«Talento Tech»

Iniciación a la

Programación con Python

Clase 08

Buenos Aires Ciudad

Clase N° 8 | Diccionarios

Temario:

- Concepto de clave y valor.
- Creación y manipulación de diccionarios.
- Métodos esenciales de diccionarios.
- Iteración sobre diccionarios.

Objetivos de la Clase

En esta clase, aprenderás a utilizar los **diccionarios**, una estructura de datos esencial en Python que permite almacenar información de una manera más organizada y eficiente. Vas a descubrir cómo funcionan los **pares clave-valor** y cómo esta característica facilita el acceso rápido a los datos sin necesidad de recurrir a posiciones específicas como en las listas.

Explorarás cómo recorrer diccionarios con bucles y cómo estos recorridos pueden aplicarse en situaciones prácticas, preparándote para desafíos más complejos. Este nuevo conocimiento te permitirá gestionar datos de manera más clara y efectiva, una habilidad que será clave para tu **Trabajo Final Integrador (TFI)**.

Octava jornada en TechLab 🚀

El día comienza con el movimiento habitual en **TalentoLab**. Mientras llegás a tu escritorio, una nota de **Luis** te indica que **Mariana** quiere verte en la sala de reuniones.

Mariana te explica que el equipo necesita una pequeña aplicación para registrar productos y sus precios en pesos. Esta vez, vas a necesitar una estructura eficiente para organizar y acceder rápidamente a los datos.

Te cuenta que la mejor opción es usar **diccionarios** en Python, una herramienta que permite asociar claves con valores.

¡Veamos cómo funcionan!

Introducción a diccionarios

En Python, los **diccionarios** son una estructura de datos que permite almacenar información en forma de **pares clave-valor**. A diferencia de las listas y las tuplas, donde los elementos se acceden mediante un índice numérico, en los diccionarios cada elemento se accede a través de una clave única que está asociada a un valor específico.

Para entender mejor los diccionarios en Python, podemos hacer una analogía con los diccionarios del mundo real, como los diccionarios de idiomas.

- **Claves (Keys):** En un diccionario de idiomas, las claves serían las palabras. Cada palabra (clave) debe ser única para evitar confusiones y garantizar que se puede encontrar rápidamente.
- **Valores (Values):** Los valores serían las definiciones o traducciones asociadas a cada palabra. Estos valores proporcionan la información o el significado correspondiente a cada clave.

Por ejemplo, en un diccionario de inglés a español:

- La clave "apple" tiene el valor "manzana".
- La clave "book" tiene el valor "libro".

Un diccionario en Python se define utilizando **llaves {}** y cada par clave-valor se separa con dos puntos :. Por ejemplo:

```
productos = {
   "manzana": 150,
   "banana": 120,
   "naranja": 180
}
```

En este diccionario, "manzana", "banana" y "naranja" son las claves, mientras que 150, 120 y 180 son los valores asociados a cada clave. Esta estructura permite acceder rápidamente al precio de un producto a través de su nombre, sin tener que recorrer toda la colección.

Para acceder a los valores de un diccionario, utilizamos las **claves**, de manera similar a como usamos los **índices** en las listas. La diferencia es que, mientras en una lista los índices son números enteros que representan la posición de cada elemento, en un diccionario las claves pueden ser palabras u otros tipos de datos. Esta flexibilidad hace que los diccionarios sean más intuitivos para ciertas tareas.

Por ejemplo, en una lista de precios, accederíamos al valor por su posición:

```
precios_lista = [150, 120, 180]
print(precios_lista[0])  # Salida: 150 (precio de la manzana)
```

En un diccionario, accedemos al valor usando una clave:

```
productos = {
   "manzana": 150,
   "banana": 120,
   "naranja": 180
}
print(productos["manzana"])  # Salida: 150
```

En este ejemplo, en lugar de recordar que el precio de la manzana está en la posición **0,** simplemente usamos la clave "**manzana**" para obtener su precio. Esto hace que el código sea más fácil de entender y menos propenso a errores, especialmente cuando trabajamos con grandes cantidades de datos.

