

PROGRAMACIÓN

Grado en Ingeniería Biomédica

TEMA . 7

Python – Diccionarios y Conjuntos



Diccionarios

- Un Diccionario es una estructura de datos y un tipo de dato en Python con características especiales que nos permite almacenar cualquier tipo de valor como enteros, cadenas, listas e incluso otras funciones.
- Para definir un diccionario, se encierra el listado de valores entre llaves {}. Responden a modelos clave-valor
- Las parejas de clave y valor se separan con comas, y la clave y el valor se separan con dos puntos ➔ clave:valor.
- La **clave es única** y tiene que ser de **tipos inmutables**: int, float, string, tuplas, bool. **Cuidado con los floats como claves!!**
- Los **valores** pueden ser de cualquier tipo, mutables e inmutables y pueden repetirse (no necesitan ser únicos).

Diccionarios

clave:valor

```
info_estudiante={"Nombre": "Carlos", "Apellidos": "Rodriguez Martínez",  
                 "Grado": "Ing. Biomedica",  
                 "Asignaturas": ["Algebra", "Cálculo", "Química", "Programación"]}  
print(info_estudiante)  
print(info_estudiante["Apellidos"],",", info_estudiante["Nombre"])  
print(info_estudiante["Asignaturas"])  
for asig in info_estudiante["Asignaturas"]:  
    print (asig)
```

```
{'Nombre': 'Carlos', 'Apellidos': 'Rodriguez Martínez', 'Grado': 'Ing. Biomedica',  
'Asignaturas': ['Algebra', 'Cálculo', 'Química', 'Programación']}
```

Rodriguez Martínez , Carlos

['Algebra', 'Cálculo', 'Química', 'Programación']

Algebra

Cálculo

Química

Programación

Métodos Diccionarios

- Constructor dict() crea un diccionario en Python
- Si es factible se crea un diccionario

Fijate en la sintaxis!!

```
necesidades=dict(material=["lapiz", "tijeras", "papel"], clase=4 )  
print(type(necesidades))  
print(necesidades)  
print(necesidades["material"])
```

```
dic_vacio=dict()  
print("Diccionario vacio",dic_vacio)
```

```
<class 'dict'>  
{'material': ['lapiz', 'tijeras', 'papel'], 'clase': 4}  
['lapiz', 'tijeras', 'papel']  
Diccionario vacio {}
```

Métodos Diccionarios

<code>clear()</code>	<code>copy()</code>	<code>fromkeys()</code>	<code>get()</code>
<code>items()</code>	<code>keys()</code>	<code>pop()</code>	<code>popitem()</code>
<code>setdefault()</code>	<code>update()</code>	<code>values()</code>	

- **items():** Devuelve una **lista de tuplas**, cada tupla se compone de dos elementos: el primero será la clave y el segundo, su valor.
- **keys():** Devuelve una lista de elementos con las claves del diccionario.
- **values():** Devuelve una lista de elementos con los valores del diccionario.
- **clear():** Elimina todos los elementos del diccionario dejándolo vacío

Métodos Diccionarios

```
diccionario = {'a' : 1, 'b' : 2, 'c' : 3 , 'd' : 4}
recurso=diccionario.items()
print(recurso)
claves=diccionario.keys()
print(claves)
valores=diccionario.values()
print(valores)

dict_items([('a', 1), ('b', 2), ('c', 3), ('d', 4)])
dict_keys(['a', 'b', 'c', 'd'])
dict_values([1, 2, 3, 4])
```

Métodos Diccionarios

- **copy():** Devuelve una copia del diccionario original
- **fromkeys():** Recibe como parámetros un iterable y un valor, devolviendo un diccionario que contiene como claves los elementos del iterable con el mismo valor ingresado.
 - Si no se ingresa valor alguno, el método devolverá None para todas las claves

```
In [43]: dic = dict.fromkeys(['a','c','d','b'],(1,2,3))
```

```
In [44]: dic
```

```
Out[44]: {'a': (1, 2, 3), 'c': (1, 2, 3), 'd': (1, 2, 3), 'b': (1, 2, 3)}
```

