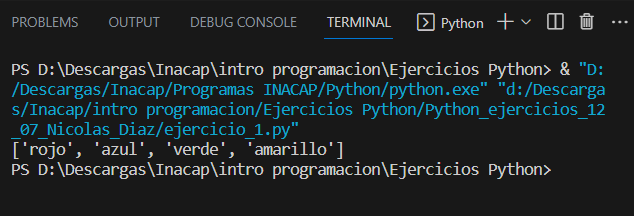


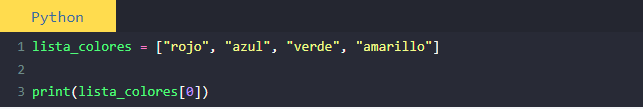
**Imprimir una lista entera**



Resultado código



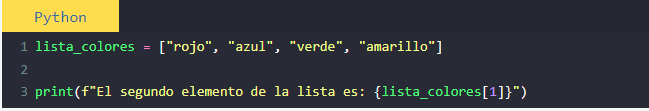
### Imprimir solo un valor concreto de una lista



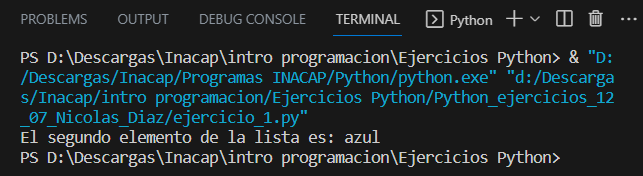
Resultado código

### 

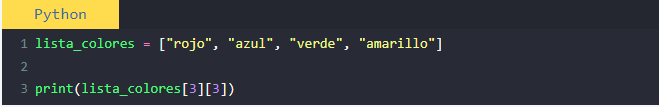
### Utilizar listas en strings formateados



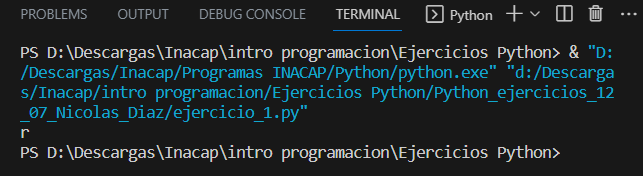
Resultado código



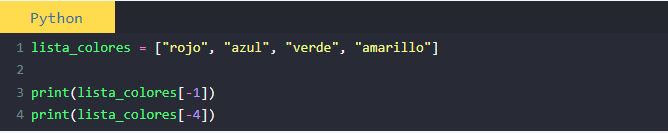
### Acceder a los caracteres de string de un elemento en una lista



