ماژول Collections

ماژول Collections یک ماژول درونی است که انواع داده‌های ویژه‌ای را ارائه می‌دهد که جایگزینی برای مجموعه‌های ساخته شده در پایتون با اهداف عمومی می‌باشند. ما قبلاً درباره‌ی مبانی آنها صحبت کردیم: set، list، dict و tuple.

حالا درباره‌ی جایگزین‌هایی که ماژول Collections ارائه می‌دهد، خواهیم آموخت.

شمارنده (Counter)

شمارنده (Counter) یک *زیرکلاس (subclass)* از *دیکشنری (dict)* است که به شمارش اشیاءی که می‌توانند هش شوند کمک می‌کند. در داخل آن، عناصر به عنوان کلیدهای دیکشنری ذخیره می‌شوند و تعداد اشیاء به عنوان مقدار ذخیره می‌شوند.

بیایید ببینیم چگونه می‌توان از آن استفاده کرد:

In [1]:

```
from collections import Counter
```

Counter() با لیست ها

In [2]:

```
lst = [1,2,2,2,2,3,3,3,1,2,1,12,3,2,32,1,21,1,223,1]

Counter(lst)
```

Out[2]:

```
Counter({1: 6, 2: 6, 3: 4, 12: 1, 32: 1, 21: 1, 223: 1})
```

Counter() با رشته ها

In [3]:

```
Counter('aabsbsbsbhshhbbsbs')
```

Out[3]:

```
Counter({'a': 2, 'b': 7, 's': 6, 'h': 3})
```

Counter با کلمات داخل یک جمله

In [4]:

```
s = 'How many times does each word show up in this sentence word times each each wor

words = s.split()

Counter(words)
```

Out[4]:

```
Counter({'How': 1,
         'many': 1,
         'times': 2,
         'does': 1,
         'each': 3,
         'word': 3,
         'show': 1,
         'up': 1,
         'in': 1,
         'this': 1,
         'sentence': 1})
```

In [6]:

```
# Methods with Counter()
c = Counter(words)

c.most_common(2)
```

Out[6]:

```
[('each', 3), ('word', 3)]
```

In [9]:

```
sum(c.values())
```

Out[9]:

```
16
```

In [10]:

```
list(c)
```

Out[10]:

```
['How',
 'many',
 'times',
 'does',
 'each',
 'word',
 'show',
 'up',
 'in',
 'this',
 'sentence']
```

In [13]:

```
1  c.most_common()[:-5-1:-1]
```

Out[13]:

```
[('sentence', 1), ('this', 1), ('in', 1), ('up', 1), ('show', 1)]
```

الگوهای مرسوم در هنگام استفاده از شی ()Counter

```
sum(c.values())                 # مجموع کل شمارش ها
c.clear()                       # ریست کردن کل شمارنده ها
list(c)                         # لیستی از مقادیر منحصر بفرد
set(c)                          # تبدیل به مجموعه
dict(c)                         # تبدیل به یک دیکشینری معمول
c.items()                       # تبدیل به لیستی از تاپل های (elem, cnt)
Counter(dict(list_of_pairs))    # تبدیل به لیستی از تاپل های (elem, cnt) جفتی
c.most_common()[:-n-1:-1]       # n تا از موارد کمتر تکرار شده
c += Counter()                  # حذف شمارش های صفر و منفی
```

defaultdict

defaultdict یک شیء مشابه دیکشنری است که تمامی متدهایی که توسط یک دیکشنری ارائه می‌شود را فراهم می‌کند، اما یک آرگومان اول (default_factory) را به عنوان نوع داده پیش‌فرض برای دیکشنری دریافت می‌کند. استفاده از defaultdict سریع‌تر از استفاده از روش dict.set_default است.

هیچگاه defaultdict یک KeyError ایجاد نخواهد کرد. هر کلیدی که وجود نداشته باشد، مقدار برگشت داده شده توسط default factory خواهد بود.

