Modulo 4: Diccionarios

¿Qué es un diccionario?

El **diccionario** es otro tipo de estructura de datos de Python. No **es una secuencia** (pero puede adaptarse fácilmente a un procesamiento secuencial) y además es **mutable**.

Para explicar lo que es un diccionario en Python, es importante comprender de manera literal lo que es un diccionario.

Un diccionario en Python funciona de la misma manera que **un diccionario bilingüe**. Por ejemplo, se tiene la palabra en español «gato» y se necesita su equivalente en francés. Lo que se haría es buscar en el diccionario para encontrar la palabra «gato». Eventualmente la encontrarás, y sabrás que la palabra equivalente en francés es «chat».

En el mundo de Python, la palabra que se esta buscando se denomina **clave(key)**. La palabra que se obtiene del diccionario es denominada **valor**.

Esto significa que un diccionario es un conjunto de pares de **claves y valores**. Nota:

- Cada clave debe de ser única. No es posible tener una clave duplicada.
- Una clave puede ser un tipo de dato de cualquier tipo: puede ser un número (entero o flotante), o incluso una cadena.
- Un diccionario no es una lista. Una lista contiene un conjunto de valores numerados, mientras que un diccionario almacena pares de valores.
- La función len() aplica también para los diccionarios, regresa la cantidad de pares (clave-valor) en el diccionario.
- Un diccionario es una herramienta de un solo sentido. Si fuese un diccionario español-francés, podríamos buscar en español para encontrar su contraparte en francés más no viceversa.

¿Cómo crear un diccionario?

Si deseas asignar algunos pares iniciales a un diccionario, utiliza la siguiente sintaxis:

```
dictionary = {"gato" : "chat", "perro" : "chien", "caballo" : "cheval"}
phone_numbers = {'jefe': 5551234567, 'Suzy': 22657854310}
empty_dictionary = {}

print(dictionary)
print(phone_numbers)
print(empty_dictionary)
```

En este primer ejemplo, el diccionario emplea claves y valores las cuales ambas son cadenas. En el segundo, las claves con cadenas pero los valores son enteros. El orden inverso (claves → números, valores → cadenas) también es posible, así como la combinación número a número.

La lista de todos los pares es **encerrada con llaves**, mientras que los pares son **separados por comas**, y **las claves y valores por dos puntos**.

El primer diccionario es muy simple, es un diccionario Español-Francés. El segundo es un directorio telefónico muy pequeño.

Los diccionarios vacíos son construidos por **un par vacío de llaves** - nada inusual.

El diccionario entero se puede imprimir con una invocación a la función print(). El fragmento de código puede producir la siguiente salida:

```
{'perro': 'chien', 'caballo': 'cheval', 'gato': 'chat'}
{'Suzy': 5557654321, 'jefe': 5551234567}
{}
```

¿Has notado que el orden de los pares impresos es diferente a la asignación inicial?, ¿Qué significa esto?

Primeramente, recordemos que **los diccionarios no son listas** - no guardan el orden de sus datos, el orden no tiene significado (a diferencia de los diccionarios reales). El orden en que un diccionario **almacena sus datos esta fuera de nuestro control**. Esto es normal. (*)

NOTA: En Python 3.6x los diccionarios se han convertido en colecciones ordenadas de manera predeterminada. Tu resultado puede variar dependiendo en la versión de Python que se este utilizando.

¿Cómo utilizar un diccionario?

Si deseas obtener cualquiera de los valores, debes de proporcionar una clave válida:

```
print(dictionary['gato'])
print(phone_numbers['Suzy'])
```

El obtener el valor de un diccionario es semejante a la indexación, gracias a los corchetes alrededor del valor de la clave.

Nota:

- Si una clave es una cadena, se tiene que especificar como una cadena.
- Las claves son sensibles a las mayúsculas y minúsculas: 'Suzy' sería diferente a 'suzy'.

El fragmento de código da las siguientes salidas:

```
chat
5557654321
```

Ahora algo muy importante: **No se puede utilizar una clave que no exista**. Hacer algo como lo siguiente:

```
print(phone_numbers['presidente'])
```

Provocará un error de ejecución. Inténtalo.

Afortunadamente, existe una manera simple de evitar dicha situación. El operador **in**, junto con su acompañante, **not in**, pueden salvarnos de esta situación.

