«Talento Tech»

Iniciación a la Programación con Python

CLASE 10

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Buenos Aires Ciudad

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Clase N° 10 | Diccionarios

Temario:

- Diccionarios: uso y métodos esenciales.
- Uso de diccionarios como medio de almacenamiento temporal de datos
- Ejercicios
- Cuestionario en Campus Virtual.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Introducción a los diccionarios.

Ya conocés listas y tuplas, que te permiten almacenar múltiples valores en una sola variable. Pero hay un tipo de estructura en Python que alcanza incluso mayor flexibilidad y es sumamente recomendable para determinadas tareas: **los diccionarios.** Mientras que las listas y tuplas almacenan elementos de forma ordenada por su índice, los diccionarios funcionan como una especie de *"caja de herramientas"* donde podés guardar valores asociados a una clave. Cada clave está asociada a un valor y esto te permite almacenar y acceder a los datos de una manera más eficiente.

En un sistema de inventario, como el que estás desarrollando para tu Proyecto Final Integrador, los diccionarios son ideales para almacenar información de productos ya que podés asociar la clave "nombre del producto" o "código" con datos como la cantidad disponible o el precio.

Crear un diccionario.

Un diccionario se define con **llaves {}**, y cada elemento tiene una clave y un valor. La clave debe ser única, mientras que el valor puede ser de cualquier tipo (números, cadenas, listas, etc.).

```
# Diccionario de productos con nombre y cantidad
productos = {
   "Manzanas": 50,
   "Peras": 30,
   "Bananas": 40
}
```

En este ejemplo, la clave es el nombre del producto ("Manzanas", "Peras", "Bananas") y el valor es la cantidad disponible en el inventario.

Acceder a los valores de un diccionario.

Para acceder a un valor en un diccionario, simplemente usás el nombre de la clave entre corchetes. Por ejemplo, para ver cuántas manzanas tenés en stock:

```
print("Cantidad de manzanas:", productos["Manzanas"])
```

Verás esto en la terminal:

```
Cantidad de manzanas: 50
```

Si la clave no existe en el diccionario, Python te dará un error, por lo que siempre es recomendable asegurarte de que la clave esté presente o utilizar el método **.get()**, que evita errores si la clave no existe:

```
# Si "Peras" no está en el diccionario, devuelve 0
cantidad_peras = productos.get("Peras", 0)
print("Cantidad de peras:", cantidad_peras)
```

El método .get() de los diccionarios en Python es una forma muy útil de acceder a los valores asociados a una clave sin preocuparse de que esa clave exista o no en el diccionario. Normalmente, cuando tratás de acceder a una clave que no está en el diccionario usando la sintaxis de corchetes (por ejemplo, productos["Peras"]), Python lanza un error que detiene la ejecución del programa, ya que no encuentra esa clave.

Esto es un problema cuando no podés estar seguro de que la clave que estás buscando realmente exista en el diccionario. Ahí es donde **.get()** se vuelve muy útil, porque en lugar de dar un error, te devuelve un valor predeterminado si la clave no está presente.

La sintaxis básica de **.get()** es:

```
valor = diccionario.get(clave, valor_por_defecto)
```

Donde:

- **clave** es la clave que estás buscando en el diccionario.
- **valor_por_defecto** es el valor que se devuelve si la clave no está en el diccionario (por defecto es *"none"* si no especificás este valor).

Imaginemos que tenés un inventario de productos, pero querés obtener la cantidad de un producto que podría o no estar en el inventario. Si usás el acceso directo con corchetes y la clave no existe, Python va a lanzar un error. En cambio, con .get(), podés manejar esto con más seguridad.

```
# Diccionario de productos
productos = {
   "Manzanas": 50,
   "Peras": 30
}

# Tratamos de obtener un producto que no existe
cantidad_uvas = productos.get("Uvas", 0)  # Si "Uvas" no está, devuelve
0
print("Cantidad de uvas:", cantidad_uvas)
```

En este caso, como "Uvas" no está en el diccionario, .get() devuelve "0" en lugar de lanzar un error.

Si no especificás un valor por defecto, .get() devuelve “None” cuando la clave no se encuentra en el diccionario:

```
# No especificamos valor por defecto
cantidad_bananas = productos.get("Bananas")
print(cantidad_bananas)  # Imprime None
```

Este método es muy útil cuando estás trabajando con datos que pueden variar, como en el caso de un sistema de inventario en el que los productos pueden estar o no presentes en un momento determinado.

Agregar o actualizar elementos.

Los diccionarios son dinámicos, lo que significa que podés agregar nuevos pares clave-valor o actualizar los existentes fácilmente.