In [14]:

```
1  from collections import defaultdict
```

In [15]:

```
1  d = {}
```

In [16]:

```
1 d['one']
```

```
---------------------------------------------------------------------------
-
KeyError                                  Traceback (most recent call las
t)
Cell In[16], line 1
----> 1 d['one']

KeyError: 'one'
```

In [17]:

```
1 d.get('one')
```

In [18]:

```
1 d = defaultdict(object)
```

In [19]:

```
1 d
```

Out[19]:

```
defaultdict(object, {})
```

In [20]:

```
1 d['one']
```

Out[20]:

```
<object at 0x1ffa2369b20>
```

In [21]:

```
1 d
```

Out[21]:

```
defaultdict(object, {'one': <object at 0x1ffa2369b20>})
```

In [22]:

```
1 for item in d:
2     print(item)
```

```
one
```

همچنین می توانین برای کلیدها مقداردهی اولیه انجام دهید:

In [23]:

```
d = defaultdict(lambda: 0)
```

In [24]:

```
d['one']
```

Out[24]:

```
0
```

In [25]:

```
d
```

Out[25]:

```
defaultdict(<function __main__.<lambda>()>, {'one': 0})
```

namedtuple

تاپل استاندارد از ایندکس‌های عددی برای دسترسی به اعضای خود استفاده می‌کند، به عنوان مثال:

In [26]:

```
t = (12,13,14)
```

In [27]:

```
t[0]
```

Out[27]:

```
12
```

برای موارد استفاده ساده، این معمولاً کافی است. از طرف دیگر، به یاد داشتن اینکه برای هر مقدار از کدام ایندکس استفاده شود ممکن است منجر به خطاها شود، به خصوص اگر تاپل دارای تعداد زیادی فیلد باشد و در محلی ساخته شود که از آن دور استفاده می‌شود. یک namedtuple نام‌ها و همچنین شاخص عددی را به هر عضو اختصاص می‌دهد.

هر نوع namedtuple توسط یک کلاس جداگانه نماینده می‌شود که با استفاده از تابع سازنده `namedtuple()` ایجاد می‌شود. آرگومان‌ها نام کلاس جدید و یک رشته حاوی نام‌های عناصر است.

می‌توانید در اصل به namedtuples به عنوان راهی بسیار سریع برای ایجاد یک نوع جدید اشیاء/کلاس با برخی فیلدهای ویژگی فکر کنید. به عنوان مثال:

In [28]:

```
from collections import namedtuple
```

In [29]:

```
Dog = namedtuple('Dog', ['age', 'breed', 'name'])

sam = Dog(age = 2, breed = 'Lab', name = 'Sammy')

frank = Dog(age=2,breed='Shepard',name="Frankie")
```

ما namedtuple را با ابتدا انتقال نام نوع شیء (سگ) و سپس انتقال رشته‌ای حاوی انواع فیلدها به عنوان یک رشته با فاصله بین نام‌های فیلد ایجاد می‌کنیم. سپس می‌توانیم به ویژگی‌های مختلف آن دسترسی پیدا کنیم:

In [30]:

```
sam
```

Out[30]:

```
Dog(age=2, breed='Lab', name='Sammy')
```

In [31]:

```
sam.age
```

Out[31]:

```
2
```

In [32]:

```
sam.breed
```

Out[32]:

```
'Lab'
```

In [34]:

```
sam[0]
```

Out[34]:

```
2
```

In [35]:

```
sam[-1]
```

Out[35]:

```
'Sammy'
```

In [37]:

```
1 sam[0] = 55
```

```
---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
Cell In[37], line 1
----> 1 sam[0] = 55

TypeError: 'Dog' object does not support item assignment
```

جمع بندی

امیدوارم که حالا ببینید چقدر ماژول مجموعه‌ها (collections) در پایتون بسیار مفید است و باید به عنوان ماژول اصلی‌تان برای انجام تعداد زیادی از کارهای متداول انتخاب شود!