تابع dict

آخرین به‌روزرسانی: ۲۹ اردیبهشت ۱۴۰۵

تابع dict در پایتون یک تابع پیش‌ساخته (built-in) است که دیکشنری جدیدی می‌سازد. این تابع پنج شکل مختلف از سینتکس را پشتیبانی می‌کند و به همین دلیل انعطاف‌پذیرترین روش برای ساخت دیکشنری از منابع داده‌ای گوناگون محسوب می‌شود؛ از آرگومان‌های کلیدواژه‌ای گرفته تا لیستی از جفت‌های کلید-مقدار یا یک mapping موجود.

فهرست مطالب:

سینتکس تابع dict

سینتکس کلی تابع dict به پنج صورت زیر است:

```
dict()
dict(**kwargs)
dict(mapping)
dict(iterable)
dict(mapping, **kwargs)
```

پارامترها یا آرگومان‌های این سینتکس عبارتند از:

- آرگومان `**kwargs` یا آرگومان‌های کلیدواژه‌ای: هر آرگومان کلیدواژه‌ای که پاس داده شود به یک جفت کلید-مقدار در دیکشنری تبدیل می‌شود. کلیدها استرینگ هستند و مقادیر می‌توانند هر نوع داده‌ای باشند. در این روش کلیدها باید شناسه‌های معتبر پایتون باشند (نمی‌توانند عدد یا دارای فاصله باشند).
- آرگومان `mapping` یک آبجکت `mapping` (مانند دیکشنری موجود) است که جفت‌های کلید-مقدار آن در دیکشنری جدید کپی می‌شوند. این روش یک کپی سطحی از `mapping` ورودی ایجاد می‌کند.
- آرگومان `iterable` یک ایتریبل است که هر عنصر آن باید دقیقاً دو آبجکت داشته باشد؛ اولی به عنوان کلید و دومی به عنوان مقدار. معمولاً لیستی از تاپل‌های دوتایی یا خروجی تابع `zip` است.

مقدار بازگشتی تابع `dict` همیشه یک دیکشنری از نوع `dict` است.

راهنمای جامع دیکشنری (dict)

در مورد نوع داده دیکشنری یا همان `dict` در پایتون بیشتر بدانید: **دیکشنری (dict)**

در صورتی که تابع `dict` بدون آرگومان فراخوانی شود، یک دیکشنری خالی `{}` برمی‌گرداند.

در صورت ترکیب `mapping` یا `iterable` با `**kwargs` در تابع `dict`، ابتدا دیکشنری از آرگومان اول ساخته می‌شود، سپس آرگومان‌های کلیدواژه‌ای روی آن اعمال می‌شوند. در صورت تکراری بودن کلید، مقدار آرگومان کلیدواژه‌ای جایگزین می‌شود.

کاربردهای تابع dict

ساده‌ترین مثال از کاربرد این تابع می‌تواند حالت زیر باشد:

```
>>> dict()
{}
>>> dict(name="Ali", age=28, city="Tehran")
{'name': 'Ali', 'age': 28, 'city': 'Tehran'}
```

یا می‌توان از یک لیست جفت کلید-مقدار دیکشنری ساخت:

```
>>> dict([("name", "Zahra"), ("age", 25)])
{'name': 'Zahra', 'age': 25}
```

در صورت تکراری بودن کلید، آخرین مقدار در دیکشنری نهایی باقی می‌ماند:

```
>>> dict([("color", "red"), ("color", "blue"), ("color", "green")])
{'color': 'green'}
```

رایج‌ترین کاربردهای تابع dict عبارتند از:

- ساخت دیکشنری از لیست کلیدها و لیست مقادیر با استفاده از تابع zip،
- تبدیل لیستی از تاپل‌های دوتایی به دیکشنری،
- ساخت یک کپی سطحی از دیکشنری موجود،
- ساخت دیکشنری پویا با کلیدهای متغیر که نمی‌توان با سینتکس {} آنها را تعریف کرد.

ساخت دیکشنری با آرگومان‌های کلیدواژه‌ای: وقتی کلیدها، استرینگ‌های ساده هستند، روش کلیدواژه‌ای خواناترین شکل است:

```
>>> user = dict(username="ali_r", email="ali@example.com", is_active=True)
>>> user
{'username': 'ali_r', 'email': 'ali@example.com', 'is_active': True}
```

محدودیت مهم: در این روش کلیدها باید شناسه‌های معتبر پایتون باشند. بنابراین نمی‌توان از کلیدهای عددی، کلیدهای دارای فاصله یا کلیدهایی که با عدد شروع می‌شوند استفاده کرد:

```
>>> dict(1key="value")   # SyntaxError
>>> dict(my key="value") # SyntaxError
```

برای چنین کلیدهایی باید از روش `iterable` یا سینتکس {} استفاده کنید.

ساخت دیکشنری از zip: یکی از پرکاربردترین الگوها، ترکیب تابع dict با تابع zip است که دو لیست موازی (کلیدها و مقادیر) را به یک دیکشنری تبدیل می‌کند:

```
>>> keys = ["name", "age", "city"]
>>> values = ["Reza", 30, "Sari"]

>>> person = dict(zip(keys, values))
>>> print(person)
# output: {'name': 'Reza', 'age': 30, 'city': 'Sari'}
```

این الگو به‌ویژه هنگام کار با داده‌های جدول‌گونه (مثل خواندن از فایل CSV) بسیار مفید است.