Si querés agregar un nuevo producto al inventario, simplemente le asignás un valor a una nueva clave:

```
# Diccionario de productos
productos = {
   "Manzanas": 50,
   "Peras": 30
}

productos["Naranjas"] = 25
print(productos)
```

Y si querés actualizar la cantidad de un producto, simplemente accedés a su clave y le asignás un nuevo valor:

```
productos["Manzanas"] = 60  # Ahora hay 60 manzanas
```

Eliminar elementos.

En ocasiones vas a necesitar eliminar elementos que ya no son útiles. Esto puede suceder, por ejemplo, en un sistema de inventario cuando un producto deja de estar en stock o cuando querés limpiar datos innecesarios. Eliminar elementos de un diccionario es un proceso muy simple y existen diferentes formas de hacerlo, cada una con características propias.

Eliminar un elemento con del.

El uso de la **palabra clave** del es la forma más directa de eliminar un elemento del diccionario. Al usarla vas a especificar la clave que querés eliminar. Si esa clave existe, se elimina tanto la clave como su valor asociado.

```
# Diccionario de productos
productos = {
    "Manzanas": 50,
    "Peras": 30,
    "Bananas": 20
}

# Eliminamos las "Bananas" del diccionario
del productos["Bananas"]

# Mostramos el diccionario después de la eliminación
print(productos)
# Resultado: {'Manzanas': 50, 'Peras': 30}
```

En este ejemplo, al usar del productos["Bananas"], estamos eliminando tanto la clave "Bananas" como el valor asociado (20). Después de ejecutar la instrucción, "Bananas" ya no está en el diccionario.

Eliminar y obtener el valor con .pop().

En ocasiones no solo buscarás eliminar un elemento sino que también te interesará obtener su valor al mismo tiempo. El método **.pop()** hace precisamente eso: elimina el elemento y te devuelve su valor.

```
# Diccionario de productos
productos = {
   "Manzanas": 50,
   "Peras": 30,
   "Bananas": 20
}

# Eliminamos "Peras" y obtenemos su valor
cantidad_peras = productos.pop("Peras", 0)
# Si no existe "Peras", devuelve 0

# Mostramos el valor eliminado y el diccionario actualizado
print("Cantidad eliminada:", cantidad_peras)
# Resultado: Cantidad eliminada: 30

print(productos)
# Resultado: {'Manzanas': 50, 'Bananas': 20}
```

En este ejemplo, .pop("Peras", 0) elimina el producto "Peras" del diccionario y devuelve su cantidad (30). Si "Peras" no hubiera estado en el diccionario, habría devuelto el valor por defecto que es 0.

Con **pop(),** todas son ventajas. Por ejemplo, si querés no solo eliminar el elemento sino también usar el valor para alguna operación posterior, con esta herramienta realizás ambas acciones al mismo tiempo. También es útil para evitar errores: en caso de que la clave no exista, podés especificar un valor por defecto para que no ocurra un error, igual que en el método **.get()**.

Eliminar todos los elementos con .clear()

Si en algún momento querés eliminar todos los elementos de un diccionario y dejarlo vacío, podés usar el método .clear(). Esto es útil cuando necesitás resetear completamente los datos.

```
# Diccionario de productos
productos = {
    "Manzanas": 50,
    "Peras": 30,
    "Bananas": 20
}

# Limpiamos el diccionario
productos.clear()

# Mostramos el diccionario después de limpiarlo
print(productos)
# Resultado: {}
```

Después de ejecutar .clear(), el diccionario queda vacío, sin ninguna clave ni valor.

Claramente, eliminar elementos de un diccionario es algo común y Python ofrece varias formas de hacerlo según lo que necesites. Aprendimos que:

- **del**: Elimina una clave específica (y su valor), pero lanzará un error si la clave no existe.
- **.pop()**: Elimina una clave y devuelve su valor. También podés usarlo para evitar errores si la clave no está.
- **.clear():** Borra todo el contenido del diccionario, dejándolo vacío.

Cada método tiene sus ventajas, y elegir el correcto depende de si necesitás recuperar el valor eliminado o si querés evitar errores en caso de que la clave no exista. Esto es muy importante cuando trabajás con datos variables, como en el caso de un inventario, donde los productos pueden cambiar constantemente.

Recorrer un diccionario con bucles

Recorrer un diccionario con bucles es una de las formas más eficientes de obtener tanto las claves como los valores almacenados en él. Esto es particularmente útil cuando estás trabajando con un inventario de productos o cualquier conjunto de datos que se almacenen de manera organizada en un diccionario.

