El futuro digital es de todos

MinTIC

Diccionarios

OPERADO POR:

Universidad Tecnológica de Pereira

Misión TIC 2022

RUTA DE APRENDIZAJE 1

Diccionarios

En Python existen diferentes **estructuras de datos** las cuales nos permiten almacenar y gestionar diferentes tipos de datos, por ejemplo tenemos a las listas, las tuplas y a los **diccionarios**.

Hoy vamos a hablar de los diccionarios, la estructura característica que posee Python, el diccionario, define una relación uno a uno entre una clave y su valor.

Los diccionarios en Python, al igual que las listas y las tuplas, nos permiten almacenar diferentes tipos de datos: Strings, enteros, flotantes, booleanos, tuplas, listas e inclusive otros diccionarios. 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.

Definición y almacenamiento de datos

```
In [2]: diccionario = {}

In [3]: print(diccionario)
        {}

In [4]: diccionario = dict()

In [5]: diccionario
Out[5]: {}
```

Para definir un diccionario haremos uso de un juego de llaves o de la función dict(). De ambas formas es correcto.

Para poder almacenar algún valor seguiremos la siguiente estructura, nuestra clave, dos puntos y el valor el cual queremos asociar.

Por ejemplo, el *string* total, dos puntos, 55, con esto le indico a Python que la llave "total" almacena 55.

```
In [6]: diccionario = {"total": 55}
        print(diccionario)
        otrodiccionario = {"copia": 123.23}
        print(otrodiccionario)

        {'total': 55}
        {'copia': 123.23}
```

Si necesitamos almacenar **nuevos valores** basta con separarlos mediante una coma.

```
In [7]: diccionario = {"total": 55, "descuento": True, 15: "15"}

        print(diccionario)

        diccionarioEjemploExcel ={"nombre":5+2,"telefono":3363692, "edad":33, "ciudad":"Pereira"}

        print(diccionarioEjemploExcel)
```

```
{'total': 55, 'descuento': True, 15: '15'}
{'nombre': 7, 'telefono': 3363692, 'edad': 33, 'ciudad': 'Pereira'}
```

Otra forma de crearlos es llamando a la función *dict*

```
In [8]: diccionario = dict(total= 55, descuento= True, subtotal= 15)

        print(diccionario)
```

```
{'total': 55, 'descuento': True, 'subtotal': 15}
```

Veamos otro ejemplo.

```
In [9]: diccionario = {"total": 55, 10: "Curso de Python", 2.0: True}

        print(diccionario)

        {'total': 55, 10: 'Curso de Python', 2.0: True}
```

En este caso ya nos complicamos un poco. Estamos almacenando tres valores con sus correspondientes llaves. Estas tres llaves son valores inmutables. Tenemos:

- Un string ("total")
- Un número entero (10)
- Un decimal (2.0)

Estamos almacenando 55, "Curso Python", True respectivamente.

Regularmente haremos uso de llaves de un mismo tipo, comúnmente **string**, sin embargo si por algún motivo necesitas almacenar otro tipo de llave sabrás que puedes hacerlo. Un dato interesante es que podemos utilizar clases como llaves.

En este curso más adelante trabajaremos con objetos **JSON**. Por lo cual los diccionarios serán muy familiares ya que de hecho el equivalente de un **JSON** en Python son los diccionarios.


```
In [10]: usuario = {
             'nombre': 'Nombre del usuario',
             'edad' : 23,
             'curso': 'Curso de Python',
             'skills':{
                 'programacion' : True,
                 'base_de_datos': False
             },
             'No medallas' : 10
         }

         print(usuario)
```

```
{'nombre': 'Nombre del usuario', 'edad': 23, 'curso': 'Curso de Python', 'ski
lls': {'programacion': True, 'base_de_datos': False}, 'No medallas': 10}
```

```
In [11]: print(usuario['curso'])
         print(usuario['skills'])
         print(usuario['skills']['base_de_datos'])
```

```
Curso de Python
{'programacion': True, 'base_de_datos': False}
False
```

```
In [12]: diccionario = dict(total=55, descuento=True, descuento5=True, subtotal="15")

         print(diccionario)
         print(diccionario['subtotal'])
```

```
{'total': 55, 'descuento': True, 'descuento5': True, 'subtotal': '15'}
15
```

Cómo ven este tipo de dato puede ser tan complejo como deseemos.