Comparación entre listas, tuplas y diccionarios

Luis, notando que no terminas de encontrar la relación entre cada uno de estos tipos de datos, te proporciona un resúmen de sus características:

Estructura	Descripción	Ejemplo	Acceso
Listas	Almacenan colecciones ordenadas de elementos que se pueden modificar.	`frutas = ["manzana", "banana", "naranja"]`	frutas[0] devuelve "manzana"
Tuplas	Similares a las listas, pero **inmutables**, es decir, no se pueden modificar.	`colores = ("rojo", "azul", "verde")`	colores[1] devuelve "azul"
Diccionarios	Permiten acceder a los elementos mediante **claves personalizadas**,	`productos = {"manzana": 150, "banana": 120}`	productos["manzana"] devuelve 150

	facilitando búsquedas más rápidas y descriptivas.		

Esta tabla resume de manera clara las diferencias entre listas, tuplas y diccionarios, ayudándote a elegir la estructura adecuada según el tipo de datos que necesites manejar.

Características de los diccionarios en Python

Característica	Descripción	Ejemplo
Estructura de Clave-Valor	Cada entrada tiene una clave única y un valor asociado. Los valores pueden repetirse.	`{"nombre": "Juan", "edad": 25}`
Acceso Rápido a los Datos	Permite acceder a los valores rápidamente mediante claves, lo cual es eficiente con grandes volúmenes de datos.	`diccionario["nombre"]` `# devuelve "Juan"`
Mutable	Se puede modificar, agregar o eliminar pares clave-valor después de crear el diccionario.	`diccionario["ciudad"] = "Buenos Aires"`
Orden	Desde Python 3.7, mantiene el orden de inserción; en versiones anteriores el orden no era garantizado.	`{"a": 1, "b": 2}`
Diversidad de Tipos	Las claves deben ser inmutables (cadenas, números, tuplas). Los valores pueden ser cualquier tipo de datos.	`{"nombre": "Ana", 1: [2, 3], (1, 2): "Par ordenado"}`
Extensible	Puede crecer o reducirse dinámicamente al agregar o eliminar pares clave-valor (veremos esto en detalle más adelante)	`diccionario.pop("edad")`

Métodos esenciales de diccionarios en Python

Al igual que con las **listas** y las **tuplas**, los diccionarios cuentan con una serie de métodos que facilitan su manipulación y te permiten realizar tareas comunes como agregar, eliminar, buscar y modificar elementos. A continuación, Luis te muestra una tabla con los métodos más importantes para trabajar con diccionarios en Python, junto con ejemplos prácticos y una breve explicación de cada uno.

Método	Ejemplo de uso	Descripción
get()	`diccionario.get("nombre")`	Devuelve el valor asociado a la clave especificada. Si la clave no existe, devuelve None o un valor por defecto.
keys()	`diccionario.keys()`	Devuelve una vista de todas las claves del diccionario.
values()	`diccionario.values()`	Devuelve una vista de todos los valores del diccionario.
items()	`diccionario.items()`	Devuelve una vista de todos los pares clave-valor como tuplas.
pop()	`diccionario.pop("edad")`	Elimina y devuelve el valor asociado a la clave especificada. Si la clave no existe, lanza un error.
popitem()	`diccionario.popitem()`	Elimina y devuelve el último par clave-valor insertado (desde Python 3.7 en adelante).
update()	`diccionario.update({"ciudad": "Buenos Aires"})`	Actualiza el diccionario con pares clave-valor de otro diccionario o iterable.
clear()	`diccionario.clear()`	Elimina todos los elementos del diccionario, dejándolo vacío.
copy()	`nuevo_diccionario = diccionario.copy()`	Devuelve una copia superficial del diccionario.

