educación IT
CODE YOUR FUTURE

Introducción a Python

Módulo 3

Colecciones

Colecciones en Python

Una colección permite **agrupar** varios objetos bajo un mismo nombre. Por ejemplo, necesitamos almacenar en nuestro programa los nombres de los alumnos de un curso de programación. Será más conveniente ubicarlos a todos dentro de una misma colección de nombre alumnos, en lugar de crear los objetos alumno1, alumno2, etc. Como estuvimos haciendo con las listas.

Las listas forman parte de lo que llamamos *"colecciones"* de Python. Las colecciones nos permiten acceder **de forma rápida y organizada** a los datos.

Colecciones en Python

- Listas (ya conocidas por nosotros).
- Tuplas.
- Diccionarios.

Para tener en cuenta...

Además de las anteriores, Python incluye un tipo de colección llamado **`conjunto(set)`**. Estos "conjuntos" son una colección no ordenada y sin elementos repetidos.

Los usos básicos de éstos incluyen verificación de pertenencia y eliminación de entradas duplicadas. Pero estos "conjuntos" no son muy empleados en el día a día como las listas, tuplas y diccionarios.

Por eso a continuación abordaremos a las tuplas y luego a los diccionarios.

Tuplas

En Python, una **tupla** es un **conjunto ordenado e inmutable** de elementos de este o de diferente tipo. Se representan escribiendo los elementos entre paréntesis y separados por comas. Se accede por el índice, al igual que sucedía en las listas.

A grandes rasgos podríamos pensar las tuplas como una **lista constante,** que no cambia. Las tuplas no permiten asignación de ítems, y tampoco tienen métodos. Más adelante, en el curso siguiente *(Python Programming)*, vamos a encontrarles un uso y un significado.

Por ahora hay que conocer que existen y que son una colección de este lenguaje.

```
>>> datos = (10,"Hola",5.5,True)

>>> type(datos)
<class 'tuple'>

>>> datos[1]
'Hola'

>>> datos[3]
True
```

Diccionarios

Los diccionarios son **colecciones**, al igual que las listas, pero sus *elementos* (o *valores*) no están ordenados ni asociados a un índice, sino a una clave. Entonces, decimos que los miembros de un diccionario **son pares de *clave-valor***.

Para crear un diccionario vamos a emplear llaves y separar sus miembros por comas, que a su vez serán pares de una clave y un valor separados por dos puntos. Por ejemplo:

```
alumno1 = {"nombre": "Pablo", "cursos": 3}
```

Este código crea un diccionario de nombre `alumno1` cuyas claves son `"nombre"` y `"cursos"`, y sus valores, respectivamente, `"Pablo"` y `3`.

Los **valores** de un diccionario pueden ser de **cualquier tipo de dato**, incluidos otros diccionarios y listas.

En cambio, las **claves tienen ciertas restricciones** (además que no pueden repetirse) que no nos interesan por el momento; no obstante, por lo general serán cadenas (tal como en este ejemplo) o bien números enteros.

Para luego acceder a alguno de sus valores, simplemente indicamos la clave entre corchetes:

```
print(alumno1["nombre"])
print(alumno1["cursos"])
```

Del mismo modo podemos **cambiar** los valores:

```
alumno1["cursos"] = 4
```

O bien **crear nuevos** pares clave-valor:

```
alumno1["curso_actual"] = "Python"
```

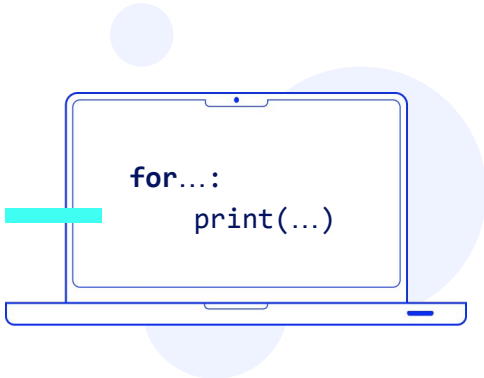
Usando un bucle **for** sobre un diccionario tenemos acceso a cada una de sus claves.

Por ejemplo:

```
for clave in alumno1:
    print(clave)
```

Como el **for** recorre las claves del diccionario, el código imprime lo siguiente:

```
nombre
cursos
curso_actual
```



Si luego de cada clave queremos imprimir su valor, podemos hacer, según lo que aprendimos recién, esto otro:

```
for clave in alumno1:
    print(clave)
    print(alumno1[clave])
```

Los diccionarios nos permiten **crear estructuras de datos** más complejas, por ejemplo, para almacenar la información de un alumno, algo que hubiese sido más engorroso de lograr usando los tipos de dato básicos o las listas.

educación IT
CODE YOUR FUTURE

¡Sigamos trabajando!