El siguiente código busca de manera segura palabras en francés:

```
dictionary = {"gato" : "chat", "perro" : "chien", "caballo" : "cheval"}
words = ['gato', 'león', 'caballo']

for word in words:
    if word in dictionary:
```

```
        print(word, "->", dictionary[word])
    else:
        print(word, "no está en el diccionario")
```

La salida del código es la siguiente:

```
gato -> chat
león no está en el diccionario
caballo -> cheval
```

Nota: Cuando escribes una expresión grande o larga, puede ser una buena idea mantenerla alineada verticalmente. Así es como puede hacer que el código sea más legible y más amigable para el programador, por ejemplo:

```
# Ejemplo 1:
dictionary = {
              "gato": "chat",
              "perro": "chien",
              "caballo": "cheval"
              }

# Ejemplo 2:
phone_numbers = {'jefe': 5551234567,
                 'Suzy': 22657854310
                 }
```

Este tipo de formato se llama **sangría francesa**.

¿Cómo utilizar un diccionario? El método keys()

¿Pueden los diccionarios ser **examinados** utilizando el bucle for, como las listas o tuplas?

No y si.

No, porque un diccionario **no es un tipo de dato secuencial** - el bucle `for` no es útil aquí.

Si, porque hay herramientas simples y muy efectivas que pueden **adaptar cualquier diccionario a los requerimientos del bucle** `for` (en otras palabras, se construye un enlace intermedio entre el diccionario y una entidad secuencial temporal).

El primero de ellos es un método denominado **keys()**, el cual es parte de todo diccionario. El método retorna o regresa una lista de todas las claves dentro del diccionario. Al tener una lista de claves se puede acceder a todo el diccionario de una manera fácil y útil.

A continuación se muestra un ejemplo:

```
dictionary = {"gato" : "chat", "perro" : "chien", "caballo" : "cheval"}

for key in dictionary.keys():
    print(key, "->", dictionary[key])
```

El código produce la siguiente salida:

```
gato -> chat
```

```
perro -> chien
caballo -> cheval
```

La función sorted()

¿Deseas que la salida este ordenada? Solo hay que agregar al bucle for lo siguiente:

```
for key in sorted(dictionary.keys()):
```

La función **sorted()** hará su mejor esfuerzo y la salida será la siguiente:

```
caballo -> cheval
gato -> chat
perro -> chien
```

¿Cómo utilizar un diccionario? Los métodos item() y values()

Otra manera de hacerlo es utilizar el método **items()**. Este método **regresa una lista de tuplas** (este es el primer ejemplo en el que las tuplas son mas que un ejemplo de si mismas) **donde cada tupla es un par de cada clave con su valor**.

Así es como funciona:

```
dictionary = {"gato" : "chat", "perro" : "chien", "caballo" : "cheval"}

for index, value in dictionary.items():
    print(index, "->", value)
```

Nota la manera en que la tupla ha sido utilizada como una variable del bucle for.

El ejemplo imprime lo siguiente:

```
gato -> chat
perro -> chien
caballo -> cheval
```

También existe un método denominado **values()**, funciona de manera muy similar al de **keys()**, pero **regresa una lista de valores**.

Este es un ejemplo sencillo:

```
dictionary = {"gato" : "chat", "perro" : "chien", "caballo" : "cheval"}

for value in dictionary.values():
    print(value)
```

Como el diccionario no es capaz de automáticamente encontrar la clave de un valor dado, el rol de este método es algo limitado.

Esta es la salida esperada:

```
chat
```

```
chien
cheval
```

¿Cómo utilizar un diccionario? Modificar, agregar y eliminar valores

El asignar un nuevo valor a una clave existente es sencillo, debido a que los diccionarios son completamente **mutables**, no existen obstáculos para modificarlos.

Se va a reemplazar el valor «chat» por «minou», lo cual no es muy adecuado, pero funcionará con nuestro ejemplo.

```
dictionary = {'gato': 'minou', 'perro': 'chien', 'caballo': 'cheval'}

dictionary['gato'] = 'minou'
print(dictionary)
```

La salida es:

```
{'gato': 'minou', 'dog': 'chien', 'caballo': 'cheval'}
```

Agregando nuevas claves

El agregar una nueva clave con su valor a un diccionario es tan simple como cambiar un valor. Solo se tiene que asignar un valor a una nueva **clave que no haya existido antes**.

Nota: este es un comportamiento muy diferente comparado a las listas, las cuales no permiten asignar valores a índices no existentes.

A continuación se agrega un par nuevo al diccionario, un poco extraño pero válido:

```
dictionary = {"gato" : "chat", "perro" : "chien", "caballo" : "cheval"}

dictionary['cisne'] = 'cygne'
print(dictionary)
```

El ejemplo muestra como salida:

```
{'gato': 'chat', 'perro': 'chien', 'caballo': 'cheval', 'cisne': 'cygne'}
```

Note: También es posible insertar un elemento al diccionario utilizando el método **update()**, por ejemplo:

```
dictionary = {"gato" : "chat", "perro" : "chien", "caballo" : "cheval"}

dictionary.update({"pato": "canard"})
print(dictionary)
```

Eliminado una clave

¿Puedes deducir como eliminar una clave de un diccionario?

