دیکشنری

دیکشنری ساختار داده‌ای در پایتون است که یک *map* از تعدادی کلید (*key*) به تعدادی مقدار (*value*) را پیاده‌سازی می‌کند. به صورت زیر می‌توان دیکشنری‌ها را مقداردهی اولیه یا فقط تعریف کرد.

```
empty_dict = {}
filled_dict = {"key_one": "value_one", "key_two": "value_2", "key_three": 3}
```

دیکشنری‌ها به این صورت نوشته می‌شوند که قبل از دو نقطه، کلید می‌آید و بعد از آن مقدار می‌آید. سپس جفت‌های کلید و مقدار با `,` از هم جدا می‌شوند.

کلیدها در دیکشنری باید از نوع غیرقابل تغییر (*immutable*) باشند، برای این که همواره خروجی تابع هش (*hash value*) برای آنها ثابت باشد ولی مقدارها می‌توانند از هر نوعی باشند.

Copy Python

```
>>> invalid_dict = {[1, 2, 3]: "123"}
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
>>> valid_dict = {(1, 2, 3):[1, 2, 3]}
```

می‌بینید که در کد بالا در جایی که کلید دیکشنری از نوع لیست (که نوعی تغییرپذیر است) می‌باشد، مفسر خطای `TypeError` می‌دهد با این مضمون که لیست، قابل هش نیست.

خواندن مقادیر از دیکشنری

دو روش برای خواندن مقادیر از دیکشنری وجود دارد.

`dict_name[key]`

می‌توان مانند لیست‌ها با قرار دادن کلید(به جای اندیس) داخل براکت، مقادیر را از دیکشنری خواند ولی در این روش اگر کلید نوشته شده در دیکشنری وجود نداشته باشد، `ValueError` می‌دهد. برای جلوگیری از این خطا، باید از روش دیگری استفاده کرد.

```
>>> dict = {"a":14 , "b":12 , "c":"promo_code"}
>>> dict["a"]
14
>>> dict["d"]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'd'
```

`dict_name.get(key[, default_value])`

متد *get* هم مانند روش بالا، کلید را به عنوان آرگومان اول خود می‌گیرد و مقدار آن کلید در دیکشنری را برمی‌گرداند. اگر این کلید در دیکشنری وجود نداشته باشد، خطایی نمی‌دهد و `None` برمی‌گرداند. آرگومان دومی نیز قبول می‌کند که در مواقعی که کلید در دیکشنری وجود ندارد به جای `None` ، آن را برمی‌گرداند.

```
>>> dict = {"a":14 , "b":12 , "c":"promo_code"}
>>> dict.get("a")
14
>>> dict.get("d")
>>> print(dict.get("d"))
None
>>> dict.get("d","default_value")
'default_value'
```

- در *shell* پایتون `None` نشان داده نمی‌شود و باید از تابع `print()` استفاده کنید.

برای بررسی وجود یک کلید در دیکشنری از کلمه‌ی `in` استفاده کنید.

```
>>> dict = {"a":14 , "b":12 , "c":"promo_code"}
>>> "a" in dict
True
>>> 14 in dict
False
```

اضافه کردن به دیکشنری

`dict_name[key] = value`

در این روش اگر کلید وجود نداشته باشد، این جفت کلید و مقدار را به دیکشنری اضافه می‌کند و اگر وجود داشته باشد، مقدار آن را به مقدار جلوی مساوی تغییر می‌دهد.

```
>>> dict = {"a":14 , "b":12 , "c":"promo_code"}
>>> dict["d"] = "v"
>>> dict
{'a': 14, 'b': 12, 'c': 'promo_code', 'd': 'v'}
>>> dict["d"] = "new_v"
>>> dict
{'a': 14, 'b': 12, 'c': 'promo_code', 'd': 'new_v'}
```

`dict_name.setdefault(key,value)`

این متد، کلید و مقداری را می‌گیرد و اگر این کلید در دیکشنری وجود نداشت، این جفت را به دیکشنری اضافه می‌کند و *value* را برمی‌گرداند. اگر کلید وجود داشت، تغییری در دیکشنری ایجاد نمی‌کند و مقدار آن کلید را برمی‌گرداند.

```
>>> dict = {"a":14 , "b":12 , "c":"promo_code"}
>>> dict.setdefault("five",5)
```

```
5
>>> dict
{'a': 14, 'b': 12, 'c': 'promo_code', 'five': 5}
>>> dict.setdefault("five",6)
5
>>> dict
{'a': 14, 'b': 12, 'c': 'promo_code', 'five': 5}
```

