9 Diccionarios

Un **diccionario** es como una lista, pero más general. En una lista, las posiciones del índice deben ser enteros; En un diccionario, los índices pueden ser (casi) de cualquier tipo.

Puedes pensar en un diccionario como una asignación entre un conjunto de índices (que se denominan **claves**) y un conjunto de valores. Cada valor se asigna a una clave. La asociación de una clave y un valor se denomina **par clave-valor** o, a veces, un **elemento**.

Como ejemplo, construiremos un diccionario que asigne palabras del inglés al español, por lo que las claves y los valores son todas cadenas.

La función `dict` crea un nuevo diccionario sin elementos. Como `dict` es el nombre de una función incorporada, debes evitar usarla como nombre de variable.

```
named_variable = dict()
# También puedes crear un diccionario de la siguiente manera
named_variable_2 = {}
```

Las llaves, `{}`, representan un diccionario vacío. Para agregar elementos al diccionario, puede usar corchetes:

```
>>> eng2sp['one'] = 'uno'
```

Esta línea crea un elemento que se asigna desde la clave `'one'` al valor "uno". Si imprimimos el diccionario nuevamente, vemos un par clave-valor con dos puntos entre la clave y el valor:

```
>>> print(eng2sp)
{'one': 'uno'}
```

Este formato de salida es también un formato de entrada. Por ejemplo, puedes crear un diccionario nuevo con tres elementos. Pero si imprimes `eng2sp`, podría sorprenderse:

```
>>> eng2sp = {'one': 'uno', 'two': 'dos', 'three': 'tres'}
>>> print(eng2sp)
{'one': 'uno', 'three': 'tres', 'two': 'dos'}
```

El orden de los pares clave-valor no es el mismo. De hecho, si escribes el mismo ejemplo en tu ordenador, puedes obtener un resultado diferente. En general, el orden de los elementos en un diccionario es impredecible.

Pero eso no es un problema porque los elementos de un diccionario nunca se indexan con índices enteros. En su lugar, utiliza las claves para buscar los valores correspondientes:

```
>>> print(eng2sp['two'])
'dos'
```

La clave `'two'` siempre contiene el valor "dos", por lo que el orden de los elementos no importa.

Si la clave no está en el diccionario, obtendrás una excepción:

```
>>> print(eng2sp['four'])
KeyError: 'four'
```

La función `len` funciona en los diccionarios; devuelve el número de pares clave-valor:

```
>>> len(eng2sp)
3
```

El operador `in` funciona también en los diccionarios; le indica si algo aparece como una **clave** en el diccionario.

```
>>> 'one' in eng2sp
True
>>> 'uno' in eng2sp
False
```

Para ver si algo aparece como un valor en un diccionario, puede usar el método `values()`, que devuelve los valores como una lista, y luego usar el operador `in`:

```
>>> 'uno' in eng2sp.values()
True
```

El operador `in` usa diferentes algoritmos para listas y diccionarios. Para listas, utiliza un algoritmo de búsqueda lineal. A medida que la lista se alarga, el tiempo de búsqueda se alarga en proporción directa a la longitud de la lista. Para los diccionarios, Python usa un algoritmo llamado **tabla hash** que tiene una propiedad notable: el operador `in` toma aproximadamente la misma cantidad de

tiempo, sin importar cuántos elementos haya en un diccionario. No explicaré por qué las funciones hash son tan mágicas, pero puedes leer más sobre esto en wikipedia.org/wiki/Hash_table.

Ejercicio 1 [lista de palabras 2]

Escribe un programa que lea las palabras en `words.txt` y las almacene como claves en un diccionario. No importa cuáles sean los valores. Luego, puedes usar el operador `in` como una forma rápida de verificar si una cadena está en el diccionario.

Diccionario como conjunto de contadores

Supongamos que se le asigna una cadena y desea contar cuántas veces aparece cada letra. Hay varias formas de hacerlo:

1. Podrías crear 27 variables, una para cada letra del alfabeto. Luego, podrías atravesar la cadena y, para cada caracter, incrementar el contador correspondiente, probablemente utilizando un condicional encadenado.
2. Podrías crear una lista con 27 elementos. Luego, puedes convertir cada carácter en un número (utilizando la función incorporada `ord`), usar el número como un índice en la lista e incrementar el contador apropiado.
3. Podrías crear un diccionario con caracteres como claves y contadores como los valores correspondientes. La primera vez que veas un caracter, agregarás un elemento al diccionario. Después de eso, incrementarías el valor de un artículo existente.

