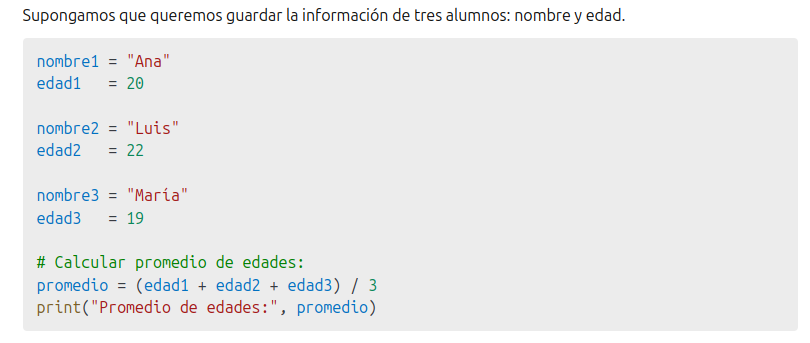
# **Estructuras de datos (Listas, Tuplas y Diccionarios)**

En Python, una **variable** es un nombre que referencia un **dato** (por ejemplo, un número, texto u objeto). Cuando empiezas a tener muchos de esos datos, manejarlos uno a uno con variables independientes se vuelve complejo. Ahí aparecen las **estructuras de datos**, que son “contenedores” que agrupan valores para que puedas:

* **Almacenar** muchos datos bajo un solo nombre.
* **Recorrer** o buscar información fácilmente.
* **Escalar** tu código sin multiplicar variables sueltas.

## 1. Ejemplo sin estructuras de datos

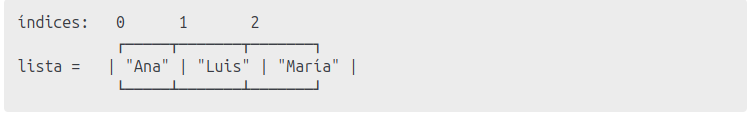
Supongamos que queremos guardar la información de tres alumnos: nombre y edad.



## 2. Con estructuras de datos

### 2.1. Lista (list)

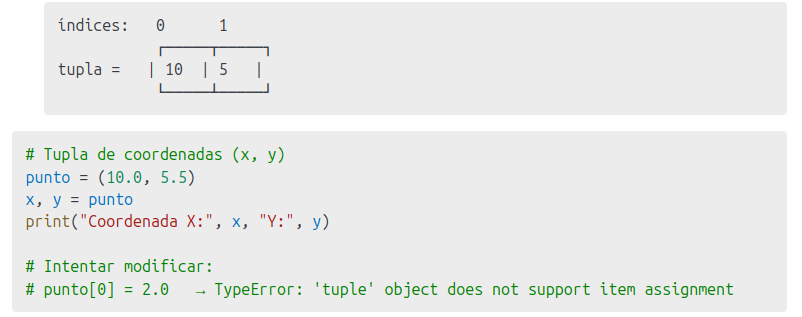
* **Concepto**: Una lista es una colección ordenada de elementos en la que cada posición tiene un índice numérico. Las listas son dinámicas, lo que significa que puedes cambiar su tamaño en tiempo de ejecución añadiendo o eliminando elementos.
* **Características**:
  + **Ordenada**: Mantiene el orden de inserción.
  + **Mutable**: Puedes modificar, insertar o eliminar elementos.
  + **Heterogénea**: Puede contener diferentes tipos de datos en la misma lista.
* **Uso típico**:
  + Almacenar secuencias de elementos cuando necesites iterar, filtrar o reordenar dinámicamente.
* **Estructura (visual)**:





### 2.2. Tupla (tuple)

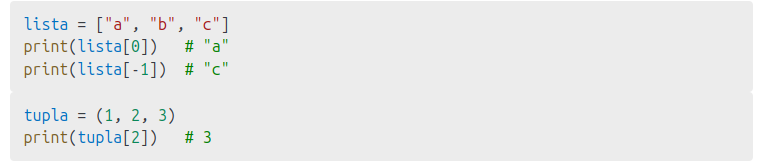
* **Concepto**: Una tupla es una secuencia ordenada de elementos similar a una lista, pero inmutable. Una vez creada, no puedes cambiar su longitud ni sus valores.
* **Características**:
  + **Ordenada**: Conserva el orden de definición.
  + **Inmutable**: La estructura y los valores no pueden modificarse.
  + **Eficiente**: Las tuplas suelen ser más rápidas y ligeras en memoria.
* **Uso típico**:
  + Representar datos constantes que no deben cambiar, como coordenadas, configuraciones o claves de diccionario.
* **Estructura (visual)**:



### 2.3. Índices y métodos

#### Índices

* **Qué son**: Los índices son números enteros que indican la posición de un elemento dentro de una secuencia ordenada (listas y tuplas).
* **Rango**: Van desde 0 hasta len(secuencia)-1 para índices positivos, y de -1 (último elemento) hasta -len(secuencia) (primer elemento) para índices negativos.
* **Uso**:



#### Métodos

Los **métodos** son funciones integradas disponibles en las estructuras de datos que permiten realizar operaciones comunes:

* **Listas**:
  + append(valor): Añade al final.
  + insert(indice, valor): Inserta en posición.
  + remove(valor): Elimina la primera aparición.
  + pop(indice): Elimina y devuelve el elemento en índice.
  + sort(): Ordena la lista in situ.
  + reverse(): Invierte el orden in situ.
* **Tuplas**:
  + count(valor): Cuenta cuántas veces aparece valor.
  + index(valor): Devuelve el índice de la primera aparición de valor.

### 2.4. Combinando listas y tuplas

A continuación, dos ejemplos sencillos:

**Ejemplo 1: Lista de tuplas para compras**

Tienes una lista de frutas con sus cantidades y quieres acceder fácilmente a cada par.



* La **lista** permite añadir o quitar tuplas de compras.
* La **tupla** garantiza que cada registro (fruta, cantidad) no cambie accidentalmente.

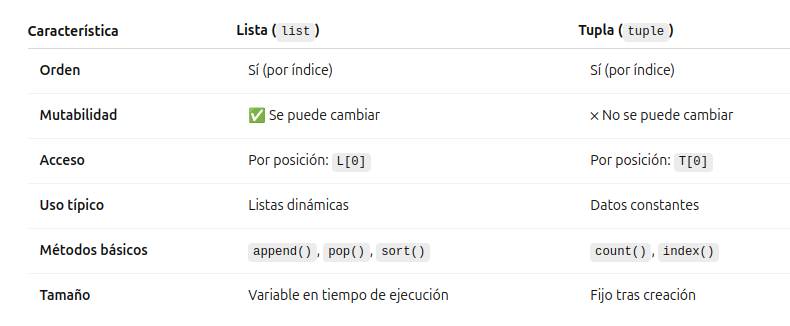
**Ejemplo 2: Tupla de listas para grupos de opciones**

Quieres agrupar botones de acción en dos categorías, pero cada categoría debe poder expandirse.



* Las **listas** dentro pueden modificarse añadiendo nuevas opciones.

## 3. Diferencias clave



A continuación, algunos ejercicios que combinan listas y tuplas junto con sus soluciones en Python:



## 1. ¿Qué es un diccionario?

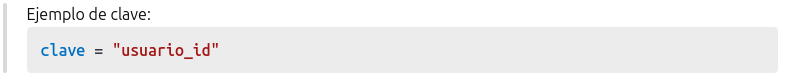
Un **diccionario** es una colección de datos que asocia cada **clave** única con un **valor** relacionado, permitiendo un acceso rápido y directo a la información. Imagina un diccionario de la vida real: al buscar una palabra (la clave), obtienes su definición (el valor). Esta estructura es ideal para representar relaciones uno a uno donde necesitas consultar o actualizar valores de forma eficiente.

