Universidad Nacional de Jujuy
Facultad de Ingeniería

Introducción a la Programación
Programación I

Diccionarios

Contenido

- Introducción
- Definición
- Creación e indexación
- Operaciones
- Métodos internos
- Archivos JSON
- Ejemplos

Introducción

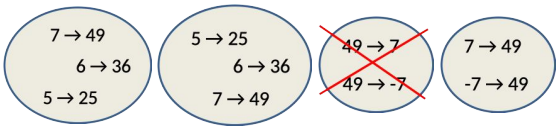
Los diccionarios son la colección de datos más poderosa de Python
Los diccionarios nos permiten realizar operaciones rápidas similares a las de una base de datos en Python
Los diccionarios tienen diferentes nombres en diferentes lenguajes.
- Matrices Asociativas - Perl / PHP
- Propiedades o Mapa o HashMap - Java
- Bolsa de propiedades - C# / .Net

Lista: Una colección lineal de valores que se mantienen en orden.

Diccionario: Un "bolso" de valores cada uno con su propia etiqueta

Definición

- Un diccionario o mapeo asigna a cada clave a un valor
- El orden no importa
- Dada una clave, puede buscar un valor
- Dado un valor, no puede buscar su clave
- Sin claves duplicadas
- Dos o más claves pueden mapear el mismo valor
- Las claves deben ser inmutables
- Se pueden agregar pares clave → valor a un diccionario
- También puede eliminar pares clave → valor



Creación

- Sintaxis:
  dic = { }
  dic = dict()
  dic = {**clave1**:**valor1**, **clave2**:**valor2**, … }

- Cada valor es referenciado por su clave
- Las claves son únicas
- Los pares clave-valor se separan por comas
- Las claves deben ser un tipo de dato inmutable
  - Números, cadenas
- Los valores pueden ser de cualquier tipo de dato
  - Números, cadenas, listas, diccionarios, etc

Anidamiento

```
pedido = {
            "tamaño": "mediana",
            "precio": 15.67,
            "toppings": ["champinones", "pepperoni", "albahaca"],
            "queso_extra": False,
            "delivery": True,
            "cliente": {
                "nombre": "Jane Doe",
                "telefono": None,
                "correo": "janedoe@email.com"
            }
        }

ordenes = [pedio01, pedido02, …]
```

Indexación

- Las listas indexan sus entradas según la posición en la lista
- Los diccionarios son como bolsas, sin orden. Así que indexamos las cosas que ponemos en el diccionario con una "etiqueta de búsqueda"

```
bolso = dict()
bolso['dinero'] = 12
bolso['caramelo'] = 3
bolso['pañuelo'] = 75

print(bolso) #{'dinero': 12, 'pañuelo': 75, 'caramelo': 3}
print(bolso['caramelo']) #3

bolso['caramelo'] = bolso['caramelo'] + 2
print(bolso) #{'dinero': 12, 'pañuelo': 75, 'caramelo': 5}
```


Listas vs. Diccionarios (operaciones)

Los diccionarios son como las listas, excepto que usan claves en lugar de números para acceder a los valores.

```
lst = list()
lst.append(21)
lst.append(183)
print(lst)
[21, 183]
lst[0] = 23
print(lst)
[23, 183]
print(len(lst))#2
```

```
dic = dict()
dic['edad'] = 21
dic['curso'] = 182
print(dic)
{'curso': 182, 'edad': 21}
dic['edad'] = 23
print(dic)
{'curso': 182, 'edad': 23}
print(len(dic))#2
```


Operaciones

```
claves= ['A', 'B', 'C']
valores = ['Alfa', 'Bravo', 'Charlie']
dic = {}
for i in range (len(claves)):
  dic[claves[i]] = valores[i]

dic = dict(zip(claves, valores))#Otra forma
print(dic)
#{'A': 'Alfa', 'B': 'Bravo', 'C': 'Charlie'}

k1, k2, k3 = dic  #A B C
v1, v2, v3 = dic.values() #Alfa Bravo Charlie
t1, t2, t3 = dic.items()
print(t1, t2, t3) #('A', 'Alfa') ('B', 'Bravo') ('C', 'Charlie')
```


