Programación I: Variables Dimensionadas

En este tema exploraremos las estructuras de datos fundamentales en Python que nos permiten almacenar y organizar colecciones de información. Aprenderemos a trabajar con listas, tuplas, conjuntos y diccionarios, comprendiendo sus características únicas y casos de uso específicos.

Contenedores: Organizando la Información

¿Qué son los contenedores?

Los contenedores son tipos de datos especiales que nos permiten agrupar múltiples elementos en una única variable. En Python, disponemos de varios tipos de contenedores, cada uno con características particulares que los hacen ideales para diferentes situaciones.

Los principales contenedores que estudiaremos son: **strings** (cadenas de texto), **listas**, **tuplas**, **conjuntos** y **diccionarios**. Cada uno tiene propiedades únicas en cuanto a orden, mutabilidad y duplicación de elementos.

$$objetos = [\ 7, 'Hola', True, 3.5]$$

Listas: Flexibilidad y Orden

Ordenadas por índice

Cada elemento ocupa una posición específica, comenzando desde 0

Elementos repetidos

Pueden contener valores duplicados sin restricciones

Tipos variados

Admiten elementos de cualquier tipo de dato en la misma lista

Mutables

Pueden modificarse después de su creación: agregar, eliminar o cambiar elementos

Las listas son uno de los tipos de datos más versátiles en Python. Admiten «rebanadas» (slices), lo que significa que podemos extraer porciones de la lista usando índices. Esta característica las convierte en herramientas fundamentales para el manejo de colecciones de datos.

```python
lista_vacia = []
lista_vacia2 = list()
lista_con_elementos = [52, 'cadena', 3.8, False, [1,2,3]]
lista_desde_string = list('probando')
lista_desde_rango = list(range(3))
```

Creación de Listas en Python

Existen varias formas de crear listas en Python. La más común es utilizando corchetes `[]` y separando los elementos con comas. También podemos crear listas vacías para posteriormente agregar elementos, o convertir otros tipos de contenedores en listas usando la función `list()`.

> **Consejo práctico:** Al nombrar tus listas, utiliza nombres descriptivos en plural que indiquen claramente qué tipo de elementos contienen. Por ejemplo: `numeros`, `nombres`, `temperaturas`.

Iterando una Lista: Tres Enfoques

Iteración por índice (for)

Utilizamos `range(len(lista))` para recorrer cada posición y acceder al elemento mediante su índice. Útil cuando necesitamos conocer la posición del elemento.

Iteración por índice (while)

Usamos una variable contador que incrementamos manualmente. Ofrece mayor control sobre el flujo de iteración, permitiendo saltos o retrocesos.

Iteración por elementos

La forma más sencilla y pythónica. Recorremos directamente cada elemento sin necesidad de índices: `for elemento in lista:`

```python
for i in range(len(lista)):
    print(lista[i])
```

```python
i=0
while i<len(lista):
    print(lista[i])
    i+=1
```

```python
for elemento in lista:
    print(elemento)
```

Operaciones Fundamentales con Listas

1

Creación y conversión

`list(contenedor)` - Convierte cualquier contenedor en lista

2

Concatenación

`lista1 + lista2` - Une dos listas creando una nueva

3

Longitud

`len(lista)` - Devuelve el número de elementos

4

Acceso y rebanadas

`lista[0]`, `lista[0:3]`, `lista[0:7:2]` - Accede a elementos o porciones

5

Modificación

`lista[posición] = nuevo_valor` - Cambia el valor en una posición específica

6

Búsqueda

`lista.index(elemento)` - Encuentra el índice de un elemento

Más Operaciones con Listas

Consulta y conteo

- `elemento in lista` - Verifica si un elemento existe (devuelve booleano)
- `lista.count(elemento)` - Cuenta cuántas veces aparece un elemento

Agregar elementos

- `lista.append(elemento)` - Añade al final de la lista
- `lista.insert(posición, elemento)` - Inserta en posición específica

Eliminar elementos

- `del lista[posición]` - Elimina por índice
- `lista.remove(elemento)` - Elimina por valor (primera ocurrencia)
- `lista.clear()` - Vacía completamente la lista

