

# Estructura de datos en Python – Diccionarios

## 1. Introducción

Un diccionario Python es una colección de elementos, similar a las listas y tuplas. Sin embargo, a diferencia de las listas y las tuplas, cada elemento de un diccionario es un par clave-valor (formado por una clave y su valor).

Las claves del diccionario deben ser inmutables, como tuplas, cadenas, enteros, etc. No podemos utilizar como claves objetos mutables (modificables) como las listas.

Las claves de un diccionario deben ser únicas.

Vamos a continuación a estudiar los diccionarios en Python.

## 2. Diccionarios – dict()

Los diccionarios pueden utilizarse para realizar diversas operaciones. Su mayor fuerte es la búsqueda de elementos.

Veamos las operaciones a realizar con los diccionarios.

### Crear un diccionario

En Python podemos crear un diccionario vacío de dos formas distintas tal y como se muestra en la siguiente figura:

<pre>9  dicc01 = dict() 10 dicc02 = {} 11 print(f'dicc01 -&gt; {dicc01}') 12 print(f'dicc02 -&gt; {dicc02}')</pre>	<pre>dicc01 -&gt; {} dicc02 -&gt; {}</pre>
--	--

La función dict() sin argumentos, permite crear un diccionario vacío. En la línea 10 se muestra otra forma de crear una lista vacía colocando una llave abierta y otra cerrada.

Para crear un diccionario con valores se añaden pares clave: valor entre llaves {}, separados por comas. Veamos manera de como hacerlo.

```
16 dicc01 = {'a':1, 'b':2, 'c':3}
17 print(f'dicc01 -> {dicc01}')
```

```
dicc01 -> {'a': 1, 'b': 2, 'c': 3}
```

```
19 country_capitals = {
20     "Germany": "Berlin",
21     "Canada": "Ottawa",
22     "England": "London"
23 }
24
25 for dato in country_capitals:
26     print(f'{dato:8} : {country_capitals[dato]}')
```

```
Germany : Berlin
Canada  : Ottawa
England  : London
```

```
28 dicc01 = {'a':1, 'b':2, 'c':3}
29 dicc02 = {1:'a', 2:'b', 3:'a'}
30 dicc03 = {(1, 'a'):3, (2, 'a'):7}
31 print(f'dicc01 -> {dicc01}')
```

```
dicc01 -> {'a': 1, 'b': 2, 'c': 3}
dicc02 -> {1: 'a', 2: 'b', 3: 'a'}
dicc03 -> {(1, 'a'): 3, (2, 'a'): 7}
```

```
32 print(f'dicc02 -> {dicc02}')
```

```
33 print(f'dicc03 -> {dicc03}')
```

Las claves del diccionario deben ser inmutables, como tuplas, cadenas, enteros, etc. No podemos utilizar como claves objetos mutables (modificables) como las listas.

Las claves de un diccionario deben ser únicas. Si hay claves duplicadas, el valor posterior de la clave sobrescribe el valor anterior.

```
39 dicc01 = {'a':1, 'b':2, 'c':3, 'a':999}
40 print(f'dicc01 -> {dicc01}')
```

```
dicc01 -> {'a': 999, 'b': 2, 'c': 3}
```

```
42 hogwarts_houses = {
43     "Harry Potter": "Gryffindor",
44     "Hermione Granger": "Gryffindor",
45     "Ron Weasley": "Gryffindor",
46     # duplicate key with a different house
47     "Harry Potter": "Slytherin"
48 }
49 for dato in hogwarts_houses:
50     print(f'{dato:8} : {hogwarts_houses[dato]}')
```

```
Harry Potter : Slytherin
Hermione Granger : Gryffindor
Ron Weasley : Gryffindor
```

Aquí, la clave Harry Potter se asigna por primera vez a Gryffindor. Sin embargo, hay una segunda entrada en la que Harry Potter se asigna a Slytherin.

