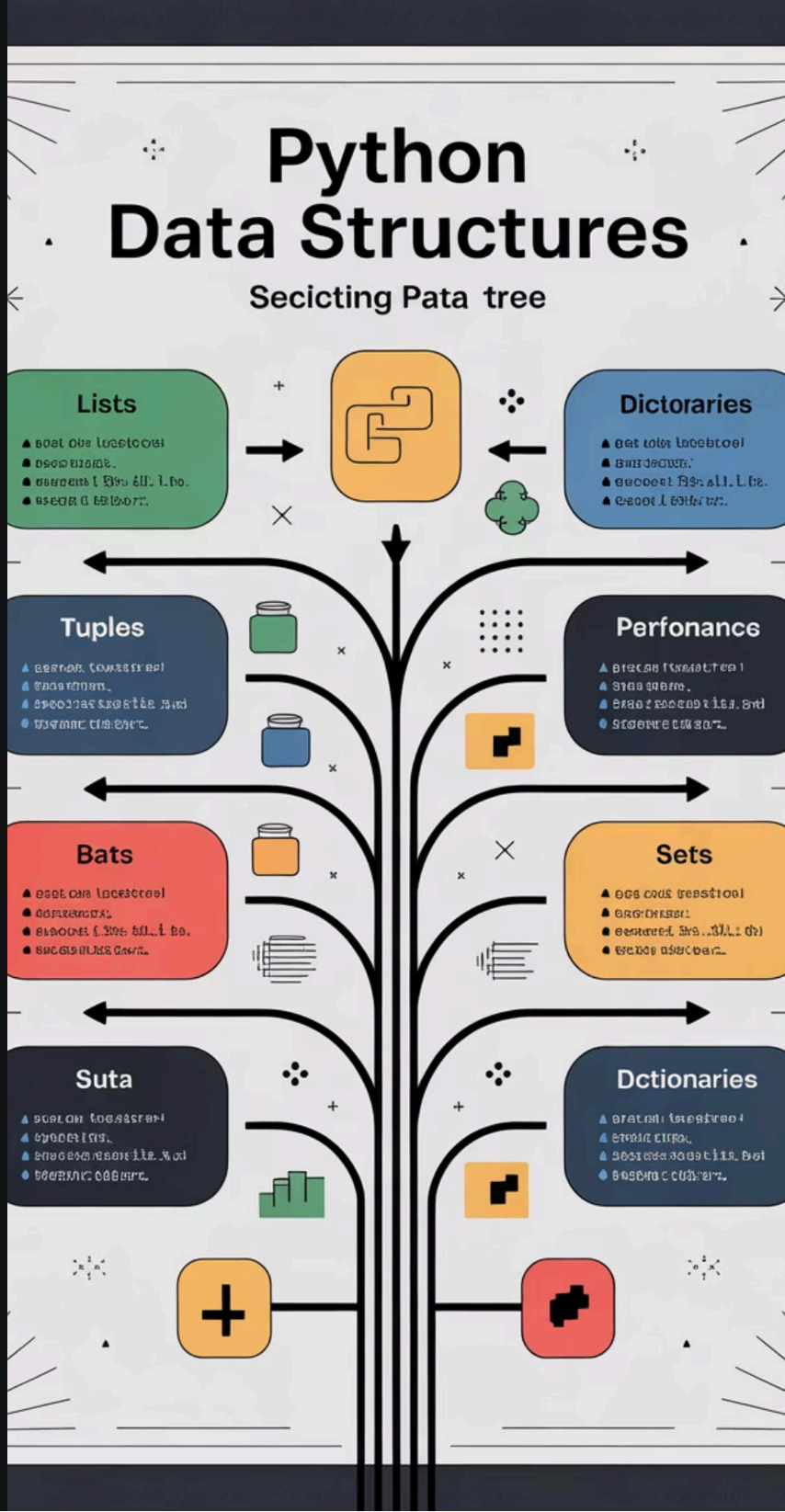
Para eliminar duplicados y realizar operaciones de conjuntos.

```
colores = {'rojo', 'verde', 'azul'}
```

Diccionario

Para recuperar valores mediante identificadores únicos.

```
usuario = {'nombre': 'Enrique', 'edad': 25, 'profesion': 'abogado'}
```



Anidación de estructuras

Listas de listas

Útiles para matrices y datos tabulares.

```
matriz = [[1, 2, 3], [4, 5, 6]]
```

Diccionarios de listas

Ideales para agrupar colecciones por categoría.

```
estudiantes = {"grupo_a": ["Ana", "Luis"]}
```

