defaultdict در پایتون

پایتونیک کدزدن به چه معناست؟
اگر در کدنویسی پایتون تجربه داشته باشید، حتما اصطلاح پایتونیک کدزدن را شنیده‌اید. در زبان پایتون مانند دیگر زبان‌ها امکان پیاده‌سازی یک برنامه به شکل های مختلفی وجود دارد، اما معمولا یک راه وجود دارد که به راه های دیگر ترجیح داده می‌شود که این راه را روش پایتونیک پیاده سازی آن راه حل می گویند. با یادگیری این راه ها و قواعد می‌توانید برنامه نویس پایتون بهتری باشید.

عرفان صابری
دانشجو مهندسی کامپیوتر
دانشکدگان فارابی دانشگاه تهران
erfansaberiow@gmail.com

بسیار پیش می‌آید که نیاز داشته باشیم از دیکشنری برای حل مسائلی مانند شمارش تعداد تکرار اشیا و ... استفاده کنیم، می‌دانیم که ممکن است برخی کلید ها در دیکشنری وجود نداشته باشند، بنابراین به روش های مختلفی مانند استفاده از بلاک های try و except سعی می کنیم به این موارد رسیدگی کنیم. به کد زیر نگاه کنید

```
>>> counts = {'A':2, 'B':3}
>>> print(counts['A'])
2
>>> print(counts['C'])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'C'
```

در اینجا چون کلید C در دیکشنری ما وجود ندارد، پایتون برای ما یک Exception از نوع KeyError ایجاد می کند که توسط بلاک های try و except قابل رسیدگی است.

```
>>> try:
...     print(counts['C'])
... except KeyError:
...     print(0)
0
```

می‌بینیم که پایتون خطای ایجاد شده را مدیریت کرده و مقدار ۰ را نمایش می‌دهد، چرا که در صورت عدم وجود کلید، قاعدتا تعداد تکرار آن نیز باید ۰ بوده باشد. اما در این مسئله شاید بلاک try و except چندان از جهت خوانایی مناسب نباشد و باعث شود تعداد خطوط کد ما افزایش پیدا کند. به جای استفاده از این بلاک، میتوانیم از متد get کمک بگیریم. این متد در صورتی که مقدار مورد نظر وجود نداشته باشد، مقدار پیشفرضی که به آن داده‌ایم را برمیگرداند.

```
>>> print(counts.get('C', 0))
0
```

همچنین در پیاده سازی کدی که برای شمارش استفاده می کنیم نیز داریم:

```
>>> for c in chars:
...     counts[c] = counts.get(c, 0) + 1
```

و می‌بینیم که لازم است هر بار مقدار پیشفرض را به متد get پاس دهیم، آیا راهی وجود دارد که از تکرار چندباره این خط کد خلاص شویم؟ بله؛ کانتینر defaultdict از کتابخانه collections. پیش‌تر راجع به دیگر کانتینر های موجود در این کتابخانه صحبت کردیم، در این شماره به بررسی deafultdict می‌پردازیم.

مدیریت کلیدهای ناموجود

کانتینر defaultdict با هدف مقداردهی خودکار کلید های ناموجود طراحی شده. این کانتینر در متد سازنده خود یک تابع دریافت می‌کند و در صورتی که کلید فراخوانی شده موجود نبود، مقدار اولیه را از آن تابع دریافت کرده و با کلید موردنظر به دیکشنری اضافه می‌کند. همچنین میتواند یک کلاس دریافت کند و با فراخوانی متد سازنده آن کلاس، یک شی از آن کلاس به عنوان مقدار اولیه بسازد.

```
>>> from collections import defaultdict
>>> counts = defaultdict(int)
>>> print(counts[1])
0
>>> print(counts)
defaultdict(<class 'int'>, {1: 0})
```