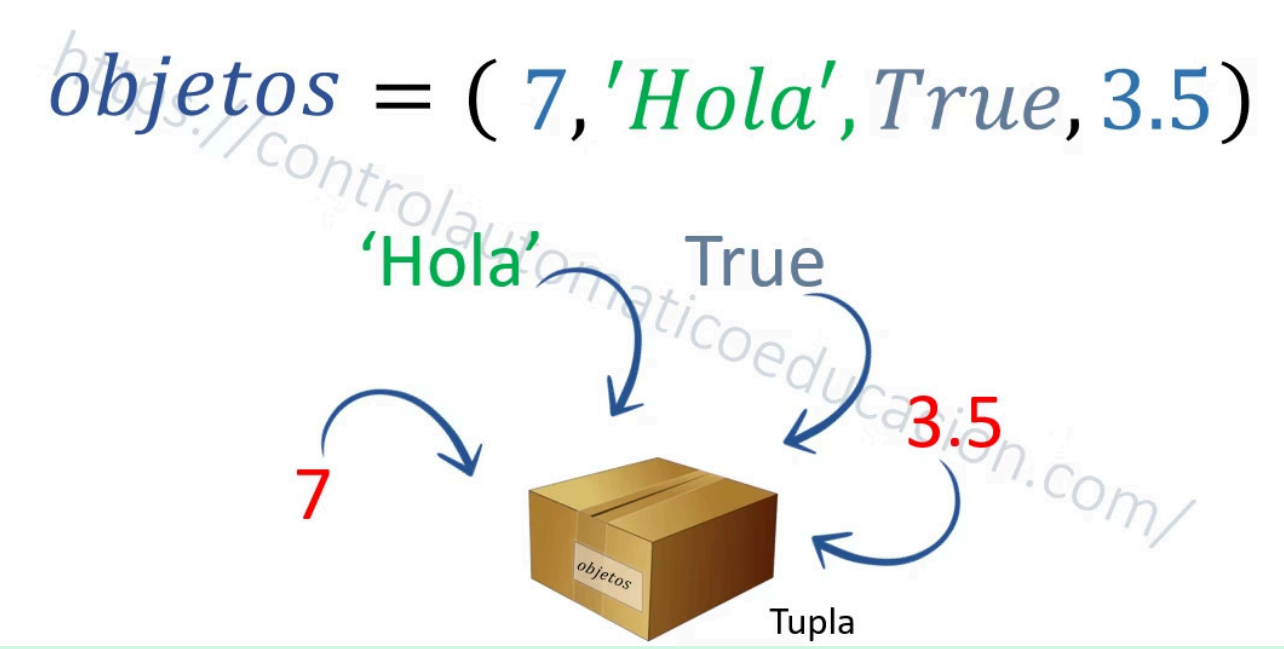
Estas operaciones son la base para manipular listas de forma efectiva. Dominarlas te permitirá resolver una gran variedad de problemas de programación.

Tuplas: Inmutabilidad y Eficiencia

Las tuplas son muy similares a las listas en cuanto a estructura: contienen elementos ordenados, admiten repeticiones, pueden incluir tipos variados y soportan rebanadas. Se iteran de la misma manera y comparten muchas operaciones.

La diferencia clave: Inmutabilidad

Una vez creada una tupla, **no puede modificarse**. No podemos agregar, eliminar ni cambiar sus elementos. Esta característica las hace más eficientes en memoria y más seguras cuando queremos garantizar que los datos no cambien accidentalmente.



¿Cuándo usar tuplas? Son ideales para representar datos que no deben cambiar, como coordenadas geográficas, configuraciones o registros de bases de datos.

Creación de Tuplas en Python

Las tuplas se definen usando paréntesis `()` en lugar de corchetes. Sin embargo, los paréntesis son opcionales: lo que realmente crea una tupla es la **coma** que separa los elementos.