Sin ingresar un valor en fromkeys()

```
In [45]: dic = dict.fromkeys(['a','c','d','b'])
```

```
In [46]: dic
```

```
Out[46]: {'a': None, 'c': None, 'd': None, 'b': None}
```

Métodos Diccionarios

- **get():** Recibe como parámetro una clave.
 - devuelve el valor de la clave.
 - Si no lo encuentra, devuelve un objeto None

```
In [47]: info_estudiante={"Nombre": "Carlos", "Apellidos": "Rodriguez Martínez",  
...:                     "Grado": "Ing. Biomedica",  
...:                     "Asignaturas": ["Algebra", "Cálculo", "Química", "Programación"]}
```

```
In [48]: info_estudiante.get("Grado")
```

```
Out[48]: 'Ing. Biomedica'
```

```
In [49]: info_estudiante.get("Curso")
```

```
In [50]: curso=info_estudiante.get("Curso")
```

```
In [51]: print(curso)
```

```
None
```


Métodos Diccionarios

- **pop():** Recibe como parámetro una clave, elimina esta y devuelve su valor. Si no lo encuentra, devuelve error.

```
In [52]: diccionario = {'a' : 1, 'b' : 2, 'c' : 3 , 'd' : 4}
```

```
In [53]: valor=diccionario.pop('b')
```

```
In [54]: print(valor)  
2
```

```
In [55]: valor=diccionario.pop('e')  
Traceback (most recent call last):
```

```
File "<ipython-input-55-45ae390696eb>", line 1, in <module>  
    valor=diccionario.pop('e')
```

```
KeyError: 'e'
```

Es posible
controlarlo
con
try:
except:

Métodos Diccionarios

- **setdefault()**. Funciona de dos formas:

- Idéntico a un get()

```
In [60]: dic = dict(nombre='nestor', apellido='Plasencia', edad=22)
```

```
In [61]: nombre=dic.setdefault('nombre')
```

```
In [62]: print(nombre)
nestor
```

- Agregar un nuevo elemento al diccionario

```
In [63]: dic = dict(nombre='nestor', apellido='Plasencia', edad=22)
```

```
In [64]: dic.setdefault("Asignaturas",["Física","Química","Álgebra", "Cálculo"])
```

```
Out[64]: ['Física', 'Química', 'Álgebra', 'Cálculo']
```

```
In [65]: print(dic)
```

```
{'nombre': 'nestor', 'apellido': 'Plasencia', 'edad': 22, 'Asignaturas': ['Física', 'Química', 'Álgebra', 'Cálculo']}
```

Métodos Diccionarios

- **update():** Recibe como parámetro otro diccionario. Si se tienen claves iguales, actualiza el valor de la clave repetida; si no hay claves iguales, este par clave-valor es agregado al diccionario.

```
In [4]: dic_1 = {'a' : 1, 'b' : 2, 'c' : 3 , 'd' : 4}
```

```
In [5]: dic_2 = {'c' : 6, 'b' : 5, 'e' : 9 , 'f' : 10}
```

```
In [6]: dic_1.update(dic_2)
```

```
In [7]: print(dic_1)
```

```
{'a': 1, 'b': 5, 'c': 6, 'd': 4, 'e': 9, 'f': 10}
```

Ejemplo Diccionarios

```
def words_frequencies(texto):
```

```
    #Eliminamos caracteres de puntuación
    texto_aux=""
    caracteres_puntuacion=":;,.»«?»!;\\"
    for c in texto:
        if c in caracteres_puntuacion:
            continue
        else:
            texto_aux+=c

    texto_aux=texto_aux.lower() #Conversión a minúsculas
    list_words=texto_aux.split() #Lista de palabras
    myDict={}
    for word in list_words:
        if word in myDict:
            myDict[word] += 1
        else:
            myDict[word] = 1
    return myDict
```

```
def most_common_words(myDict):
```

```
    frequency=myDict.values()
    max_frequency=max(frequency)

    words=[]
    for clave in myDict:
        if myDict[clave]==max_frequency:
            words.append(clave)

    return(words,max_frequency)
```