در این مثال، defaultdict در هنگام مواجهه با کلید ۱ که ناموجود است، متد سازنده int را فراخوانی می‌کند. این متد به طور پیشفرض مقدار صفر را به شی ساخته شده می‌دهد، بنابراین مقدار ۰ به کلید موردنظر داده می‌شود. همچنین می‌توان به جای دادن کلاس، یک تابع به defaultdict داد.

```
>>> def default():
…     return 0
>>> counts = defaultdict(default)
```

این کانتینر نیز مانند کانتینر قبلی، در مواجهه با کلید های ناموجود، کلید مورد نظر را با مقدار اولیه ۰ وارد دیکشنری می کند. برای کوتاه تر شدن خطوط کد میتوان از توابع لامبدا نیز استفاده کرد. فرض کنیم میخواهیم در صورت عدم وجود یک کلید، مقدار اولیه "Not exist" در دیکشنری قرار داده شود، میتوانیم به کمک یک تابع لامبدا این کار را انجام دهیم.

```
>>> mydict = defaultdict(lambda: "Not exist")
>>> mydict[“Test”]
“Not exist”
```

حل مسئله شمارش با استفاده از defaultdict

حالا سعی می کنیم مسئله شمارش را با استفاده از defaultdict انجام دهیم. (در شماره قبل، راجع به Counter نیز صحبت کردیم، در صورتی که قبلا آن را مطالعه نکرده اید، پیشنهاد میکنیم پس از مطالعه این مطلب نگاهی نیز به مطلب Counter در شماره قبلی داشته باشید)

راه حلی که برای پیاده سازی مسئله شمارش استفاده میکنیم، کمک گرفتن از یک دیکشنری و استفاده از یک حلقه بر روی ورودی مورد نظر است. به این صورت که روی تک تک آیتم های ورودی مورد نظرمان یک بار حلقه زده و برای هر آیتم، اگر در دیکشنری موجود نبود مقدار آن را برابر صفر قرار دهیم و سپس به مقدار آن یکی اضافه کنیم.

```
>>> text = “mississippi”
>>> counts = {}
>>> for c in text:
…     if c not in counts:
…         counts[c] = 0
…     counts[c] += 1
>>> print(counts)
{'m': 1, 'i': 4, 's': 4, 'p': 2}
```

حالا به جای اینکه از دیکشنری عادی استفاده کنیم و کلید های ناموجود را با استفاده از if مدیریت کنیم، از defaultdict استفاده می کنیم و کدمان را بازنویسی می کنیم. اینبار اجازه می دهیم defaultdict کلید های ناموجود را مدیریت کند.

```
>>> from collections import defaultdict
>>> text = “mississippi"
>>> counts = defaultdict(int)
>>> for c in text:
…     counts[c] += 1
>>> print(counts)
defaultdict(<class 'int'>, {'m': 1, 'i':
4, 's': 4, 'p': 2})
>>> print(counts[‘i’])
4
>>> print(dict(counts))
{'m': 1, 'i': 4, 's': 4, 'p': 2}
```

مشاهده میکنیم که defaultdict، مقادیری که از قبل داخل دیکشنری وجود نداشتند را با مقدار اولیه ۰ داخل دیکشنری قرار می دهد تا یکی به مقدار آن اضافه شود و در نهایت، تعداد تکرار هر کدام از آیتم ها برای ما مشخص شوند. در حل این مسئله روش بهتری نیز با استفاده از Counter از همین کتابخانه وجود دارد، پیشنهاد میکنیم با مراجعه به شماره قبلی و مطالعه مطلب Counter با آن روش و دیگر روش های حل این مسئله آشنا شوید. کاربرد defaultdict صرفا به حل مسئله شمارش منحصر نمی شود، ممکن است به مسائل دیگری نیز برخورد کنید که حالت مشابهی با این مسئله داشته باشند و آنجا، defaultdict به شما کمک می کند با تعداد خط کد کمتر، مسئله مورد نظر را حل کنید.

منابع

https://docs.python.org
https://realpython.com
https://www.geeksforgeeks.org