Los diccionarios son mutables


```
estudiantes = {88:'Mary', 29:'Anna', 59:'Juan'}
nuevo = estudiantes
#agregar el estudiante 37 Sara en nuevo
nuevo [37] = 'Sara'
#mostrar estudiantes
print(estudiantes) #{88:'Mary', 29:'Anna', 59:'Juan', 37:'Sara'}
#mostrar nuevo
print(nuevo)    #{88:'Mary', 29:'Anna', 59:'Juan', 37:'Sara'}
```


Métodos

Método	Descripción
dict.*clear*()	Elimina todos los elementos del diccionario.
dict.*copy*()	Devuelve una copia del diccionario.
dict.*get*(key[, default=None])	Devuelve el valor correspondiente a la clave. Si la clave no existe en el diccionario se devuelve el valor establecido por defecto.
dict.*items*()	Devuelve una lista de los pares de tuplas (clave, valor).
dict.*keys*()	Devuelve la lista de las claves del diccionario.
dict.*pop*(key[, default=None])	Elimina el valor de la clave y devuelve su valor. Si la clave no existe en el diccionario se devuelve el valor establecido por defecto.
dict.*popitem*()	Elimina un elemento al azar y devuelve una tupla (clave, valor). Lanza un error *KeyError* si el diccionario está vacío.
dict.*setdefault*(key, default = None)	Similar a *get*(), pero si la clave no existe en el diccionario la crea con el valor dado por defecto y devuelve el valor por defecto.
dict.*update*(dict2)	Añade los pares clave:valor del diccionario dict2 a dict. Se reescriben los valores con la misma clave.
dict.*values*()	Devuelve la lista de los valores del diccionario.

Métodos

```
estudiantes = {88:'Mary', 29:'Anna', 59:'Juan'}

for clave, valor in estudiantes.items():
  print(clave, valor) #Ver todo el diccionario

print(estudiantes.get(53)) #Juan
print(estudiantes[99]) #Error
print(estudiantes.get(99)) #None
print(estudiantes.get(99,'No existe'))
print(estudiantes.setdefault(99,'NN')) #se agrega 99

claves = estudiantes.keys() #no es una lista
listaClaves = list(estudiantes.keys()) #es una lista
valores = estudiantes.values() #no es una lista
listaValores = list(estudiantes.values()) #es una lista
```

Métodos continuación

```
estudiantes = {88:'Mary', 29:'Anna', 59:'Juan'}
aux = {15: 'Pedro', 37: 'Sara'}
estudiantes.update(aux)
print(estudiantes)
#{88: 'Mary', 29: 'Anna', 59: 'Juan', 15: 'Pedro', 37: 'Sara'}
print(59 in estudiantes)#True
print('Juan' in estudiantes)#False
print('Juan' in estudiantes.values())#True
nuevo = estudiantes
estudiantes_copia = estudiantes.copy()
print(estudiantes is nuevo)#True
print(estudiantes is estudiantes_copia)#False
estudiantes_copia.clear()
print(estudiantes_copia)#{}
```


Métodos continuación

```
estudiantes = {88: 'Mary', 29: 'Anna', 59: 'Juan', 37: 'Sara'}

del estudiantes[100]#Error
item = estudiantes.pop(100,'No existe')
print(item)#No existe
item = estudiantes.popitem() #remueve el último
print(estudiantes)#{88: 'Mary', 29: 'Anna', 59: 'Juan'}
```


Diccionarios por comprensión

<diccionario> = {<expresión> for <elemento> in <iterable> <filtro>}

```
d1 = {x: x ** 2 for x in range(10) if x%2 == 0}
print(d1) #{0: 0, 2: 4, 4: 16, 6: 36, 8: 64}
d2 = {i : chr(65+i) for i in range(4)}
print(d2) #{0: 'A', 1: 'B', 2: 'C', 3: 'D'}

'''Dada la lista  lst = [5, 6, 7], obtener dic =  {5:25, 6:36, 7:49}
Luego producir el reverso de dic, cid = {25:5, 36:6, 49:7}'''

lst = [5, 6, 7]
dic = {x:x**2 for x in lst}
print(dic) #{5: 25, 6: 36, 7: 49}

cid = {v : k for k, v in dic.items()}
print(cid) #{25: 5, 36: 6, 49: 7}
```


¿Qué es JSON?

