دیکشنری (dict)

آخرین به‌روزرسانی: ۲۹ اردیبهشت ۱۴۰۵

نوع داده دیکشنری (dict) در پایتون یک مجموعه تغییرپذیر از جفت‌های کلید-مقدار است. دیکشنری‌ها یکی از پرکاربردترین انواع داده در پایتون هستند و پیاده‌سازی اصلی نوع Mapping در این زبان به شمار می‌روند. ویژگی اصلی دیکشنری آن است که به جای اندیس عددی، از کلیدهای دلخواه برای دسترسی به مقادیر استفاده می‌کند که این امر جستجو را با پیچیدگی O(1) انجام می‌دهد. از پایتون 3.7 به بعد، دیکشنری‌ها ترتیب درج کلیدها را حفظ می‌کنند و این ویژگی، بخشی رسمی از مشخصات زبان است. کلیدهای دیکشنری باید قابل هَش باشند (اعداد، استرینگ‌ها و تاپل‌ها)، در حالی که مقادیر می‌توانند از هر نوعی باشند.

فهرست مطالب:

ایجاد دیکشنری در پایتون

داده‌هایی از نوع دیکشنری در پایتون را می‌توان به روش‌های مختلفی ساخت.

۱) نوشتار مستقیم با آکولاد:

```
>>> empty = {}
>>> person = {"name": "Ali", "age": 30, "city": "Tehran"}
>>> mixed = {1: "one", "two": 2, (3, 4): [3, 4]}
```

۲) روش Dict Comprehension: پایتونیک‌ترین و بهینه‌ترین روش برای ساخت این نوع داده به حساب می‌آید:

```
>>> squares = {x: x ** 2 for x in range(1, 6)}
>>> squares
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

>>> words = ["apple", "banana", "cherry"]
>>> word_lengths = {w: len(w) for w in words}
>>> word_lengths
{'apple': 5, 'banana': 6, 'cherry': 6}

>>> inventory = {"laptop": 5, "mouse": 0, "keyboard": 3, "monitor": 0}
>>> in_stock = {k: v for k, v in inventory.items() if v > 0}
>>> in_stock
{'laptop': 5, 'keyboard': 3}
```

۳) تابع dict: سازنده نوع داده دیکشنری در پایتون، تابع `dict` است:

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

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

راهنمای جامع تابع dict

در مورد تابع `dict` که سازنده یا `Constructor` برای دیکشنری در پایتون است بیشتر بدانید: **آشنایی با تابع dict در پایتون**

نکته: کلیدهای دیکشنری باید قابل هَش (hashable) باشند. اعداد، رشته‌های متنی و تاپل‌های حاوی اشیای قابل هش می‌توانند کلید باشند. لیست‌ها و دیکشنری‌های دیگر نمی‌توانند کلید باشند، زیرا تغییرپذیرند.

عملیات پایه روی دیکشنری

در ادامه، تعدادی مثال از عملیات روی دیکشنری‌ها را با هم مرور خواهیم کرد.

دسترسی به مقادیر: مشابه لیست‌ها، دسترسی به اعضای دیکشنری به وسیله کلید امکانپذیر است.

```
>>> person = {"name": "Ali", "age": 30, "city": "Tehran"}

>>> person["name"]
'Ali'

>>> person["age"]
30
```

افزودن و تغییر مقادیر: با سینتکس بالا، می‌توان جفت‌های کلید-مقدار جدید نیز تعریف کرد یا تغییراتی اعمال کرد.

```
>>> person["email"] = "ali@example.com"
>>> person["age"] = 31

>>> person
{'name': 'Ali', 'age': 31, 'city': 'Tehran', 'email': 'ali@example.com'}
```

طول دیکشنری و بررسی عضویت: با استفاده از عملگرهای منطقی `in` و `not in` می‌توان بررسی کرد که آیا یک آیتم درون کلیدهای دیکشنری قرار دارد یا خیر:

```
>>> person = {"name": "Ali", "age": 30, "city": "Tehran"}

>>> "name" in person
True

>>> "phone" not in person
True

>>> len(person)
3
```

ادغام: از نسخه 3.9 پایتون به بعد، عملگر `|` را می‌توان برای ادغام دیکشنری‌ها استفاده کرد.

```
>>> defaults = {"theme": "dark", "lang": "en", "timeout": 30}
>>> user_prefs = {"lang": "fa", "timeout": 60}

>>> merged = defaults | user_prefs
>>> merged
{'theme': 'dark', 'lang': 'fa', 'timeout': 60}

>>> defaults |= user_prefs
>>> defaults
{'theme': 'dark', 'lang': 'fa', 'timeout': 60}
```

هنگام ادغام ۲ دیکشنری با عملگر `|`، دیکشنری دوم تمامی کلیدهای تکراری دیکشنری اول را بازنویسی (`override`) می‌کند.