Tupla con paréntesis

`tupla = (1, 2, 3, 4)`

La forma más clara y recomendada

Tupla sin paréntesis

`tupla = 1, 2, 3, 4`

Funcionalmente idéntica, las comas definen la tupla

Tupla de un elemento

`tupla = (5,)` o `tupla = 5,`

La coma final es *obligatoria* para diferenciarla de una expresión entre paréntesis

```python
tupla_vacia = ()
tupla_vacia2 = tuple()
tupla_con_elementos = (52, 'cadena', 3.8, False, [1,2,3])
tupla_con_elementos2 = 52, 'cadena', 3.8, False, [1,2,3]
tupla_desde_string = tuple('probando')
tupla_desde_rango = tuple(range(3))
```

Conjuntos: Elementos Únicos sin Orden

Los conjuntos son colecciones **sin orden** definido donde cada elemento puede aparecer **una sola vez**. Son perfectos para eliminar duplicados o realizar operaciones matemáticas de conjuntos como uniones e intersecciones.

- **Sin orden**

  Los elementos no tienen índice ni posición fija

- **Sin duplicados**

  Cada elemento aparece una sola vez

- **Solo inmutables**

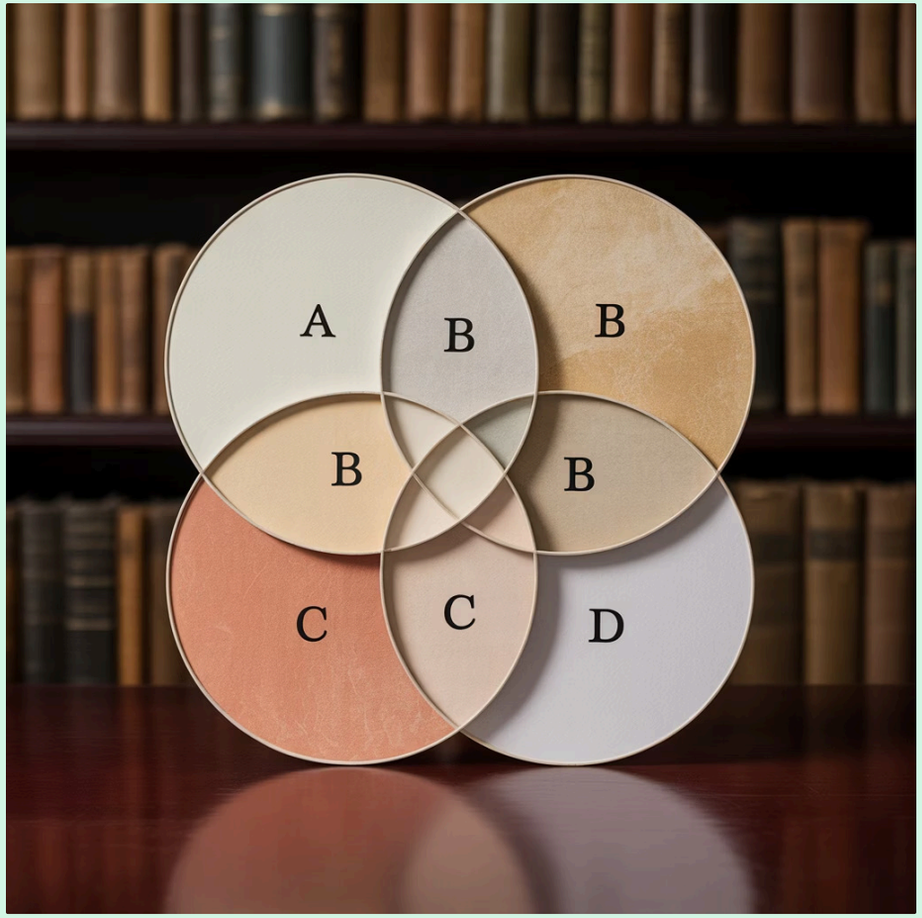
  Solo pueden contener tipos inmutables (números, strings, tuplas)

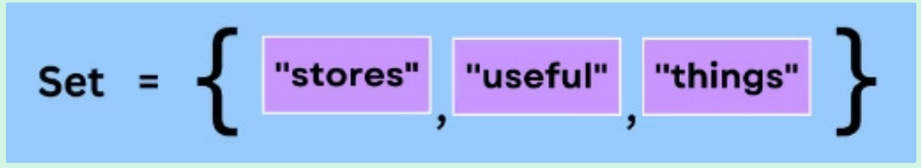
- **No admiten rebanadas**

  Al no tener orden, no podemos usar índices

- **Modificables**

  Podemos agregar y eliminar elementos del conjunto





Creación e Iteración de Conjuntos

Los conjuntos se definen usando llaves `{}`. Para crear un conjunto vacío, debemos usar `set()`, ya que `{}` crea un diccionario vacío.