Recorrer un diccionario con .items()

El método **.items()** es clave cuando querés obtener simultáneamente tanto las claves como los valores de un diccionario. Devuelve cada par clave-valor en forma de tuplas, lo que nos permite descomponerlas fácilmente en un bucle for. Así, podemos ver el nombre del producto (clave) y su cantidad (valor) en el inventario, por ejemplo.

```
# Diccionario de productos con cantidad en stock
productos = {
    "Manzanas": 50,
    "Peras": 30,
    "Bananas": 20
}
# Usamos un bucle for para recorrer claves y valores
for producto, cantidad in productos.items():
    print("Producto:", producto, "Cantidad en stock:", cantidad)
```

En este ejemplo, productos.items() devuelve cada clave y su valor como una tupla: ("Manzanas", 50), ("Peras", 30), y así sucesivamente. Al descomponerlas en producto y cantidad, podemos imprimir ambos en el bucle. Esto es lo que ves en la terminal:

```
Producto: Manzanas Cantidad en stock: 50
Producto: Peras Cantidad en stock: 30
Producto: Bananas Cantidad en stock: 20
```

Esto es ideal para cuando necesitás trabajar con los dos elementos al mismo tiempo: el producto y la cantidad. Si, por ejemplo, estás gestionando un inventario, esta es la forma más intuitiva de obtener toda la información del diccionario.

Recorrer sólo las claves con .keys()

A veces sólo necesitarás obtener las claves de un diccionario sin los valores asociados. Para eso, podés usar .keys() que devuelve sólo las claves del diccionario. Esto es útil, por ejemplo, si únicamente querés listar qué productos tenés en el inventario sin preocuparte por cuántos quedan.

```
# Diccionario de productos con cantidad en stock
productos = {
    "Manzanas": 50,
    "Peras": 30,
    "Bananas": 20
}

# Usamos un bucle for para recorrer solo las claves
for producto in productos.keys():
    print("Producto disponible:", producto)
```

En este caso, **.keys()** devuelve todas las claves del diccionario, que son los nombres de los productos. Esto te permite listar sólo los nombres, lo que puede ser útil en un sistema de inventarios cuando querés que quien le de uso al mismo seleccione un producto sin mostrar las cantidades. Esto es lo que ves en la terminal:

```
Producto disponible: Manzanas
Producto disponible: Peras
Producto disponible: Bananas
```

Recorrer solo los valores con .values()

Por otro lado, si lo único que te interesa son los valores de un diccionario, podés utilizar **.values()**. Esto te permite obtener directamente los valores, sin preocuparte por las claves. Imaginá que querés saber cuántas unidades en total tenés en tu inventario. Podrías usar .values() para trabajar con los números (cantidades) sin necesidad de usar los nombres de los productos.

```
# Diccionario de productos con cantidad en stock
productos = {
    "Manzanas": 50,
    "Peras": 30,
    "Bananas": 20
}

# Usamos un bucle for para recorrer solo los valores
total = 0
for cantidad in productos.values():
    total = total + cantidad

print("Cantidad total de productos en stock:", total)
```

En este caso, productos.values() devuelve los valores del diccionario, es decir, las cantidades de cada producto (50, 30, 20). Luego, simplemente sumamos esos valores para obtener el total del inventario. Esto es lo que ves en la terminal:

```
Cantidad total de productos en stock: 100
```

Diferencia entre .items(), .keys(), y .values()

Como podés ver, cada método tiene su uso dependiendo de lo que necesites hacer:

- **.items()** te da tanto las claves como los valores. Es perfecto para cuando querés trabajar con ambos al mismo tiempo, como al recorrer un inventario completo.
- **.keys() t**e da solo las claves. Esto es útil si querés listar o trabajar únicamente con los nombres de los productos o claves de cualquier tipo.
- **.values()** te da solo los valores. Es ideal si necesitás trabajar solo con las cantidades, precios, o cualquier dato numérico o no asociado a una clave.

Ejemplo combinado.

Podemos combinar estos métodos según nuestras necesidades. Supongamos que querés mostrar los productos disponibles en el inventario, pero también calcular cuántas unidades en total hay:

```
# Diccionario de productos con cantidad en stock
productos = {
    "Manzanas": 50,
    "Peras": 30,
    "Bananas": 20
}

# Recorremos el diccionario para mostrar los productos
```

```
print("Inventario de productos:")
for producto, cantidad in productos.items():
   print("Producto:", producto, "- Cantidad en stock:", cantidad)

# Calculamos el total de productos en stock
total_productos = sum(productos.values())
print("Total de productos en stock:", total_productos)
```

Acá utilizamos **.items()** para recorrer y mostrar el nombre del producto y su cantidad en stock, y luego usamos **.values()** para obtener las cantidades y sumarlas de manera tal de mostrar el total de productos en el inventario.