Como en un diccionario no se permiten claves duplicadas, la última entrada Slytherin sobrescribe el valor anterior Gryffindor.

## Acceder a los elementos de un diccionario

Podemos acceder al valor de una clave del diccionario colocand la clave entre corchetes.

<pre>54 dicc01 = {'a':1,'b':2,'c':3} 55 dicc02 = {1:'a',2:'b', 3:'a'} 56 57 print(f'El valor de la clave "a" en dicc01 es: {dicc01['a']}') 58 print(f'El valor de la clave "3" en dicc02 es: {dicc02[3]}')</pre>	<pre>El valor de la clave "a" en dicc01 es: 1 El valor de la clave "3" en dicc02 es: a</pre>
--	--

Cuando se intenta acceder a una clave que no exista en un diccionario se produce un error. Veamos algunos ejemplos:

<pre>60 dicc01 = {'a':1,'b':2,'c':3} 61 dicc02 = {1:'a',2:'b', 3:'a'} 62 print(f'Valor de la clave "a" en dicc01: {dicc01['f']}')</pre>	<pre>Traceback (most recent call last):   File "/Users/rghinf/ACADEMICO/CURSO_24_25/PRO_2024/COD-PYTH 63, in &lt;module&gt;     print(f'Valor de la clave "a" en dicc01: {dicc01['f']}')                                      ~~~~~^~~~~~ KeyError: 'f' rghinf@MacBook-Pro-de-Reinaldo UT02 %</pre>
---	---

### Uso de get()

El método get() devuelve el valor de la clave especificada en el diccionario. Toma un máximo de dos parámetros get(key[, value]):

key - Clave a buscar en el diccionario.

value (opcional) - Valor que se devolverá si no se encuentra la clave. El valor por defecto es None.

El método get() devuelve

- el valor de la clave especificada si la clave está en el diccionario.
- None si no se encuentra la clave y no se especifica el valor.
- valor si no se encuentra la clave y se especifica el valor.

Veamos ejemplos de su uso

<pre>66 scores = { 67     'Physics': 67, 68     'Maths': 87, 69     'History': 75 70 } 71 print(f'Valor de la clave "Maths" es: {scores.get('Physics')}')</pre>	<pre>Valor de la clave "Maths" es: 67</pre>
---	---

```

77 person = {'name': 'Phill', 'age': 22}
78 print('Name: ', person.get('name'))
79 print('Age: ', person.get('age'))
80 # clave que no está
81 print('Salary: ', person.get('salary'))
82 # se indica valor a devolver
83 print('Salary: ', person.get('salary', 0.0))

```

```

Name:  Phill
Age:  22
Salary:  None
Salary:  0.0

```

## Modificación de los elementos de un diccionario

Los diccionarios de Python son mutables (modificables). Podemos cambiar el valor de un elemento del diccionario haciendo referencia a su clave. Por ejemplo,

```

87 country_capitals = {
88     "Germany": "Berlin",
89     "Italy": "Naples",
90     "England": "London"
91 }

```

```
{'Germany': 'Berlin', 'Italy': 'Rome', 'England': 'London'}
```

En caso de que la clave no existiera en el diccionario se agregará al mismo.

## Asignaciones con diccionarios

Hay que tener cuidado a la hora de asignar un diccionario a una nueva variable, veamos el comportamiento que se produce al hacerlo.

```

98 d1 = {'a':1, 'b':2}
99 d2 = d1
100 print(f'd1 es: {d1}')
101 print(f'd2 es: {d2}')
102 d1['a'] = 9
103 d2['b'] = '*'
104 print(f'd1 es: {d1}')
105 print(f'd2 es: {d2}')

```

```

d1 es: {'a': 1, 'b': 2}
d1 es: {'a': 1, 'b': 2}
d1 es: {'a': 9, 'b': '*'}
d1 es: {'a': 9, 'b': '*'}

```

Se crea el diccionario **d1** con los elementos que se aprecian en la línea 98 de código. Luego asignamos a una nueva variable **d2** el diccionario **d1**. Posteriormente se cambian de valor una clave en d1 y otra clave distinta en d2. Como resultado se aprecia que los cambios hechos se reproducen en los dos diccionarios que comparten la misma zona de memoria.

