ESTUDIOS SUPERIORES DE BUENOS AIRES
ESBA

Diccionarios

Definición

Los diccionarios son un tipo de estructura de datos que permite guardar un conjunto no ordenado de pares **clave-valor**, siendo las claves **únicas** dentro de un mismo diccionario (es decir que no pueden existir dos elementos con una misma clave).

Los diccionarios son mutables, es decir, es posible modificar su longitud, podemos agregar o quitar elementos de él; de igual forma todos los valores almacenados en el diccionario pueden ser modificados.

A diferencias de las listas y de las tuplas los diccionarios no se rigen por la regla de los índices, en este caso todos los valores que se almacenen en el diccionario no corresponderá a un índice, si no a una clave (llave o key). Todos los valores necesitan tener una clave y cada clave necesita tener un valor.

```
diccionario = { clave₁ : valor₁ , clave₂ : valor₂ , claveₙ : valorₙ }
```

```
alumno = {
    "nombre" : "Oliver",
    "edad" : 5,
    "nacionalidad" : "Argentina",
    "materias" : ("matemáticas", "inglés")
}
```

```python
alumno = {}
#Agregamos elementos
alumno["nombre"] = "Oliver"
alumno["edad"] = 5
alumno["nacionalidad"] = "Argentina"
alumno["materias"] = ("matemáticas", "inglés")
alumno["edad"] = 6 #Modificamos un valor

print(alumno["nombre"])
```

Acceder a un elemento de un diccionario es una de las principales operaciones por las que existe este tipo de dato. El acceso a un valor se realiza mediante **indexación** de la clave. Para ello, simplemente encierra entre corchetes la clave del elemento, por ejemplo: alumno["edad"] siguiendo el ejemplo del código anterior. Si la clave ya existe en el diccionario, se actualiza su valor.

En caso de que la clave no exista, se lanzará la excepción **KeyError**.

Comparaciones

Se puede utilizar el operador de igualdad **==** para comparar si dos diccionarios son iguales. Dos diccionarios son iguales si contienen el mismo conjunto de pares clave:valor, independientemente del orden que tengan.

```
persona1 = {
    "nombre":"Guido van Rossum",
    "edad":56
}
persona2 = {
    "nombre":"Richard M. Stallman",
    "edad":68
}
if (persona1 == persona2):
    print("Son iguales")
else:
    print("Son distintos")
```

Otro tipo de comparaciones entre diccionarios no están permitidas. Si se intenta, el intérprete lanzará la excepción **TypeError**.

Métodos

El método .**get**(*clave*, *valorDef*) devuelve el valor correspondiente a la clave. En caso de que la clave no exista no lanza ningún error, sino que devuelve el segundo argumento valor por defecto. Si no se proporciona este argumento, se devuelve el valor None.

```
personaje = {
    "nombre":"Guido",
    "edad":56,
    "nacionalidad":"Holandés",
    "nacimiento":(1967, 4, 10)
}
print(personaje.get("nombre")) #Guido
print(personaje.get("edad")) #56
print(personaje.get("apellido", "-")) #-
print(personaje.get("año")) #None
```

El método .**items**() devuelve una lista (de la clase dict_items) de tuplas compuestas por la clave y el valor.

El método .**keys**() devuelve una lista de claves como elementos.

El método .**values**() devuelve una lista de valores sin sus claves.

```
print( personaje.items())
print( personaje.keys())
print( personaje.values())
```

```
dict_items([('nombre', 'Guido'), ('edad', 56), ('nacionalidad', 'Holandés'), ('nacimiento', (1967, 4, 10))])
dict_keys(['nombre', 'edad', 'nacionalidad', 'nacimiento'])
dict_values(['Guido', 56, 'Holandés', (1967, 4, 10)])
```

Recorrido

Existen varias alternativas para recorrer los elementos de un diccionario:

```
personaje = {
    "nombre":"Guido",
    "edad":56,
    "nacionalidad":"Holandés",
    "nacimiento":(1967, 4, 10)
}

for clave in personaje:
    print( clave, ' -> ', personaje[clave] )

for clave in personaje.keys():
    print( clave, ' -> ', personaje[clave] )

for clave, valor in personaje.items():
    print( clave, ' -> ', valor )

for valor in personaje.values():
    print( valor )
```

```
nombre  ->  Guido
edad  ->  56
nacionalidad  ->  Holandés
nacimiento  ->  (1967, 4, 10)
```

Por defecto, se utilizará la clave como iterador.