Nota: al eliminar la clave también se **removerá el valor asociado. Los valores no pueden existir sin sus claves**.

Esto se logra con la instrucción **del**.

A continuación un ejemplo:

```
dictionary = {"gato" : "chat", "perro" : "chien", "caballo" : "cheval"}

del dictionary['perro']
print(dictionary)
```

Nota: **el eliminar una clave no existente, provocará un error**.

El ejemplo da como salida:

```
{'gato': 'chat', 'caballo': 'cheval'}
```

EXTRA

Para eliminar el ultimo elemento de la lista, se puede emplear el método **popitem()**:

```
dictionary = {"gato" : "chat", "perro" : "chien", "caballo" : "cheval"}

dictionary.popitem()
print(dictionary)    # salida: {'gato': 'chat', 'perro': 'chien'}
```

En versiones anteriores de Python, por ejemplo, antes de la 3.6.7, el método **popitem()** elimina un elemento al azar del diccionario.

Las tuplas y los diccionarios pueden trabajar juntos

Se ha preparado un ejemplo sencillo, mostrando como las tuplas y los diccionarios pueden trabajar juntos.

Imaginemos el siguiente problema:

Necesitas un programa para calcular los promedios de tus alumnos. El programa pide el nombre del alumno seguido de su calificación. Los nombres son ingresados en cualquier orden. El ingresar un nombre vacío finaliza el ingreso de los datos (nota 1: ingresar una puntuación vacía generará la excepción ValueError, pero no te preocupes por eso ahora, verás cómo manejar tales casos cuando hablemos de excepciones en el segundo parte de la serie del curso). Una lista con todos los nombre y el promedio de cada alumno debe ser mostrada al final.

```
school_class = {}

while True:
    name = input("Ingresa el nombre del estudiante: ")
    if name == '':
        break
    score = int(input("Ingresa la calificación del estudiante (0-10): "))
    if score not in range(0, 11):
        break
```

```
    if name in school_class:
        school_class[name] += (score,)
    else:
        school_class[name] = (score,)
for name in sorted(school_class.keys()):
    adding = 0
    counter = 0
    for score in school_class[name]:
        adding += score
        counter += 1
    print(name, ":", adding / counter)
```

Ahora se analizará línea por línea:

- Línea 1: crea un diccionario vacío para ingresar los datos: el nombre del alumno es empleado como clave, mientras que todas las calificaciones asociadas son almacenadas en una tupla (la tupla puede ser el valor de un diccionario, esto no es un problema).
- Línea 3: se ingresa a un bucle «infinito» (no te preocupes, saldrémos de el en el momento indicado).
- Línea 4: se lee el nombre del alumno aquí.
- Línea 5-6: si el nombre es una cadena vacía (), salimos del bucle.
- Línea 8: se pide la calificación del estudiante (un valor entero en el rango del 1-10).
- Línea 9-10: si la puntuación ingresada no está dentro del rango de 0 a 10, se abandona el bucle.
- Línea 12-13: si el nombre del estudiante ya se encuentra en el diccionario, se alarga la tupla asociada con la nueva calificación (observa el operador +).
- Línea 14-15: si el estudiante es nuevo (desconocido para el diccionario), se crea una entrada nueva, su valor es una tupla de un solo elemento la cual contiene la calificación ingresada.
- Línea 17: se itera a través de los nombres ordenados de los estudiantes.
- Línea 18-19: inicializa los datos necesarios para calcular el promedio (sum y counter).
- Línea 20-22: se itera a través de la tupla, tomado todas las calificaciones subsecuentes y actualizando la suma junto con el contador.
- Línea 23: se calcula e imprime el promedio del alumno junto con su nombre.

Este es un ejemplo del programa:

```
Ingresa el nombre del estudiante: Bob
Ingresa la calificación del estudiante (0-10): 7
Ingresa el nombre del estudiante: Andy
Ingresa la calificación del estudiante (0-10): 3
Ingresa el nombre del estudiante: Bob
Ingresa la calificación del estudiante (0-10): 2
Ingresa el nombre del estudiante: Andy
Ingresa la calificación del estudiante (0-10): 10
Ingresa el nombre del estudiante: Andy
Ingresa la calificación del estudiante (0-10): 3
Ingresa el nombre del estudiante: Bob
Ingresa la calificación del estudiante (0-10): 9
Ingresa el nombre del estudiante:
Andy : 5.333333333333333
Bob : 6.0
```

Puntos Clave: Diccionarios

1. Los diccionarios son *colecciones indexadas de datos, mutables y desordenadas. (*En Python 3.6x los diccionarios están ordenados de manera predeterminada.