Esto ocurre porque en la línea 102 y 103 se realizan cambios en los valores de las claves de los diccionarios. Por tanto las dos variables siguen apuntando a la misma zona de memoria en la que se encuentra el diccionario.

Una forma de resolver este es utilizando el método de copy a la hora de copiar el diccionario. Veamos un ejemplo.

```

111 d1 = {'a':1, 'b':2}
112 d2 = d1.copy()
113 print(f'd1 es: {d1}')
114 print(f'd2 es: {d2}')
115 d1['a'] = 9
116 d2['b'] = '*'
117 print(f'd1 es: {d1}')
118 print(f'd2 es: {d2}')

```

```

d1 es: {'a': 1, 'b': 2}
d2 es: {'a': 1, 'b': 2}
d1 es: {'a': 9, 'b': 2}
d2 es: {'a': 1, 'b': '*'}

```

Veamos un nuevo ejemplo en el que ahora se asigna un nuevo diccionario a una de las variables

```

113 d1 = {'a':1, 'b':2}
114 d2 = d1
115 print(f'd1 es: {d1}')
116 print(f'd2 es: {d2}')
117 d1 = {0: 3, 1:4}
118 d2['b'] = 7
119 print(f'd1 es: {d1}')
120 print(f'd2 es: {d2}')

```

```

d1 es: {'a': 1, 'b': 2}
d2 es: {'a': 1, 'b': 2}
d1 es: {0: 3, 1: 4}
d2 es: {'a': 1, 'b': 7}

```

Como se aprecia en el código (línea 55) ahora a la variable a se le asigna una nueva lista. Eso hace que ahora apunta a la zona de memoria de la nueva lista y b mantiene la referencia a la lista anterior, tal y como se muestra en la segunda imagen.

Problemos ahora que pasa si el valor de la clave del diccionario es de tipo mutable y realizamos una copia del diccionario usando el método `copy()` y modificamos uno de los elementos del valor mutable.

```
137 d1 = {'a':[3,4], 'b':2}
138 d2 = d1.copy()
139 print(f'd1 es: {d1}')
140 print(f'd2 es: {d2}')
141 d1['a'][0] = 100
142 d2['b'] = '*'
143 print(f'd1 es: {d1}')
144 print(f'd2 es: {d2}')
```

```
d1 es: {'a': [3, 4], 'b': 2}
d2 es: {'a': [3, 4], 'b': 2}
d1 es: {'a': [100, 4], 'b': 2}
d2 es: {'a': [100, 4], 'b': '*'}
```

Para solucionarlo usamos la función `deepcopy()` de la librería `copy` tal y como se muestra en la siguiente imagen

```
148 from copy import deepcopy
149 d1 = {'a':[3,4], 'b':2}
150 d2 = deepcopy(d1)
151 print(f'd1 es: {d1}')
152 print(f'd2 es: {d2}')
153 d1['a'][0] = 100
154 d2['b'] = '*'
155 print(f'd1 es: {d1}')
156 print(f'd2 es: {d2}')
```

```
d1 es: {'a': [3, 4], 'b': 2}
d2 es: {'a': [3, 4], 'b': 2}
d1 es: {'a': [100, 4], 'b': 2}
d2 es: {'a': [3, 4], 'b': '*'}
```

## Diccionarios dentro de un diccionario

Veamos como trabajar con diccionarios anidados. Veamos un ejemplo.

```
160 dicc01 = {1:{'a':[1,3], 'b':[2,4]}, 2:{'c':[5,7]}}
161 for elto in dicc01:
162     print(f'clave = {elto}:{' ' ', end=' ' ')
163     for elto_int in dicc01[elto]:
164         valor = f'{elto_int}:{' ' + f'{dicc01[elto][elto_int]}'
165         print(valor, end=' ')
166     print('')
```

```
clave = 1: { a:[1, 3] b:[2, 4] }
clave = 2: { c:[5, 7] }
```

