I dizionari in Python

In questo tutorial imparerai che cosa sono i dizionari in Python e come manipolarli per gestire coppie di valori.

Indice

Definizione: che cosa sono i dizionari in Python?

Un **dizionario** (*Python dictionary*) è una **sequenza di coppie di elementi** separati da una virgola e racchiusi all'interno di due parentesi graffe { }.

Ciascuna coppia è composta da una **chiave** (*key*) e da un **valore** (*value*) che sono separati tra loro dal simbolo :.

I **valori** possono essere di qualsiasi tipo, mentre le **chiavi**:

- devono essere un dato immutabile (numeri, stringhe, tuple o frozenset);
- non possono ripetersi (in altre parole, sono uniche).

Per comprendere cosa sono i dizionari in Python, ripensa ad un qualunque **dizionario bilingue**: questi non sono altro che una sequenza di termini (chiavi) a cui sono associate le relative traduzioni (valori).

`Esempio: Python dictionary`

```
1  # dizionario composto da stringhe
2  ita_eng = {'gatto': 'cat', 'cane': 'dog', 'uccello': 'bird'}
3  # dizionario composto da stringhe, numeri e liste
4  studente = {'nome': 'Alessandro', 'anni': 28, 'voti': [6.5, 7, 8]}
```

A cosa servono i dizionari

I dizionari, a differenza di tutte le altre tipologie di elenchi (liste, tuple e set), ti consentono di memorizzare **coppie di elementi** anziché elementi singoli.

Ciò è particolarmente utile per accedere in modo molto rapido alle informazioni.

Infatti, per estrarre un qualunque valore da un dizionario ti basterà richiamare la relativa chiave.

Facciamo un esempio in cui, per svolgere la stessa operazione, utilizzeremo dapprima una lista e poi un dizionario: ipotizziamo che tu debba mostrare la traduzione di cane.

Esempio: Python list vs dictionary

```
# lista
ita_eng = [['gatto', 'cat'], ['cane', 'dog'], ['uccello', 'bird']]
print(ita_eng[1][1])
# dizionario
ita_eng = {'gatto': 'cat', 'cane': 'dog', 'uccello': 'bird'}
print(ita_eng['cane'])
```

```
OUTPUT
dog
```

L'uso del dizionario è nettamente più pratico rispetto alle liste perché, anziché doverti ricordare la posizione dell'elemento, è sufficiente indicare la chiave a cui è accoppiato `['cane']`.

Come ordinare gli elementi di un dizionario in Python

Come ordinare un dizionario in funzione delle chiavi

Per **ordinare** gli elementi di un dizionario **in funzione delle sue chiavi** devi:

- estrarre una lista ordinata delle chiavi utilizzando la funzione *sorted()* e il metodo *keys()*;
- scorrere la lista che hai appena realizzato e compilare il nuovo dizionario.

Esempio: How to sort Python dictionary by key (ascending)

```
dizionario = {'c': 20, 'a': 10, 'b': 30}
dizionario_ordinato = {}
# estraggo la lista di chiavi e la ordino
chiavi = sorted(dizionario.keys())
# scorro la lista appena realizzata
for chiave in chiavi:
    # individuo il valore corrispondente alla chiave
    valore = dizionario[chiave]
    # aggiungo la coppia chiave/valore al nuovo dizionario
    dizionario_ordinato[chiave] = valore
# mostro il contenuto del nuovo dizionario
print(dizionario_ordinato)
```

```
OUTPUT
{'a': 10, 'b': 30, 'c': 20}
```

Per ordinare gli elementi **in senso decrescente** ti basta aggiungere il parametro *reverse = True* all'interno delle parentesi tonde di *sorted()*.