Para poder agregar, obtener o modificar algún valor del diccionario haremos uso de corchetes.

In [13]:
```
diccionario = dict()
print(diccionario)

diccionario['marca'] = 'Mazda'
print(diccionario['marca'])

diccionario['marca'] = 'Subaru'
print(diccionario['marca'])

print(diccionario)
```

```
{}
Mazda
Subaru
{'marca': 'Subaru'}
```

podemos obtener todas las llaves de nuestro diccionario utilizando el método keys, de igual forma podremos obtener todos los valores el diccionario con le método values.

```
In [14]: diccionario = {'Eduardo': 1, 'Fernando':2, 'Uriel':3, 'Rafael': 4}
         print(diccionario)

         {'Eduardo': 1, 'Fernando': 2, 'Uriel': 3, 'Rafael': 4}

In [15]: diccionario.keys()

Out[15]: dict_keys(['Eduardo', 'Fernando', 'Uriel', 'Rafael'])

In [16]: diccionario.values()

Out[16]: dict_values([1, 2, 3, 4])
```

Veamos otro ejemplo

```
In [17]: diccionario = {
             "clave1":234,
             "cadena":True,
             "clave3":"Valor 1",
         }
         print(diccionario)
         print(type(diccionario))
```

```
{'clave1': 234, 'cadena': True, 'clave3': 'Valor 1'}
<class 'dict'>
```

Usted puede acceder a los valores del diccionario usando cada clave, a continuación se presentan unos ejemplos:

```
In [18]: diccionario['clave1']
```

```
Out[18]: 234
```

```
In [20]: diccionario['clave3']
```

```
Out[20]: 'Valor 1'
```

Acceder a valor de clave

Esta operación le permite acceder a un valor especifico del diccionario mediante su clave. Indicamos entonces que clave queremos buscar en el diccionario, y nos entregara como resultado asociado a esa palabra clave:

```
In [1]: versiones = dict(python=2.7, zope=2.13, plone=5.1, django=2.1)
        versiones['zope']

Out[1]: 2.13
```

Métodos

Los objetos de tipo diccionario tienen por defecto una serie de métodos integrados, a continuación:

clear()

Este método remueve todos los elementos desde el diccionario.

```
In [23]: versiones = dict(python=2.7, zope=2.13, plone=5.1)
         print(versiones)

         {'python': 2.7, 'zope': 2.13, 'plone': 5.1}

In [24]: versiones.clear()
         print(versiones)

         {}
```

copy()

Este método devuelve una copia superficial del tipo diccionario:

```
In [25]: versiones = dict(python=2.7, zope=2.13, plone=5.1)
         otra_versiones = versiones.copy()
         print(versiones == otra_versiones)

         True

In [26]: print(versiones)
         print(otra_versiones)

         {'python': 2.7, 'zope': 2.13, 'plone': 5.1}
         {'python': 2.7, 'zope': 2.13, 'plone': 5.1}
```

fromkeys()

Este método crea un nuevo diccionario con claves a partir de un tipo de dato secuencia. El valor de value por defecto es el tipo None.

```
In [27]: secuencia = ('python', 'zope', 'plone')
         versiones = dict.fromkeys(secuencia)
         print("Nuevo Diccionario : %s" %  str(versiones))
         print("Nuevo Diccionario : {}".format(str(versiones)))

         Nuevo Diccionario : {'python': None, 'zope': None, 'plone': None}
         Nuevo Diccionario : {'python': None, 'zope': None, 'plone': None}
```

En el ejemplo anterior inicializa los valores de cada clave a None, mas puede inicializar un valor común por defecto para cada clave:

```
In [28]: versiones = dict.fromkeys(secuencia, 0.1)
         print("Nuevo Diccionario : %s" %  str(versiones))

         Nuevo Diccionario : {'python': 0.1, 'zope': 0.1, 'plone': 0.1}
```

get()

Este método devuelve el valor basado en la coincidencia de búsqueda en un diccionario mediante una clave, de lo contrario devuelve el objeto None.

```
In [29]: versiones = dict(python=2.7, zope=2.13, plone=5.1)
         versiones.get('plone')

Out[29]: 5.1

In [30]: print(versiones.get('php'))

         None
```

items()