Estos métodos te permiten manejar diccionarios de forma eficiente y adaptarlos a las necesidades específicas de tus programas. Entender cómo funcionan te ayudará a escribir

código más claro, conciso y robusto, habilidades que serán fundamentales para completar tu Trabajo Final Integrador (TFI). Es por esto que Luis te invita a escribir un script que utilice varios métodos de los diccionarios.

```
# Creamos un diccionario con información de una cliente
cliente = {
   "nombre": "Lucía",
   "edad": 30,
   "ciudad": "Córdoba"
}

# Usamos get() para acceder a una clave existente y a una que no existe
print(cliente.get("nombre"))      # Salida: Lucía
print(cliente.get("telefono"))    # Salida: None

# Usamos keys() para ver todas las claves del diccionario
print(cliente.keys())
# Salida: dict_keys(['nombre', 'edad', 'ciudad'])

# Usamos values() para ver todos los valores
print(cliente.values())
# Salida: dict_values(['Lucía', 30, 'Córdoba'])

# Usamos items() para ver los pares clave-valor
print(cliente.items())
# Salida: dict_items([('nombre', 'Lucía'), ('edad', 30), ('ciudad',
'Córdoba')])

# Agregamos una nueva clave con setdefault()
cliente.setdefault("telefono", "123-4567")
print(cliente)
# Salida: {'nombre': 'Lucía', 'edad': 30, 'ciudad': 'Córdoba',
'telefono': '123-4567'}

# Eliminamos una clave con pop()
cliente.pop("edad")
print(cliente)
# Salida: {'nombre': 'Lucía', 'ciudad': 'Córdoba', 'telefono':
'123-4567'}
```

Mientras mirás el código que acabas de escribir, Luis se acerca con su café y, con su tono relajado de siempre, te dice que arrancaste creando un diccionario llamado **cliente** con información básica como el nombre, la edad y la ciudad. Después, usaste **get()** para acceder al valor de la clave "**nombre**", lo que te devolvió "**Lucía**", y también probaste con una clave que no existía, "**telefono**", que por defecto te devolvió **None** en lugar de tirar un error. Es una gran forma de manejar claves inexistentes sin que el programa se rompa.

Luis toma un sorbo de su café y sigue dando cuenta de lo trabajado: además le pediste a Python que te muestre todas las claves con **keys()**, los valores con **values()**, y los pares clave-valor con **items()**. Eso es re útil cuando necesitás ver qué hay en el diccionario sin recorrerlo manualmente. También agregaste una nueva clave "**telefono**" usando **setdefault()**, que solo la creó si no existía ya. Finalmente, eliminaste la clave "**edad**" con **pop()**, lo que te permitió modificar el diccionario sin problemas. Todo esto te da flexibilidad para manejar la información sin importar cuán compleja sea.

Ejemplo práctico 1: Registro de clientes con diccionarios y listas

Ahora, vamos a crear un programa que registre la información de varios clientes. Se nos pedirá ingresar el código del cliente, su nombre y su ciudad. Cada conjunto de datos se almacenará en un diccionario, y todos estos diccionarios se agregarán a una lista. Al final, mostraremos toda la información registrada. Vamos a ver cómo se hace.

Este ejemplo es una excelente manera de recordar que las listas pueden contener cualquier tipo de dato, incluidos diccionarios.

```
# Creamos una lista vacía para almacenar los diccionarios
clientes = []

# Bucle para ingresar los datos de varios clientes
while True:
   print("\nIngresá los datos del cliente.[vacío para finalizar]:")
   codigo = input("Código del cliente: ")

   # Condición para salir del bucle si el código está vacío
   if codigo == "":
       break

   nombre = input("Nombre del cliente: ")
   ciudad = input("Ciudad del cliente: ")

   # Creamos un diccionario con los datos ingresados
```

```
    cliente = {
        "código": codigo,
        "nombre": nombre,
        "ciudad": ciudad
    }

    # Agregamos el diccionario a la lista de clientes
    clientes.append(cliente)

# Mostramos los datos de todos los clientes registrados
print("\n--- Clientes Registrados ---")
for cliente in clientes:
    print(f"Código: {cliente['código']}, Nombre: {cliente['nombre']},
Ciudad: {cliente['ciudad']}")
```

