

Diccionarios en Python

¿Qué Es Una Colección?



- Una colección es genial porque podemos poner más de un valor en ella y moverla alrededor en un paquete muy conveniente
- Tenemos un grupo de valores en una sola “variable”
- Hacemos esto al tener más de un solo lugar “en” la variable
- Tenemos forma de definir los diferentes lugares en la variable

¿Qué No Es Una “Colección”?

- La mayoría de nuestras **variables** tienen un único valor en ellas – cuando ponemos un nuevo valor en la **variable** – el valor anterior se sobrescribe

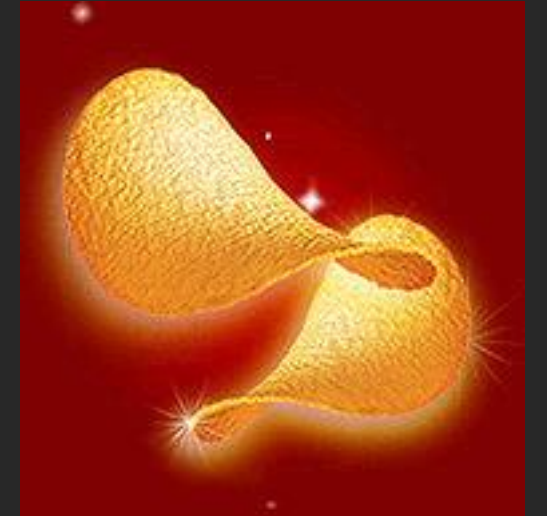
```
$ python
>>> x = 2
>>> x = 4
>>> print(x)
4
```



Una Historia De Dos Colecciones...

- Lista

- Una colección lineal de valores que mantienen un orden



- Diccionario

- Una “bolsa” de valores, cada uno con una etiqueta



Diccionarios



https://es.wikipedia.org/wiki/Vector_asociativo

Diccionarios



- Los diccionarios son la colección de datos más poderosa de Python
- Los diccionarios nos permiten hacer operaciones rápidas similares a una base de datos en Python
- Los diccionarios tienen diferentes nombres en diferentes lenguajes
 - Vectores Asociativos - Perl / PHP
 - Propiedades o Mapas o HashMap - Java
 - Bolsa de Propiedades - C# / .Net

Diccionarios

- Las listas **indexan** sus entradas basadas en la posición en la lista
- Los **Diccionarios** son como bolsas – no tienen orden
- Así que **indexamos** las cosas que ponemos en un **diccionario** con una “etiqueta de búsqueda”

```
>>> bolsa = dict()
>>> bolsa['dinero'] = 12
>>> bolsa['dulce'] = 3
>>> bolsa['papel'] = 75
>>> print(bolsa)
{'dinero': 12, 'papel': 75, 'dulce': 3}
>>> print(bolsa['dulce'])
3
>>> bolsa['dulce'] = bolsa['dulce'] + 2
>>> print(bolsa)
{'dinero': 12, 'papel': 75, 'dulce': 5}
```

Comparación de Listas y Diccionarios

Los **Diccionarios** son como **listas** a excepción de que utilizan **claves** en vez de números para buscar **valores**

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

```
>>> ddd = dict()
>>> ddd['edad'] = 21
>>> ddd['curso'] = 182
>>> print(ddd)
{'curso': 182, 'edad': 21}
>>> ddd['edad'] = 23
>>> print(ddd)
{'curso': 182, 'edad': 23}
```



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

```
>>> ddd = dict()
>>> ddd['edad'] = 21
>>> ddd['curso'] = 182
>>> print(ddd)
{'curso': 182, 'edad': 21}
>>> ddd['edad'] = 23
>>> print(ddd)
{'curso': 182, 'edad': 23}
```

Lista

Clave	Valor
-------	-------

[0]	21
[1]	183

lst

Diccionario

Clave	Valor
-------	-------

['curso']	182
['edad']	21

ddd

Literales de Diccionarios (Constantes)

- Las literales de diccionarios se escriben con llaves y tienen una lista en par tipo **clave** : **valor**
- Puedes inicializar un **diccionario vacío** escribiendo corchetes vacíos

```
>>> jjj = { 'chuck' : 1 , 'fred' : 42, 'jan': 100 }
>>> print(jjj)
{'jan': 100, 'chuck': 1, 'fred': 42}
>>> ooo = { }
>>> print(ooo)
{}
>>>
```

¿El Nombre Más Común?