Esempio: How to sort Python dictionary by key (descending)

```
dizionario = {'c': 20, 'a': 10, 'b': 30}
dizionario_ordinato = {}
# estraggo la lista di chiavi e la ordino
chiavi = sorted(dizionario.keys(), reverse=True)
# scorro la lista appena realizzata
for chiave in chiavi:
    # individuo il valore corrispondente alla chiave
    valore = dizionario[chiave]
    # aggiungo la coppia chiave/valore al nuovo dizionario
    dizionario_ordinato[chiave] = valore
```

```
# mostro il contenuto del nuovo dizionario
print(dizionario_ordinato)
```

```
OUTPUT
{'c': 20, 'b': 30, 'a': 10}
```

Come ordinare un dizionario in funzione dei valori

Per **ordinare** gli elementi di un dizionario **in funzione dei suoi valori** devi:

- estrarre una lista ordinata (in funzione dei valori) di coppie chiave/valore utilizzando la funzione *sorted()* e il metodo *items()*;
- scorrere la lista che hai appena realizzato e compilare il nuovo dizionario.

```
Esempio: How to sort Python dictionary by value
```

```
dizionario = {'c': 20, 'a': 10, 'b': 30}
dizionario_ordinato = {}
# estraggo una lista ordinata di coppie chiave/valore
chiavi_valori = sorted(dizionario.items(), key = lambda x: x[1])
# scorro la lista appena realizzata
for chiave, valore in chiavi_valori:
    # compilo il nuovo dizionario
    dizionario_ordinato[chiave] = valore
# mostro il contenuto del nuovo dizionario
print(dizionario_ordinato)
```

```
OUTPUT
{'a': 10, 'c': 20, 'b': 30}
```

> Analogamente a quanto visto nel precedente paragrafo, per ordinare gli elementi **in senso decrescente** ti basta aggiungere il parametro **reverse = True** all'interno delle parentesi tonde di *sorted()*.

Come verificare se un elemento è presente nel dizionario

Per **verificare** se una **chiave** è **presente** nel dizionario, devi utilizzare l'operatore in (*Python membership operator*).

In questo modo è come se stessi domandando al tuo computer: "la chiave X è nel dizionario Y?"

Il computer mostrerà True se è presente oppure False se assente.

Ad esempio, vuoi sapere se il dizionario "ita_eng" contiene la chiave "gatto"?

> Ricordati che Python è case sensitive, ovvero distingue tra maiuscole e minuscole. Cercare gatto è diverso dal cercare Gatto oppure GATTO.

```
Esempio: Python membership operator
```

```
ita_eng = {'gatto': 'cat', 'cane': 'dog'}
print('gatto' in ita_eng)
print('Gatto' in ita_eng)
```

```
OUTPUT
True
False
```

Purtroppo non esiste un metodo universale per **verificare** se un **valore** è **presente** nel dizionario, perché dipende dalla sua tipologia.

Dovrai trovare di volta in volta una soluzione che si adatti alla tua esigenza.

Come estrarre gli elementi di un dizionario in Python

Come estrarre le chiavi di un dizionario

Per **estrarre** una lista di tutte le **chiavi** che compongono un dizionario, devi:

- isolare le chiavi con il metodo *keys()*;
- salvarle in una lista con il metodo *list()*.

`Esempio: Python keys() method`

```
1  ita_eng = {'gatto': 'cat', 'cane': 'dog'}
2  chiavi = list(ita_eng.keys())
3  print(chiavi)
```

```
OUTPUT
['gatto', 'cane']
```

Come estrarre i valori di un dizionario

Per **estrarre** una lista di tutti i **valori** che compongono un dizionario, devi:

- isolare i valori con il metodo *values()*;
- salvarli in una lista con il metodo *list()*.

`Esempio: Python values() method`

```
1  ita_eng = {'gatto': 'cat', 'cane': 'dog'}
2  valori = list(ita_eng.values())
3  print(valori)
```

```
OUTPUT
['cat', 'dog']
```

Come estrarre i valori associati ad una chiave specifica

Per **estrarre** i **valori associati ad una chiave** specifica, devi utilizzare il metodo *get()* e inserire all'interno delle sue parentesi tonde la chiave in questione.

`Esempio: Python get() method`

```
1  ita_eng = {'gatto': 'cat', 'cane': 'dog'}
2  # estraggo la traduzione di 'gatto'
3  print(ita_eng.get('gatto'))
4  # estraggo la traduzione di un termine inesistente
5  print(ita_eng.get('leone'))
```

```
OUTPUT
cat
None
```

In alternativa, potresti indicare la chiave all'interno di due parentesi quadre [ ].

Tuttavia, lo **svantaggio** di questa soluzione è che, nel caso di chiave inesistente, Python blocca l'esecuzione del programma mostrando un **messaggio di errore**.