## Borrar elementos de un diccionario

### Uso de del

Veamos ejemplos de uso del comando del para borrar elementos.

```
170 country_capitals = {
171     "Germany": "Berlin",
172     "Canada": "Ottawa",
173 }
174 print(country_capitals)
175 del country_capitals["Germany"]
176 print(country_capitals)
```

```
{'Germany': 'Berlin', 'Canada': 'Ottawa'}
{'Canada': 'Ottawa'}
```

Si se intenta borrar una clave que no existe en el diccionario se produce un error.

```
179 country_capitals = {
180     "Germany": "Berlin",
181     "Canada": "Ottawa",
182 }
183 print(country_capitals)
184 del country_capitals["Spain"]
```

```
{'Germany': 'Berlin', 'Canada': 'Ottawa'}
Traceback (most recent call last):
  File "/Users/rghinf/ACADEMICO/CU
in <module>
    del country_capitals["Spain"]
    ~~~~~^~~~~~
KeyError: 'Spain'
```

## Borrar todas las claves de un diccionario

### Uso de clear()

El siguiente ejemplo muestra el uso del método clear para borrar todas las claves de un diccionario.

```

188 country_capitals = {
189     "Germany": "Berlin",
190     "Canada": "Ottawa",
191 }
192 country_capitals.clear()
193 print(country_capitals)

```



### Uso de pop()

El método pop() se utiliza para eliminar la clave del diccionario que se suministra como parámetro. Este método retorna el valor que tiene la clave que se elimina. Si la clave no existe da un error.

Veamos a continuación ejemplos de uso de pop()

```

197 marks = { 'Physics': 67, 'Chemistry': 72, 'Math': 89 }
198 print(marks)
199 element = marks.pop('Chemistry')
200 print(marks)
201 print('Popped Marks:', element)

```

```

{'Physics': 67, 'Chemistry': 72, 'Math': 89}
{'Physics': 67, 'Math': 89}

```

```

205 marks = { 'Physics': 67, 'Chemistry': 72, 'Math': 89 }
206 element = marks.pop('Chemistry 2')

```

```

Traceback (most recent call last):
  File "/Users/rghinf/ACADEMICO/CURSO_24
in <module>
    element = marks.pop('Chemistry 2')
KeyError: 'Chemistry 2'

```

### Recorrer un diccionario

Existen diversas formas para recorrer un diccionario

#### Uso de for..in

Esta forma de recorrer la lista permite pasar por todas las claves del diccionario. Veamos un ejemplo:

```

217 country_capitals = {
218     "United States": "Washington D.C.",
219     "Italy": "Rome"
220 }
221 # print claves una por una
222 print('Las claves')
223 for country in country_capitals:
224     print(country)

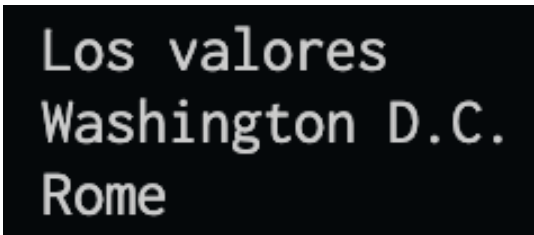
```

Las claves  
United States  
Italy

Como se aprecia en cada iteración, el valor de la variable `country` toma el valor de una de las claves del diccionario.