Esta es la apariencia de la terminal luego de correr el script:

```
Inventario de productos:
Producto: Manzanas - Cantidad en stock: 50
Producto: Peras - Cantidad en stock: 30
Producto: Bananas - Cantidad en stock: 20
Total de productos en stock: 100
```

El recorrido de diccionarios es una herramienta poderosísima cuando trabajás con colecciones de datos en Python. Con **.items(), .keys(), y .values()**, podés adaptar tu bucle for para obtener exactamente la información que necesitás de tu diccionario, lo que lo hace extremadamente flexible y útil en muchos casos, como puede ser la gestión de inventarios o cualquier sistema que maneje datos clave-valor.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Uso de diccionarios como almacenamiento temporal.

Aprendimos que una de las grandes ventajas de los diccionarios es su capacidad para almacenar datos de manera organizada permitiendo acceder a ellos rápidamente. Esto es ideal cuando necesitás guardar información temporalmente y procesarla. En el caso del inventario de una tienda, por ejemplo, podés usar diccionarios para guardar los datos de los productos que se están vendiendo en un día y luego actualizar el inventario principal con la cantidad vendida. Así que te proponemos el siguiente ejemplo para finalizar esta clase:

Actualización de inventario diario

Supongamos que querés registrar las ventas del día y luego actualizar el inventario principal al final del día. **Usamos dos diccionarios:** uno para el inventario y otro para las ventas diarias.

```
# Inventario inicial
inventario = {
    "Manzanas": 50,
    "Peras": 30,
    "Bananas": 40
}

# Diccionario para registrar ventas del día
ventas_dia = {}

# Pedimos al usuario que ingrese el producto y la cantidad vendida
# Supongamos que vendemos 3 productos diferentes en el día
for _ in range(3):
    producto = input("Ingresá el producto vendido: ")
```

```
    cantidad_vendida = int(input("Ingresá la cantidad vendida: "))

    # Si el producto está en el inventario, registramos la venta
    if producto in inventario:
        # Actualizamos el diccionario de ventas
        ventas_dia[producto] = cantidad_vendida
    else:
        print("Producto no encontrado en inventario.")

# Actualizamos el inventario con las ventas del día
for producto, cantidad in ventas_dia.items():
    inventario[producto] = inventario[producto] - cantidad

print("Inventario actualizado:", inventario)
```

Este ejemplo muestra cómo podés usar diccionarios para almacenar datos temporalmente y luego utilizarlos para actualizar una estructura de datos más grande (el inventario). Al final del día, el inventario refleja las ventas realizadas y se ajusta correctamente.

```
Ingresá el producto vendido: Peras
Ingresá la cantidad vendida: 5
Ingresá el producto vendido: Bananas
Ingresá la cantidad vendida: 10
Ingresá el producto vendido: Peras
Ingresá la cantidad vendida: 3
Inventario actualizado: {'Manzanas': 50, 'Peras': 27, 'Bananas': 30}
```

Hemos visto que los diccionarios son una herramienta poderosa que te permite organizar y gestionar información de manera eficiente. En el contexto del Proyecto Final Integrador, podrías utilizarlos para almacenar datos de productos, ventas, y cualquier otra

información que tu sistema de inventario requiera. A medida que avanzás, vas a ver cómo los diccionarios facilitan la gestión de datos y pronto vas a poder combinarlos con otros conceptos que aprenderemos, como las bases de datos.

Ejercicios prácticos:

Gestión de inventario con diccionarios.

En un comercio, se necesita gestionar los productos y sus precios. Desarrollá un programa que permita:

1. Ingresar el nombre de tres productos y su precio correspondiente, guardándolos en un diccionario donde la clave es el nombre del producto y el valor es su precio.
2. Una vez ingresados, mostrará todos los productos y sus precios en pantalla.

Ejemplo de salida esperada:

```
Producto: Manzanas, Precio: 100
Producto: Naranjas, Precio: 150
Producto: Peras, Precio: 120
```

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Consultar stock en inventario

El inventario de una tienda está almacenado en un diccionario, donde las claves son los nombres de los productos y los valores son las cantidades disponibles en stock. Creá un programa que:

1. Te permita ingresar el nombre de un producto.
2. Utilice el método .get() para buscar el stock disponible. Si el producto no existe, deberá mostrar un mensaje diciendo "Producto no encontrado".
3. Si el producto está disponible, mostrará cuántas unidades quedan en stock.

Diccionario inicial:

```
productos = {
   "Manzanas": 50,
   "Naranjas": 30,
   "Peras": 25
}
```

Ejemplo de salida esperada:

```
Ingresá el nombre del producto: Peras
Stock disponible de Peras: 25
```

Buenos Aires aprende

Agencia de Habilidades para el Futuro

Buenos Aires Ciudad