¿El Nombre Más Común?

marquard

cwen

cwen

zhen

marquard

zhen

csev

zhen

csev

marquard

zhen

csev

zhen

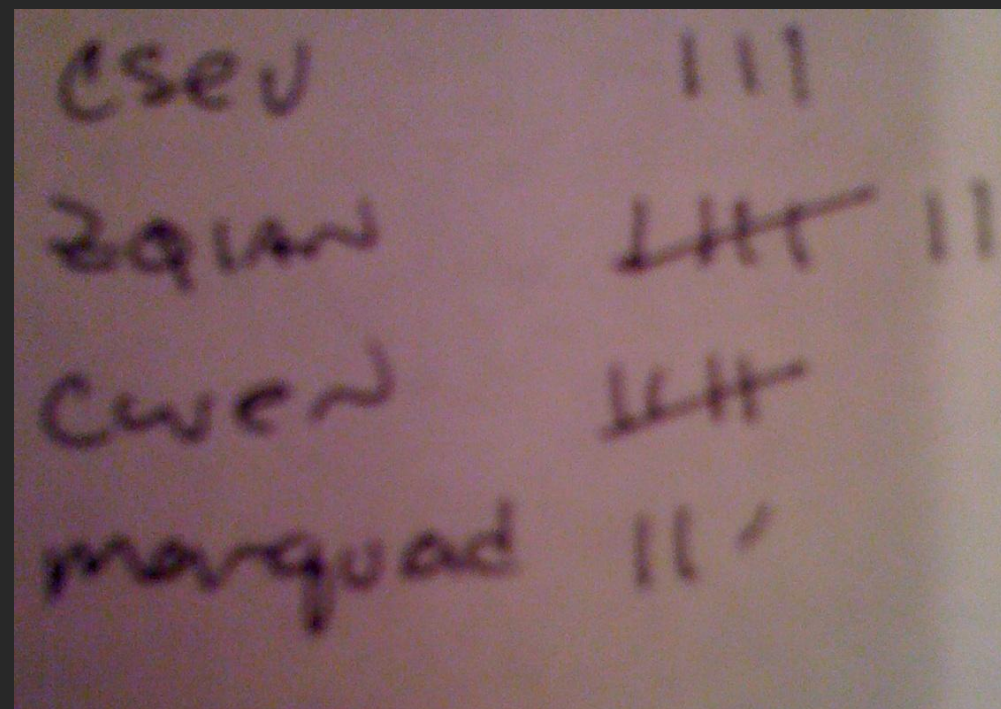
¿El Nombre Más Común?

