ColeCCiones

En el capítulo anterior vimos algunos tipos básicos, como los números, las cadenas de texto y los booleanos. En esta lección veremos algunos tipos de colecciones de datos: listas, tuplas y diccionarios.

Listas

La lista es un tipo de colección ordenada. Sería equivalente a lo que en otros lenguajes se conoce por arrays, o vectores.

Las listas pueden contener cualquier tipo de dato: números, cadenas, booleanos, ... y también listas.

Crear una lista es tan sencillo como indicar entre corchetes, y separados por comas, los valores que queremos incluir en la lista:

l = [22, True, “una lista”, [1, 2]]

Podemos acceder a cada uno de los elementos de la lista escribiendo el nombre de la lista e indicando el índice del elemento entre corchetes. Ten en cuenta sin embargo que el índice del primer elemento de la lista es 0, y no 1:

l = [11, False]

mi\_var = l[0] # mi\_var vale 11

Si queremos acceder a un elemento de una lista incluida dentro de otra lista tendremos que utilizar dos veces este operador, primero para indicar a qué posición de la lista exterior queremos acceder, y el segundo para seleccionar el elemento de la lista interior:

l = [“una lista”, [1, 2]]

mi\_var = l[1][0] # mi\_var vale 1

También podemos utilizar este operador para modificar un elemento de la lista si lo colocamos en la parte izquierda de una asignación:

l = [22, True]

l[0] = 99 # Con esto l valdrá [99, True]

El uso de los corchetes para acceder y modificar los elementos de una lista es común en muchos lenguajes, pero Python nos depara varias sorpresas muy agradables.

Una curiosidad sobre el operador [] de Python es que podemos utilizar también números negativos. Si se utiliza un número negativo como índice, esto se traduce en que el índice empieza a contar desde el final, hacia la izquierda; es decir, con [-1] accederíamos al último elemento de la lista, con [-2] al penúltimo, con [-3], al antepenúltimo, y así sucesivamente.

Otra cosa inusual es lo que en Python se conoce como *slicing* o particionado, y que consiste en ampliar este mecanismo para permitir seleccionar porciones de la lista. Si en lugar de un número escribimos dos números inicio y fin separados por dos puntos (inicio:fin) Python interpretará que queremos una lista que vaya desde la posición inicio a la posición fin, sin incluir este último. Si escribimos tres números (inicio:fin:salto) en lugar de dos, el tercero se utiliza para determinar cada cuantas posiciones añadir un elemento a la lista.

l = [99, True, “una lista”, [1, 2]]

mi\_var = l[0:2] # mi\_var vale [99, True]

mi\_var = l[0:4:2] # mi\_var vale [99, “una lista”]

Los números negativos también se pueden utilizar en un slicing, con el mismo comportamiento que se comentó anteriormente.

Hay que mencionar así mismo que no es necesario indicar el principio y el final del slicing, sino que, si estos se omiten, se usarán por defecto las posiciones de inicio y fin de la lista, respectivamente:

l = [99, True, “una lista”]

mi\_var = l[1:] # mi\_var vale [True, “una lista”]

mi\_var = l[:2] # mi\_var vale [99, True]

mi\_var = l[:] # mi\_var vale [99, True, “una lista”]

mi\_var = l[::2] # mi\_var vale [99, “una lista”]

También podemos utilizar este mecanismo para modificar la lista:

l = [99, True, “una lista”, [1, 2]]

l[0:2] = [0, 1] # l vale [0, 1, “una lista”, [1, 2]]

pudiendo incluso modificar el tamaño de la lista si la lista de la parte derecha de la asignación tiene un tamaño menor o mayor que el de la selección de la parte izquierda de la asignación:

l[0:2] = [False] # l vale [False, “una lista”, [1, 2]]

En todo caso las listas ofrecen mecanismos más cómodos para ser modificadas a través de las funciones de la clase correspondiente, aunque no veremos estos mecanismos hasta más adelante, después de explicar lo que son las clases, los objetos y las funciones.

Tuplas

Todo lo que hemos explicado sobre las listas se aplica también a las tuplas, a excepción de la forma de definirla, para lo que se utilizan paréntesis en lugar de corchetes.

t = (1, 2, True, “python”)

En realidad el constructor de la tupla es la coma, no el paréntesis, pero el intérprete muestra los paréntesis, y nosotros deberíamos utilizarlos, por claridad.

>>> t = 1, 2, 3

>>> type(t)

type “tuple”

Además hay que tener en cuenta que es necesario añadir una coma para tuplas de un solo elemento, para diferenciarlo de un elemento entre paréntesis.

>>> t = (1)

>>> type(t)

type “int”

>>> t = (1,)

>>> type(t)

type “tuple”

Para referirnos a elementos de una tupla, como en una lista, se usa el operador []:

mi\_var = t[0] # mi\_var es 1

mi\_var = t[0:2] # mi\_var es (1, 2)

Podemos utilizar el operador [] debido a que las tuplas, al igual que las listas, forman parte de un tipo de objetos llamados secuencias. Permitirme un pequeño inciso para indicaros que las cadenas de texto también son secuencias, por lo que no os extrañará que podamos hacer cosas como estas:

c = “hola mundo”

c[0] # h

c[5:] # mundo

c[::3] # hauo

Volviendo al tema de las tuplas, su diferencia con las listas estriba en que las tuplas no poseen estos mecanismos de modificación a través de funciones tan útiles de los que hablábamos al final de la anterior sección.

Además son inmutables, es decir, sus valores no se pueden modificar una vez creada; y tienen un tamaño fijo.

A cambio de estas limitaciones las tuplas son más “ligeras” que las listas, por lo que si el uso que le vamos a dar a una colección es muy básico, puedes utilizar tuplas en lugar de listas y ahorrar memoria.

Diccionarios

Los diccionarios, también llamados matrices asociativas, deben su nombre a que son colecciones que relacionan una clave y un valor. Por ejemplo, veamos un diccionario de películas y directores:

d = {“Love Actually “: “Richard Curtis”, “Kill Bill”: “Tarantino”,

“Amélie”: “Jean-Pierre Jeunet”}

El primer valor se trata de la clave y el segundo del valor asociado a la clave. Como clave podemos utilizar cualquier valor inmutable: podríamos usar números, cadenas, booleanos, tuplas, ... pero no listas o diccionarios, dado que son mutables. Esto es así porque los diccionarios se implementan como tablas hash, y a la hora de introducir un nuevo par clave-valor en el diccionario se calcula el hash de la clave para después poder encontrar la entrada correspondiente rápidamente. Si se modificara el objeto clave después de haber sido introducido en el diccionario, evidentemente, su hash también cambiaría y no podría ser encontrado.

La diferencia principal entre los diccionarios y las listas o las tuplas es que a los valores almacenados en un diccionario se les accede no por su índice, porque de hecho no tienen orden, sino por su clave, utilizando de nuevo el operador [].

d[“Love Actually “] # devuelve “Richard Curtis”

Al igual que en listas y tuplas también se puede utilizar este operador para reasignar valores.

d[“Kill Bill”] = “Quentin Tarantino”

Sin embargo en este caso no se puede utilizar slicing, entre otras cosas porque los diccionarios no son secuencias, si no *mappings* (mapeados, asociaciones).