Ejemplos Diccionarios (cont.)

```
#####
nombre_fichero=input("Nombre del fichero a analizar: ")
try:
    fichero=open(nombre_fichero,"r") # Apertura
    texto=fichero.read()             # Lectura
    fichero.close()                   # Cierre

    diccionario=words_frequencies(texto) # Creación de diccionario

    word_max,maximo=most_common_words(diccionario) # Más frecuentes

    if(len(word_max)==1):
        print("La palabra \"",word_max[0], "\" es la más frecuente")
    else:
        print("Las palabras", word_max, "son las más frecuentes")

    print("Las palabras encontradas y su frecuencia son: ")
    for word in sorted(diccionario):
        print(f'{word:12} ==> {diccionario.get(word):4d}')

except IOError:
    print("El fichero %s no se ha podido abrir"% nombre_fichero)
```

Ejemplos Diccionarios (cont.)

Nombre del fichero a analizar: Piratas.txt
La palabra " en " es la más frecuente
Las palabras encontradas y su frecuencia son:

a	==>	3
al	==>	2
alcanza	==>	1
alegre	==>	1
allá	==>	1
alza	==>	1
asia	==>	1
azul	==>	1
bajel	==>	1
banda	==>	1
bergantín	==>	1
blando	==>	1
bonanza	==>	1
bravura	==>	1
cantando	==>	1
capitán	==>	1
cañones	==>	1
cien	==>	1
con	==>	1
confin	==>	1
conocido	==>	1
corta	==>	1
.		.

de	==>	1
del	==>	2
despecho	==>	1
diez	==>	1
el	==>	5
en	==>	6
enemigo	==>	1
europa	==>	1
frente	==>	1
gime	==>	1
han	==>	1
hecho	==>	1
hemos	==>	1
inglés	==>	1
la	==>	3
lado	==>	1
llaman	==>	1
lona	==>	1
luna	==>	1
mar	==>	3
mi	==>	1
mio	==>	1
mis	==>	1
movimiento	==>	1
naciones	==>	1
navega	==>	1
navío	==>	1

ni	==>	4
no	==>	1
olas	==>	1
otro	==>	2
pendones	==>	1
pies	==>	1
pirata	==>	2
plata	==>	1
popa	==>	2
por	==>	2
presas	==>	1
que	==>	2
rendido	==>	1
riela	==>	1
rumbo	==>	1
sin	==>	1
sino	==>	1
stambul	==>	1
su	==>	2
sujetar	==>	1
sus	==>	1
temido	==>	1
temor	==>	1
toda	==>	1
todo	==>	1
torcer	==>	1

continua ...

Listas vs. Diccionarios

Listas:

- Secuencia “ordenada” de elementos
- Los elementos de la lista se buscan con un índice entero, i.e. `lista[4]`
- Los índices tienen un orden
- Índices tienen que ser enteros

Diccionarios:

- Empareja claves con valores
- Busca un elemento por medio de la clave:
`myDict[“Nombre”]`
- No se garantiza ningún orden
- Las claves pueden ser de cualquier tipo inmutable

Conjuntos

- Un conjunto es una colección no ordenada de objetos únicos.
- Hay dos tipos de conjuntos:
 - Conjuntos mutables → `s1=set([1,2,3,5])`
 - Conjuntos inmutables → `s2=frozenset([2,3,5,True])`
- Para crear un conjunto mutable también se puede especificar sus elementos entre llaves: `s={1,3,5,7,True}`
- Los elementos de un conjunto pueden ser de distintos tipos y siempre inmutables

`s = {True, 3.14, None, False, "Hola mundo", (1, 2)}`
- No puede incluir objetos mutables como listas, diccionarios, e incluso otros conjuntos mutables.

