# \*\*Estructuras de Datos de Python\*\*

Introducción a Python para Visualización de Datos

Dr. Gerardo Ramón Fox

Julio 2022

Python nos proporciona cuatro estructuras de datos básicas:

1. Listas

2. Tuplas

3. Diccionarios

4. Sets

En este módulo, veremos diccionarios y sets.

## \*\*Diccionarios\*\*

Los diccionarios son estructuras de datos que permiten almacenar datos de la forma llave-valor (key-value). Técnicamente, están implementadas como \*hash tables\* por lo que son estructuras muy eficientes.

Se declaran de la siguiente manera:

# Las tres generan el mismo resultado.

mi\_dic\_a = dict(a=1, b=2)

mi\_dic\_b = {'a': 1, 'b': 2}

# La tercera forma permite inicializar diccionarios de una forma general y programable.

mi\_dic\_c = dict([('a', 1), ('b', 2)]) # Note que estamos inicializando con una lista de tuplas!

# Cada tupla contiene un par llave-valor (llave, valor).

# Imprimimos el diccionario (sin formato)

print(mi\_dic\_a)

print(mi\_dic\_b)

print(mi\_dic\_c)

# Verificar igualdad.

print(mi\_dic\_a == mi\_dic\_b)

print(mi\_dic\_a == mi\_dic\_c)

print(mi\_dic\_b == mi\_dic\_c)

La función len() retorna el número de pares almacenados.

mi\_dic = {'a': 1, 'b': 2}

print(len(mi\_dic))

Podemos agregar más pares con la función update:

# Forma 1: comp parametros

mi\_dic.update(c=3, d=4)

print(mi\_dic, len(mi\_dic), end="\n")

# Forma 2: pasando un diccionario

mi\_dic.update({'e': 5, 'f': 6})

print(mi\_dic, len(mi\_dic), end="\n")

# Forma 3: pasando una lista de tuplas (la mas general y que funciona de manera programatica)

mi\_dic.update([('g', 7), ('h', 8)])

print(mi\_dic, len(mi\_dic), end="\n")

Podemos acceder a elementos individuales usando la llave:

print(mi\_dic['a'])

print(mi\_dic['c'])

print(mi\_dic['g'])

#print(mi\_dic['z']) # Arroja un error si la llave no existe.

Eliminamos pares con la instrucción del:

del mi\_dic['h']

print(mi\_dic, len(mi\_dic))

### \*\*Vistas de un Diccionario\*\*

Como los diccionarios son tablas, podemos extraer vistas según la información que necesitemos.

Extraer una vista de las llaves en el diccionario:

```

mi\_dic.keys()

```

Extraer una vista de los valores solamente en el diccionario:

```

mi\_dic.values()

```

Extraer pares y valores:

```

mi\_dic.items()

```

La última genera una iterador, por lo que hay que usar un ciclo for especial para extraer los elementos individuales.

Veamos los ejemplos implementados

mi\_dic = {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5, 'f': 6, 'g': 7, 'h': 8}

print(mi\_dic.keys(), type(mi\_dic.keys()))

print(mi\_dic.values(), type(mi\_dic.values()))

print(mi\_dic.items(), type(mi\_dic.items()))

Nótese que son estructuras de datos especiales, se pueden convertir o \*castear\* a listas con la función list():

keys\_list = list(mi\_dic.keys())

values\_list = list(mi\_dic.values())

pairs\_list = list(mi\_dic.items())

print(keys\_list, type(keys\_list))

print(values\_list, type(values\_list))

print(pairs\_list, type(pairs\_list))

¿Cómo usar las vistas para iterar? Veamos unos ejemplos:

mi\_dic = {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5, 'f': 6, 'g': 7, 'h': 8}

# Recorrer las llaves.

for clave in mi\_dic.keys():

print(clave, end=" ")

print(" ")

# Recorrer los valores

for valor in mi\_dic.values():

print(valor, end=" ")

print(" ")

# Extraer llaves y valores en paralelo

for clave, valor in mi\_dic.items():

print(clave, valor)

### \*\*Ejemplo: Contar Frecuencias de Caracteres\*\*

Versión 1

letras = ['h','o','l','a',' ','m','u','n','d','o',' ','d','e',' ',\

'P','y','t','h','o','n'] # Podemos usar la barra invertida para dividir lineas largas.

# Inicializamos la tabla de que cuentas las frecuencias como un diccionario vacio.

tabla = dict()

# Recorrer la lista de letras.

for i in range(len(letras)):

# Si la letra ya esta en la tabla, contarla.

if letras[i] in tabla:

tabla[letras[i]] = tabla[letras[i]] + 1

# Si la letra no esta en la tabla, agregarla e inicializar el contador.

else:

tabla[letras[i]] = 1

# Imprimir los resultados

print("caracter: frecuencia")

for letra, frec in tabla.items():

print(letra + " : " + str(frec))

Versión 2

letras = ['h','o','l','a',' ','m','u','n','d','o',' ','d','e',' ',\

'P','y','t','h','o','n'] # Podemos usar la barra invertida para dividir lineas largas.

