📝 Apuntes de Programación: Los Diccionarios (Dictionaries) en Python

Los Diccionarios (`dict`) son colecciones de datos **desordenadas** que almacenan información en pares de **clave-valor (key-value)**. Son la estructura ideal para modelar objetos del mundo real y para búsquedas de información extremadamente rápidas.

1. Creación y Características Esenciales

Los diccionarios se crean asociando una clave única con un valor.

A. Formas de Declarar un Diccionario

Código de Ejemplo	Descripción y Uso
`dict1 = {"Nombre" : "Jose Maria", "Edad" : 57, "Activo" : True}`	**Uso de llaves** `{}` **:** La forma más común. Los pares se separan con comas y la clave del valor con dos puntos.
`dict2 = dict(color="azul",modelo="Canddy",motor=2.0)`	**Uso del constructor** `dict()` **:** Útil cuando las claves son cadenas de texto sencillas (sin espacios ni caracteres especiales).

B. Reglas de las Claves (Keys)

1. **Unicidad:** Las claves deben ser únicas. Si introduces una clave repetida, Python **sustituye el valor anterior** por el nuevo.
2. **Tipos de Claves:** La clave puede ser de cualquier tipo inmutable (cadenas, números enteros, tuplas).

```
dict3 = {24:"Charcuteria Mano",26:"Medias Puri",28:"Bar el Torrezno"} # Claves numé
```

2. Acceso y Recuperación de Valores

Para obtener un valor, debes referenciar su clave correspondiente.

A. Acceso por Clave (`[]`)

- **Sintaxis:** `diccionario[clave]`

```
print(dict3[26]) # Recupera "Medias Puri"
```

- **Advertencia:** Si la clave no existe, Python lanzará un error (`KeyError`) y detendrá el programa.

B. El Método `get()` (Acceso Seguro)

- **Sintaxis:** `diccionario.get(clave, valor_defecto)`

```
print(dict2.get("motor"))
```

- **Ventaja clave:** Si la clave no se encuentra, `get()` devuelve `None` (o el valor por defecto que especifiques), **sin detener el programa**. Esto es mucho más seguro que usar `[]`.

3. Iteración y Vistas

Existen varias formas de recorrer un diccionario o de obtener sus componentes como listas.

A. Recorrido Básico

1. **Solo Claves (Por defecto):** Iterar sobre el diccionario directamente devuelve solo las claves.

```
for elemento in dict3: # 'elemento' es la clave (24, 26, 28)
    print(elemento)
```

2. **Clave y Valor (Acceso por Clave):** Usando la clave dentro del bucle.

```
for elemento in dict3:
    print(elemento, dict3[elemento]) # Imprime clave y luego valor
```

B. Vistas del Diccionario

Los métodos de vista son esenciales y devuelven colecciones dinámicas de los componentes del diccionario (puedes convertirlas a `list` para ver su contenido).

Método	Tipo de Contenido	Propósito
`dict.keys()`	Claves	Devuelve una lista de **todas las claves**.
`dict.values()`	Valores	Devuelve una lista de **todos los valores**.
`dict.items()`	Pares (Tuplas)	Devuelve una lista de **tuplas (clave, valor)**. Ideal para iterar.

- **Mejor Práctica para Iterar:** Usar `dict.items()` es la forma más limpia y eficiente de obtener clave y valor simultáneamente:

```
for clave, valor in dict3.items():
    print(f"La clave {clave} tiene el valor {valor}")
```

4. Modificación y Manipulación

A. Insertar y Actualizar

- **Asignación Directa:**

```
dict3[30] = "Peluqueria Canina el galgo" # Crea una nueva clave (30)
dict3[24] = "Nuevo Valor"                # Reemplaza el valor de la clave 24
```

- **Método** `update()` **:** Permite fusionar otro diccionario, o pares clave-valor, en el diccionario actual. Si las claves existen, se actualizan; si no, se añaden.

```
dict4 = {"activo":False, "dni":"28777666x"}
dict1.update(dict4) # dict1 ahora tiene 'dni' y el valor de 'Activo' se actualiza a
```

B. Eliminar Elementos

Método	Propósito	Comportamiento Clave
`dict.pop(clave)`	Elimina el par clave-valor especificado y **devuelve el valor** de la clave eliminada.	Lanza `KeyError` si la clave no existe.
`dict.popitem()`	Elimina y **devuelve el último par (tupla)** insertado (en Python 3.7+).	
`dict.clear()`	Elimina todos los elementos del diccionario.	

5. Consejos para Mejorar la Programación con Diccionarios (Examen Pro-Tip)

Los diccionarios son fundamentales en Python. Usarlos correctamente demuestra un alto nivel de programación.

🎯 **Principio 1: ¡Los Diccionarios son Búsqueda O(1)!**

- La búsqueda, inserción y eliminación de elementos en un diccionario es **casi instantánea** (tiempo constante, O(1)), sin importar qué tan grande sea.
- **Consejo de Examen:** Si necesitas revisar repetidamente la existencia de una clave o recuperar un valor por identificador, **siempre usa un diccionario** en lugar de una lista de objetos.

🎯 **Principio 2: Usar** `get()` **para Prevenir Fallos**

- Siempre que intentes acceder a una clave que podría no existir, usa `diccionario.get('clave', 'Valor de Reserva')` en lugar de `diccionario['clave']` . Esto hace que tu código sea más robusto y evita la necesidad de usar bloques `try...except` para errores simples de clave.

🎯 **Principio 3: Usar** `items()` **para Bucle Eficiente**

- Para iterar tanto la clave como el valor, no uses `for key in dict: value = dict[key]` . En su lugar, usa el desempaquetado de tuplas de `items()` :

```
for clave, valor in mi_diccionario.items():
    # Código aquí
```

```
        pass
```

Esto es más rápido y mucho más legible.