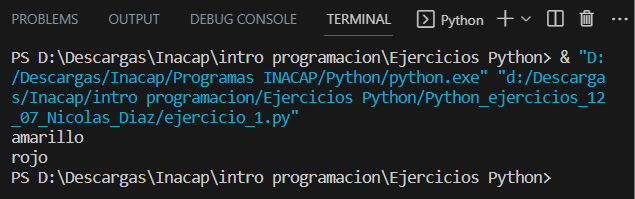
Resultado código



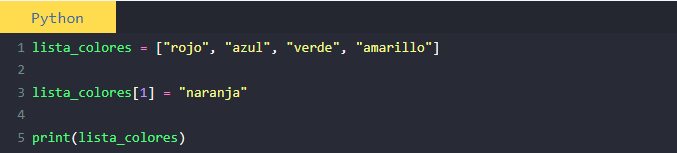
## Posiciones negativas en las listas



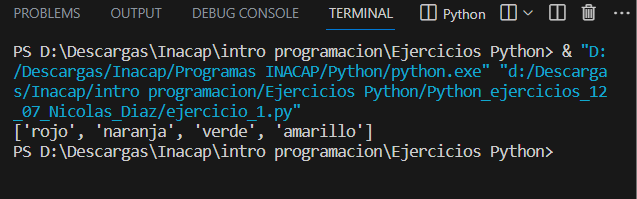
Resultado código



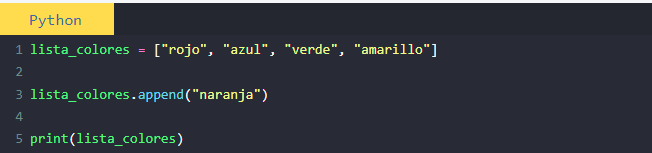
## Reemplazar valores en la lista



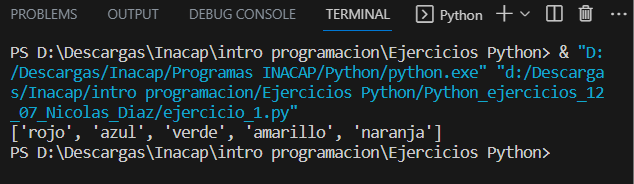
Resultado código

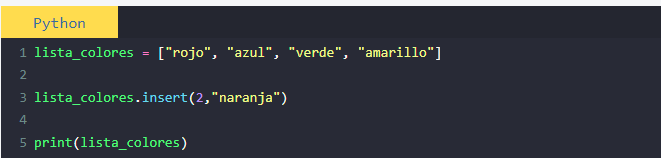


## Añadir elementos a las listas

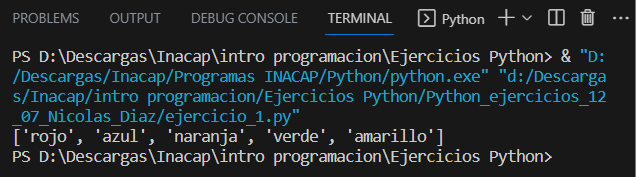


Resultado código

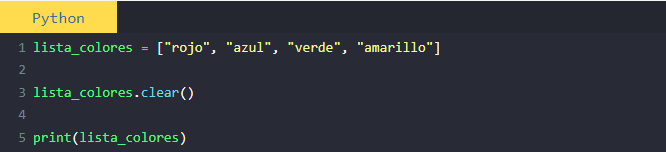




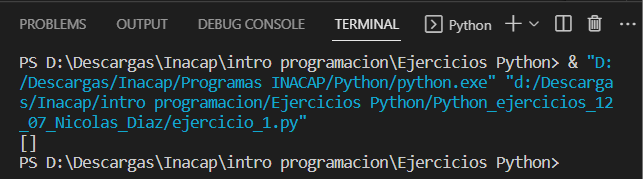
Resultado código



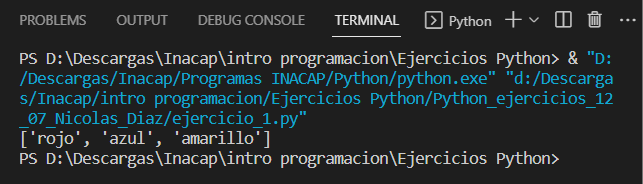
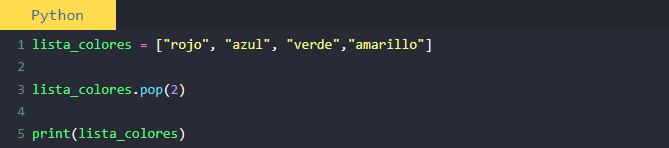
## Eliminar elementos de las listas