Al ejecutarlo e interactuar con el programa, ves que una salida posible es la siguiente:

```
Ingresá los datos del cliente. Para finalizar, dejá el código vacío.
Código del cliente: 1
Nombre del cliente: Ana
Ciudad del cliente: CABA

Ingresá los datos del cliente. Para finalizar, dejá el código vacío.
Código del cliente: 3
Nombre del cliente: Juan
Ciudad del cliente: Rosario

Ingresá los datos del cliente. Para finalizar, dejá el código vacío.
Código del cliente: 8
Nombre del cliente: Luis
Ciudad del cliente: Quilmes

Ingresá los datos del cliente. Para finalizar, dejá el código vacío.
Código del cliente:

--- Clientes Registrados ---
Código: 1, Nombre: Ana, Ciudad: CABA
Código: 3, Nombre: Juan, Ciudad: Rosario
Código: 8, Nombre: Luis, Ciudad: Quilmes
```

Veamos la explicación detallada de lo que hace este código. El programa empieza creando una lista vacía llamada **clientes**. Esta lista se usará para almacenar los diccionarios que contienen los datos de cada cliente.

Luego, utilizamos un bucle **while** para permitir que la persona ingrese los datos de varios clientes. En cada iteración, se le pide el código, el nombre, y la ciudad del cliente. Si se deja el campo del código vacío y presiona Enter, el bucle se interrumpe gracias a la sentencia **break**.

Una vez que se obtienen los datos, se crea un diccionario llamado cliente con las claves "**código**", "**nombre**" y "**ciudad**", y se asignan los valores ingresados por el usuario a cada una de estas claves. Este diccionario se agrega a la lista **clientes** usando el método **append()**.

Al final del programa, recorremos la lista de diccionarios con un bucle **for**. Para cada cliente en la lista, mostramos la información usando una **f-string** para formatear la salida de manera ordenada. Este ejemplo, dice Luis, te muestra cómo combinar listas y diccionarios para gestionar datos más complejos. Es una técnica muy útil para tu Trabajo Final Integrador (TFI) o cualquier proyecto donde necesites manejar múltiples registros de información organizada.

Ejemplo práctico 2: Eliminando elementos de un diccionario con un bucle

Analicemos, nos propone Luis, un último ejemplo. Esta vez partimos de un diccionario ya definido con algunos productos y sus respectivos precios. Utilizaremos un bucle para mostrar los productos disponibles y permitir que el usuario elija cuál quiere eliminar. Al finalizar, el programa mostrará el estado final del diccionario con los productos restantes.

```
# Diccionario inicial con productos y precios
productos = {
    "lápiz": 50,
    "cuaderno": 200,
    "goma": 30,
    "marcador": 100
}

# Mostramos los productos disponibles al inicio
print("Productos disponibles:")
for nombre, precio in productos.items():
    print(f"{nombre.capitalize()}: ${precio}")

# Bucle para permitir al usuario eliminar productos
```

```
while True:
    print("\nIngresá el nombre del producto que querés eliminar (o
presioná Enter para terminar):")
    eliminar = input("Producto a eliminar: ").lower()

    # Si el usuario presiona Enter sin escribir nada, se sale del bucle
    if eliminar == "":
        break

    # Verificamos si el producto está en el diccionario y lo eliminamos
    if eliminar in productos:
        productos.pop(eliminar)
        print(f"El producto '{eliminar}' fue eliminado del inventario.")
    else:
        print(f"El producto '{eliminar}' no existe en el inventario.")

    # Mostramos el estado actual del diccionario
    print("\nProductos restantes:")
    for nombre, precio in productos.items():
        print(f"{nombre.capitalize()}: ${precio}")

# Estado final del diccionario
print("\n--- Estado final del inventario ---")
for nombre, precio in productos.items():
    print(f"{nombre.capitalize()}: ${precio}")
```

El programa comienza con un diccionario llamado productos, que contiene algunos artículos junto con sus precios. Utilizamos un bucle **for** para mostrar todos los productos disponibles al inicio del programa.