ساخت دیکشنری از لیست تاپل‌ها: وقتی داده‌ها به صورت جفت‌های کلید-مقدار در یک لیست هستند، می‌توان مستقیماً آن را به تابع dict داد. این روش برای کلیدهایی مناسب است که شناسه معتبر پایتون نیستند:

```
>>> codes = dict([("200 OK", "OK"), ("404 Not Found", "Not Found"), ("500 Error",
"Server Error")])
>>> codes
{'200 OK': 'OK', '404 Not Found': 'Not Found', '500 Error': 'Server Error'}
```

کپی کردن دیکشنری با تابع dict: پاس دادن یک دیکشنری موجود به تابع dict یک کپی سطحی (shallow copy) از آن می‌سازد:

```
original = {"name": "Mina", "scores": [95, 87, 91]}
copy = dict(original)

copy["name"] = "Karim"

print(original["name"])  # output: Mina
print(copy["name"])      # output: Karim
```

نکته: چون کپی سطحی است، آبجکت‌های تودرتو (مثل لیست scores) هنوز به منبع اصلی اشاره دارند. برای کپی کامل باید از تابع deepcopy ماژول copy استفاده کنید.

ترکیب mapping با kwargs: می‌توان یک دیکشنری پایه را با آرگومان‌های کلیدواژه‌ای گسترش داد یا کلیدهایی از آن را بازنویسی کرد:

```
>>> defaults = {"theme": "light", "language": "fa", "notifications": True}
>>> user_settings = dict(defaults, language="en", font_size=14)

>>> print(user_settings)
# output: {'theme': 'light', 'language': 'en', 'notifications': True, 'font_size':
14}
```

در این مثال کلید language بازنویسی شده و font_size جدید اضافه شده است.

مثال واقعی از تابع dict

در یک سناریوی فرضی، شما داده‌های کاربران را از یک API دریافت می‌کنید که هر رکورد به صورت دو لیست جداگانه (فیلدها و مقادیر) ارائه می‌شود. باید این داده‌ها را به لیستی از دیکشنری‌های قابل استفاده تبدیل کنید:

```
# data from API
fields = ["id", "name", "email", "role"]
raw_users = [
    [101, "Ali Rezaei", "ali@example.com", "admin"],
    [102, "Mehdi Karimi", "mehdi@example.com", "editor"],
    [103, "Jafar Jalebi", "jafar@example.com", "viewer"],
]

# convert each record to a dictionary
users = [dict(zip(fields, row)) for row in raw_users]

# filtering
admins = [u for u in users if u["role"] == "admin"]

for admin in admins:
    print(f"Admin: {admin['name']} - Email: {admin['email']}")
```

نمونه خروجی این کد به صورت زیر خواهد بود:

```
Admin: Ali Rezaei - Email: ali@example.com
```

در این مثال، `dict(zip(fields, row))` هسته اصلی تبدیل داده است که در یک خط، دو لیست موازی را به یک دیکشنری ساختاریافته تبدیل می‌کند.

تفاوت تابع dict با {} و dict comprehension

هر سه روش دیکشنری می‌سازند، اما کاربرد متفاوتی دارند:

```python
person = {"name": "Ali", "age": 28}
person = dict(name="Ali", age=28)
person = dict(zip(["name", "age"], ["Ali", 28]))

squares = {x: x**2 for x in range(1, 6)}
```

راهنمای انتخاب:

- سینتکس `{}` برای وقتی مناسب است که کلیدها و مقادیر از پیش مشخص هستند.
- اگر کلیدها، رشته‌های متنی ساده هستند و خوانایی کد مهم است، `dict(**kwargs)` توصیه می‌شود.
- در صورتی که داده‌ها از دو لیست موازی یا لیست و تاپل می‌آید، سینتکس `dict(zip(...))` یا `dict(iterable)` مناسب‌تر است.
- اگر نیاز است تبدیل یا فیلتری روی داده‌ها اعمال شود، روش `dict comprehension` بهترین گزینه است.

سوالات متداول

1. تفاوت تابع `dict` و `{}` در پایتون چیست؟

هر دو دیکشنری می‌سازند، اما `{}` برای لیترال‌های ثابت سریع‌تر است چون مستقیم توسط مفسر پردازش می‌شود. تابع `dict` زمانی مفیدتر است که بخواهید دیکشنری را از یک ایتریبل، `mapping` موجود یا آرگومان‌های کلیدواژه‌ای بسازید.

---

2. آیا تابع `dict` می‌تواند کلیدهای عددی داشته باشد؟

با روش `kwargs` خیر، اما با روش‌های `iterable` یا `mapping` بله.

---

3. اگر کلید تکراری به تابع `dict` بدهیم چه اتفاقی می‌افتد؟

آخرین مقداری که برای آن کلید پاس داده شده در دیکشنری نهایی باقی می‌ماند و مقادیر قبلی بازنویسی می‌شوند.

---

4. آیا تابع `dict` یک کپی واقعی از دیکشنری می‌سازد؟

خیر، یک کپی سطحی می‌سازد. یعنی دیکشنری جدید یک آبجکت مستقل است، اما مقادیر تودرتو (مثل لیست‌ها یا دیکشنری‌های داخلی) هنوز به آبجکت‌های اصلی اشاره دارند.

---

5. چه زمانی باید از `dict(zip(...))` استفاده کنم؟

وقتی کلیدها و مقادیر در دو لیست یا ایتریبل جداگانه هستند و می‌خواهید آن‌ها را با هم ترکیب کنید. این الگو در پردازش داده‌های CSV، پاسخ‌های API و هر جایی که داده به صورت ستونی ذخیره شده رایج است.

جهت کسب اطلاعات بیشتر می‌توانید به **مستندات رسمی پایتون برای تابع `dict`** مراجعه کنید.

دسته: توابع پیش‌ساخته - پایتون - دانش‌نامه برنامه‌نویسی