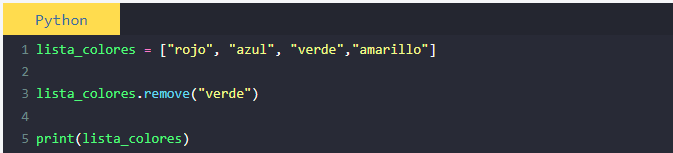
Resultado código



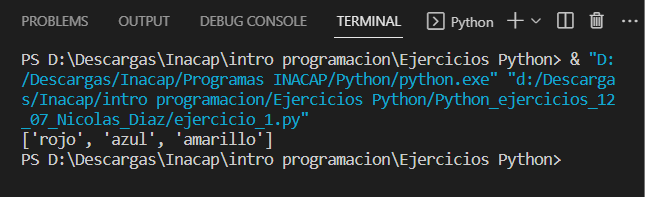
OTRO METODO



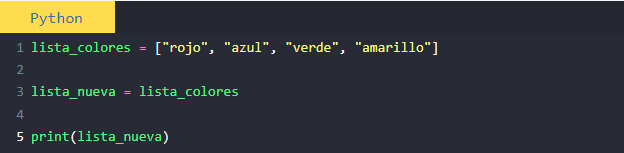
Resultado código



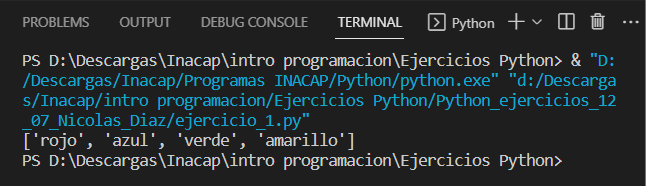
Resultado código



## Duplicar listas



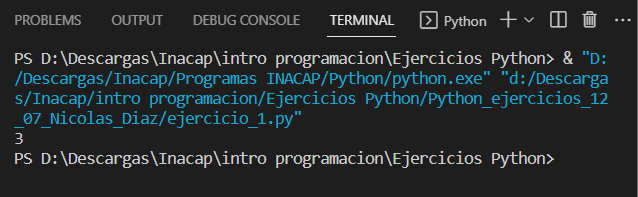
Resultado código



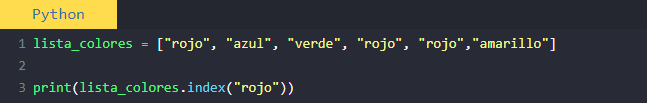
## Encontrar valores repetidos en listas



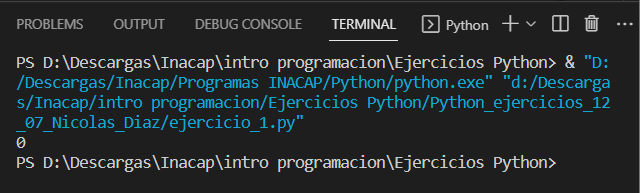
Resultado código



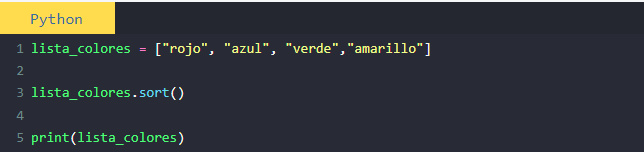
## Encontrar la posición de un elemento literal en las listas



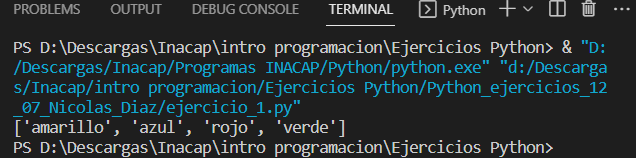
Resultado código



## Ordenar listas en orden alfabético



Resultado código

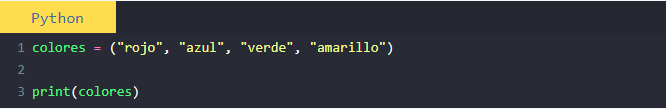


## ¿Qué son las tuplas?

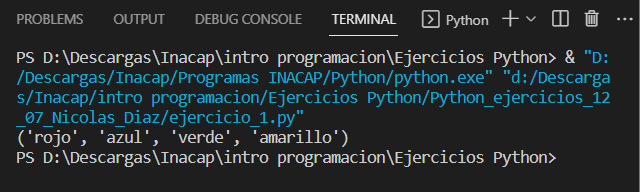
Las tuplas son elementos similares a las listas, con la gran diferencia en que las listas son mutables (pueden variar de valores) y las tuplas son inmutables (no pueden cambiar de valores). Una vez las declaramos y les damos un valor, queda durante toda la ejecución del programa.

La diferencia sintáctica es escribirlas con paréntesis en lugar de con corchetes.

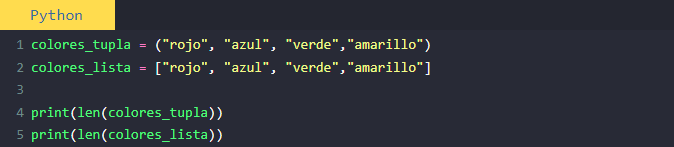
**Imprimir una Tupla entera**



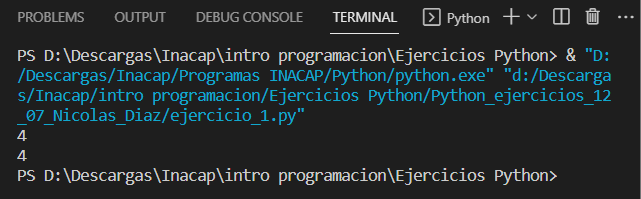
Resultado código



## Saber la longitud de una lista o tupla



Resultado código



El método count() cuenta el número de veces que el objeto pasado como parámetro se ha encontrado en la lista.

l = [1, 1, 1, 3, 5]

Resultado código

print(l.count(1))

# Diccionario

Los diccionarios en Python son una estructura de datos que permite almacenar su contenido en forma de llave y valor.