Luego, entramos en un bucle **while** donde le pedimos al usuario que ingrese el nombre del producto que quiere eliminar. La entrada se convierte a minúsculas con **.lower()** para evitar problemas de mayúsculas o minúsculas. Si el usuario presiona Enter sin escribir nada, el bucle termina gracias a la sentencia **break**.

Dentro del bucle, verificamos si el producto ingresado existe en el diccionario. Si existe, lo eliminamos usando el método **pop()**. Si no existe, se muestra un mensaje indicando que el producto no fue encontrado. Después de cada eliminación, mostramos el estado actual del diccionario para que el usuario vea los productos restantes.

Al finalizar el bucle, mostramos el estado final del inventario con todos los productos que quedan en el diccionario.

Esta línea: `if eliminar in productos:` comprueba si la clave almacenada en la variable **eliminar** existe dentro del diccionario **productos**. La palabra clave **in** se usa para verificar si una clave específica está presente en el diccionario. Si la clave se encuentra, la condición se evalúa como **True** y se ejecuta el bloque de código que elimina ese elemento del diccionario. Si la clave no está presente, la condición se evalúa como **False** y el programa pasa a ejecutar el bloque correspondiente al caso en que el producto no existe.

Ruta de avance

Llegaste a un punto muy importante en tu aprendizaje dentro de **TalentoLab**. A lo largo de las clases anteriores, adquiriste conocimientos fundamentales que te permiten gestionar datos de manera más eficiente y estructurada. En esta clase, aprendiste a utilizar **diccionarios**, una herramienta poderosa para asociar claves con valores, lo que facilita el almacenamiento y la recuperación de información sin tener que depender de índices numéricos.

Con lo que ya sabés, podés empezar a combinar **listas** y **diccionarios** para manejar conjuntos más complejos de datos. Por ejemplo, podrías almacenar una lista de diccionarios donde cada diccionario contenga información detallada de un producto, como su **nombre**, **precio** y **cantidad disponible**. Esta estructura te permitirá llevar un registro más organizado y fácilmente accesible de los datos que necesites en tu **Trabajo Final Integrador (TFI)**.

Hasta ahora, tus herramientas incluyen:

1. **Listas** para almacenar múltiples elementos ordenados y mutables.
2. **Tuplas** para almacenar elementos inmutables donde el orden es importante y no necesitas modificar los datos.
3. **Diccionarios** para asociar claves con valores, facilitando búsquedas rápidas y organizadas.
4. **Bucles while y for** para recorrer y manipular listas y diccionarios de manera dinámica.
5. **Condicionales** para validar y procesar datos según distintos criterios.

Para tu **TFI**, podrías pensar en una aplicación que permita registrar clientes, productos o cualquier otro conjunto de datos relevante. Algunas tareas que podrías implementar con lo que aprendiste hasta ahora son:

- **Agregar registros** a una lista de diccionarios, donde cada diccionario represente un cliente o un producto con información como **nombre**, **código**, y **precio o datos de contacto**.

- **Validar datos** ingresados por el usuario, como asegurarte de que los campos no estén vacíos o que los valores sean coherentes (por ejemplo, precios que no sean negativos).
- **Recorrer y mostrar registros** usando bucles **for** para listar todos los clientes o productos almacenados.
- **Actualizar o eliminar registros** existentes en la lista de diccionarios, asegurándote de que el programa sea dinámico y permita gestionar la información sin reiniciar el proceso.