Cada diccionario es un par de clave : valor. Se puede crear empleado la siguiente sintaxis:

```
my_dictionary = {
    key1: value1,
    key2: value2,
    key3: value3,
    }
```

2. Si se desea acceder a un elemento del diccionario, se puede hacer haciendo referencia a su clave colocándola dentro de corchetes (Ejemplo 1) o utilizando el método `get()` (Ejemplo 2):

```
pol_esp_dictionary = {
    "kwiat": "flor",
    "woda": "agua",
    "gleba": "tierra"
    }

item_1 = pol_esp_dictionary["gleba"]    # Ejemplo 1.
print(item_1)    # salida: tierra

item_2 = pol_esp_dictionary.get("woda")    # Ejemplo 2.
print(item_2)    # salida: agua
```

3. Si se desea cambiar el valor asociado a una clave específica, se puede hacer haciendo referencia a la clave del elemento, a continuación se muestra un ejemplo:

```
pol_esp_dictionary = {
    "zamek" : "castillo",
    "woda"  : "agua",
    "gleba" : "tierra"
    }

pol_esp_dictionary["zamek"] = "cerradura"
item = pol_esp_dictionary["zamek"]
print(item)  # salida: cerradura
```

4. Para agregar o eliminar una clave (junto con su valor asociado), emplea la siguiente sintaxis:

```
phonebook = {}    # un diccionario vacío

phonebook["Adán"] = 3456783958    # crear/agregar un par clave-valor
print(phonebook)    # salida: {'Adán': 3456783958}

del phonebook["Adán"]
print(phonebook)    # salida: {}
```

Además, se puede insertar un elemento a un diccionario utilizando el método `update()`, y eliminar el ultimo elemento con el método `popitem()`, por ejemplo:

```
pol_esp_dictionary = {"kwiat": "flor"}

pol_esp_dictionary.update({"gleba": "tierra"})
print(pol_esp_dictionary)    # salida: {'kwiat': 'flor', 'gleba': 'tierra'}

pol_esp_dictionary.popitem()
```

```
print(pol_esp_dictionary)    # salida: {'kwiat': 'flor'}
```

5. Se puede emplear el bucle `for` para iterar a través del diccionario, por ejemplo:

```
pol_esp_dictionary = {
    "zamek": "castillo",
    "woda": "agua",
    "gleba": "tierra"
    }

for item in pol_esp_dictionary:
    print(item)

# salida: zamek
#         woda
#         gleba
```

6. Si deseas examinar los elementos (claves y valores) del diccionario, puedes emplear el método `items()`, por ejemplo:

```
pol_esp_dictionary = {
    "zamek" : "castillo",
    "woda"  : "agua",
    "gleba" : "tierra"
    }

for key, value in pol_esp_dictionary.items():
    print("Pol/Esp ->", key, ":", value)
```

7. Para comprobar si una clave existe en un diccionario, se puede emplear la palabra clave reservada `in`:

```
pol_esp_dictionary = {
    "zamek" : "castillo",
    "woda"  : "agua",
    "gleba" : "tierra"
    }

if "zamek" in pol_esp_dictionary:
    print("Si")
else:
    print("No")
```

8. Se puede emplear la palabra reservada `del` para eliminar un elemento, o un diccionario entero. Para eliminar todos los elementos de un diccionario se debe emplear el método `clear()`:

```
pol_esp_dictionary = {
    "zamek" : "castillo",
    "woda"  : "agua",
    "gleba" : "tierra"
    }

print(len(pol_esp_dictionary))    # salida: 3
del pol_esp_dictionary["zamek"]    # eliminar un elemento
print(len(pol_esp_dictionary))    # salida: 2

pol_esp_dictionary.clear()   # eliminar todos los elementos
```

```
print(len(pol_esp_dictionary))    # salida: 0

del pol_esp_dictionary    # elimina el diccionario
```

9. Para copiar un diccionario, emplea el método `copy()`:

```
pol_esp_dictionary = {
    "zamek" : "castillo",
    "woda"  : "agua",
    "gleba" : "tierra"
    }

copy_dictionary = pol_esp_dictionary.copy()
```

From:
https://matebcn.eu/ - **miguel angel torres egea**

Permanent link:
https://matebcn.eu/info:cursos:netacad:python:pe1m4:diccionarios

Last update: **08/09/2022 12:03**