Listas de diccionarios

Perfectas para colecciones de objetos con atributos.

```
usuarios = [{"id": 1, "nombre": "Pablo"}]
```

Diccionarios anidados

Para representar jerarquías complejas.

```
escuela = {"1A": {"tutor": "Ana"}}
```

Métodos importantes: Listas

append(), insert(), extend()

Añaden elementos al final, en posición específica o desde otra lista.

remove(), pop(), clear()

Eliminan por valor, por índice o todos los elementos.

index(), count(), in

Buscan posición, cuentan ocurrencias o verifican existencia.

sort(), reverse()

Ordenan elementos o invierten su orden.

Métodos importantes: Diccionarios

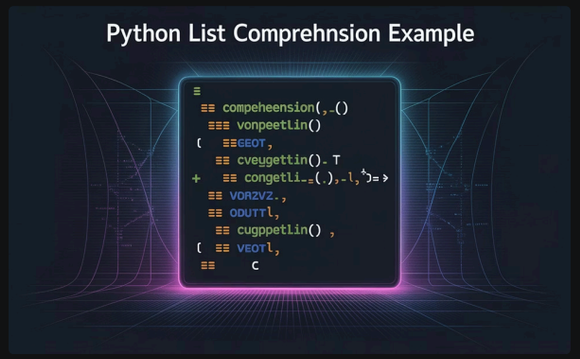
Acceso y modificación

- get(key, default): acceso seguro
- update(): combinar diccionarios
- setdefault(): valor si no existe
- pop(), popitem(): eliminar y retornar

Vistas e iteración

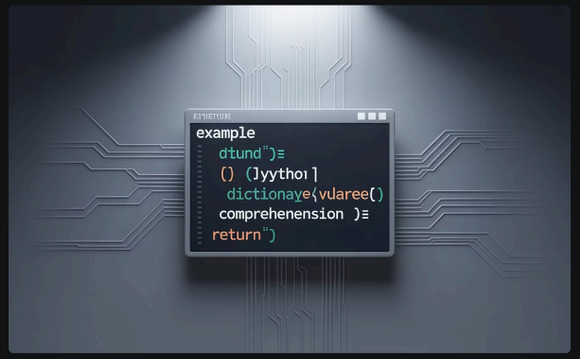
- keys(): vista de claves
- values(): vista de valores
- items(): vista de pares (clave, valor)
- for k, v in d.items(): recorrer

Comprensiones: creación eficiente



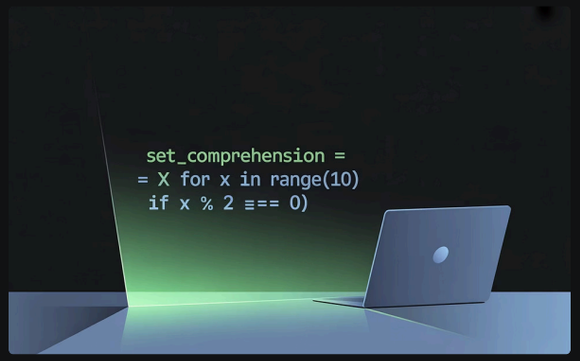
Listas

```
[x*2 for x in range(5)]
# [0, 2, 4, 6, 8]
```