Utilizando el método .**keys**() es equivalente y más claro pero menos oṕtimo.

Se puede iterar desempaquetando cada item mediante el método .**item**()

Se puede iterar sólo a través de sus valores ignorando la clave mediante el método .**values**()

Eliminar contenido

Si quisiéramos eliminar un elemento del diccionario, utilizamos el comando **del**. Si no existe la clave, se lanza una excepción **KeyError**

```
alumno = {
"nombre":"Oliver",
"edad":5,
"nacionalidad":"Argentina",
"materias":("matemáticas", "inglés")
}
del alumno["nacionalidad"]
print(alumno) #{'nombre': 'Oliver', 'edad': 5, 'materias': ('matemáticas', 'inglés')}
```

Se puede verificar la existencia de la clave con el operador **in**:

```
if "nacionalidad" in alumno:
    del alumno["nacionalidad"]
```

También se pueden eliminar todos los valores del diccionario con el método .**clear**():

```
alumno.clear()
print(alumno) #{}
```

Listas de diccionarios

Se podría interpretar la visualización de una lista de diccionarios como una grilla; en donde la clave es el título de la columna y, los valores, los registros (filas) que corresponden a cada una de ellas. Cada elemento (intersección de fila y columna) se accederá por la combinación de la posición de la fila (elemento de lista) y la columna (clave).

```
ciudades = [
        {
            "cp": 1425,
            "nombre": "Capital Federal"
        },
        {
            "cp": 1900,
            "nombre": "La Plata"
        }
    ]
#Agregar un elemento
ciudades.append({"cp":1878, "nombre":"Quilmes"})
```

cp	nombre
1425	Capital Federal
1900	La Plata
1878	Quilmes

```
#Acceso a un elemento
print("Código postal La Plata: ", ciudades[1]["cp"])
```

Anidamiento

Es posible, también, crear un diccionario cuyo valor contiene otro diccionario. En dicha situación, se debe acceder a los valores por medio de las distintas claves que lo componen como si fuera una matriz.

```
pais = {
    "id": 1,
    "nombre": "Argentina",
    "coordenadas": {
        "latitud": -34,
        "longitud": -64
    }
}
print(pais["nombre"], " ->     (",
    pais["coordenadas"]["latitud"], ", ",
    pais["coordenadas"]["longitud"], ")")
# Argentina  ->   ( -34 ,  -64 )
```

Orden de claves

Cómo hemos visto, las claves de un diccionario no están ordenadas, si se desea ordenar un diccionario mediante sus claves, se puede utilizar la función **sorted**() en combinación con el método .**items**() del diccionario.

```
alumno = {
    "nombre":"Oliver",
    "edad":5,
    "nacionalidad":"Argentina",
    "materias":("matemáticas")
}
alumnoOrdenado = dict(sorted(alumno.items()))
print(alumnoOrdenado)
#{'edad': 5, 'materias': 'matemáticas', 'nacionalidad': 'Argentina', 'nombre': 'Oliver'}
```

Extender diccionario

El método update() se utiliza para agregar elementos a un diccionario o para actualizar los valores existentes en el diccionario con nuevos valores.

Recibe como parámetro otro diccionario que contiene pares clave-valor que se agregarán al diccionario existente. Aquí hay un ejemplo de cómo utilizarlo:

```
diccionario = {'a': 1, 'b': 2, 'c': 3}
diccionario.update({'d': 4, 'e': 5})

print(diccionario)
#{'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5}

diccionarioB = {'b': 10}
diccionario.update(diccionarioB)
```