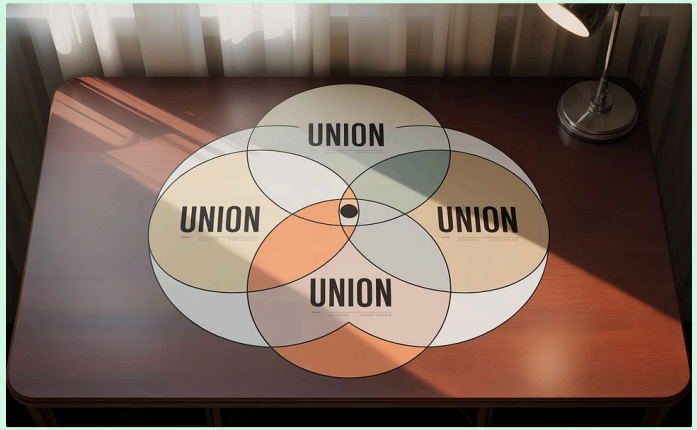
```
conjunto_vacio = set()
conjunto_con_elementos = {52, 'cadena', 3.8, False, [1,2,3]}
conjunto_desde_string = set('probando')
conjunto_desde_rango = set(range(3))
conjunto_desde_lista = set([1,2,3,4])
conjunto_desde_tupla = set(('a','b','c'))
```

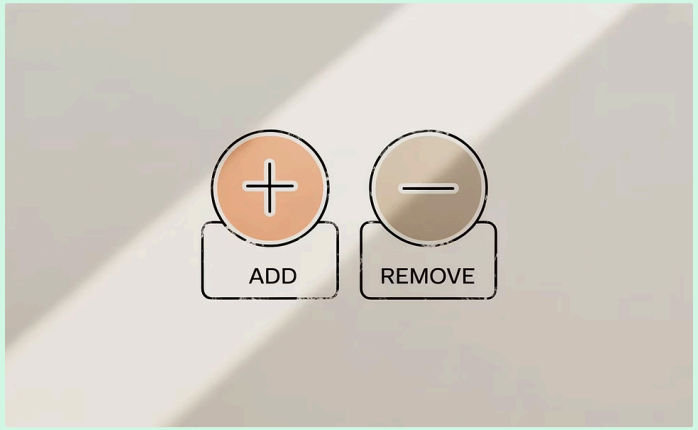
Iteración por elementos

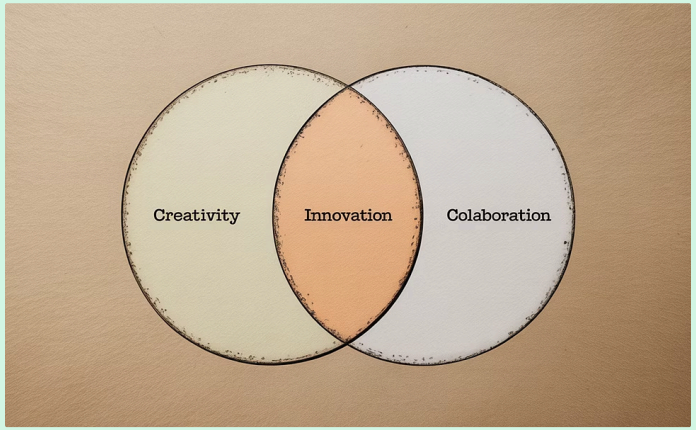
Como los conjuntos no tienen índice, solo podemos iterar directamente por sus elementos. El orden en que aparecen es arbitrario y puede variar:

```
for elemento in conjunto:
    print(elemento)
```