Diccionarios

```
{x: x**2 for x in range(5)}
# {0:0, 1:1, 2:4, 3:9, 4:16}
```

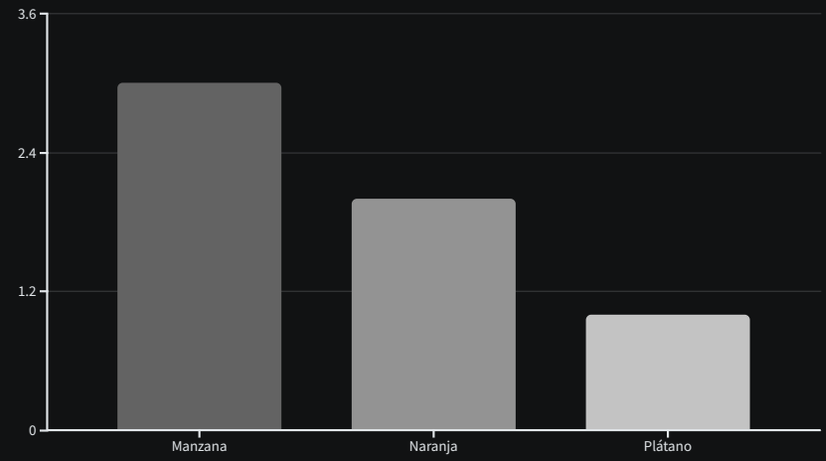


Sets

```
{x%3 for x in range(10)}
# {0, 1, 2}  sin repetidos
```

Uso de Counter para análisis

```
from collections import Counter

# Contar elementos
frutas = ["manzana", "naranja",
          "manzana", "plátano",
          "manzana", "naranja"]

conteo = Counter(frutas)
print(conteo)
# Counter({'manzana': 3,
#          'naranja': 2,
#          'plátano': 1})

# Elementos más comunes
print(conteo.most_common(2))
# [('manzana', 3), ('naranja', 2)]
```



Aplicaciones prácticas: Análisis de texto

Código de ejemplo

```
texto = """Python es un lenguaje
de programación muy popular.
Python es fácil de aprender.
Python tiene muchas bibliotecas."""

# Contar palabras
palabras = texto.lower().split()

1111from collections import Counter

frecuencia = Counter(palabras)

# Palabras más comunes
print(frecuencia.most_common(3))
```

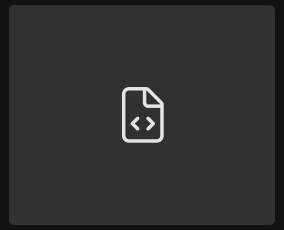
Resultado

Las palabras más frecuentes son:

1. "python" (3 veces)
2. "es" (2 veces)
3. "de" (2 veces)

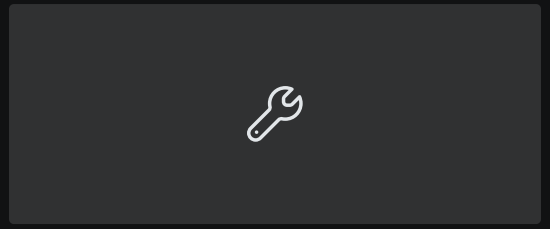
Podemos usar esta información para análisis de sentimiento, clasificación de textos o sistemas de recomendación.

Buenas prácticas



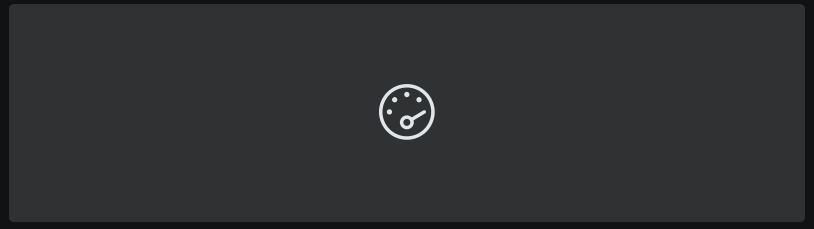
Elegir la estructura adecuada

Selecciona según el problema, no por costumbre.



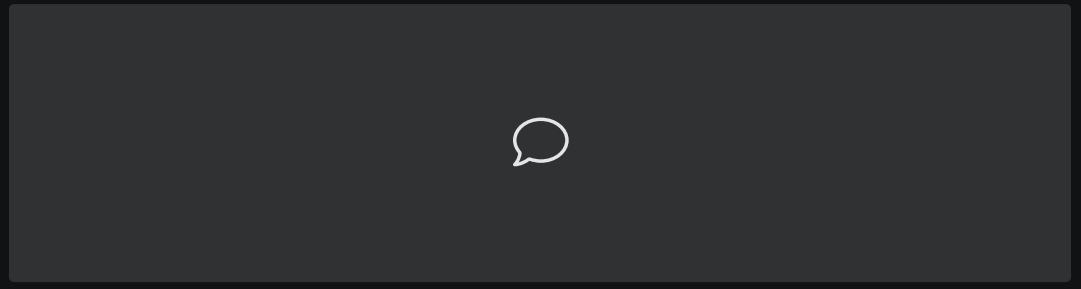
Métodos específicos

Aprovecha los métodos integrados en lugar de reinventarlos.



Considerar el rendimiento

Para grandes volúmenes de datos, elige estructuras eficientes.



Documentar decisiones

Explicar por qué se usa cada estructura para facilitar mantenimiento.