# Inicializamos la tabla de que cuentas las frecuencias como un diccionario vacio.

tabla = dict()

# Recorrer la lista de letras.

for letra in letras:

# Si la letra ya esta en la tabla, contarla.

if letra in tabla:

tabla[letra] = tabla[letra] + 1

# Si la letra no esta en la tabla, agregarla e inicializar el contador.

else:

tabla[letra] = 1

# Imprimir los resultados

print("caracter: frecuencia")

for letra, frec in tabla.items():

print(letra + " : " + str(frec))

Para más información sobre el uso de diccionarios, consultar la documentación: https://docs.python.org/3/tutorial/datastructures.html#dictionaries

## \*\*Sets\*\*

Un set es una estructura que almacena sin repetición objetos "hashables", es decir, que son inmutables. Los podemos inicializar de las siguientes maneras:

```

letras = {'a', 'd', 'c', 'b'}

letras = set(['a', 'd', 'c', 'b']) # A partir de una lista.

letras = set("adcb") # A partir de un string.

```

Veamos una serie de operaciones que se pueden hacer sobre un set:

# Inicializamos un set:

letras = {'a', 'b', 'b', 'c', 'd', 'd', 'd', 'e'}

print(letras) # Notese la falta de elementos repetidos.

# Obtenemos el numero de elementos con len.

print(len(letras))

# Verificamos membresia

print('a' in letras) # True, 'a' esta en el set.

print('z' in letras) # False, 'z' no esta en el set.

# Agregar un elemento

letras.add('f')

print(letras)

letras.add('g')

print(letras)

# Eliminar un elemento

letras.remove('g')

print(letras)

# Tambien podemos usar la funcion letras.discard('g'), la diferencia

# es que remove arroja un error si el argumento no esta en el set, mientras

# que discard simplemente no hace nada si no se encuentra el argumento.

# Vaciamos el set con

Los sets son utiles para hacer operaciones entre conjuntos. Veamos el siguiente ejemplo:

s1 = {1, 2, 3, 4, 5}

s2 = {-1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

print("s1 = ", s1)

print("s2 = ", s2)

print(" ")

# Union

print("s1.union(s2) = ", s1.union(s2)) # Equivalente: s1 | s2

print(" ")

# Interseccion

print("s1.intersection(s2) = ", s1.intersection(s2)) # Equivalente: s1 & s2

print(" ")

# Diferencia: retorna los elementos del que llama que no estan en el set del argumento

print("s1.difference(s2) = ", s1.difference(s2)) # Equivalente: s1 - s2

print("s2.difference(s1) = ", s2.difference(s1)) # Equivalente: s2 - s1

print(" ")

# Diferencia Simetrica:

print("s1.symmetric\_difference(s2) = ", s1.symmetric\_difference(s2)) # Equivalente: s1 ^ s2

print("s2.symmetric\_difference(s1) = ", s2.symmetric\_difference(s1)) # Equivalente: s2 ^ s1

print(" ")

Los sets no son estructuras que se pueden acceder por indices como uno lo hace en listas o tuplas. La unica manera de extraer valores individuales es iterar, por ejemplo:

nums = set([1,2,3,4,5])

# Recorrer el set con un ciclo for

print("Valores en el set nums: ")

for val in nums:

print(val)

print("----")

# Calcular la suma

tot = 0

for val in nums:

tot = tot + val

print("La suma de nums es: " + str(tot))

Para más información sobre el uso de Sets, consultar la documentación: https://docs.python.org/3/tutorial/datastructures.html#sets

## \*\*Conversión o Casteo Entre Estructuras de Datos\*\*

Las funciones generadores de cada estuctura se usan para convertir una estructura en otra:

```

list(), tuple(), dict(), set()

```

Veamos algunos ejemplos.

lst = [1,2,2,2,3,3,4] # Generamos una lista

print(lst)

tp = tuple(lst) # La convertimos a tupla

print(tp)

set\_lst = set(lst) # La convertimos a un set. Ojo: perderemos elementos repetidos.

print(set\_lst)

s = {'a', 'b', 'c', 'd'} # Generamos un set.

print(s)

ls\_s = list(s) # Lo convertimos en lista

print(ls\_s)

Pasar de diccionario a otras estructuras. Como el diccionario, es una estructura tabular (técnicamente una hash table), las funciones list(), tuple() y set() solo convertiran a su forma respectiva las llaves que el diccionario tenga definidas. Veamos un ejemplo:

d = {'a': 1, 'b': 2, 'c': 3}

print(list(d))

print(tuple(d))

print(set(d)) # Naturalmente, el diccionario por definicion no tiene repeticion.

Hay que usar las vistas keys(), values() y items() según la conversión que queramos realizar.

La funcion dict() no acepta cualquier argumento, como ya habiamos visto, solo acepta una lista de tuplas, donde cada tupla es un par llave-valor.

## \*\*Comprehensions o Comprensiones con Diccionarios\*\*

Hay algunas maneras cortas de inicializar listas o diccionarios que llamamos comprensiones:

# Comprensiones de Diccionarios

n = { x: x\*\*3 for x in range(0, 6) } # Aqui cada llave es un entero, es valido. Esto puede ser util para emular arreglos con indices diferentes.

print(n)

w = { str(x): x\*\*3 for x in range(0, 6)} # Aqui cada llave es un string.

print(w)

La ventaja de usar una comprension es indicar al interpretador una operacion iterativa en una sola linea. De esta manera, se evita usar ciclos for "tradicionales", que son menos eficientes, para generar listas o diccionarios.

## \*\*Copias de Estructuras de Datos\*\*

Los diccionarios y sets también deben copiarse usando el método copy propio de la estructura, o usar el módulo copy para crear una copia de la estructura y evitar hacer una referencia.