

دیکشنری‌ها (Dictionaries) در پایتون

دیکشنری (Dictionary) یکی از مهم‌ترین ساختارهای داده‌ای در پایتون است که برای ذخیره داده‌های کلید-مقدار استفاده می‌شود. برخلاف لیست‌ها که فقط بر اساس ایندکس مقدارها را ذخیره می‌کنند، دیکشنری‌ها داده‌ها را بر اساس کلید (Key) ذخیره کرده و به آن‌ها دسترسی سریع می‌دهند.

۱. ایجاد دیکشنری

✓ تعریف دیکشنری با {}

```
# تعریف دیکشنری شامل اطلاعات یک فرد
person = {
    "name": "Ali",
    "age": 25,
    "city": "Tehran"
}
```

✓ تعریف دیکشنری با dict()

```
person = dict(name="Ali", age=25, city="Tehran")
```

✓ دیکشنری خالی

```
empty_dict = {}
```

۲. دسترسی به مقدارها

برای دسترسی به مقدار یک کلید، می‌توان از براکت [] یا متد get() استفاده کرد.

```
person = {"name": "Ali", "age": 25, "city": "Tehran"}
```

```
# روش ۱: دسترسی با []
print(person["name"]) # خروجی: Ali
```

```
# روش ۲: دسترسی با get()
print(person.get("age")) # خروجی: 25
```

🔥 تفاوت [] و get()

• اگر کلید وجود نداشته باشد، [] خطا می‌دهد ولی get() مقدار None برمی‌گرداند.

```
print(person.get("country")) # خروجی: None
# print(person["country"]) # خطا: KeyError: 'country'
```

۳. افزودن یا تغییر مقادارها

```
person["job"] = "Engineer" # افزودن کلید جدید
person["age"] = 26 # تغییر مقدار کلید موجود

print(person)
# خروجی: {'name': 'Ali', 'age': 26, 'city': 'Tehran', 'job': 'Engineer'}
```

۴. حذف مقادارها

حذف یک کلید → `del` ✓

```
del person["city"]
print(person) # خروجی: {'name': 'Ali', 'age': 26, 'job': 'Engineer'}
```

حذف و دریافت مقدار → `pop()` ✓

```
age = person.pop("age")
print(age) # خروجی: 26
print(person) # خروجی: {'name': 'Ali', 'job': 'Engineer'}
```

حذف آخرین عنصر → `popitem()` ✓

```
last_item = person.popitem()
print(last_item) # خروجی: ('job', 'Engineer')
print(person) # خروجی: {'name': 'Ali'}
```

تفاوت `del` و `pop()` :

- `del` فقط مقدار را حذف می‌کند.
- `pop()` مقدار حذف‌شده را برمی‌گرداند.
- `popitem()` آخرین مقدار را حذف و برمی‌گرداند (در پایتون ۳.۷ به بعد، دیکشنری‌ها ترتیب‌دار هستند).

۵. بررسی وجود کلید در دیکشنری

```
if "name" in person:
    print("Key exists")
print("salary" in person) # خروجی: False
```

۶. توابع کاربردی دیکشنری

دریافت لیست کلیدها → `keys()` 

```
print(person.keys()) # خروجی: dict_keys(['name', 'age', 'city'])
```

دریافت لیست مقادیر → `values()` 

```
print(person.values()) # خروجی: dict_values(['Ali', 25, 'Tehran'])
```

دریافت کلیدها و مقادیر به صورت tuple → `items()` 

```
print(person.items())  
# خروجی: dict_items([('name', 'Ali'), ('age', 25), ('city', 'Tehran')])
```

۷. پیمایش (Loop) در دیکشنری

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

خروجی:

```
name: Ali  
age: 25  
city: Tehran
```

۸. ادغام دو دیکشنری

ترکیب دو دیکشنری → `update()` 

```
extra_info = {"gender": "Male", "city": "Shiraz"}  
person.update(extra_info)  
  
print(person)  
# خروجی: {'name': 'Ali', 'age': 25, 'city': 'Shiraz', 'gender': 'Male'}
```

🔥 اگر کلیدی مشترک باشد، مقدار جدید جایگزین مقدار قبلی می‌شود.

۹. استفاده از `defaultdict` برای مقادیرهای پیش‌فرض

`defaultdict` از ماژول `collections` کمک می‌کند که اگر کلید وجود نداشت، مقدار پیش‌فرض مشخصی برگردد.

```
from collections import defaultdict

# دیکشنری معمولی خطا می‌دهد اگر کلید وجود نداشته باشد
normal_dict = {}
# print(normal_dict["age"]) # ❌ KeyError

# استفاده از defaultdict
default_dict = defaultdict(int) # فرض 0 است
print(default_dict["age"]) # خروجی: 0
```

مثال دیگر با لیست:

```
from collections import defaultdict

word_count = defaultdict(list)
word_count["python"].append("awesome")
word_count["java"].append("good")

print(word_count)
# خروجی: {'python': ['awesome'], 'java': ['good']}
```

۱۰. دیکشنری‌های تو در تو (Nested Dictionaries)

```
students = {
    "student1": {"name": "Ali", "age": 20},
    "student2": {"name": "Sara", "age": 22}
}

print(students["student1"]["name"]) # خروجی: Ali
```

۱۱. تبدیل لیست به دیکشنری

استفاده از zip () 

```
keys = ["name", "age", "city"]
values = ["Ali", 25, "Tehran"]

person = dict(zip(keys, values))
print(person)
# خروجی: {'name': 'Ali', 'age': 25, 'city': 'Tehran'}
```

۱۲. حذف تمام مقادیر دیکشنری


`() clear` 

```
person.clear()
print(person) # خروجی: {}
```

۱۳. کپی کردن دیکشنری

`() copy` → کپی سطحی (Shallow Copy) 

```
new_dict = person.copy()
```

`() deepcopy` → کپی عمیق (Deep Copy) 

```
import copy

deep_copy_dict = copy.deepcopy(students)
```

🚀 در `() copy` تغییر مقادیر تو در تو روی هر دو دیکشنری اعمال می‌شود، اما در `() deepcopy` این مشکل وجود ندارد.

جمع‌بندی

- ✓ دیکشنری‌ها ساختاری سریع و منعطف برای نگهداری داده‌ها هستند.
- ✓ دسترسی به داده‌ها بر اساس کلید انجام می‌شود، نه ایندکس.
- ✓ عملیات افزودن، حذف، و تغییر مقدار در آن‌ها بسیار ساده است.
- ✓ توابع کاربردی مانند `keys()`، `values()`، `items()` و `defaultdict` بهینه‌سازی کار با دیکشنری را آسان‌تر می‌کنند.
- ✓ برای داده‌های مرتبط با هم و دارای ساختار کلید-مقدار، دیکشنری بهترین انتخاب است. 🚀