متدهای پرکاربرد دیکشنری

نوع `dict` در پایتون، مجموعه‌ای غنی از متدها دارد که عملیات دسترسی ایمن، پیمایش، تغییر و ادغام را پوشش می‌دهند.

۱) متدهای دسترسی ایمن:

- متد `get` – دسترسی به مقدار با کلید مشخص
- متد `setdefault` – دسترسی به مقدار با کلید مشخص یا تعریف کلید-مقدار جدید

```
>>> person = {"name": "Ali", "age": 30}

>>> person.get("name")
'Ali'

>>> person.get("phone")
None

>>> person.get("phone", "N/A")
'N/A'

>>> person.setdefault("name", "Unknown")
'Ali'

>>> person.setdefault("city", "Tehran")
'Tehran'
>>> person
{'name': 'Ali', 'age': 30, 'city': 'Tehran'}
```

هنگام دسترسی به مقادیر در دیکشنری، اگر کلیدی که داخل سینتکس person["name"] وارد شده، وجود نداشته باشد، پایتون خطای KeyError می‌دهد. متد get اگر کلید وجود نداشته باشد، None یا مقدار پیش‌فرض را برمی‌گرداند. هنگام استفاده از متد setdefault، اگر کلید وجود داشته باشد، پایتون مقدارش را برمی‌گرداند. اگر وجود نداشته باشد، کلید را با مقدار پیش‌فرض داخل دیکشنری درج می‌کند و همان مقدار را برمی‌گرداند.

نکته: در اکثر موارد که می‌خواهید مقدار یک کلید را بخوانید، get از [] ایمن‌تر است چون برنامه را متوقف نمی‌کند.

۲) متدهای نما: این سه متد، نما (view) برمی‌گردانند؛ اشیایی که به صورت زنده محتوای دیکشنری را منعکس می‌کنند و با تغییر دیکشنری به‌روز می‌شوند.

- متد keys – نمای تمام کلیدها
- متد values – نمای تمام مقادیر
- متد items – نمای تمام جفت‌های کلید-مقدار به صورت تاپل

```
>>> person = {"name": "Ali", "age": 30, "city": "Tehran"}

>>> person.keys()
dict_keys(['name', 'age', 'city'])

>>> person.values()
dict_values(['Ali', 30, 'Tehran'])

>>> person.items()
dict_items([('name', 'Ali'), ('age', 30), ('city', 'Tehran')])
```

نکته مهم: نماهای keys و values و items اشیای زنده هستند! اگر دیکشنری بعد از ساخت نما تغییر کند، نما هم به‌روز می‌شود.

```
>>> d = {"a": 1}
>>> keys_view = d.keys()

>>> d["b"] = 2
>>> keys_view              # updated view
dict_keys(['a', 'b'])
```

۳) متدهای حذف:

- متد pop – حذف کلید مشخص
- متد popitem – حذف آخرین جفت کلید-مقدار درج‌شده
- متد clear – حذف تمامی کلیدها و مقادیر (دیکشنری خالی باقی می‌ماند)

```
>>> person = {"name": "Ali", "age": 30, "city": "Tehran"}

>>> person.pop("city")
'Tehran'
>>> person
{'name': 'Ali', 'age': 30}
>>> person.pop("phone", "N/A")
'N/A'

>>> person.popitem()
('age', 30)
>>> person
{'name': 'Ali'}

>>> person.clear()
>>> person
{}
```

۴) متد به‌روزرسانی و ادغام: متد `update` به‌روزرسانی دیکشنری با کلیدها و مقادیر از منبع دیگر را انجام می‌دهد. کلیدهای موجود بازنویسی می‌شوند، کلیدهای جدید اضافه می‌شوند:

```
>>> person = {"name": "Ali", "age": 30}

>>> person.update({"age": 31, "city": "Tehran"})
>>> person
{'name': 'Ali', 'age': 31, 'city': 'Tehran'}

>>> person.update(email="ali@example.com", phone="09123456789")
>>> person
{'name': 'Ali', 'age': 31, 'city': 'Tehran', 'email': 'ali@example.com', 'phone':
'09123456789'}
```

۵) متد کپی: متد `copy` برای ایجاد یک کپی سطحی (`shallow copy`) از `dict` در پایتون استفاده می‌شود.

```
>>> original = {"name": "Ali", "scores": [90, 85, 92]}

>>> shallow = original.copy()
>>> shallow
{'name': 'Ali', 'scores': [90, 85, 92]}
```

برای کپی کامل باید از تابع deepcopy ماژول copy استفاده کنید.

