Diccionarios y otras colecciones

En este documento vamos a analizar otras estructuras de datos de Python que, si bien no son las más habituales, nos van a permitir organizar la información de cierta forma para poder acceder a ella.

1. Tuplas

Las tuplas son un tipo especial de datos en Python, que se parecen algo a las estructuras o *structs* de C o C#. Permiten definir un conjunto de datos heterogéneos, pero esos datos, una vez se definen, son **inmutables**, es decir, no podemos modificar su valor. Cada dato dentro de la tupla tiene un índice (empezando por cero), para poder acceder a cada dato.

Para definir una tupla, especificamos sus datos (entre paréntesis si queremos, aunque no es obligatorio):

```
nombres = ("John", "Helen", "Mary")
nombres2 = "Susan", "Adam", "Peter"
```

Como decimos, podemos especificar también diferentes datos en cada posición. Por ejemplo, para almacenar información personal de una persona:

```
datosPersonales = ("John Doe", 36, "611223344")
```

Después, podemos acceder a cada elemento con un índice entre corchetes, empezando por el 0:

```
print(datosPersonales[0])   # John Doe
```

Las tuplas ofrecen mucha flexibilidad a la hora de acceder a sus elementos, o asignarlos a variables separadas. Observemos este ejemplo:

```
datos = 20, "Nacho", 55.4
print(datos)                # (20, "Nacho", 55.4)
elem1, elem2, elem3 = datos
print(elem2)                # Nacho
elem2 = "May"
print(elem2)                # May
datos[1] = "May"            # Error: tupla inmutable
```

Como podemos ver, podemos sacar datos de una tupla de forma individual, y luego modificar esas variables. O bien tratar la tupla como un todo (variable `datos` anterior), en cuyo caso los valores almacenados son inmutables.

> **Ejercicio 1**:
>
> Crea un programa llamado **Tuplas.py** donde le pidas al usuario que rellene su dirección postal, que estará formada por su nombre de calle (texto), número de puerta (entero) y número de piso (entero). Almacena los datos en una tupla y luego muestra por pantalla el resultado (campo a campo).

2. Diccionarios

Los diccionarios, también denominados mapas o tablas *hash* en muchos lenguajes de programación, permiten almacenar información en forma de pares *clave-valor*, donde cada valor almacenado en la colección tiene asociada una clave, y accedemos a dicho valor a través de su clave.

Para definir un diccionario inicialmente vacío, se utiliza una pareja de llaves:

```
datos = {}
```

Después, para acceder a cada posición del diccionario y darle un valor o consultar el valor que tiene, debemos hacerlo siempre por su clave. Por ejemplo, imaginemos que estamos gestionando un diccionario con un catálogo de productos de una tienda. La clave para identificar cada producto será su código (alfanumérico, por ejemplo), y el valor asociado serán los datos del producto, que podemos representar como una tupla con su código, título y precio, por ejemplo. De este modo podríamos crear e inicializar el diccionario:

```
productos = {}
productos['111A'] = ('111A', 'Monitor LG 22 pulgadas', 99.95)
productos['222B'] = ('222B', 'Disco duro 512GB SSD', 109.45)
productos['333C'] = ('333C', 'Ratón bluetooth', 19.35)
```

Para acceder a un elemento en concreto, siempre tendrá que ser con su clave. Podemos utilizar el operador `in` para verificar previamente si existe la clave en el diccionario:

```
if '111A' in productos:
    print(productos['111A'][1]) # Monitor LG 22 pulgadas
```

También podemos recorrer todo el diccionario. En este caso podemos usar algunas alternativas:

```
# Recorrer todas las claves y con ellas sacar los valores
for clave in productos:
    print(productos[clave][1])  # Títulos de los productos

# Recorrer el diccionario sacando clave y valor
for clave, valor in productos.items():
    print(valor[1])             # Títulos de los productos
```

Existen otras operaciones habituales sobre diccionarios, como por ejemplo:

- La operación `pop` elimina la clave que indiquemos del diccionario, junto con su valor asociado.

```
productos.pop('111A')
```

- Las operaciones `len` y `clear` sirven, respectivamente, para obtener el número de elementos del diccionario, o para borrarlos todos.

```
print("El diccionario tiene", len(productos), "productos")
productos.clear()
```

> **Ejercicio 2**:
>
> Crea un programa llamado **DiccionarioTuplas.py** donde le pidas al usuario que rellene direcciones de 4 usuarios, identificados por su clave que será el DNI. Así, para cada usuario rellenará dicho DNI, y luego los datos de la dirección como en el ejercicio anterior (nombre de calle, número de puerta y número de piso). Almacenará los datos en un diccionario (asociando cada DNI con su dirección) y luego le pedirá al usuario que escriba un DNI y mostrará los datos de su dirección, o el mensaje "El DNI no se encuentra almacenado" si no existe dicha clave.
>
> **AYUDA:** vídeo con la solución del ejercicio

3. Conjuntos

Los conjuntos son un tipo de colección que no admite duplicados. Se pueden crear directamente pasando los elementos entre llaves, y luego con las operaciones `add` y `remove` añadimos o quitamos elementos al conjunto, respectivamente. También podemos crear conjuntos a partir de listas usando la función `set`, que se encarga automáticamente de eliminar duplicados.

Conviene tener presente que en los conjuntos no existe un orden ni una secuencia numerada de elementos, como sí ocurre en las listas. Veamos un ejemplo:

```
lista = ["Uno", "Dos", "Dos", "Tres"]
conjunto = set(lista)
print(conjunto)
# {"Dos", "Uno", "Tres"}
conjunto.add("Uno")
conjunto.add("Cuatro")
print(conjunto)
# {"Dos", "Uno", "Tres", "Cuatro"}
conjunto.remove("Tres")
print(conjunto)
# {"Dos", "Uno", "Cuatro"}
```