PRO-UT2-7.Diccionarios en Python

Los diccionarios en Python son una estructura de datos que permite almacenar su contenido en forma de clave y valor.

Crear diccionario Python

Un diccionario en Python es una colección de elementos, donde cada uno tiene una clave `key` y un valor asociado `value`.

Los diccionarios se pueden crear con llaves `{}` separando con una coma cada par `key: value`.

En el siguiente ejemplo tenemos un diccionario con 3 elementos `keys` que son el nombre, la edad y el id.

```
d1 = {
  "Nombre": "Sara",
  "Edad": 27,
  "id": 1003882
}
print(d1)     # {'Nombre': 'Sara', 'Edad': 27, 'id': 1003882}
```

Otra forma equivalente de crear un diccionario en Python es usando la función `dict()` e introduciendo en una **lista** los pares `key: value` entre paréntesis (en t-uplas).

```
d2 = dict([
       ('Nombre', 'Sara'),
       ('Edad', 27),
       ('id', 1003882),
])
print(d2)     # {'Nombre': 'Sara', 'Edad': '27', 'id': '1003882'}
```

También es posible usar el constructor de la clase `dict` para crear un diccionario.

```
d3 = dict(Nombre='Sara',
          Edad=27,
          id=1003882)
print(d3)     # {'Nombre': 'Sara', 'Edad': 27, 'id': 1003882}
```

Algunas propiedades de los diccionario en Python son las siguientes:

- Son **mutables**, pueden crecer o decrecer, se pueden añadir o eliminar elementos.
- Son **indexados**, los elementos del diccionario son accesibles a través del `key`.
- Y son **anidados**, un diccionario puede contener a otro diccionario en su campo `value`.

Acceder y modificar elementos

Se puede acceder a sus elementos con el operador de indexación `[]` o también con el método `get()`. Pasamos como parámetro la **clave** y obtenemos el **valor** asociado a la misma.

```
print(d1['Nombre'])             # Sara
print(d1.get('Nombre'))         # Sara
```

Para modificar un elemento lo podemos hacer también con el operador de indexación `[]` con el nombre de la clave y asignar el valor que queremos.

```
d1['Nombre'] = "Laura"
print(d1)   # {'Nombre': Laura', 'Edad': 27, 'id': 1003882}
```

Si el `key` al que accedemos no existe, se añade el par de valores al diccionario.

```
d1['Direccion'] = "Calle 123"
print(d1)   # {'Nombre': 'Laura', 'Edad': 27, 'id': 1003882, 'Direccion':
'Calle 123'}
```

Eliminar elementos de un diccionario

Con `del` podemos eliminar tanto un par clave-valor de un diccionario:

```
d1 = {
  "Nombre": "Sara",
  "Edad": 27,
  "id": 1003882
}
del(d1["id"])
print(d1)
```

Desaparece el par `"id" - 1003882` de `d1`:

```
{'Nombre': 'Sara', 'Edad': 27}
```

Como una variable diccionario completa:

```
d2 = {
      'Nombre': 'María',
      'Edad': 33,
      'id': 1003083,
}
del(d2)
print(d2)               # No muestra nada
```

Iterar diccionario

Recorrer para obtener claves

Los diccionarios se pueden iterar de manera muy similar a las listas u otras estructuras de datos. Para imprimir las claves:

```
for key in d1:
    print(key)

# Nombre
# Edad
# id
# Direccion
```

Recorrer para obtener valores

Se puede recorrer y mostrar solo los valores.

```
for key in d1:
    print(d1[key])

# Laura
# 27
# 1003882
# Calle 123
```

Recorrer claves y valores

El método `items()` aplicado a un diccionaro devuelve un **objeto vista** que contiene los pares `clave-valor` de un diccionario como una lista de `t-uplas` en una lista.

```
d2 = dict([
       ('Nombre', 'Sara'),
       ('Edad', 27),
       ('id', 1003882),
])
print(d2.items())
```

Devuelve:

```
dict_items([('Nombre', 'Sara'), ('Edad', 27), ('id', 1003882)])
```

Esto nos permite recorrer un diccionario en un bucle y tener acceso tanto a la **clave** como a su **valor** asociado:

```
d2 = dict([
       ('Nombre', 'Sara'),
       ('Edad', 27),
       ('id', 1003882),
])

for key, value in d2.items():
    print(f"{key} \t {value}")
```