* Las **claves** deben ser de un tipo inmutable (por ejemplo, cadenas, números o tuplas).
* Los **valores** pueden ser de cualquier tipo, incluso otros diccionarios.
* No mantienen orden (hasta Python 3.6 eran completamente desordenados; a partir de 3.7 conservan el orden de inserción, aunque esto no debe emplearse como característica principal).

### 1.1 Clave (Key)

Una **clave** es el identificador único que se utiliza para asociar un valor dentro de un diccionario. Sus características son:

* **Unicidad:** Cada clave debe ser única dentro del mismo diccionario.
* **Inmutabilidad:** Puede ser de tipo cadena (str), número (int, float), tupla (tuple) o cualquier objeto inmutable.
* **Acceso:** Se emplea para obtener o modificar su valor asociado.



### 1.2 Valor (Value)

Un **valor** es el elemento asociado a una clave dentro del diccionario. Sus características son:

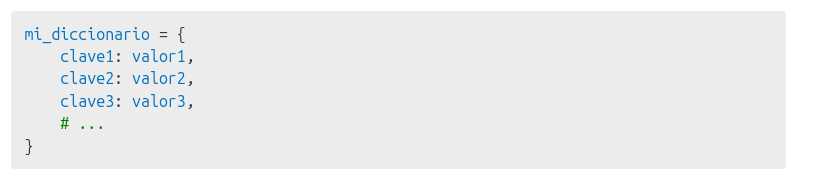
* **Flexibilidad:** Puede ser de cualquier tipo: numérico, cadena, lista, otro diccionario, objeto, etc.
* **Puede repetirse:** A diferencia de las claves, varios elementos pueden compartir el mismo valor.
* **Asociación:** Se accede o modifica a través de su clave correspondiente.



## 2. Creación de diccionarios

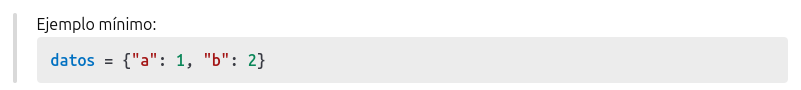
### 2.1 Formato general de un diccionario

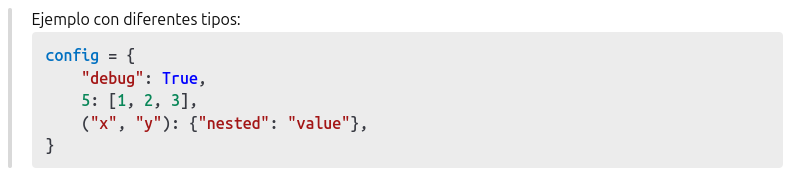
La sintaxis básica para definir un diccionario en Python es la siguiente:



**Componentes del formato:**

* **Llaves** {}: Delimitan el inicio y fin de la definición.
* **Pares** clave: valor: Cada par separa la **clave** (izquierda del :) y el **valor** (derecha del :).
* **Comas** ,: Separan cada par, excepto el último (opcionalmente puede llevar una coma final).





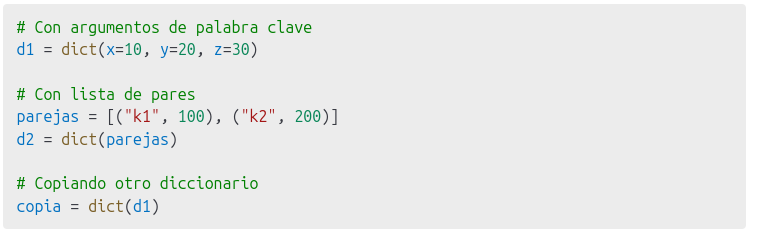
1. **Literal con llaves**



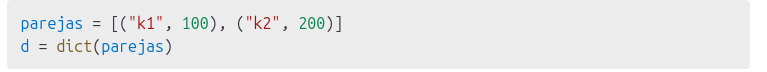
1. \*\*Constructor \*\*dict

Un **constructor** es una función o clase especial que crea e inicializa una instancia de un tipo de dato. En este caso, dict() es el constructor de la clase dict, el cual puede recibir:

* + **Argumentos de palabra clave:** Cada clave se toma del nombre del argumento y su valor del argumento.
  + **Un iterable de pares (clave, valor):** Puede ser una lista, tupla u otro iterable de pares.
  + **Otro diccionario o mapeo:** Copia las claves y valores.