Java Script Object Notation. JSON es un formato usado para almacenar o representar datos. Sus usos más frecuentes incluyen desarrollo web y creación de archivos de configuración.

Desarrollo web: JSON se usa comúnmente en aplicaciones web para enviar información desde el servidor al cliente o desde el cliente al servidor.

Archivos de configuración: para crear una aplicación web se debe incluir un archivo JSON llamado manifest.json para especificar el nombre de la aplicación, su descripción, versión actual y otras propiedades.

Estructura y formato JSON

```
{
    "tamano": "mediana",
    "precio": 15.67,
    "toppings": ["champinones", "pepperoni", "albahaca"],
    "queso_extra": false,
    "delivery": true,
    "cliente": {
        "nombre": "Jane Doe",
        "telefono": null,
        "correo": "janedoe@email.com"
    }
}
```

- Las claves deben ser cadenas de caracteres.
- Los valores pueden ser cadenas de caracteres, números, arreglos, valores Booleanos (true / false), null, o un objeto JSON.

JSON vs. Diccionarios

Las claves en los pares clave/valor de JSON siempre son de tipo str. Cuando se convierte un diccionario a JSON, todas las claves del diccionario se convierten en *strings*. Hay una guía de estilo

- JSON y los diccionarios pueden parecer muy similares inicialmente (visualmente) pero en realidad son muy diferentes. Veamos cómo están relacionados y cómo se complementan para lograr que Python sea una herramienta poderosa para trabajar con archivos JSON.
- JSON es un formato de archivo usado para representar y almacenar información mientras que un diccionario de Python es una estructura de datos (objeto) que se almacena en la memoria del dispositivo mientras se ejecuta el programa de Python.

Conversion de tipos

Python	JSON
dict	Object
list	Array
tuple	Array
str	String
int	Number
float	Number
True	true
False	false
None	null

Módulos

```
import json
```

```
json.<funcion>(<args>)
```

Módulo

Argumentos

De cadena de caracteres JSON a un diccionario de Python

```
json.loads(<cadena_JSON>)
```

Módulo | Función | Cadena de caracteres JSON

De diccionario de Python a cadena de caracteres JSON

```
json.dumps(<obj>)
```

Módulo | Función | Objeto

JSON y archivos

Abrir ordenes.json en modo de lectura | Objeto archivo

```
with open("ordenes.json") as archivo:
    datos = json.load(archivo)
```

Leer el archivo JSON y crear un diccionario

Abrir orders.json en modo de escritura | Objeto archivo

```
with open("ordenes.json", "w") as archivo:
    # Agregar informacion al archivo JSON
    json.dump(<obj>, <archivo>)
```

Objeto que será guardado en formato JSON | Archivo donde será guardado

	load()	loads()
Propósito	Crear un objeto de Python a partir de un archivo JSON	Crear un objeto de Python a partir de una cadena de caracteres
Argumento	Archivo JSON	Cadena de caracteres
Valor retornado	Objeto de Python	Objeto de Python

	dump()	dumps()
Propósito	Escribir un objeto en formato JSON a un archivo	Obtener una cadena de caracteres JSON a partir de un objeto
Argumento	Objeto + Archivo	Objeto
Valor retornado	None	Cadena de caracteres

Bibliografía

- Aprende Python
- https://www.w3schools.com/python/python_dictionaries.asp
- https://docs.python.org/es/3/
- https://www.freecodecamp.org/espanol/news/python-leer-archivo-json-como-cargar-json-desde-un-archivo-y-procesar-dumps/