Obtenemos:

```
Nombre    Sara
Edad      27
id        1003882
```

Diccionarios anidados

Un diccionario en Python pueden contener, a su vez, otros diccionarios como valores:

```
anidado1 = {"a": 1, "b": 2}
anidado2 = {"a": 1, "b": 2}
d = {
  "anidado1" : anidado1,
  "anidado2" : anidado2
}
print(d)    # {'anidado1': {'a': 1, 'b': 2}, 'anidado2': {'a': 1, 'b': 2}}
```

Listas de diccionarios

También es posible crear listas cuyos elementos sean diccionarios:

```
empleados = []
d1 = {
  "Nombre": "Sara",
  "Edad": 27,
  "id": 1003882
}
d2 = {
      'Nombre': 'María',
      'Edad': 33,
      'id': 1003083,
}

empleados.append(d1)
empleados.append(d2)

for empleado in empleados:
    print(empleado)

print(f"Id empleado 2: {empleados[1]['id']}")
```

Obtenemos:

```
{'Nombre': 'Sara', 'Edad': 27, 'id': 1003882}
{'Nombre': 'María', 'Edad': 33, 'id': 1003083}
Id empleado 2: 1003083
```

Otros métodos de diccionarios

`clear()`

El método `clear()` elimina todo el contenido del diccionario.

```
d = {'a': 1, 'b': 2}
d.clear()
print(d)                                # {}
```

`get(<key>[,<default>])`

El método `get()` nos permite consultar el `value` para un `key` determinado. El segundo parámetro es opcional, y en el caso de proporcionarlo es el valor a devolver si no se encuentra la `key`.

```
d = {'a': 1, 'b': 2}
print(d.get('a'))                       # 1
print(d.get('z', 'No encontrado'))      # No encontrado
```

`items()`

El método `items()` devuelve un **objeto vista** que contiene los pares `clave-valor` de un diccionario como `t-uplas` en una lista.

Si se convierte en `list` se puede indexar como si de una lista normal se tratase, siendo los primeros elementos las `key` y los segundos los `value`.

```
d = {'a': 1, 'b': 2}
it = d.items()
print(it)                       # dict_items([('a', 1), ('b', 2)])
print(list(it))                 # [('a', 1), ('b', 2)]
print(list(it)[0][0])           # a
```

`keys()`

El método `keys()` devuelve un **objeto vista** que contiene una lista con todas las claves del diccionario.

```
d = {'a': 1, 'b': 2}
k = d.keys()
print(k)                        # dict_keys(['a', 'b'])
print(list(k))                  # ['a', 'b']
```

`values()`

El método `values()` devuelve un **objeto vista** que contiene una lista con todos los `values` o valores del diccionario.

```
d = {'a': 1, 'b': 2}
print(list(d.values()))         # [1, 2]
```

`pop(<key>[,<default>])`

El método `pop()` busca y **elimina** el elemento del diccionario que contiene la `key` que se pasa como parámetro y devuelve su valor asociado.

Si la clave no existe y no le pasamos un segundo parámetro se produciría un error

```
d = {'a': 1, 'b': 2}
r = d.pop('c', 'no encontrado')
print(r)                          # no encontrado
r = d.pop('a')
print(r)                          # 1
print(d)                          # {'b': 2}
```

También se puede pasar un segundo parámetro que es el valor a devolver si la clave no se ha encontrado. En este caso si no se encuentra no habría error.

```
d = {'a': 1, 'b': 2}
d.pop('c', -1)
print(d)                          # {'a': 1, 'b': 2}
```

`popitem()`

El método `popitem()` elimina el último elemento de un diccionario. Si la lista está vacía se genera un error.

```
d = {'a': 1, 'b': 2}
d.popitem()
print(d)                          # {'a': 1}
```

`update(<obj>)`

El método `update()` se llama sobre un diccionario y tiene como entrada otro diccionario. Los `value` son actualizados y si alguna `key` del nuevo diccionario no esta en el diccionario se añade el par al diccionario.

```
d1 = {'a': 1, 'b': 2}
d2 = {'a': 0, 'd': 400}
d1.update(d2)
print(d1)                         #{'a': 0, 'b': 2, 'd': 400}
```

Recursos

- Dictionaries - RealPython
- Diccionarios en Python
- Ejercicios de diccionarios - w3resource
- 15 things you should know about Dictionaries in Python - towardsdatascience.com

tags: `pro` `ut2` `diccionarios` `dictionaries`