Diccionarios

Otro tipo de dato útil incluído en Python es el *diccionario* (ver *Tipos integrados*). Los diccionarios se encuentran a veces en otros lenguajes como "memorias asociativas" o "arreglos asociativos". A diferencia de las secuencias, que se indexan mediante un rango numérico, los diccionarios se indexan con *claves*, que pueden ser cualquier tipo inmutable; las cadenas y números siempre pueden ser claves. Las tuplas pueden usarse como claves si solamente contienen cadenas, números o tuplas; si una tupla contiene cualquier objeto mutable directa o indirectamente, no puede usarse como clave. No podés usar listas como claves, ya que las listas pueden modificarse usando asignación por índice, asignación por sección, o métodos como **`append()`** y **`extend()`**.

Lo mejor es pensar en un diccionario como un conjunto no ordenado de pares *clave: valor*, con el requerimiento de que las claves sean únicas (dentro de un diccionario en particular). Un par de llaves crean un diccionario vacío: `{}`. Colocar una lista de pares clave:valor separados por comas entre las llaves añade pares clave:valor iniciales al diccionario; esta también es la forma en que los diccionarios se presentan en la salida.

Las operaciones principales sobre un diccionario son guardar un valor con una clave y extraer ese valor dada la clave. También es posible borrar un par clave:valor con `del`. Si usás una clave que ya está en uso para guardar un valor, el valor que estaba asociado con esa clave se pierde. Es un error extraer un valor usando una clave no existente.

Hacer `list(d.keys())` en un diccionario devuelve una lista de todas las claves usadas en el diccionario, en un orden arbitrario (si las querés ordenadas, usá en cambio `sorted(d.keys())`.[6] Para controlar si una clave está en el diccionario, usá el **`in`**.

Un pequeño ejemplo de uso de un diccionario:

```
>>> tel = {'jack': 4098, 'sape': 4139}
>>> tel['guido'] = 4127
>>> tel
```

```
{'sape': 4139, 'jack': 4098, 'guido': 4127}
>>> tel['jack']
4098
>>> del tel['sape']
>>> tel['irv'] = 4127
>>> tel
{'jack': 4098, 'irv': 4127, 'guido': 4127}
>>> list(tel.keys())
['irv', 'guido', 'jack']
>>> sorted(tel.keys())
['guido', 'irv', 'jack']
>>> 'guido' in tel
True
>>> 'jack' not in tel
False
```

El constructor `dict()` crea un diccionario directamente desde secuencias de pares clave-valor:

```
>>> dict([('sape', 4139), ('guido', 4127), ('jack', 4098)])
{'sape': 4139, 'jack': 4098, 'guido': 4127}
```

Además, las comprensiones de diccionarios se pueden usar para crear diccionarios desde expresiones arbitrarias de clave y valor:

```
>>> {x: x ** 2 for x in (2, 4, 6)}
{2: 4, 4: 16, 6: 36}
```

Cuando las claves son cadenas simples, a veces resulta más fácil especificar los pares usando argumentos por palabra clave:

```
>>> dict(sape=4139, guido=4127, jack=4098)
{'sape': 4139, 'jack': 4098, 'guido': 4127}
```

Técnicas de iteración

Cuando iteramos sobre diccionarios, se pueden obtener al mismo tiempo la clave y su valor correspondiente usando el método `items()`.

```
>>> caballeros = {'gallahad': 'el puro', 'robin': 'el valiente'}
>>> for k, v in caballeros.items():
...     print(k, v)
...
gallahad el puro
robin el valiente
```

Cuando se itera sobre una secuencia, se puede obtener el índice de posición junto a su valor correspondiente usando la función `enumerate()`.

```
>>> for i, v in enumerate(['ta', 'te', 'ti']):
...     print(i, v)
...
0 ta
1 te
2 ti
```

Para iterar sobre dos o más secuencias al mismo tiempo, los valores pueden emparejarse con la función`zip()`.

```
>>> preguntas = ['nombre', 'objetivo', 'color favorito']
>>> respuestas = ['lancelot', 'el santo grial', 'azul']
>>> for p, r in zip(preguntas, respuestas):
```

```
...     print('Cual es tu {0}?  {1}.'.format(p, r))
...
Cual es tu nombre?  lancelot.
Cual es tu objetivo?  el santo grial.
Cual es tu color favorito?  azul.
```

Para iterar sobre una secuencia en orden inverso, se especifica primero la secuencia al derecho y luego se llama a la función **`reversed()`**.

```
>>> for i in reversed(range(1, 10, 2)):
...     print(i)
...
9
7
5
3
1
```

Para iterar sobre una secuencia ordenada, se utiliza la función **`sorted()`** la cual devuelve una nueva lista ordenada dejando a la original intacta.

```
>>> canasta = ['manzana', 'naranja', 'manzana', 'pera', 'naranja', 'banana']
>>> for f in sorted(set(canasta)):
...     print(f)
...
banana
manzana
naranja
pera
```

A veces uno intenta cambiar una lista mientras la está iterando; sin embargo, a menudo es más simple y seguro crear una nueva lista:

```
>>> datos = [56.2, float('NaN'), 51.7, 55.3, 52.5, float('NaN'), 47.8]
>>> datos_filtrados = []
>>> for valor in datos:
...     if not math.isnan(valor):
...         datos_filtrados.append(valor)
...
>>> datos_filtrados
[56.2, 51.7, 55.3, 52.5, 47.8]
```