Cada una de estas opciones realiza el mismo cálculo, pero cada una de ellas implementa ese cálculo de una manera diferente.

Una **implementación** es una forma de realizar un cálculo. Algunas implementaciones son mejores que otras. Por ejemplo, una ventaja de la implementación del diccionario es que no tenemos que saber con antelación qué letras aparecen en la cadena y solo tenemos que crear espacio para las letras que van apareciendo.

Así es cómo se vería el código:

Estamos efectivamente calculando un **histograma** , que es un término estadístico para un conjunto de contadores (o frecuencias).

```
word = 'brontosaurus'
d = {}
for c in word:
  if c not in d:
    d[c] = 1
  else:
    d[c] = d[c] + 1
print(d)
```

El bucle `for` atraviesa la cadena. Cada vez que recorre el bucle, si el caracter `c` no está en el diccionario, creamos un nuevo elemento con la clave `c` y el valor inicial 1 (ya que hemos visto esta letra una vez). Si `c` ya está en el diccionario, incrementamos `d[c]`.

Aquí está la salida del programa:

```
{'a': 1, 'b': 1, 'o': 2, 'n': 1, 's': 2, 'r': 2, 'u': 2, 't': 1}
```

El histograma indica que las letras `'a'` y `'b'` aparecen una vez; "O" aparece dos veces, y así sucesivamente.

Los diccionarios tienen un método llamado `get` que toma una clave y un valor predeterminado. Si la clave aparece en el diccionario, `get` devuelve el valor correspondiente; de lo contrario, devuelve el valor predeterminado. Por ejemplo:

```
>>> counts = { 'chuck' : 1 , 'annie' : 42, 'jan': 100}
>>> print(counts.get('jan', 0))
100
>>> print(counts.get('tim', 0))
0
```

Podemos usar `get` para escribir nuestro bucle de histograma de forma más concisa. Debido a que el método `get` maneja automáticamente el caso en el que una clave no esté en un diccionario, podemos reducir cuatro líneas a una y eliminar la instrucción `if`.

```
word = 'brontosaurus'
d = dict()
for c in word:
    d[c] = d.get(c,0) + 1
```

```
print(d)
```

El uso del método `get` para simplificar este bucle de conteo termina siendo un "idiom" muy usado en Python y lo usaremos muchas veces en el resto del libro. Así que debes tomarte un momento y comparar el bucle usando la instrucción `if` y el operador `in` con el bucle usando el método `get`. Hacen exactamente lo mismo, pero uno es más sucinto.

Diccionarios y archivos

Uno de los usos comunes de un diccionario es contar la aparición de palabras en un archivo con algún texto escrito. Comencemos con un archivo muy simple de palabras tomadas del texto de **Romeo y Julieta**.

Para el primer conjunto de ejemplos, usaremos una versión abreviada y simplificada del texto sin puntuación. Posteriormente trabajaremos con el texto de la escena con puntuación incluida.

```
But soft what light through yonder window breaks
It is the east and Juliet is the sun
Arise fair sun and kill the envious moon
Who is already sick and pale with grief
```

Escribiremos un programa en Python para leer las líneas del archivo, dividir cada línea en una lista de palabras y luego recorrer cada una de las palabras en la línea y contar cada palabra con un diccionario.

Verás que tenemos dos bucles `for`. El bucle externo está leyendo las líneas del archivo y el bucle interno está iterando a través de cada una de las palabras en esa línea en particular. Este es un ejemplo de un patrón llamado **bucles anidados** porque uno de los bucles es el bucle **externo** y el otro bucle es el bucle **interno**.

Debido a que el bucle interno ejecuta todas sus iteraciones cada vez que el bucle externo realiza una única iteración, pensamos que el bucle interno se repite "más rápidamente" y que el bucle externo se repite más lentamente.

La combinación de los dos bucles anidados asegura que contaremos cada palabra en cada línea del archivo de entrada.

https://trinket.io/embed/python3/17a4334148

Cuando ejecutamos el programa, vemos un volcado sin formato de todos los recuentos en orden hash. (El archivo `romeo.txt` está disponible en www.py4e.com/code3/romeo.txt)

```
python count1.py
Enter the file name: romeo.txt
{'and': 3, 'envious': 1, 'already': 1, 'fair': 1,
'is': 3, 'through': 1, 'pale': 1, 'yonder': 1,
'what': 1, 'sun': 2, 'Who': 1, 'But': 1, 'moon': 1,
'window': 1, 'sick': 1, 'east': 1, 'breaks': 1,
'grief': 1, 'with': 1, 'light': 1, 'It': 1, 'Arise': 1,
'kill': 1, 'the': 3, 'soft': 1, 'Juliet': 1}
```

Es un poco incómodo mirar en el diccionario para encontrar las palabras más comunes y sus conteos, por lo que necesitamos agregar un poco más de código de Python para obtener el resultado que será más útil.