## Crear diccionario Python

Un diccionario en Python es una colección de elementos, donde cada uno tiene una llave key y un valor value. Los diccionarios se pueden crear con paréntesis {} separando con una coma cada par key: value. En el siguiente ejemplo tenemos tres keys que son el nombre, la edad y el documento.

d1 = {

"Nombre": "Sara",

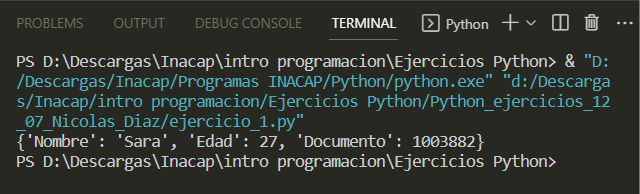
"Edad": 27,

"Documento": 1003882

}

print(d1)

#{'Nombre': 'Sara', 'Edad': 27, 'Documento': 1003882}



Otra forma equivalente de crear un diccionario en Python es usando dict() e introduciendo los pares key: value entre paréntesis.

d2 = dict([

('Nombre', 'Sara'),

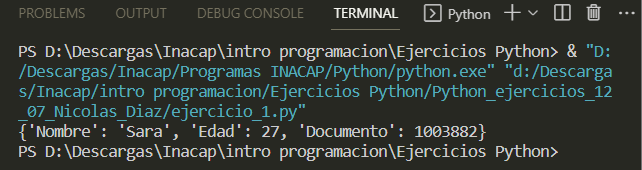
('Edad', 27),

('Documento', 1003882),

])

print(d2)

#{'Nombre': 'Sara', 'Edad': '27', 'Documento': '1003882'}



También es posible usar el constructor dict() para crear un diccionario.

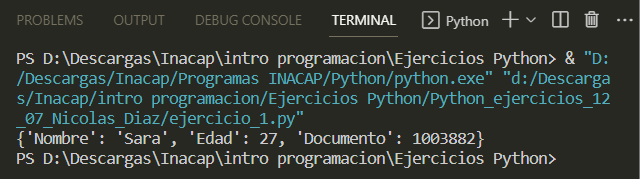
d3 = dict(Nombre='Sara',

Edad=27,

Documento=1003882)

print(d3)

#{'Nombre': 'Sara', 'Edad': 27, 'Documento': 1003882}



Algunas propiedades de los diccionario en Python son las siguientes:

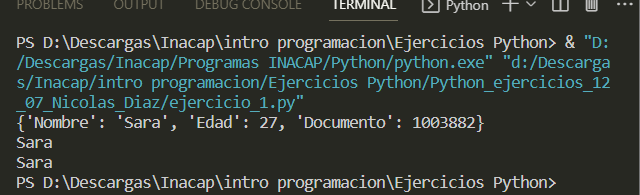
* Son **dinámicos**, pueden crecer o decrecer, se pueden añadir o eliminar elementos.
* Son **indexados**, los elementos del diccionario son accesibles a través del key.
* Y son **anidados**, un diccionario puede contener a otro diccionario en su campo value.

## Acceder y modificar elementos

Se puede acceder a sus elementos con [] o también con la función get()

print(d1['Nombre']) #Sara

print(d1.get('Nombre')) #Sara

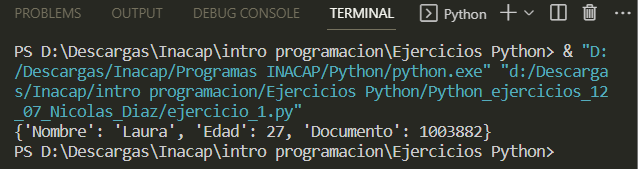


Para modificar un elemento basta con usar [] con el nombre del key y asignar el valor que queremos.

d1['Nombre'] = "Laura"

print(d1)

#{'Nombre': Laura', 'Edad': 27, 'Documento': 1003882}

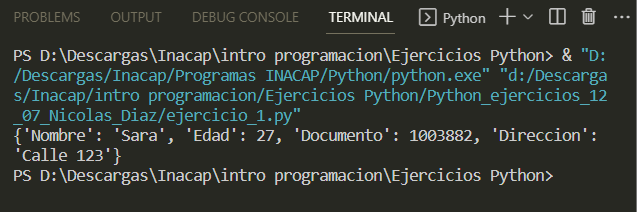


Si el key al que accedemos no existe, se añade automáticamente.

d1['Direccion'] = "Calle 123"

print(d1)

#{'Nombre': 'Laura', 'Edad': 27, 'Documento': 1003882, 'Direccion': 'Calle 123'}



## Iterar diccionario

Los diccionarios se pueden iterar de manera muy similar a las listas u otras estructuras de datos. Para imprimir los key.