marquard

cwen

cwen

zhen



zhen

csev

csev

marquard

zhen

csev

zhen

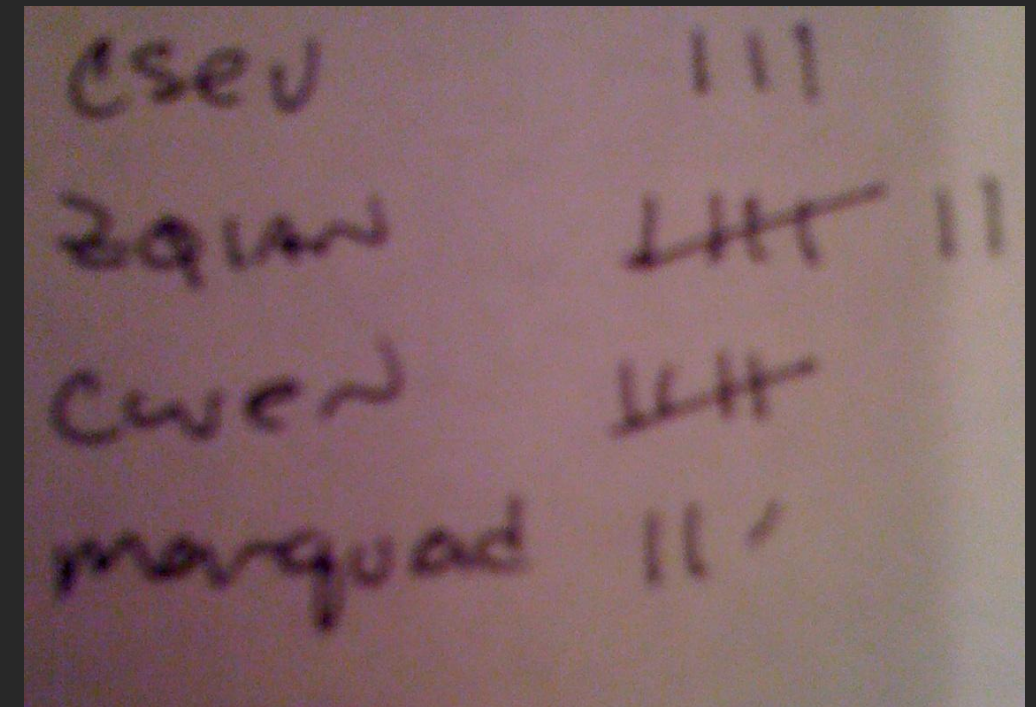
Múltiples Contadores con un Diccionario

Un uso común de diccionarios es **contar** con qué frecuencia “vemos” algo

```
>>> ccc = dict()
>>> ccc['csev'] = 1
>>> ccc['cwen'] = 1
>>> print(ccc)
{'csev': 1, 'cwen': 1}
>>> ccc['cwen'] = ccc['cwen'] + 1
>>> print(ccc)
{'csev': 1, 'cwen': 2}
```

Clave

Valor



A photograph of a piece of paper with handwritten entries and tally marks, illustrating the concept of counting frequencies. The entries are 'csev', 'cwen', and 'marquard'. 'csev' has three vertical tally marks, 'cwen' has two, and 'marquard' has two. There is also a crossed-out entry 'Zqian' with two vertical tally marks.

csev	
Zqian	
cwen	
marquard	

Errores de Diccionarios

- Es un **error** hacer referencia a una clave que no existe en un diccionario
- Podemos usar el operador **in** para comprobar si una clave se encuentra en un diccionario

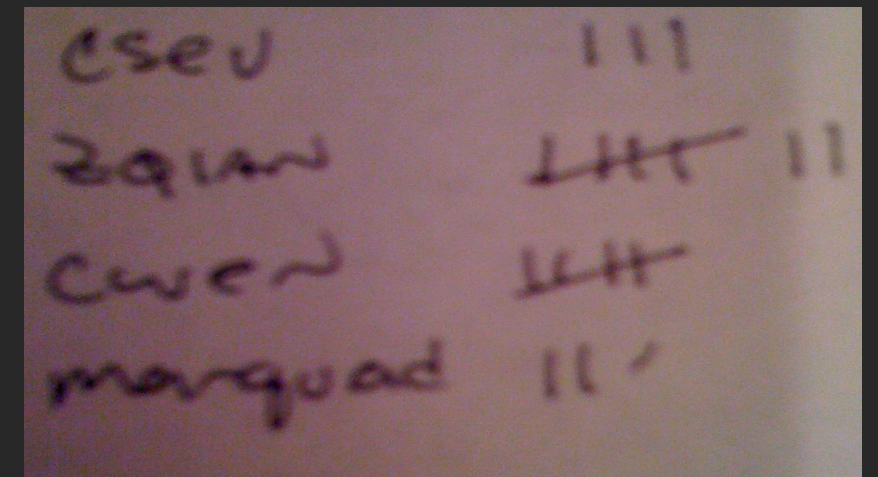
```
>>> ccc = dict()
>>> print(ccc['csev'])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'csev'
>>> 'csev' in ccc
False
```