`Esempio: How to access elements from Python dictionary`

```
1  ita_eng = {'gatto': 'cat', 'cane': 'dog'}
2  # estraggo la traduzione di 'gatto'
3  print(ita_eng['gatto'])
4  # estraggo la traduzione di un termine inesistente
5  print(ita_eng['leone'])
```

```
OUTPUT
cat
Traceback (most recent call last):
  File "/.../.../Desktop/ex.py", line 5, in <module>
    print(ita_eng['leone'])
KeyError: 'leone'
```

Come modificare gli elementi dei dizionari in Python

Come modificare le chiavi di un dizionario

Per **modificare** la **chiave** di un dizionario devi:

- indicare la nuova chiave all'interno di due parentesi quadre [ ];
- rimuovere quella vecchia utilizzando il metodo *pop()*.

`Esempio: How to change Python dictionary's keys`

```
1  ita_eng = {'leone': 'cat', 'cane': 'dog'}
2  ita_eng['gatto'] = ita_eng.pop('leone')
3  print(ita_eng)
```

```
OUTPUT
{'cane': 'dog', 'gatto': 'cat'}
```

Come modificare i valori di un dizionario

Per **modificare** i **valori** di un dizionario devi:

- selezionare la chiave a cui sono assegnati i vecchi valori;
- assegnarle i nuovi valori.

`Esempio: How to change Python dictionary's values`

```
1  ita_eng = {'gatto': 'lion', 'cane': 'dog'}
2  ita_eng['gatto'] = 'cat'
3  print(ita_eng)
```

```
OUTPUT
{'gatto': 'cat', 'cane': 'dog'}
```

Come aggiungere elementi ai dizionari in Python

Per **aggiungere** una nuova **coppia chiave/valore** devi indicare la nuova chiave e assegnarle i relativi valori.

Esempio: How to change Python dictionary's values

```
1  ita_eng = {'gatto': 'cat', 'cane': 'dog'}
2  ita_eng['uccello'] = 'bird'
3  print(ita_eng)
```

```
OUTPUT
{'gatto': 'cat', 'cane': 'dog', 'uccello': 'bird'}
```

A differenza delle liste, i dizionari **non possono essere concatenati**.

Se ci provassi, Python ti mostrerebbe il seguente **messaggio di errore**.

```
TypeError: unsupported operand type(s) for +: 'dict' and 'dict'
```

```
1  ita_eng = {'gatto': 'cat'} + {'cane': 'dog'}
```

```
OUTPUT
Traceback (most recent call last):
  File "/.../.../Desktop/ex.py", line 1, in <module>
    ita_eng = {'gatto': 'cat'} + {'cane': 'dog'}
TypeError: unsupported operand type(s) for +: 'dict' and 'dict'
```

Come eliminare elementi dai dizionari in Python

Come eliminare una chiave da un dizionario

Per **eliminare** una **chiave** e tutti i suoi valori puoi utilizzare *pop()* oppure *del*.

Esempio: How to remove Python dictionary's elements

```
1  ita_eng = {'gatto': 'cat', 'cane': 'dog'}
2  ita_eng.pop('gatto') # del ita_eng['gatto']
3  print(ita_eng)
```

```
OUTPUT
{'cane': 'dog'}
```

Come eliminare un valore da un dizionario

Per **eliminare** un **valore** associato ad una chiave, devi utilizzare:

- *pop()* oppure del se conosci la **posizione** dell'elemento da eliminare.

Esempio: Python pop() method

```
1  ita_eng = {'gatto': ['cat', 'dog'], 'cane': 'dog'}
2  ita_eng['gatto'].pop(-1) # del ita_eng['gatto'][-1]
3  print(ita_eng)
```

```
OUTPUT
{'gatto': ['cat'], 'cane': 'dog'}
```

- *remove()* se conosci il **valore** dell'elemento da eliminare.

Esempio: Python remove() method

```
1  ita_eng = {'gatto': ['cat', 'dog'], 'cane': 'dog'}
2  ita_eng['gatto'].remove('dog')
3  print(ita_eng)
```

```
OUTPUT
{'gatto': ['cat'], 'cane': 'dog'}
```

Come eliminare tutti gli elementi di un dizionario

Per **eliminare tutti gli elementi** di un dizionario, devi utilizzare *clear()*.

