عبارات لامبدا، نگاشت (map) و فیلتر (filter)

حالا وقت آن است که در مورد دو تابع داخلی، فیلتر و نقشه سریع یاد بگیریم. پس از اینکه در مورد چگونگی عملکرد آنها یاد گرفتیم، می توانیم در مورد عبارت لامبدا یاد بگیریم که هنگامی که مهارت های خود را بیشتر توسعه می دهید، به درد می خورد!

تابع نگاشت

تابع **نگاشت** به شما اجازه می دهد تا یک تابع را به یک شئ قابل تکرار "نگاشت" کنید. به این معناست که شما می توانید به سرعت همان تابع را به هر آیتم در یک قابل تکرار، مانند یک لیست، فراخوانی کنید. به عنوان مثال:

In [1]:

```
def square(num):
    return num ** 2
```

In [2]:

```
my_nums = [1,2,3,4,5]
```

In [3]:

```
map(square, my_nums)
```

Out[3]:

```
<map at 0x1d811b505e0>
```

In [4]:

```
# To get the results, either iterate through map()
# or just cast to a list
list(map(square, my_nums))
```

Out[4]:

```
[1, 4, 9, 16, 25]
```

In [6]:

```
[*map(square, my_nums)]
```

Out[6]:

```
[1, 4, 9, 16, 25]
```

تابع مورد استفاده در نگاشت می تواند پیچیده تر نیز باشد:

In [7]:

```
def splicer(mystring):
    if len(mystring) % 2 == 0:
        return 'even'
    else:
        return mystring[0]
```

In [8]:

```
mynames = ['John','Cindy','Sarah','Kelly','Mike']
```

In [9]:

```
[* map(splicer, mynames)]
```

Out[9]:

```
['even', 'C', 'S', 'K', 'even']
```

تابع فیلتر

تابع فیلتر یک تکرار کننده را برمی گرداند که شامل اعضای از دنباله ورودی خواهند بود که مقدار تابع فیلتر کننده برای آنها True باشد. به این معنی که شما باید با یک تابع فیلتر کنید که یا True یا False را بازگرداند. سپس تابع مذکور و دنباله خود را بعنوان ورودی به تابع فیلتر بدهید و در مقابل فقط اعضایی را دریافت خواهید نمود که تابع برای آنها مقدار True را برگردانده است.

In [10]:

```
def check_even(num):
    return num % 2 == 0
```

In [11]:

```
nums = [0,1,2,3,4,5,6,7,8,9,10]
```

In [12]:

```
filter(check_even, nums)
```

Out[12]:

```
<filter at 0x1d8130250f0>
```

In [13]:

```
[* filter(check_even, nums)]
```

Out[13]:

```
[0, 2, 4, 6, 8, 10]
```

عبارت لامبدا

یکی از ابزارهای مفید (و برای مبتدیان، گیج کننده) پایتون عبارت لامبدا است. عبارات لامبدا به ما امکان می دهد تا توابع "ناشناس" ایجاد کنیم. این در اصل به معنای این است که می توانیم به سرعت توابع ad-hoc ایجاد کنیم بدون نیاز به تعریف صحیح یک تابع با استفاده از def.

شی های تابع با اجرای عبارات لامبدا دقیقاً همانطور که توسط defs ایجاد و اختصاص داده شده است، کار می کنند. تفاوت کلیدی وجود دارد که لامبدا را در نقش های تخصصی مفید می کند:

بدنه لامبدا یک عبارت منفرد است، نه یک بلوک از بیانات.

- بدنه لامبدا شبیه به آنچه که در بیان بازگشت def body قرار می دهیم است. ما به سادگی نتیجه را به عنوان یک عبارت تایپ می کنیم به جای بازگشت آن به صورت صریح. به دلیل محدود شدن به یک دستور، استفاده از آن نسبت به توابع معمولی کمتر است. لامبدا برای پیاده سازی توابع ساده طراحی شده است و def وظایف بزرگتر را انجام می دهد.بیایید به آرامی یک عبارت لامبدا را با تجزیه یک تابع بسازیم:

In [14]:

```
def square(num):
    result = num**2
    return result
```

In [16]:

```
square(5)
```

Out[16]:

```
25
```

تابع بالا را می توانیم خلاصه تر نماییم:

In [17]:

```
def square(num):
    return num**2
```

In [18]:

```
square(5)
```

Out[18]:

```
25
```

جالبه که می تونیم این تابع را حتی در یک سطر خلاصه کنیم

In [19]:

```
def square(num): return num**2
```

In [20]:

```
square(7)
```

Out[20]:

```
49
```

در واقع هدف عبارات لامبدا ساخت توابعی مشابه بالا می باشد. تابع بالا را می توانیم با استفاده از لامبدا بصورت زیر بازنویسی کنیم.

In [21]:

```
lambda num: num ** 2
```

Out[21]:

```
<function __main__.<lambda>(num)>
```

In [22]:

```
# You wouldn't usually assign a name to a lambda expression, this is just for demonst
lbd_square = lambda num: num ** 2
```

In [23]:

```
lbd_square(8)
```

Out[23]:

```
64
```

سوالی که شاید پیش بیاید این است که چرا باید از چنین تابعی استفاده نماییم. در بسیاری از توابع مانند نگاشت و فیلتر ما از تابع ورودی که غالبا ساختار ساده ای دارند یکبار استفاده می کنیم، از این رو معمولا از عبارات لامبدا در اینگونه سناریوها استفاده می نماییم.

In [24]:

```
[* map(lambda num: num ** 2, my_nums)]
```

Out[24]:

```
[1, 4, 9, 16, 25]
```

In [25]:

```
[* filter(lambda n: n % 2 == 0, nums)]
```

Out[25]:

```
[0, 2, 4, 6, 8, 10]
```

در ادامه به بررسی چند مثال می پردازیم. توجه به این نکته ضروری است که هرچه یک تابع پیچیده تر باشد که تبدیل آن به یک عبارت لامبدا سخت تر خواهد بود و گاهی اوقات تنها راه استفاده از یک تابع معمولی می باشد.

مثال 1: یک عبارت لامبدا برای استخراج اولین کاراکتر یک رشته:

In [26]:

```
first_letter = lambda s: s[0]
```

In [27]:

```
first_letter('Hello')
```

Out[27]:

```
'H'
```

مثال 2: یک عبارت لامبدا برای معکوس کردن یک رشته:

In [ ]:

```
lambda s: s[::-1]
```

همچنین ما می توانیم تعداد پارامتر ورودی بیشتری را برای لامبدا تعریف کنیم:

In [28]:

```
add  = lambda x,y : x + y
```

In [29]:

```
add(5, 6)
```

Out[29]:

```
11
```

عبارات لامبدا در بسیاری از کتابخانه مورد استفاده می شود برای مثلا در کتابخانه Pandas از این عبارات بسیار استفاده می شود. این کتابخانه در بحث یادگیری ماشین، داده کاوی و هوش مصنوعی بشدت کاربردی می باشد.