# Imprime los key del diccionario

for x in d1:

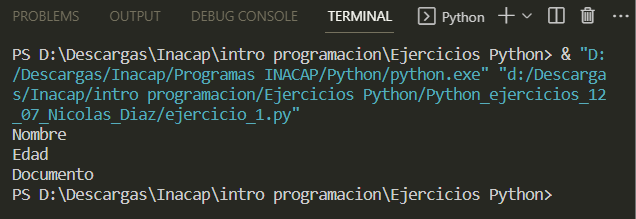
print(x)

#Nombre

#Edad

#Documento

#Direccion



Se puede imprimir también solo el value.

# Impre los value del diccionario

for x in d1:

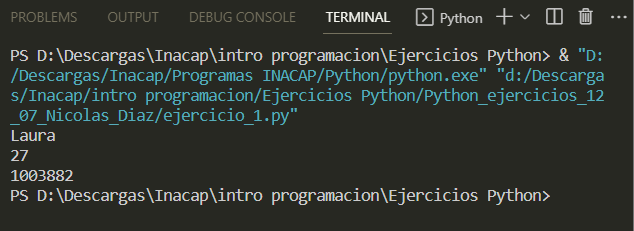
print(d1[x])

#Laura

#27

#1003882

#Calle 123



O si queremos imprimir el key y el value a la vez.

# Imprime los key y value del diccionario

for x, y in d1.items():

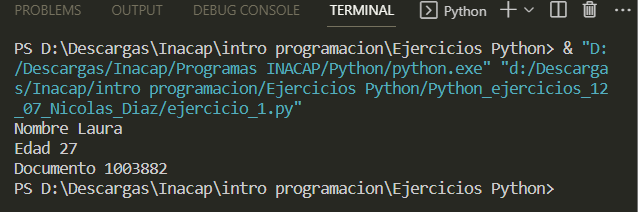
print(x, y)

#Nombre Laura

#Edad 27

#Documento 1003882

#Direccion Calle 123



Diccionarios anidados

Los diccionarios en Python pueden contener uno dentro de otro. Podemos ver como los valores anidado uno y dos del diccionario d contienen a su vez otro diccionario.

anidado1 = {"a": 1, "b": 2}

anidado2 = {"a": 1, "b": 2}

d = {

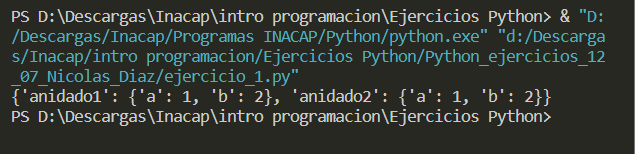
"anidado1" : anidado1,

"anidado2" : anidado2

}

print(d)

#{'anidado1': {'a': 1, 'b': 2}, 'anidado2': {'a': 1, 'b': 2}}



## Métodos diccionarios Python

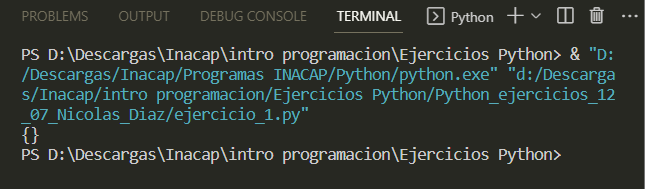
### clear()

El método clear() elimina todo el contenido del diccionario.

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

d.clear()

print(d) #{}



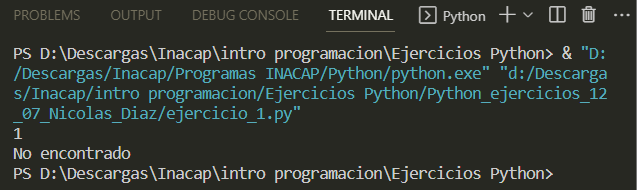
### get(<key>[,<default>])

El método get() nos permite consultar el value para un key determinado. El segundo parámetro es opcional, y en el caso de proporcionarlo es el valor a devolver si no se encuentra la key.