`dict_name.update(another_dict)`

این متد دیکشنری دیگری را به عنوان آرگومان می‌گیرد و برای هر کلید آن، اگر در این دیکشنری وجود نداشته باشد به همراه مقدار آن اضافه می‌شود و اگر وجود داشته باشد، مقدار آن به‌روزرسانی می‌شود (یعنی مقدار داخل `another_dict` جایگزین مقدار قبل می‌شود). به این کار به‌روزرسانی یک دیکشنری بر اساس دیگری می‌گویند.

```
>>> dict = {"a":14 , "b":12 , "c":"promo_code"}
>>> dict.update({"a":15 , "d":20})
>>> dict
{'a': 15, 'b': 12, 'c': 'promo_code', 'd': 20}
```

- برای حذف کردن کلیدی از دیکشنری از کلمه‌ی `del` استفاده می‌کنیم.

```
>>> dict = {"a":14 , "b":12 , "c":"promo_code"}
>>> del dict["a"]
>>> dict
{'b': 12, 'c': 'promo_code'}
```

بسته بندی و باز کردن (*packing/unpacking*)

با باز کردن یک دیکشنری داخل دیگری (به وسیله‌ی قرار دادن `**` قبل از آن) می‌توان دیکشنری بیرونی را به‌روزرسانی کرد. بسته‌بندی آرگومان‌ها به وسیله‌ی دیکشنری نیز بحث دیگری است که جلوتر در درسنامه‌ی تابع گفته خواهد شد.

```
>>> a = {"one":1 , "two":2 , "three" : 3}
>>> b = { "one":0 , **a}
>>> b
{'one': 1, 'two': 2, 'three': 3}
```

می‌بینید که مقادیر کلیدهای موجود، اصلاح شده و کلیدهای ناموجود به همراه مقادیرشان اضافه شده‌اند.

پیمایش دیکشنری

میتوان به وسیله‌ی حلقه‌ی `for` کلیدها، مقدارها یا هردو را در دیکشنری پیمود. نوشتار ساده `for` روی دیکشنری، فقط کلیدها را می‌دهد اما می‌توان با استفاده از متدهای دیگری به مقادیر نیز دست یافت.

```
>>> d = {"one":1,"two":2,"three":3,"four":4}
>>> for key in d:
...     print(key)
...
one
two
three
four
```

تابع `dict_name.keys()`

این متد یک شی قابل پیمایش (*iterable*) برمی‌گرداند که می‌توان از آن در حلقه استفاده کرد یا با اعمال تابع *list* روی آن تمام کلیدها را در یک لیست قرار داد.

```
>>> d = {"one":1,"two":2,"three":3,"four":4}
>>> type(d.keys())
<class 'dict_keys'>
>>> for item in d.keys():
...     print(item)
...
one
two
three
four
>>> list(d.keys())
['one', 'two', 'three', 'four']
```

- در پایتون از نسخه‌ی ۳.۷ کلیدهای دیکشنری ترتیب ورود خود را حفظ می‌کنند ولی در نسخه‌های قبل تضمینی برای حفظ ترتیب ورود وجود ندارد.

تابع `dict_name.values()`

این متد هم مشابه متد بالا عمل می‌کند ولی برای مقادیر دیکشنری

```
>>> d = {"one":1,"two":2,"three":3,"four":4}
>>> type(d.values())
<class 'dict_values'>
>>> for item in d.values():
...     print(item)
...
1
2
3
4
>>> list(d.values())
[1, 2, 3, 4]
```

- در این متد هم پایتون ۳.۷ و بالاتر ترتیب موجود در کلیدها را برای مقادیر حفظ می‌کند

تابع `dict_name.items()`

این متد هم مشابه متدهای بالا عمل میکند ولی به جای کلید یا مقدار جفت(تاپل)های کلید-مقدار را می‌دهد.

```
>>> d = {"one":1,"two":2,"three":3,"four":4}
>>> type(d.items())
<class 'dict_items'>
>>> for key,val in d.items():
...     print(key,val,sep=":")
...
one:1
two:2
three:3
four:4
>>> list(d.items())
[('one', 1), ('two', 2), ('three', 3), ('four', 4)]
```

در کد بالا می‌بینید که جفت کلید و مقدار به صورت تاپل برگردانده می‌شود و این تاپل به دو متغیر `key` و `val` باز شده است. این استفاده‌ی دیگر از *unpacking* است.

- در این متد نیز در پایتون ۳.۷ و بالاتر، ترتیب کلیدها رعایت می‌شود.