Veamos ahora como mostrar todos los valores del diccionario.

```
227 country_capitals = {
228     "United States": "Washington D.C.",
229     "Italy": "Rome"
230 }
231 print('Los valores')
232 # print valores uno por uno
233 for country in country_capitals:
234     capital = country_capitals[country]
235     print(capital)
```



Los valores  
Washington D.C.  
Rome

## Longitud de un diccionario

### Uso de `len()`

El método `len()` se usa para calcular el número de claves que tiene el diccionario. Veamos ejemplos de su uso.

```
239 country_capitals = {"England": "London", "Italy": "Rome"}
240 print(len(country_capitals)) # Output: 2
241 numbers = {10: "ten", 20: "twenty", 30: "thirty"}
242 print(len(numbers))
243 countries = {}
244 print(len(countries))
```



2  
3  
0

## Otros recorridos de un diccionario usando otros métodos

En Python existen varios métodos que nos permiten hacer recorridos y obtener los valores, las claves o los dos elementos en cada iteración. Veamos cada uno de ellos.

### Uso de `values()`

El método `values()` retorna un `dict_values` con todos los valores de las claves asociadas al diccionario. Este `dict_values` después puede ser iterado, transformado en lista, tupla, etc.

Veamos un ejemplo de su uso.

```
251 marks = {'Physics':67, 'Maths':87}
252 print(iterador:= marks.values()) # operador walrus
253 for valor in iterador:
254     print(f'El valor es: {valor}')
```

```
dict_values([67, 87])
El valor es: 67
El valor es: 87
```

### Uso de keys()

El método `keys()` retorna un `dict_keys` con todas las claves asociadas al diccionario. Este `dict_keys` después puede ser iterado, transformado en lista, tupla, etc.

Veamos un ejemplo de su uso.

```
261 marks = {'Physics':67, 'Maths':87}
262 print(iterador:= marks.keys()) # operador walrus
263 for clave in iterador:
264     print(f'El valor es: {clave}')
```

```
dict_keys(['Physics', 'Maths'])
El valor es: Physics
El valor es: Maths
```

### Uso de items()

El método `items()` retorna un `dict_items` con todas las tuplas (clave, valor) formadas con los elementos del diccionario. Este `dict_items` después puede ser iterado, transformado en lista, tupla, etc.

Veamos un ejemplo de su uso.

```
271 marks = {'Physics':67, 'Maths':87}
272 print(iterador:= marks.items()) # operador walrus
273 for tupla in iterador:
274     print(f'Elemento del diccionario: {tupla}')
```

```
dict_items([('Physics', 67), ('Maths', 87)])
Elemento del diccionario: ('Physics', 67)
Elemento del diccionario: ('Maths', 87)
```

## Ordenar las claves de un diccionario

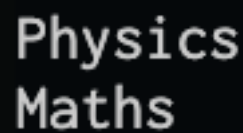
Los diccionarios no tienen método para ordenar sus claves, pero esto se puede hacer mediante la función `sorted()`. Veamos un ejemplo de como utilizarla para ordenar las claves de un diccionario.

### Uso de sorted()

```
276 marks = {'Physics':67, 'Maths':87}
277 ordered_marks = sorted(marks)
278 for i in ordered_marks:
279     print(i)
```

```
Maths
Physics
```

```
281 marks = {'Physics':67, 'Maths':87}
282 ordered_marks = sorted(marks, reverse=True)
283 for i in ordered_marks:
284     print(i)
```



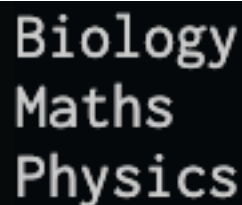
Physics  
Maths

## Invertir las claves de un diccionario

Los diccionarios no tienen método para invertir sus claves, pero esto se puede hacer mediante la función `reversed()`. Veamos un ejemplo de como utilizarla para invertir las claves de un diccionario.