Este método devuelve una lista de pares de diccionarios (clave, valor), como tuplas (después véremos que son tuplas en general).

```
In [31]: versiones = dict(python=2.7, zope=2.13, plone=5.1)
         versiones.items()

Out[31]: dict_items([('python', 2.7), ('zope', 2.13), ('plone', 5.1)])
```

pop()

Este método remueve específicamente una clave de diccionario y devuelve valor correspondiente. Lanza una excepción KeyError si la clave no es encontrada.

```
In [32]: versiones = dict(python=2.7, zope=2.13, plone=5.1)
         versiones

Out[32]: {'python': 2.7, 'zope': 2.13, 'plone': 5.1}

In [33]: print(versiones.pop('zope'))
         versiones

         2.13

Out[33]: {'python': 2.7, 'plone': 5.1}
```

update()

Este método actualiza un diccionario agregando los pares clave-valores en un segundo diccionario. Este método no devuelve nada.

El método update() toma un diccionario o un objeto iterable de pares clave/valor (generalmente tuplas). Si se llama a update() sin pasar parámetros, el diccionario permanece sin cambios.

```
In [34]: versiones = dict(python=2.7, zope=2.13, plone=5.1)
         print(versiones)

         {'python': 2.7, 'zope': 2.13, 'plone': 5.1}

In [35]: versiones_adicional = dict(django=2.1)
         print(versiones_adicional)

         {'django': 2.1}

In [36]: versiones.update(versiones_adicional)
         print(versiones)

         {'python': 2.7, 'zope': 2.13, 'plone': 5.1, 'django': 2.1}
```

Como puede apreciar este método no devuelve nada, pero si mostramos de nuevo el diccionario usando la funcion *print* versiones puede ver que este fue actualizado con el otro diccionario versiones_adicional.

Funciones:

Los objetos de tipo diccionario tienen disponibles una serie de funciones integradas en el interprete Python para su tratamiento, a continuación algunas de estas:

len()

Esta función cuenta el numero de llaves de primer nivel de nuestro diccionario:

```
In [37]: versiones = dict(python=2.7, zope=2.13, plone=5.1)
         len(versiones)

Out[37]: 3

In [38]: usuario = {
             'nombre': 'Nombre del usuario',
             'edad' : 23,
             'curso': 'Curso de Python',
             'skills':{
                 'programacion' : True,
                 'base_de_datos': False
             },
             'No medallas' : 10
         }

         print(len(usuario))

         5
```

Sentencias

Los objetos de tipo diccionario tienen disponibles una serie de sentencias integradas en el interprete Python para su tratamiento, a continuación algunas de estas:

Del

Elimina una clave y su valor del diccionario si logra encontrar la clave dentro del diccionario.

```
In [3]: versiones = dict(python=2.7, zope=2.13, plone=5.1, django=2.1)
        print(versiones)

        {'python': 2.7, 'zope': 2.13, 'plone': 5.1, 'django': 2.1}

In [4]: del versiones["python"]
        print(versiones)

        {'zope': 2.13, 'plone': 5.1, 'django': 2.1}
```

Ejercicio

Tenemos los siguientes diccionarios:

```
In [1]: Estudiantes = {  'Alumno1': {'nombre':'Daniel', 'edad':11, 'estatura':1.75, 'grado':'Master'},
                         'Alumno2':{'nombre':'David', 'edad':32, 'estatura':1.85, 'grado':'Doctor'}   }

        print(Estudiantes)
```

```
{'Alumno1': {'nombre': 'Daniel', 'edad': 11, 'estatura': 1.75, 'grado': 'Master'}, 'Alumno2': {'nombre': 'David', 'edad': 32,
'estatura': 1.85, 'grado': 'Doctor'}}
```

Comparemos los nombres de los estudiantes

```
In [2]: if Estudiantes['Alumno1']['nombre'] == Estudiantes['Alumno2']['nombre']:
            print("Los nombres son iguales")
        else:
            print('Los nombres son diferentes')
```

```
Los nombres son diferentes
```

El futuro digital es de todos

MinTIC

GRACIAS

OPERADO POR:

Universidad Tecnológica de Pereira

MisiónTIC2022