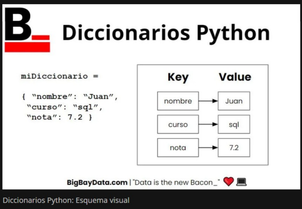
Qué es un diccionario en programación y Python

Los diccionarios son una estructura de datos en Python que te permite almacenar pares de clave-valor. Cada elemento en un diccionario **está compuesto por una clave única y su respectivo valor asociado**. La clave puede ser de cualquier tipo inmutable, como una **cadena de texto o un número,** mientras que el **valor puede ser cualquier objeto válido** en Python.

Los diccionarios en Python se definen utilizando llaves `{}` y los pares clave-valor se separan por dos puntos `:`. Por ejemplo:

Ejemplo de diccionario en Python

```
mi_diccionario = { "nombre" : "Juan", "curso" : "sql", "nota" : 7.2 }
```



Diccionarios Python: Esquema visual

Cómo funcionan los diccionarios en Python

Los diccionarios en Python se implementan **internamente utilizando una estructura de datos llamada tabla hash**. Esta estructura permite **una búsqueda rápida y eficiente de elementos en el diccionario**, incluso para un gran número de elementos.

Cuando se agrega un nuevo par clave-valor a un diccionario, **Python calcula un valor hash único para la clave proporcionada**. Este valor se utiliza para determinar la posición donde se debe almacenar el par clave-valor en la tabla hash. **Cuando se busca un elemento en el diccionario, Python calcula el valor hash de la clave de búsqueda y busca en la tabla hash la posición correspondiente**.

> 💡El Hash es un valor único, por eso, al igual que un diccionario tradicional si no hay palabra (clave) Python indica rápido que no habrá definición (valor).

Operaciones Básicas en Diccionarios

Los **diccionarios en Python admiten una variedad de operaciones básicas** que te permiten manipular y trabajar con los elementos contenidos en ellos. Algunas de estas operaciones incluyen **agregar elementos, eliminar elementos, actualizar valores y comprobar si una clave está** presente en el diccionario.

Añadir elemento en diccionario

Para agregar un **nuevo par clave-valor a un diccionario**, puedes utilizar la siguiente sintaxis:

```
mi_diccionario["nueva_clave"] = nuevo_valor
```

Como eliminar diccionarios en Python

Para **eliminar un elemento del diccionario**, puedes utilizar la palabra clave `del` seguida de la clave que deseas eliminar:

```
del mi_diccionario["clave_a_eliminar"]
```

Actualizar valores en diccionario

Para **actualizar el valor de un elemento existente en el diccionario**, simplemente asigna un nuevo valor a la clave correspondiente:

```
mi_diccionario["clave_existente"] = nuevo_valor
```

Comprobar si existe valor en diccionario en Python

Para **verificar si una clave está presente en el diccionario**, puedes utilizar el operador `in`:

```
if "clave" in mi_diccionario:

# hacer algo si la clave está presente
```

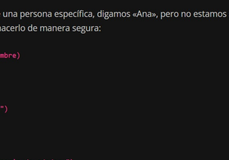
Funciones comunes en Python para diccionarios

get() para obtener el valor

El método `get()` en Python se utiliza para **acceder al valor asociado a una clave en un diccionario** de manera segura, es decir, **sin generar errores si la clave no existe** en el diccionario. Además, este método permite especificar un valor por defecto que se devolverá en caso de que la clave no se encuentre en el diccionario. Aquí tienes **un ejemplo para entenderlo** mejor:

Supongamos que tenemos un diccionario que almacena información sobre personas y sus edades:

```
edades = { "Juan": 30, "María": 25, "Carlos": 35 }
```



Ejemplo de diccionario Python

Ahora, queremos obtener la edad de una persona específica, digamos «Ana», pero no estamos seguros de si «Ana» está en el diccionario. Usar `get()` nos permite hacerlo de manera segura:

```
nombre = "Ana" edad = edades.get(nombre)

if edad is not None:

print(nombre, "tiene", edad, "años.")

else:

print(nombre, "no está en el diccionario de edades.")
```

Luego, verificamos si `edad` es `None` para determinar si «Ana» está en el diccionario o no. Si `edad` no es `None`, imprimimos su edad. **Si `edad` es `None`, imprimimos un mensaje que indica que «Ana» no está en el diccionario**.

Este enfoque es útil para evitar errores de tipo `KeyError` que ocurrirían si intentáramos acceder directamente a una clave que no existe en el diccionario sin usar `get()`. Con `get()`, podemos proporcionar un valor por defecto o manejar de manera adecuada la ausencia de la clave en el diccionario.

Tamaño del diccionario: len()

Siguiendo con el ejemplo anterior, podríamos utilizar la variable `edad` de antes y evaluar el número de elementos de la siguiente manera:

```
print( len(edad) )
```

`len()` es una función útil para **determinar rápidamente el tamaño de un diccionario**, lo que puede ser especialmente útil cuando necesitas realizar operaciones que dependen del número de elementos en el diccionario, como **bucles o validaciones**.

> 💡Aplicando la función type( ) a un diccionario Python nos dirá que es de tipo dict.

Diccionarios en Python: Métodos más usados

Los diccionarios son estructuras de datos que se utilizan en programación para almacenar pares clave-valor, donde **cada clave está asociada a un valor**. Los métodos más comunes de los diccionarios en lenguajes de programación como Python son los siguientes:

Resumen de funciones diccionarios Python con ejemplos

Método	Descripción	Ejemplo
`dict.keys()`	Devuelve una **lista de las claves** del diccionario	`python diccionario = {"nombre": "Juan", "edad": 30} claves = diccionario.keys()`
`dict.values()`	Devuelve **una lista de los valores** del diccionario	`python diccionario = {"nombre": "Juan", "edad": 30} valores = diccionario.values()`
`dict.items()`	Recibe**una lista de tuplas** (clave, valor)	`python diccionario = {"nombre": "Juan", "edad": 30} items = diccionario.items()`
`dict.get(key)`	**Devuelve el valor asociado a una clave** o un valor por defecto si la clave no existe	`python diccionario = {"nombre": "Juan", "edad": 30} nombre = diccionario.get("nombre") edad = diccionario.get("edad", 0)`
`dict.pop(key)`	**Elimina una clave y su valor del diccionario y devuelve el valor eliminado**	`python diccionario = {"nombre": "Juan", "edad": 30} valor_eliminado = diccionario.pop("nombre")`
`dict.update(other_dict)`	**Actualiza el diccionario con pares clave-valor de otro diccionario**	`python diccionario = {"nombre": "Juan", "edad": 30} otro_diccionario = {"ciudad": "Madrid"} diccionario.update(otro_diccionario)`
`len(dict)`	Devuelve la cantidad de elementos en el diccionario	`python diccionario = {"nombre": "Juan", "edad": 30} cantidad_elementos = len(diccionario)`

Métodos de diccionarios en Python más comunes

En resumen, estos son algunos de los **métodos más comunes en los diccionarios en Python**, pero **hay muchos más** disponibles para realizar diferentes operaciones y manipulaciones de datos en diccionarios. Estos métodos son útiles para acceder, modificar y trabajar con la información almacenada en diccionarios de manera eficiente.