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

print(d.get('a')) #1

print(d.get('z', 'No encontrado')) #No encontrado



### items()

El método items() devuelve una lista con los keys y values del diccionario. Si se convierte en list se puede indexar como si de una lista normal se tratase, siendo los primeros elementos las key y los segundos los value.

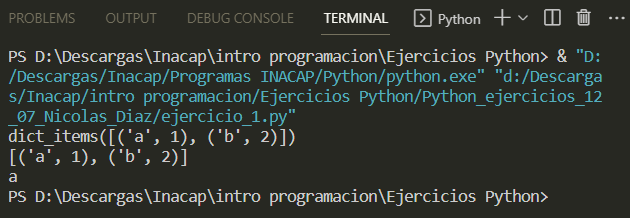
d = {'a': 1, 'b': 2}

it = d.items()

print(it) #dict\_items([('a', 1), ('b', 2)])

print(list(it)) #[('a', 1), ('b', 2)]

print(list(it)[0][0]) #a



### keys()

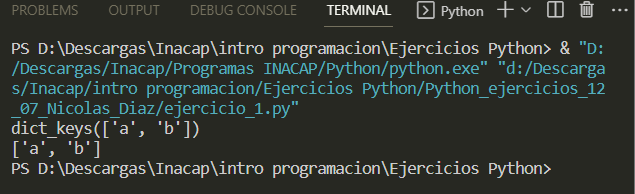
El método keys() devuelve una lista con todas las keys del diccionario.

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

k = d.keys()

print(k) #dict\_keys(['a', 'b'])

print(list(k)) #['a', 'b']

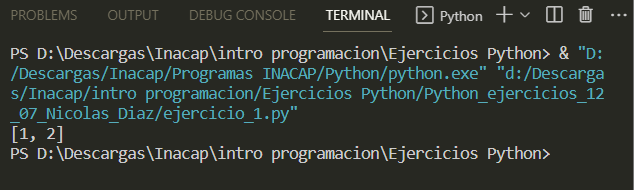


### values()

El método values() devuelve una lista con todos los values o valores del diccionario.

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

print(list(d.values())) #[1, 2]



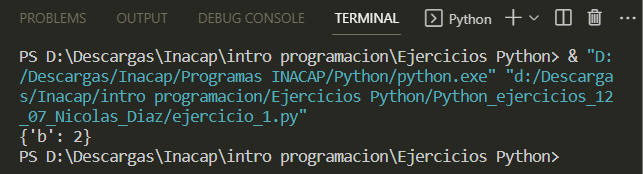
### pop(<key>[,<default>])

El método pop() busca y elimina la key que se pasa como parámetro y devuelve su valor asociado. Daría un error si se intenta eliminar una key que no existe.

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

d.pop('a')

print(d) #{'b': 2}

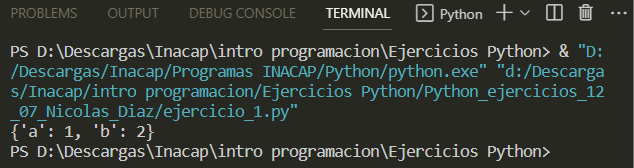


También se puede pasar un segundo parámetro que es el valor a devolver si la key no se ha encontrado. En este caso si no se encuentra no habría error.

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

d.pop('c', -1)

print(d) #{'a': 1, 'b': 2}



### popitem()

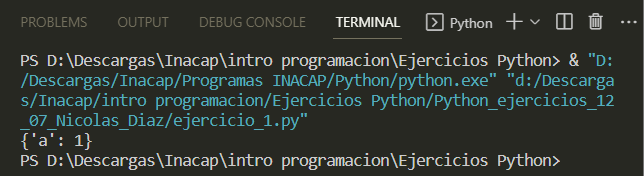
El método popitem() elimina de manera aleatoria un elemento del diccionario.

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

d.popitem()

print(d)

#{'a': 1}



### update(<obj>)

El método update() se llama sobre un diccionario y tiene como entrada otro diccionario. Los value son actualizados y si alguna key del nuevo diccionario no esta, es añadida.

d1 = {'a': 1, 'b': 2}

d2 = {'a': 0, 'd': 400}

d1.update(d2)

print(d1)

#{'a': 0, 'b': 2, 'd': 400}