Un argumento, por ejemplo una lista

Conjuntos

- Para generar un conjunto vacío, directamente creamos una instancia de la clase set → `s=set()`, **{}** **está reservado para un diccionario vacío**.
- Podemos obtener un conjunto a partir de cualquier objeto iterable

`s1=set([2,3,4,5])` → No está incluyendo una lista, `s1={2,3,4,5}`

`s2=set(range(10))`

```
In [15]: s2=frozenset(range(9))
```

```
In [16]: s2
```

```
Out[16]: frozenset({0, 1, 2, 3, 4, 5, 6, 7, 8})
```

```
In [17]: type(s2)
```

```
Out[17]: frozenset
```

s2 es un conjunto
inmutable

```
In [18]: s3={s2,1}
```

```
In [19]: s3
```

```
Out[19]: {1, frozenset({0, 1, 2, 3, 4, 5, 6, 7, 8})}
```

Métodos Conjuntos

SET

add
clear
copy
difference
difference_update
discard
intersection
intersection_update
isdisjoint
issubset
issuperset
pop
remove
symmetric_difference
symmetric_difference_update
union
update

FROZENSET

copy
difference
intersection
isdisjoint
issubset
issuperset
symmetric_difference
union

Métodos Conjuntos

- **add():** agrega un elemento a un **conjunto mutable**. Esto no tiene efecto si el elemento ya está presente.
- **clear():** vacía un **conjunto mutable**.
- **copy():** devuelve una copia de un **conjunto mutable o inmutable**.

```
In [36]: s1={2,4,True,'Pan'}
```

```
In [37]: s1_copy=s1.copy()
```


```
In [38]: s1==s1_copy
```

```
Out[38]: True
```

```
In [39]: s1_copy.add(89)
```

```
In [40]: s1==s1_copy
```

```
Out[40]: False
```

```
In [41]: s1=s1_copy #alias 
```

```
In [42]: s1==s1_copy
```

```
Out[42]: True
```

```
In [43]: s1.clear()
```

```
In [44]: s1
```

```
Out[44]: set()
```

Métodos Conjuntos

- **difference():** devuelve la diferencia entre dos **conjunto mutable** o **conjunto inmutable**: todos los elementos que están en el primero, pero no en el argumento del método.
- **difference_update():** actualiza un tipo **conjunto mutable** llamando al método `difference_update()` con la diferencia de los conjuntos.

```
In [46]: set_mutable1 = set([4, 3, 11, 7, 5, 2, 1, 4])
....:
....: set_mutable2 = set([11, 5, 9, 2, 4, 8])
```

```
In [47]: print(set_mutable1.difference(set_mutable2))
{1, 3, 7}
```

```
In [48]: print(set_mutable2.difference(set_mutable1))
{8, 9}
```

```
In [49]: set_mutable1
Out[49]: {1, 2, 3, 4, 5, 7, 11}
```

```
In [50]: set_mutable1.difference_update(set_mutable2)
```

```
In [51]: set_mutable1
Out[51]: {1, 3, 7}
```

No modifica los
conjuntos por
eso es válido
para conjuntos
mutables o no
mutables

Modifica el
conjunto

Métodos Conjuntos

- **discard():** elimina un elemento de un **conjunto mutable** si está presente.

```
>>> paquetes = {'python', 'zope', 'plone', 'django'}
>>> paquetes
set(['python', 'zope', 'plone', 'django'])
>>> paquetes.discard('django')
>>> paquetes
set(['python', 'zope', 'plone'])
```

```
>>> paquetes = {'python', 'zope', 'plone', 'django'}
>>> paquetes.discard('php')
>>> paquetes
set(['python', 'zope', 'plone'])
```

El elemento a eliminar no está presente en el conjunto

Métodos Conjuntos

- **intersection()**: devuelve la intersección entre dos **conjunto mutable** o **conjunto inmutable**: todos los elementos que están en ambos conjuntos.
- **intersection_update()**: actualiza un tipo **conjunto mutable** con los elementos comunes de los conjuntos.

```
In [58]: set_mutable1 = set([4, 3, 11, 7, 5, 2, 1, 4])  
...: set_mutable2 = set([11, 5, 9, 2, 4, 8])
```

```
In [59]: print(set_mutable1.intersection(set_mutable2))  
{2, 11, 4, 5}
```

```
In [60]: print(set_mutable2.intersection(set_mutable1))  
{2, 11, 4, 5}
```

```
In [61]: set_mutable1.intersection_update(set_mutable2)
```

```
In [62]: set_mutable1  
Out[62]: {2, 4, 5, 11}
```

No modifica los
conjuntos por eso es
válido para conjuntos
mutables o no
mutables