`Esempio: Python clear() method`

```
1  ita_eng = {'gatto': ['cat', 'dog'], 'cane': 'dog'}
2  ita_eng.clear()
3  print(ita_eng)
```

```
OUTPUT
{}
```

Come eliminare un dizionario

Per **eliminare il dizionario** stesso, devi utilizzare la keyword del seguita dal nome della variabile in cui lo hai salvato.

`Esempio: Python del statement`

```
1  ita_eng = {'gatto': 'cat', 'cane': 'dog'}
2  del ita_eng
```

Definizione: che cos’è la dictionary comprehension

La ***dictionary comprehension*** è una sintassi semplice ed elegante che ti consente di creare un **nuovo dizionario** a partire da un iterabile (stringa, lista, tupla, set o dizionario).

Di seguito un esempio: supponi di avere a disposizione una lista A contenente dei numeri e di voler realizzare un dizionario B che contenga i quadrati di questi numeri.

`Esempio: Python dictionary comprehension`

```
1  A = [1, 2, 3]
2  B = {x: x ** 2 for x in A}
3  print(B)
```

```
OUTPUT
{1: 1, 2: 4, 3: 9}
```

Le principali funzioni da applicare ai dizionari in Python

dict() crea un nuovo dizionario.

`Esempio: Python dict() function`

```
1  # Creo un dizionario vuoto
2  ita_eng = dict()
3  print(ita_eng)
4  # Creo un dizionario con dei valori
5  ita_eng = dict(gatto = 'cat', cane = 'dog')
6  print(ita_eng)
```

```
OUTPUT
{}
{'gatto': 'cat', 'cane': 'dog'}
```

len() indica il numero di chiavi che compongono il dizionario.

`Esempio: Python len() function`

```
1  ita_eng = {'gatto': 'cat', 'cane': 'dog'}
2  print(len(ita_eng))
```

```
OUTPUT
2
```

Ulteriori metodi da applicare ai dizionari in Python

copy() crea una copia del dizionario.

`Esempio: Python copy() method`

```
1  ita_eng = {'gatto': 'cat', 'cane': 'dog'}
2  ita_eng_new = ita_eng.copy()
3  print(ita_eng_new)
```

```
OUTPUT
{'gatto': 'cat', 'cane': 'dog'}
```

items() crea un elenco di coppie chiave/valore sotto forma di tuple.

`Esempio: Python items() method`

```
1  ita_eng = {'gatto': 'cat', 'cane': 'dog'}
2  for coppia in ita_eng.items():
3      print(coppia)
```

```
OUTPUT
('gatto', 'cat')
('cane', 'dog')
```

update() aggiorna gli elementi di un dizionario con quelli di un altro.

Nello specifico:

- aggiunge le coppie chiave/valore inesistenti;
- aggiorna i valori delle chiavi già esistenti.

`Esempio: Python update() method`

```
1  ita_eng = {'gatto': 'cat', 'cane': 'dog'}
2  # aggiungo una nuova coppia chiave/valore
3  aggiornamento_01 = {'uccello': 'bird'}
4  ita_eng.update(aggiornamento_01)
5  print(ita_eng)
6  # aggiorno una chiave già esistente
7  aggiornamento_02 = {'gatto': 'kitty'}
8  ita_eng.update(aggiornamento_02)
9  print(ita_eng)
```

```
OUTPUT
{'gatto': 'cat', 'cane': 'dog', 'uccello': 'bird'}
{'gatto': 'kitty', 'cane': 'dog', 'uccello': 'bird'}
```

fromkeys() crea un nuovo dizionario, utilizzando le chiavi e i valori che gli vengono forniti come argomenti.

Solitamente viene utilizzato per inizializzare un dizionario con dei valori di default.

Esempio: Python fromkeys() method

```
1  ita = ['gatto', 'cane']
2  ita_eng = {}.fromkeys(ita, 'English: ...')
3  print(ita_eng)
```

OUTPUT
{'gatto': 'English: ...', 'cane': 'English: ...'}

liberamente tratto da https://www.pythoncollege.it/tutorial/