Cuando Encontramos un Nuevo Valor

Cuando encontramos un nuevo **nombre**, necesitamos agregar una nueva entrada en el **diccionario** y si es la segunda vez o después encontramos de nuevo el **nombre**, simplemente sumamos uno al contador en el **diccionario** bajo ese **nombre**

```
contadores = dict()
nombres = ['csev', 'cwen', 'csev', 'zqian', 'cwen']
for nombre in nombres:
    if nombre not in contadores:
        contadores[nombre] = 1
    else:
        contadores[nombre] = contadores[nombre] + 1
print(contadores)
```

{'csev': 2, 'zqian': 1, 'cwen': 2}



El Método `get` de un Diccionario

El patrón de verificar si una **clave** ya existe en un diccionario y asumir un valor por defecto si la **clave** no se encuentra es tan común, que hay un **método** llamado `get()` que hace esto por nosotros

Valor por defecto si la clave no existe (y no produce errores).

```
if nombre in contadores:  
    x = contadores[nombre]  
else :  
    x = 0
```

```
x = contadores.get(name, 0)
```

```
{'csev': 2, 'zqian': 1, 'cwen': 2}
```

Conteo Simplificado usando

get()

Podemos usar `get()` y proveer un **valor por defecto de cero** cuando la **clave** no existe aún en el diccionario - y después sumar uno

```
contadores = dict()
nombres = ['csev', 'cwen', 'csev', 'zqian', 'cwen']
for nombre in nombres :
    contadores[nombre] = contadores.get(nombre, 0) + 1
print(contadores)
```

Valor por
defecto

`{'csev': 2, 'zqian': 1, 'cwen': 2}`

Conteo Simplificado usando get()

```
contadores = dict()  
nombres = ['csev', 'cwen', 'csev', 'zqian', 'cwen']  
for nombre in nombres :  
    contadores[nombre] = contadores.get(nombre, 0) + 1  
print(contadores)
```



<http://www.youtube.com/watch?v=EHJ9uYx5L58>

Conteo de Palabras en un Texto

Escribir programas (o programar) es una actividad muy creativa y gratificante. Puedes escribir programas por muchas razones, desde resolver un problema complicado de análisis de datos hasta pasar un rato divertido con alguien resolviendo un problema. Este curso asume que todos necesitan saber cómo programar, y que, una vez que aprendes a programar, serás capaz de encontrar qué quieres hacer con ese nuevo conocimiento.

En nuestra vida diaria nos encontramos rodeados de computadoras, desde computadoras portátiles hasta teléfonos celulares. Podemos pensar en esas computadoras como “asistentes personales” que pueden ocuparse de muchas cosas por nosotros. El hardware en las computadoras de hoy es esencialmente construido para preguntarnos continuamente, “¿Qué te gustaría que haga ahora?”

Nuestras computadoras son rápidas y tienen cantidades grandes de memoria, y pueden sernos muy útiles solamente si sabemos hablar el lenguaje correcto para explicarle a la computadora lo que queremos que haga ahora. Si supiéramos este lenguaje podríamos decirle a la computadora que se encargue de las tareas que repetimos con frecuencia. Es interesante saber que las cosas que las computadoras pueden hacer mejor son con frecuencia las cosas que los humanos encontramos aburridas y poco interesantes.

Patrón del Contador

```
contadores = dict()
print('Ingresa una línea de texto:')
lineaa = input(' ')

palabras = lineaa.split()

print('Palabras:', palabras)

print('Contando...')
for palabra in palabras:
    contadores[palabra] = contadores.get(palabra, 0) + 1
print('Contadores', contadores)
```

El patrón general para contar las palabras en una línea de texto es **dividir** la línea en palabras, y después recorrer las palabras y usar un **diccionario** para mantener la cuenta de cada palabra de forma independiente.

```
python contador_palabras.py
```

Ingresa una línea de texto:

```
el payaso corrio detras del carro y el carro corrio dentro  
de la tienda y la tienda cayo sobre el payaso y el carro
```

```
Palabras: ['el', 'payaso', 'corrio', 'detras', 'del',  
'carro', 'y', 'el', 'carro', 'corrio', 'dentro', 'de',  
'la', 'tienda', 'y', 'la', 'tienda', 'cayo', 'sobre',  
'el', 'payaso', 'y', 'el', 'carro']
```