Bucles y diccionarios

Si utilizas un diccionario como la secuencia en una instrucción `for`, atraviesa las claves del diccionario. Este bucle imprime cada clave y el valor correspondiente:

```
counts = { 'chuck' : 1 , 'annie' : 42, 'jan': 100}
for key in counts:
    print(key, counts[key])
```

Así es como se ve la salida:

```
jan 100
chuck 1
annie 42
```

Una vez más, las claves no están en ningún orden en particular.

Podemos usar este patrón para implementar los diversos modismos de bucle que hemos descrito anteriormente. Por ejemplo, si quisiéramos encontrar todas las entradas en un diccionario con un

valor superior a diez, podríamos escribir el siguiente código:

```
counts = { 'chuck' : 1 , 'annie' : 42, 'jan': 100}
for key in counts:
    if counts[key] > 10 :
        print(key, counts[key])
```

El bucle `for` recorre las **claves** del diccionario, por lo que debemos utilizar el operador de índice para recuperar el valor correspondiente para cada clave. Así es como se ve la salida:

```
jan 100
annie 42
```

Solo vemos las entradas con un valor superior a 10.

Si deseas imprimir las claves en orden alfabético, primero haz una lista de las claves en el diccionario usando el método `keys` disponible en los objetos de tipo diccionario, y luego ordena esa lista y recorre la lista ordenada, mirando cada clave e imprimiendo pares clave-valor ordenados de la siguiente manera:

```
counts = { 'chuck' : 1 , 'annie' : 42, 'jan': 100}
lst = list(counts.keys())
print(lst)
lst.sort()
for key in lst:
    print(key, counts[key])
```

Así es como se ve la salida:

```
['jan', 'chuck', 'annie']
annie 42
chuck 1
jan 100
```

Primero ve la lista desordenada de claves que obtenemos del método `keys`. Luego vemos los pares clave-valor en orden desde el bucle `for`.

Análisis de texto avanzado

En el ejemplo anterior usando el archivo `romeo.txt`, hicimos el archivo lo más simple posible eliminando toda puntuación a mano. El texto real tiene mucha puntuación, como se muestra a continuación.

```
But, soft! what light through yonder window breaks?
It is the east, and Juliet is the sun.
Arise, fair sun, and kill the envious moon,
Who is already sick and pale with grief,
```

Ya que la función `split` de Python busca espacios y trata las palabras como tokens separados por espacios, trataríamos las palabras "soft!" y "soft" como **diferentes** palabras y crearíamos una entrada de diccionario separada para cada palabra.

Además, dado que el archivo tiene mayúsculas, trataríamos "who" y "Who" como palabras diferentes con diferentes valores.

Podemos resolver ambos problemas utilizando los métodos de cadena `lower`, `punctuation` y `translate`. `translate` es el más sutil de los métodos. Aquí está la documentación para `translate`:

`line.translate(str.maketrans(fromstr, tostr, deletestr))`

Reemplaza los caracteres en `fromstr` con el carácter en la misma posición en `tostr` y borra todos los caracteres que estén en `deletestr`. `Fromstr` y `tostr` pueden ser cadenas vacías y el parámetro `deletestr` puede omitirse.

No especificaremos la 'tabla', pero usaremos el parámetro `deletechars` para eliminar toda la puntuación. Incluso dejaremos que Python nos diga la lista de caracteres que considera "puntuación":

```
>>> import string
>>> string.punctuation
'!"#$%&amp;\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

Los parámetros utilizados por `translate` eran diferentes en Python 2.0.

Realizamos las siguientes modificaciones a nuestro programa:

https://trinket.io/embed/python3/25874ef3fd

Parte de aprender el "Arte de Python" o "Thinking Pythonically" es darse cuenta de que Python a menudo tiene capacidades incorporadas para muchos problemas comunes de análisis de datos. Con el tiempo, verás suficiente código de ejemplo y leerás suficiente documentación para saber dónde buscar para ver si alguien ya ha escrito algo que facilita mucho tu trabajo.

La siguiente es una versión abreviada de la salida:

```
Enter the file name: romeo-full.txt
{'swearst': 1, 'all': 6, 'afeard': 1, 'leave': 2, 'these': 2,
'kinsmen': 2, 'what': 11, 'thinkst': 1, 'love': 24, 'cloak': 1,
a': 24, 'orchard': 2, 'light': 5, 'lovers': 2, 'romeo': 40,
'maiden': 1, 'whiteupturned': 1, 'juliet': 32, 'gentleman': 1,
'it': 22, 'leans': 1, 'canst': 1, 'having': 1, ...}
```

La salida aún es difícil de manejar y podemos usar Python para darnos exactamente lo que estamos buscando, pero para hacerlo, necesitamos aprender sobre las tuplas de Python. Retomaremos este ejemplo una vez que aprendamos sobre las tuplas.

Depurando

A medida que trabajas con conjuntos de datos más grandes, puede volverse difícil manejar la depuración imprimiendo y verificando los datos a mano. Aquí hay algunas sugerencias para depurar grandes conjuntos de datos:

Reduce la entrada

Si es posible, reduzce el tamaño del conjunto de datos. Por ejemplo, si el programa lee un archivo de texto, comienza solo con las primeras 10 líneas o con el ejemplo más pequeño que puedas encontrar. Puedes editar los archivos por sí mismos o (mejor) modificar el programa para que solo lea las primeras líneas `n`.

Si hay un error, puede reducir `n` al valor más pequeño que manifiesta el error, y luego aumentarlo gradualmente a medida que encuentre y corrija los errores.

Comprueba sumarios y tipos

En lugar de imprimir y verificar todo el conjunto de datos, considera imprimir resúmenes de los datos: por ejemplo, la cantidad de elementos en un diccionario o el total de una lista de números.

Una causa común de errores en tiempo de ejecución es un valor que no es el tipo correcto. Para depurar este tipo de error, a menudo es suficiente imprimir el tipo de un valor.

Otro tipo de verificación compara los resultados de dos cálculos diferentes para ver si son consistentes. Esto se llama "verificación de consistencia".

Imprime la salida con formato

Dar formato a la salida de depuración puede facilitar la detección un error.

Nuevamente, el tiempo que dedicas a construir andamios puede reducir el tiempo que dedicas a la depuración.

Ejercicios

Ejercicio 2: Escribe un programa que categorice cada mensaje de correo según el día de la semana en que se realizó la confirmación. Para hacer esto, busca líneas que comiencen con "From", luego busca la tercera palabra y manten un conteo de cada uno de los días de la semana. Al final del programa, imprime el contenido de tu diccionario (el orden no importa).

Línea de muestra:

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
```

Ejecución de la muestra:

```
python dow.py
Enter a file name: mbox-short.txt
{'Fri': 20, 'Thu': 6, 'Sat': 1}
```

Ejercicio 3: Escribe un programa que lea un registro de correo, cree un histograma usando un diccionario para contar cuántos mensajes has recibido de cada dirección de correo electrónico y luego imprima el diccionario.

```
Enter file name: mbox-short.txt
{'gopal.ramasammycook@gmail.com': 1, 'louis@media.berkeley.edu': 3,
```

```
'cwen@iupui.edu': 5, 'antranig@caret.cam.ac.uk': 1,
'rjlowe@iupui.edu': 2, 'gsilver@umich.edu': 3,
'david.horwitz@uct.ac.za': 4, 'wagnermr@iupui.edu': 1,
'zqian@umich.edu': 4, 'stephen.marquard@uct.ac.za': 2,
'ray@media.berkeley.edu': 1}
```

Ejercicio4: Agrega código al programa anterior para averiguar quién tiene más mensajes en el archivo.

Una vez que hayas leído todos los datos y se haya creado el diccionario, examina el diccionario utilizando un bucle máximo para encontrar quién tiene más mensajes e imprime cuántos mensajes tiene la persona.

```
Enter a file name: mbox-short.txt
cwen@iupui.edu 5

Enter a file name: mbox.txt
zqian@umich.edu 195
```

Ejercicio 5: Este programa registra el nombre de dominio (en lugar de la dirección) desde donde se envió el mensaje en lugar de a quién le llegó el correo (es decir, la dirección de correo electrónico completa). Al final del programa, imprime el contenido de tu diccionario.

```
python schoolcount.py
Enter a file name: mbox-short.txt
{'media.berkeley.edu': 4, 'uct.ac.za': 6, 'umich.edu': 7,
'gmail.com': 1, 'caret.cam.ac.uk': 1, 'iupui.edu': 8}
```

Revision #1
Created 2025-04-05 11:10:33 CEST by Javier Quintana
Updated 2025-04-05 11:11:24 CEST by Javier Quintana