← Modifica el
conjunto

Métodos Conjuntos

- **isdisjoint():** devuelve el valor True si no hay elementos comunes entre los **conjuntos mutables** o **conjuntos inmutables**, en caso contrario devuelve False.
- **issubset():** devuelve el valor True si el conjunto sobre el que aplica el método es un *subconjunto* del conjunto dado como argumento, en caso contrario devuelve False
- **issuperset():** devuelve el valor True si el conjunto sobre el que aplica el método es un *superset* del conjunto dado como argumento, en caso contrario devuelve False.

```
In [68]: s1=frozenset([1,2,3])
```

```
In [69]: s2=frozenset([1,3,5,6])
```

```
In [70]: s1.isdisjoint(s2)
```

```
Out[70]: False
```

```
In [71]: s1.issubset(s2)
```

```
Out[71]: False
```

```
In [72]: s1.issuperset(s2)
```

```
Out[72]: False
```

```
In [73]: s2.issuperset(s1)
```

```
Out[73]: False
```

```
In [74]: s1=frozenset([1,2,3])
```

```
In [75]: s2=frozenset([1,3,5,6,2])
```

```
In [76]: s1.isdisjoint(s2)
```

```
Out[76]: False
```

```
In [77]: s1.issuperset(s2)
```

```
Out[77]: False
```

```
In [78]: s1.issubset(s2)
```

```
Out[78]: True
```

Métodos Conjuntos

- **pop()**: devuelve aleatoriamente un elemento de **conjunto mutable**. El método pop() no tiene argumentos. Si el **conjunto mutable** esta vacío se lanza una excepción KeyError.
- **remove()**: elimina un elemento de un **conjunto mutable**, si el elemento a eliminar no está en el conjunto se produce una excepción KeyError.

```
In [79]: set_mutable1 = set([4, 3, 11, 7, 5, 2, 1,9])
```

```
In [80]: set_mutable1.pop()
```

```
Out[80]: 1
```

```
In [81]: set_mutable1
```

```
Out[81]: {2, 3, 4, 5, 7, 9, 11}
```

```
In [82]: set_mutable1.remove(4)
```

```
In [83]: set_mutable1
```

```
Out[83]: {2, 3, 5, 7, 9, 11}
```

```
In [84]: set_mutable1.remove(6)
```

```
Traceback (most recent call last):
```

```
File "<ipython-input-84-e823ea16c245>", line 1, in <module>
```

```
    set_mutable1.remove(6)
```

```
KeyError: 6
```

Es equivalente a discard() con la salvedad de que discard() no produce una excepción cuando el conjunto no contiene el elemento a eliminar.

Métodos Conjuntos

- **`symmetric_difference()`**: devuelve los elementos que están en un **conjunto (mutable o no)** u otro, pero no en ambos.

La diferencia simétrica de dos conjuntos es el conjunto de elementos que están en cualquiera de los conjuntos pero no en ambos.

- **`symmetric_difference_update()`**: actualiza un **conjunto mutable** con la diferencia simétrica de los conjuntos.

```
In [96]: set_mutable1 = set([4, 3, 11, 7, 5, 2, 1, 4])
...: set_mutable2 = set([11, 5, 9, 2, 4, 8])
```

```
In [97]: print(set_mutable1.symmetric_difference(set_mutable2))
{1, 3, 7, 8, 9}
```

```
In [98]: print(set_mutable2.symmetric_difference(set_mutable1))
{1, 3, 7, 8, 9}
```

```
In [99]: set_mutable1.symmetric_difference_update(set_mutable2)
```

```
In [100]: set_mutable1
Out[100]: {1, 3, 7, 8, 9}
```

No modifica los conjuntos por eso es válido para conjuntos mutables o no mutables

← Modifica el conjunto

Métodos Conjuntos

- **union()**: devuelve un **conjunto** con todos los elementos que están en alguno de los **conjuntos** (mutables o inmutables).

```
set_mutable1 = set([4, 3, 11, 7, 5, 2, 1, 4])  
set_mutable2 = set([11, 5, 9, 2, 4, 8])
```

```
set_mutable1.union(set_mutable2)  
{1, 2, 3, 4, 5, 7, 8, 9, 11}
```

- **update()**: agrega elementos a un **conjunto mutable**, los argumentos del método pueden ser de los siguientes tipos: tupla, lista, diccionario o un conjunto mutable.