Por ejemplo, podrías crear un programa que permita registrar una lista de productos con sus precios, mostrar el inventario actual y permitir eliminar un producto si ya no está disponible. Esta es una aplicación realista y práctica que te prepara para el desafío final.

Consejo de Luis: Recordá siempre pensar en cómo organizar tus datos para que sean fáciles de acceder y modificar. Los diccionarios son ideales cuando necesitás buscar elementos específicos sin recorrer toda la lista, mientras que las listas son útiles para manejar conjuntos ordenados de datos.

Cada clase te acerca más al desarrollo de una aplicación completa. ¡Seguís por el camino correcto y estás construyendo una base sólida para el **Trabajo Final Integrador**! 🚀

Ejercicio práctico.

Has aprendido y conocido todo sobre los diccionarios y su enorme importancia en el mundo de la programación. Mariana te escribió una nota donde te explica claramente la tarea que debe resolver el programa que deberás escribir el día de hoy:

¡Hola!

En **TalentoLab** nos han pedido desarrollar una aplicación que registre productos y sus precios utilizando diccionarios. Necesitamos que tu programa cumpla con estas instrucciones:

- Crear un diccionario llamado **productos** donde las claves sean los nombres de los productos y los valores sean sus precios.
- Permitir agregar productos y sus precios hasta que se decida finalizar.
- Mostrar el contenido del diccionario después de cada operación.

Consejo de Luis: Recordá que los diccionarios te permiten acceder a los datos rápidamente usando claves y que podés modificar sus valores sin problemas. Aprovechá esto para hacer que tu programa sea eficiente y fácil de usar.

Materiales y Recursos Adicionales:

Artículos:
El libro de Python: Diccionarios
Platzi: Cómo agregar, modificar y eliminar elementos de listas y diccionarios
Medium: Dominando los Diccionarios en Python

Videos:
La Geekipedia de Ernesto: Diccionarios en Python
Píldoras Informáticas: ¿Qué son los diccionarios?

Preguntas para Reflexionar:

1. ¿En qué situaciones es más útil usar un diccionario en lugar de una lista o una tupla? Reflexioná sobre cómo los diccionarios facilitan el acceso a los datos cuando necesitás buscar elementos específicos a partir de una clave.
2. ¿Cómo te ayuda la estructura de clave-valor de los diccionarios a organizar y manejar información de manera más eficiente? Pensá en el tipo de problemas que podrías resolver más rápidamente con esta estructura en comparación con otras.
3. ¿Qué ventajas ofrece el uso de diccionarios para registrar información en tu Trabajo Final Integrador (TFI)? Considerá cómo podrías aplicar los diccionarios para gestionar datos de clientes, productos u otra información relevante para tu proyecto.

Próximos pasos:

Llegaste al final de esta clase con una herramienta poderosa en tu haber: los **diccionarios**. Ahora sabés cómo almacenar, acceder y manipular datos utilizando claves y valores, lo que te permite gestionar información de una manera más eficiente y estructurada. Con estas habilidades, estás mucho más cerca de construir una aplicación completa para el **Trabajo Final Integrador (TFI)**.

En la próxima clase, vas a dar un paso fundamental en tu camino como especialista en desarrollo de software: aprenderás a crear **funciones definidas por el usuario o usuaria**. Las funciones te permitirán **organizar y modularizar tu código**, dividiéndolo en bloques

más manejables y reutilizables. Esto hará que tus programas sean más claros, eficientes y fáciles de mantener.

Descubrirás cómo definir funciones con y sin **parámetros** y entenderás el concepto de **alcance de variables**, diferenciando entre variables locales y globales. Esta nueva habilidad te ayudará a evitar la repetición de código y a estructurar tus proyectos de manera más profesional.

¡Estás avanzando a pasos firmes y cada vez más cerca de dominar el desarrollo en Python! 🚀

Buenos Aires aprende

Buenos Aires Ciudad