Operaciones con Conjuntos







Operaciones básicas

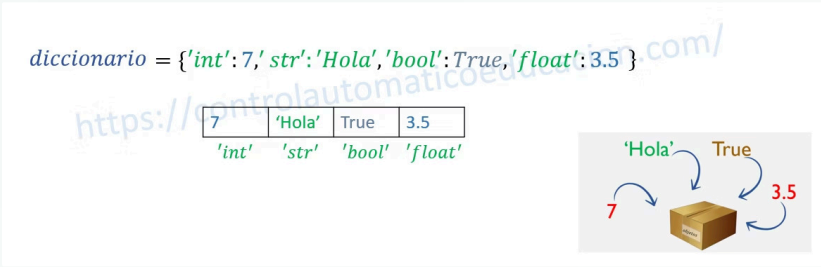
- `set(contenedor)` - Crea conjunto desde otro contenedor
- `len(conjunto)` - Número de elementos
- `elemento in conjunto` - Verifica pertenencia

Modificación

- `conjunto.add(elemento)` - Agrega un elemento
- `conjunto.update(contenedor)` - Agrega múltiples elementos
- `conjunto.remove(elemento)` - Elimina (error si no existe)
- `conjunto.discard(elemento)` - Elimina (sin error)

Operaciones matemáticas

- `conjunto1 | conjunto2` - Unión
- `conjunto1 & conjunto2` - Intersección
- `conjunto1 - conjunto2` - Diferencia
- `conjunto1 ^ conjunto2` - Diferencia simétrica



Diccionarios: Pares Clave-Valor

Los diccionarios almacenan información en **pares clave-valor**, donde cada clave única se asocia con un valor. Son ideales para representar relaciones, configuraciones o cualquier dato que requiera acceso rápido mediante identificadores.

Características principales

- Elementos sin orden definido
- Las **claves** no pueden repetirse
- Los **valores** sí pueden repetirse
- Claves de cualquier tipo inmutable
- No admiten rebanadas
- Son mutables: pueden agregarse y eliminarse pares

Acceso mediante claves

Las claves actúan como un «índice personalizado», permitiéndonos acceder directamente a los valores usando corchetes: `diccionario[clave]`

Trabajando con Diccionarios

Creación de diccionarios

Los diccionarios se definen con llaves `{}`, separando cada par clave-valor con dos puntos `:` y los pares entre sí con comas:

```
diccionario_vacio = {}
diccionario_vacio2 = dict()
diccionario_con_elementos = {'hola':'hello', 'adios':'bye'}
diccionario_desde_contenedor = dict([('hola','hello'), ('adios','bye')])
```

```
#Iteracion por claves
for clave in diccionario.keys():
    print(clave)
for clave in diccionario:
    print(clave)

#Iteracion por valores
for valor in diccionario.values():
    print(valor)
for clave in diccionario:
    print(diccionario[clave])

#Iteracion por clave y valor a la vez
for clave, valor in diccionario.items():
    print(clave, valor)
for par in diccionario.items():
    print(par[0], par[1])
```

Iteración por pares

Para recorrer un diccionario accediendo tanto a claves como valores, usamos el método `.items()`:

🗝

Iterar solo claves: `for clave in diccionario:` o `diccionario.keys()`

Iterar solo valores: `for valor in diccionario.values():`

Iterar pares: `for clave, valor in diccionario.items():`

Operaciones con Diccionarios y Errores Comunes

Operaciones principales

- `len(diccionario)` - Número de pares
- `clave in diccionario` - Verifica si existe la clave
- `valor in diccionario.values()` - Busca un valor
- `diccionario[clave]` - Obtiene valor (error si no existe)
- `diccionario.get(clave, default)` - Obtiene valor (devuelve default si no existe)
- `diccionario[clave] = valor` - Modifica o agrega par
- `diccionario.update({c1:v1, c2:v2})` - Agrega múltiples pares
- `del diccionario[clave]` - Elimina par
- `diccionario.clear()` - Vacía el diccionario

Errores comunes a evitar

1. Concatenar una lista con un dato de otro tipo
2. Imprimir estructuras sin iterar (mala práctica)
3. No considerar casos extremos (listas vacías, primer/último elemento)
4. Buscar elementos en la estructura incorrecta (olvidar estructuras anidadas)
5. Iterar por un diccionario buscando una clave (usar `in` directamente)
6. Cargar datos diferentes en la misma lista
7. Modificar el tamaño de la estructura mientras se itera

Consejo final: Practica con cada tipo de contenedor. La experiencia te ayudará a elegir la estructura más adecuada para cada problema.