Contando...

```
Contadores {'el': 4, 'payaso': 2, 'corrio': 2, 'detras':  
1, 'del': 1, 'carro': 3, 'y': 3, 'dentro': 1, 'de': 1,  
'la': 2, 'tienda': 2, 'cayo': 1, 'sobre': 1}
```



```
contadores = dict()
lineaa = input('Ingresa una línea de
texto:')
palabras = lineaa.split()

print('Palabras:', palabras)
print('Contando...')

for palabra in palabras:
    contadores[palabra] =
contadores.get(palabra,0) + 1
print('Contadores', contadores)
```



```
python contador_palabras.py
Ingresa una línea de texto:
el payaso corrio detras del carro y el carro
corrio dentro de la tienda y la tienda cayo
sobre el payaso y el carro
```

```
Palabras: ['el', 'payaso', 'corrio', 'detras',
'del', 'carro', 'y', 'el', 'carro', 'corrio', 'dentro',
'de', 'la', 'tienda', 'y', 'la', 'tienda', 'cayo',
'sobre', 'el', 'payaso', 'y', 'el', 'carro']
Contando...
Contadores {'el': 4, 'payaso': 2, 'corrio': 2,
'detras': 1, 'del': 1, 'carro': 3, 'y': 3, 'dentro':
1, 'de': 1, 'la': 2, 'tienda': 2, 'cayo': 1, 'sobre':
1}
```

Bucles Finitos y Diccionarios

A pesar de que los **diccionarios** no se almacenan en orden, podemos escribir un bucle **for** que recorre todas las **entradas** en un **diccionario** – de hecho recorre todas las **claves** en el **diccionario** y **busca** los valores

```
>>> contadores = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
>>> for clave in contadores:
...     print(clave, contadores[clave])
...
jan 100
chuck 1
fred 42
>>>
```

Recuperando listas de Claves y Valores

Puedes obtener una lista de **claves**, **valores**, o **ítems (ambos)** de un diccionario

```
>>> jjj = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
>>> print(list(jjj))
['jan', 'chuck', 'fred']
>>> print(jjj.keys())
['jan', 'chuck', 'fred']
>>> print(jjj.values())
[100, 1, 42]
>>> print(jjj.items())
[('jan', 100), ('chuck', 1), ('fred', 42)]
>>>
```

¿Qué es una
“tupla”? - próximamente...



Bonus: Dos Variables de Iteración!

- Iteramos a través de los pares **clave-valor** en un diccionario usando ***dos*** variables de iteración

```
jjj = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}  
for aaa,bbb in jjj.items() :  
    print(aaa, bbb)
```

- En cada iteración, la primera variable es la **clave** y la segunda variable es el **valor** correspondiente a la clave

```
jan 100  
chuck 1  
fred 42
```

aaa	bbb
[jan]	100
[chuck]	1
[fred]	42

```
nombre = input('Ingresa un nombre de archivo:')  
manejador = open(nombre)
```

```
contadores = dict()  
for linea in manejador:  
    palabras = linea.split()  
    for palabra in palabras:  
        contadores[palabra] =  
contadores.get(palabra,0) + 1
```

```
grancontador = None  
granpalabra = None  
for palabra,contador in contadores.items():  
    if grancontador is None or contador >  
grancontador:  
        granpalabra = palabra  
        grancontador = contador  
  
print(granpalabra, grancontador)
```

```
python palabras.py  
Enter file: palabras.txt  
a 16
```

```
python palabras.py  
Enter file: payaso.txt  
el 4
```

Usando dos bucles anidados

Resumen

- ¿Qué es una “colección”?
- Listas contra Diccionarios
- Constantes de Diccionarios
- La palabra más común
- Usando el método `get()`
- Indexado y falta de orden
- Escribiendo bucles de diccionarios
- Un vistazo: tuplas
- Ordenando diccionarios