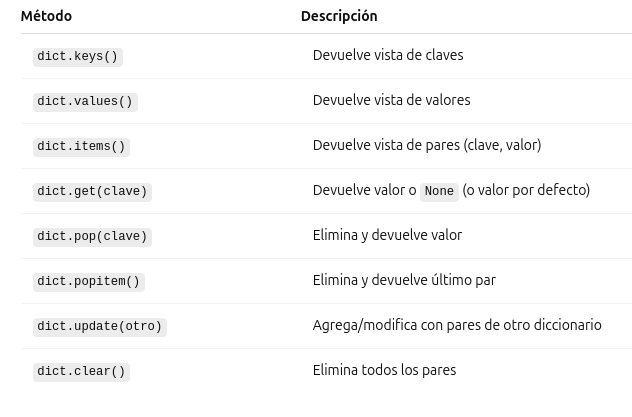
1. **A partir de lista de pares** **A partir de lista de pares**



## 3. Acceso y modificación



## 4. Métodos más comunes

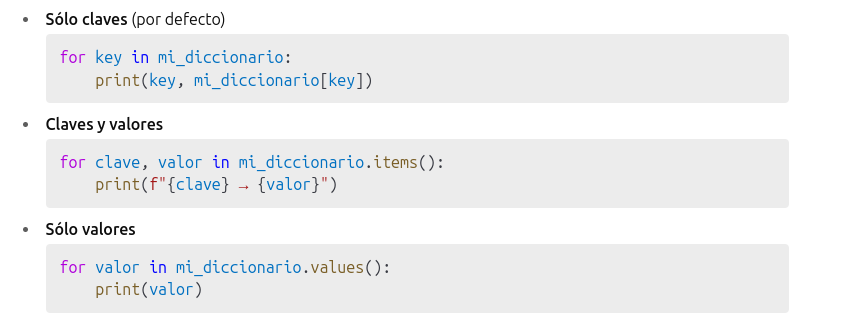


## 5. Iteración

La **iteración** es el proceso de ejecutar repetidamente un conjunto de instrucciones para recorrer o procesar elementos de una colección, como listas, tuplas o diccionarios. Permite realizar operaciones sobre cada elemento sin duplicar código.

**Características de la iteración:**

* Repite un bloque de código hasta que se agote la secuencia o se cumpla una condición.
* Facilita la manipulación y análisis de colecciones de datos.
* En Python, se implementa principalmente con el bucle for.



## 6. Ejemplo completo

**Enunciado del problema:** Diseña un programa que gestione el inventario de frutas de una tienda. Debe permitir:

1. Crear un diccionario con las cantidades iniciales de cada fruta.
2. Acceder a la cantidad disponible de una fruta.
3. Modificar la cantidad de frutas existentes.
4. Recorrer todos los elementos para mostrar el inventario.
5. Eliminar frutas del inventario cuando se agoten.



## ¿Cuándo usar lista, tupla o diccionario?

Al diseñar tus estructuras de datos, elige la más adecuada según estos criterios:

* **Lista (**list**)**
  + Ordenada y modificable (se pueden añadir, eliminar o cambiar elementos).
  + Ideal para colecciones de elementos homogéneos donde el orden importa y se requiere manipulación frecuente.
  + Sintaxis: [1, 2, 3]
* **Tupla (**tuple**)**
  + Ordenada e inmutable (no se pueden modificar tras su creación).
  + Útil para datos heterogéneos o de longitud fija que representan registros o coordenadas.
  + Garantiza que los datos no cambien accidentalmente.
  + Sintaxis: ("x", "y")
* **Diccionario (**dict**)**
  + Mapea **claves** únicas a **valores**, sin depender del orden.
  + Perfecto para relaciones uno a uno, búsquedas rápidas y acceso por identificador.
  + Sintaxis: { "clave": valor }