کتابخانه `collections`

کتابخانه `collections` در پایتون انواع داده‌ای ویژه‌ای ارائه می‌دهد که برای انجام عملیات‌های خاص بهینه‌شده‌اند. در این بخش، به معرفی برخی از انواع داده‌ای مهم این کتابخانه پرداخته‌ایم.

---

1. استفاده از `Counter`

`Counter` یک کلاس است که برای شمارش تعداد تکرار هر عنصر در یک داده (مانند لیست یا رشته) استفاده می‌شود. این نوع داده به‌ویژه برای شمارش فراوانی عناصر بسیار مفید است.

ویژگی‌ها و استفاده‌ها:

- شمارش تعداد تکرار هر عنصر در داده.
- استفاده از متد `()most_common` برای یافتن پر تکرارترین عناصر.

```
from collections import Counter

# شمارش تعداد تکرار عناصر در یک لیست
data = ['apple', 'banana', 'apple', 'orange', 'banana', 'banana']
counter = Counter(data)

print(counter)  # خروجی: Counter({'banana': 3, 'apple': 2, 'orange': 1})
print(counter.most_common(2))  # خروجی: [('banana', 3), ('apple', 2)]
```

`Counter` مناسب برای انجام تحلیل‌های ساده‌ای مانند پیدا کردن پر تکرارترین کلمات در یک متن است.

---

2. استفاده از `deque`

`deque` (short for "double-ended queue") یک نوع داده است که به شما امکان می‌دهد به‌طور مؤثر از هر دو انتهای لیست به‌طور همزمان استفاده کنید. برخلاف لیست‌ها، `deque` برای افزودن و حذف عناصر از ابتدا و انتهای لیست بهینه‌شده است.

ویژگی‌ها و استفاده‌ها:

- افزودن و حذف سریع از دو طرف لیست.
- کارایی بالا در عملیات‌هایی که نیاز به افزودن یا حذف از ابتدا و انتهای داده دارند.

```
from collections import deque

# و افزودن و حذف از دو طرف deque ایجاد یک
d = deque([1, 2, 3])
d.append(4)  # افزودن به انتهای لیست
d.appendleft(0)  # افزودن به ابتدای لیست
print(d)  # خروجی: deque([0, 1, 2, 3, 4])

d.pop()  # حذف از انتهای لیست
d.popleft()  # حذف از ابتدای لیست
print(d)  # خروجی: deque([1, 2, 3])
```

مقایسه `deque` با لیست‌ها:

- عملیات افزودن و حذف در `deque` از هر دو طرف به مراتب سریع‌تر از لیست‌ها است.
- `deque` برای صف‌ها (queues) و پشته‌ها (stacks) مناسب است.

---

3. استفاده از `defaultdict`

`defaultdict` یک نوع دیکشنری است که به شما امکان می‌دهد برای هر کلید یک مقدار پیش‌فرض تعیین کنید. این ابزار برای جلوگیری از بروز خطاها هنگام دسترسی به کلیدهای غیرموجود مفید است.

ویژگی‌ها و استفاده‌ها:

- جلوگیری از خطاهای `KeyError` هنگام دسترسی به کلیدهای غیرموجود.
- استفاده از انواع مختلف به‌عنوان پیش‌فرض (مانند لیست، عدد صحیح و غیره).

```
from collections import defaultdict

# استفاده از defaultdict با لیست به‌عنوان پیش‌فرض
dd = defaultdict(int)  # مقدار پیش‌فرض صفر برای هر کلید
dd['apple'] += 1
dd['banana'] += 2

print(dd)  # خروجی: defaultdict(<class 'int'>, {'apple': 1, 'banana': 2})

# استفاده از defaultdict با لیست به‌عنوان پیش‌فرض
dd_list = defaultdict(list)
dd_list['fruits'].append('apple')
dd_list['fruits'].append('banana')
print(dd_list)  # خروجی: defaultdict(<class 'list'>, {'fruits': ['apple', 'banana']})
```

`defaultdict` برای شمارش و گروه‌بندی داده‌ها به‌ویژه در مواردی که کلیدهای جدید به‌طور مکرر اضافه می‌شوند، بسیار مفید است.

---

4. استفاده از `namedtuple`

`namedtuple` یک نوع داده شبیه به کلاس‌ها است که ویژگی‌های نامگذاری‌شده به داده‌ها می‌دهد، اما با مصرف حافظه کمتر و کارایی بالاتر نسبت به استفاده از کلاس‌ها یا دیکشنری‌ها.

ویژگی‌ها و استفاده‌ها:

- تعریف ساختارهای داده‌ای مشابه به کلاس‌ها با ویژگی‌های نامگذاری‌شده.
- مقایسه با دیکشنری‌ها: `namedtuple` ساده‌تر و سبک‌تر است.

```python
from collections import namedtuple

# تعریف یک namedtuple به نام Point
Point = namedtuple('Point', ['x', 'y'])

# ایجاد یک شیء از Point
p = Point(1, 2)

print(p)  # خروجی: Point(x=1, y=2)
print(p.x)  # خروجی: 1
```

`namedtuple` برای تعریف ساختارهای داده‌ای با ویژگی‌های مشخص مفید است و جایگزین مناسبی برای دیکشنری‌ها یا کلاس‌ها در مواردی که ویژگی‌ها ثابت هستند، می‌باشد.

---

5. استفاده از `OrderedDict`

`OrderedDict` دیکشنری‌ای است که ترتیب اضافه شدن عناصر را حفظ می‌کند. در دیکشنری‌های معمولی از نسخه‌های قدیمی‌تر پایتون، ترتیب عناصر به‌طور پیش‌فرض حفظ نمی‌شد، اما `OrderedDict` این مشکل را حل می‌کند.

ویژگی‌ها و استفاده‌ها:

- حفظ ترتیب اضافه شدن عناصر در دیکشنری.
- مفید برای استفاده‌هایی که به ترتیب ورودی‌ها نیاز دارند.

```python
from collections import OrderedDict

# ایجاد یک OrderedDict
od = OrderedDict([('apple', 1), ('banana', 2), ('orange', 3)])

print(od)  # خروجی: OrderedDict([('apple', 1), ('banana', 2), ('orange', 3)])

# اضافه کردن عنصر جدید و حفظ ترتیب
od['grape'] = 4
print(od)  # خروجی: OrderedDict([('apple', 1), ('banana', 2), ('orange', 3), ('grape', 4)])
```

`OrderedDict` برای استفاده‌هایی که به ترتیب وارد شدن داده‌ها نیاز دارند، مانند مدیریت تاریخچه فعالیت‌ها یا پیکربندی‌های سفارشی، مناسب است.

نتیجه‌گیری

کتابخانه `collections` در پایتون ابزارهای قدرتمندی برای کار با انواع داده‌های خاص فراهم می‌کند که به شما کمک می‌کند تا عملکرد برنامه‌های خود را بهبود بخشید. `Counter` برای شمارش فراوانی عناصر، `deque` برای کار با صف‌ها و پشته‌ها، `defaultdict` برای مدیریت دیکشنری‌ها بدون خطا، `namedtuple` برای تعریف داده‌های نامگذاری‌شده، و `OrderedDict` برای حفظ ترتیب داده‌ها استفاده می‌شود.