### Uso de `reversed()`

```
290 marks = {'Physics':67, 'Maths':87, 'Biology':75}
291 reversed_marks = reversed(marks)
292 for i in reversed_marks:
293     print(i)
```



Biology  
Maths  
Physics

## Otros métodos a usar en diccionarios

### Uso de `setdefault()`

Permite buscar una clave en el diccionario y da la posibilidad de insertarla si no existe. Su sintaxis es la siguiente:

**`dict.setdefault(key[, default_value])`**

`setdefault()` toma un máximo de dos parámetros:

- `key` - la clave a buscar en el diccionario
- `default_value` (opcional) - clave con un valor `default_value` se inserta en el diccionario si la clave no está en el diccionario.

Si no se proporciona, `default_value` será `None`.

Este método puede retornar los siguientes valores:

- valor de la clave si está en el diccionario
- `None` si la clave no está en el diccionario y no se especifica `default_value`
- `default_value` si la clave no está en el diccionario y se especifica `default_value`

Veamos ejemplos de su uso:

```

248 person = {'name': 'Phill', 'age': 22}
249 age = person.setdefault('age')
250 print('person = ',person)
251 print('Age = ',age)

```

```

person = {'name': 'Phill', 'age': 22}
Age = 22

```

```

257 person = {'name': 'Phill'}
258 salary = person.setdefault('salary')
259 print('person = ',person)
260 print('salary = ',salary)

```

```

person = {'name': 'Phill', 'salary': None}
salary = None

```

```

264 person = {'name': 'Phill'}
265 age = person.setdefault('age', 22)
266 print('person = ',person)
267 print('age = ',age)

```

```

person = {'name': 'Phill', 'age': 22}
age = 22

```

El primero de los códigos se utiliza `setdefault()` con una clave que existe en el diccionario, en este caso se retorna el valor de dicha clave.

En el segundo código se utiliza `setdefault()` con una clave que no existe en el diccionario y no se pone un valor por defecto, en este caso el valor retornado será `None`.

En el tercero de los códigos se utiliza `setdefault()` con una clave que no existe en el diccionario y se pone un valor por defecto, en este caso el valor retornado será el valor puesto por defecto.

### Uso de `fromkeys()`

El método `fromkeys()` crea un diccionario a partir de la secuencia dada de claves y un único valor para la clave. Si no se suministra ningún valor se asigna `None`. Si el valor que se asigna se una lista, hay que tener cuidado porque si se modifica el valor, éste se cambiará para todos los valores de la clave.

```

271 # claves para el diccionario
272 alphabets = {'a', 'b', 'c'}
273 # valores a asignar a la clave
274 number = 1
275 # crea el diccionario con todas las claves
276 # tienen el mismo valor
277 dictionary = dict.fromkeys(alphabets, number)
278 print(dictionary)

```

```

{'b': 1, 'c': 1, 'a': 1}

```

```

280 keys = [1, 2, 4]
281 numbers = dict.fromkeys(keys)
282 print(numbers)

```

```

{1: None, 2: None, 4: None}

```

```

284 keys = {'a', 'e', 'i', 'o', 'u' }
285 value = [1]
286 vowels = dict.fromkeys(keys, value)
287 print(vowels)
288 value.append(2)
289 print(vowels)

```

```

{'i': [1], 'u': [1], 'e': [1], 'a': [1], 'o': [1]}
{'i': [1, 2], 'u': [1, 2], 'e': [1, 2], 'a': [1, 2], 'o': [1, 2]}

```

Si analizamos el último código resulta que al modificar la variable value añadiendo un nuevo elemento a la lista, se modifica todos los valores asignados a la clave. Para corregir este efecto debemos usar el método `copy()` a la hora de asignar el valor en el método `fromkeys()` tal y como se muestra en el siguiente ejemplo:

```

296 keys = {'a', 'e', 'i', 'o', 'u' }
297 value = [1]
298 vowels = dict.fromkeys(keys, value.copy())
299 print(vowels)
300 value.append(2)
301 print(vowels)
302 vowels['a'] = [3]
303 print(vowels)