Métodos Conjuntos

```
In [20]: lista=[4,5,67,9]
```

```
In [21]: s=set(lista)
```

```
In [22]: lista2=[3,5]
```

```
In [23]: s.update(lista2)
```

```
In [24]: s
```

```
Out[24]: {3, 4, 5, 9, 67}
```

```
In [25]: s2={89,98}
```

```
In [26]: s.update(s2)
```

```
In [27]: s|
```

```
Out[27]: {3, 4, 5, 9, 67, 89, 98}
```

```
In [28]: s2=frozenset([2,23])
```

```
In [29]: s.update(s2)
```

```
In [30]: s
```

```
Out[30]: {2, 3, 4, 5, 9, 23, 67, 89, 98}
```

```
In [31]: d={'a':2,'b':3,'d':4}
```

```
In [32]: s.update(d)
```

```
In [33]: s
```

```
Out[33]: {2, 23, 3, 4, 5, 67, 89, 9, 98, 'a', 'b', 'd'}
```

Ejemplos con el método update. Se han utilizado:

- Listas
- Conjuntos mutables
- Conjuntos inmutables
- Diccionarios, en estos últimos lo que añade son las claves, si quisiera los valores antes tendría que obtener los valores con el método values() del objeto diccionario.

Función integrada zip()

- La función zip() es una función integrada (built-in) que actúa sobre un conjunto de iterables (tuplas, listas, diccionarios, conjuntos) y genera un objeto zip mapeando los índices de los iterables pasados por argumento.

Sintaxis: zip(iterable1, iterable2, iterable3, iterable4,...)

```
In [39]: x = ("Joey", "Monica", "Ross")
...:
...: y = ("Chandler", "Pheobe")
...:
...: z = ("David", "Rachel", "Courtney")
...:
...: result = zip(x, y, z)
...:
...: print(result)
...:
...: print(tuple(result))
<zip object at 0x0000024D2D99C748>
(('Joey', 'Chandler', 'David'), ('Monica', 'Pheobe', 'Rachel'))
```

Función integrada zip()

Ejemplos:

```
In [40]: coin = ('Bitcoin', 'Ether', 'Ripple', 'Litecoin')
...:
...: code = ('BTC', 'ETH', 'XRP', 'LTC')
```

```
In [41]: criptomonedas=dict(zip(coin,code))
```

```
In [42]: criptomonedas
```

```
Out[42]: {'Bitcoin': 'BTC', 'Ether': 'ETH', 'Ripple': 'XRP', 'Litecoin': 'LTC'}
```

Iterador múltiple dentro de un for:

```
names = ['Alice', 'Bob', 'Charlie']
ages = [24, 50, 18]

for name, age in zip(names, ages):
    print(name, age)
# Alice 24
# Bob 50
# Charlie 18
```

Función integrada enumerate()

- La función enumerate() es una función integrada (built-in) que actúa sobre un iterable (tuplas, listas, diccionarios, conjuntos) y genera un objeto enumerate que añade al iterable un contador.

Sintaxis: enumerate (iterable, start=0)

```
In [61]: grocery = ['bread', 'milk', 'butter']
...: enumerateGrocery = enumerate(grocery)
...:
...: print(type(enumerateGrocery))
...:
...: # converting to List
...: print(list(enumerateGrocery))
...:
...: # changing the default counter
...: enumerateGrocery = enumerate(grocery, 10)
...: print(list(enumerateGrocery))
<class 'enumerate'>
[(0, 'bread'), (1, 'milk'), (2, 'butter')]
[(10, 'bread'), (11, 'milk'), (12, 'butter')]
```

Función integrada enumerate()

Ejemplos:

```
In [56]: Factory=("Siemens", "General Electric", "Toshiba", "Samsung")
```

```
In [57]: Factory_sort=sorted(Factory)
```

```
In [58]: for i,j in enumerate(Factory_sort,1):
```

```
...:     print(i,j)
```

```
...:
```

```
1 General Electric
```

```
2 Samsung
```

```
3 Siemens
```

```
4 Toshiba
```