Errores comunes a evitar

Modificar durante iteración

Nunca modifiques un diccionario o lista mientras la recorres.

Confundir inmutabilidad

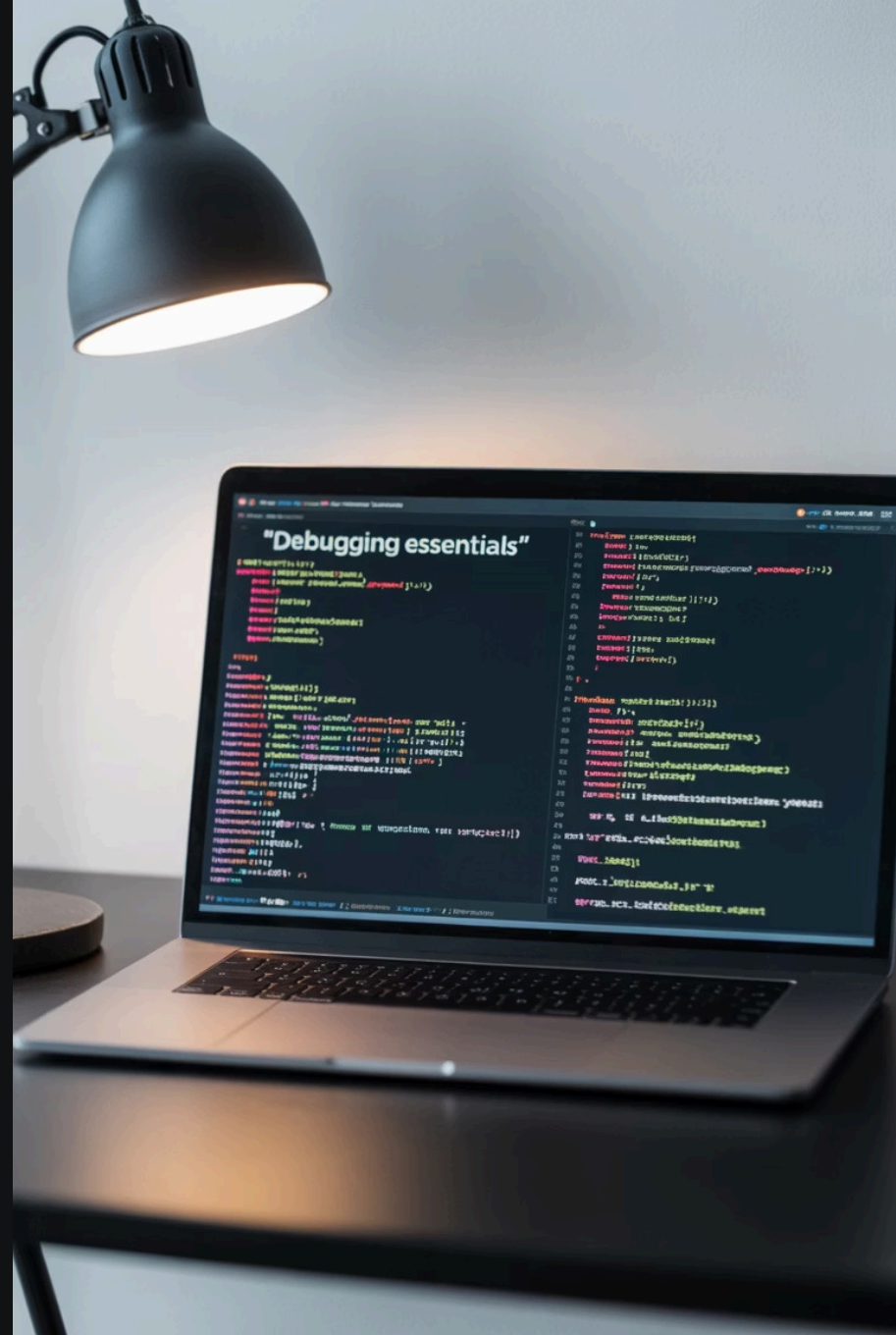
Recordar que las tuplas no pueden modificarse después de crearse.

Olvidar claves de diccionario

Usar dict.get() o try/except para acceder a claves que podrían no existir.

Ignorar rendimiento

En conjuntos grandes, buscar en listas es muy ineficiente.



¡GRACIAS!

Esperamos que hayas disfrutado este recorrido por las estructuras de datos en Python.

¿Preguntas? Estamos aquí para ayudarte en tu camino de aprendizaje.