۶) متد fromkeys: این متد برای ساخت دیکشنری جدید با کلیدهای مشخص و یک مقدار ثابت به کار می‌رود.

```
>>> dict.fromkeys(["name", "age", "city"])
{'name': None, 'age': None, 'city': None}

>>> dict.fromkeys(["a", "b", "c"], 0)
{'a': 0, 'b': 0, 'c': 0}
```

تذکر: اگر مقدار پیش‌فرض یک شیء تغییرپذیر مثل لیست باشد، تمام کلیدها همان یک شیء را به اشتراک می‌گذارند؛ نه کپی از آن!

```
>>> d = dict.fromkeys(["a", "b", "c"], [])
>>> d["a"].append(1)
>>> d
{'a': [1], 'b': [1], 'c': [1]}
```

پیمایش دیکشنری

روش‌های مختلفی برای پیمایش dict در پایتون وجود دارد. می‌توانید فقط روی کلیدها پیمایش کنید یا روی مقادیر یا روی زوج‌های مرتبی از کلیدها و مقادیر:

```
>>> scores = {"Ali": 92, "Sina": 88, "Reza": 75, "Maryam": 95}

>>> for key in scores:
...     print(key)

>>> for key in scores.keys():
...     print(key)

>>> for value in scores.values():
...     print(value)

>>> for key, value in scores.items():
...     print(f"{key}: {value}")
```

دیکشنری‌های تودرتو

دیکشنری‌ها می‌توانند شامل دیکشنری‌های دیگر باشند که ساختارهای داده‌ای پیچیده‌تر را نمایش می‌دهند:

```
>>> company = {
...     "name": "Tech Corp",
...     "departments": {
...         "engineering": {"head": "Sina", "count": 15},
...         "marketing":   {"head": "Ali",  "count": 8},
...         "hr":          {"head": "Reza", "count": 5},
...     },
... }

>>> company["departments"]["engineering"]["head"]
'Sina'

>>> company["departments"]["marketing"]["count"]
8

>>> for dept, info in company["departments"].items():
...     print(f"{dept}: {info['count']} employees, head: {info['head']}")

engineering: 15 employees, head: Sina
marketing: 8 employees, head: Ali
hr: 5 employees, head: Reza
```

دسترسی ایمن به دیکشنری‌های تودرتو (nested) از طریق زیر ممکن است:

```
>>> company.get("departments", {}).get("finance", {}).get("head", "N/A")
'N/A'
```

مثال واقعی از کاربرد دیکشنری

در یک سناریوی فرضی، شما سیستمی برای مدیریت موجودی فروشگاه می‌نویسید. این سیستم باید محصولات را ذخیره، جستجو، به‌روزرسانی و گزارش‌گیری کند. در این مثال از متد get برای دسترسی ایمن، متد setdefault برای گروه‌بندی، نمای items برای پیمایش، تابع update_stock برای تغییر مستقیم مقادیر درون دیکشنری‌های تودرتو و مفهوم dict comprehension برای اعمال فیلتر استفاده شده است.

کدهای این مثال به همراه خروجی نهایی را می‌توانید از اینجا دانلود کنید.

سوالات متداول

1. تفاوت `d[key]` و `d.get(key)` چیست؟

هنگام دسترسی به مقادیر دیکشنری با `d[key]` اگر کلید وجود نداشته باشد خطای `KeyError` می‌دهد و برنامه را متوقف می‌شود اما `d.get(key)` مقدار `None` (یا مقدار پیش‌فرضی که خودتان تعیین کنید) را برمی‌گرداند. از `d[key]` زمانی استفاده کنید که مطمئنید کلید وجود دارد و نبودش خطای منطقی است. از `get` زمانی استفاده کنید که نبود کلید، حالت عادی است.

---

2. چرا از پایتون نسخه 3.7 به بعد دیکشنری‌ها ترتیب درج را حفظ می‌کنند؟

پیاده‌سازی داخلی `dict` در `CPython` 3.6 به گونه‌ای بهینه‌سازی شد که ترتیب درج را به عنوان اثر جانبی حفظ می‌کرد. در پایتون 3.7 این رفتار بخشی رسمی از مشخصات زبان شد؛ یعنی تمام پیاده‌سازی‌های سازگار با پایتون، ملزم به حفظ این ترتیب هستند.

---

3. تفاوت متد `update` با عملگر `|` چیست؟

متد `update` دیکشنری را درجا تغییر می‌دهد و `None` برمی‌گرداند. عملگر `|` یک دیکشنری جدید می‌سازد و دو دیکشنری اصلی را دست نخورده می‌گذارد. عملگر `|=` معادل `update` است.

---

4. آیا می‌توان روی دیکشنری حین پیمایش تغییر داد؟

خیر. تغییر اندازه دیکشنری (افزودن یا حذف کلید) در حین پیمایش مستقیم خطای `RuntimeError` می‌دهد. راه‌حل: روی یک کپی از کلیدها پیمایش کنید.

---

5. متد `setdefault` چه مزیتی نسبت به بررسی دستی دارد؟

متد `setdefault`عملیات بررسی وجود کلید و درج مقدار پیش‌فرض را در یک فراخوانی اتمیک انجام می‌دهد.

جهت کسب اطلاعات بیشتر می‌توانید به **مستندات رسمی پایتون برای نوع داده دیکشنری (`dict`)** مراجعه کنید.

دسته: انواع داده - پایتون - دانش‌نامه برنامه‌نویسی