```

```

{'u': [1], 'o': [1], 'e': [1], 'a': [1], 'i': [1]}
{'u': [1], 'o': [1], 'e': [1], 'a': [1], 'i': [1]}
{'u': [1], 'o': [1], 'e': [1], 'a': [3], 'i': [1]}

```

Otra forma de corregirlo es usando lo que se explica a continuación,

## Comprensión del diccionario para objetos mutables

Podemos utilizar la comprensión de diccionario y evitar la actualización del diccionario cuando se actualiza el objeto mutable (lista, diccionario, etc). La forma de hacerlo es como se muestra a continuación:

```

305 keys = {'a', 'e', 'i', 'o', 'u' }
306 value = [1]
307 # creates dictionary using dictionary comprehension
308 vowels = { key : list(value) for key in keys }
309 print(vowels)
310 value.append(2)
311 print(vowels)

```

```

{'o': [1], 'e': [1], 'i': [1], 'u': [1], 'a': [1]}
{'o': [1], 'e': [1], 'i': [1], 'u': [1], 'a': [1]}

```

## Uso de `popitem()`

El método `popitem()` elimina y devuelve la siguiente tupla (clave, valor) del elemento que se elimina del diccionario. El valor eliminado se determina mediante un LIFO (Último en entrar, primero en salir).

- Devuelve el último par de elementos (clave,valor) insertado del diccionario.
- Elimina del diccionario el par de elementos devuelto.

El método `popitem()` genera un error `KeyError` si el diccionario está vacío.

Veamos un ejemplo de su uso

```
313 person = {'name': 'Phill', 'age': 22, 'salary': 3500.0}
314 result = person.popitem()
315 print('Return Value = ', result)
316 print('person = ', person)
317 person['profession'] = 'Plumber'
318 result = person.popitem()
319 print('Return Value = ', result)
320 print('person = ', person)
```

```
Return Value = ('salary', 3500.0)
person = {'name': 'Phill', 'age': 22}
Return Value = ('profession', 'Plumber')
person = {'name': 'Phill', 'age': 22}
```

### Uso de `update()`

El método `update()` actualiza el diccionario con los elementos de otro objeto diccionario o de un iterable de pares clave/valor.

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 inalterado.

Veamos los siguientes ejemplos:

```
324 dicc01 = {1:'a', 2:'b'}
325 dicc02 = {3:'c'}
326 dicc01.update(dicc02)
327 print(dicc01)
```

```
{1: 'a', 2: 'b', 3: 'c'}
```

```
333 dicc01 = {1:'a', 2:'b'}
334 dicc02 = {3:'c', 1:'z'}
335 dicc01.update(dicc02)
336 print(dicc01)
```

```
{1: 'z', 2: 'b', 3: 'c'}
```

```
338 dicc01 = {1:'a', 2:'b'}
339 lista = [(2,'z'), (3,'c')]
340 dicc01.update(lista)
341 print(dicc01)
```

```
{1: 'a', 2: 'z', 3: 'c'}
```

## Métodos avanzados para usar en listas `zip()`, `filter()`, `map()`

### *Uso de `zip()`*

`zip()` es un método que retorna un iterador que genera tuplas con los elementos de varias listas que se le pasan como parámetros. La primera tupla generada tiene los primeros elementos de cada lista, la segunda los segundos y así sucesivamente. Si las listas no tienen la misma longitud el iterador genera tantas tuplas como la lista de menor longitud.

Veamos unos ejemplos.

### *Uso de `map()`*

La función `map()` ejecuta una función dada a cada elemento de un iterable (como listas, tuplas, etc.).

Veamos un ejemplo de su uso

En el ejemplo anterior se ha definido la función `potencia_2` para calcular el cuadrado del número que se le pasa como parámetro. Esta función es la que se va a usar con la función `map` para aplicársela a cada uno de los elementos de la lista obteniendo así un iterador que luego pasaremos a una lista.

### *Uso de `filter`*

La función `filter()` selecciona elementos de un iterable basándose en el resultado de una función. Veamos un ejemplo de su uso