توابع مرتبه بالا

تابع مرتبه بالا تابعی است که یک یا چند تابع به عنوان ورودی بگیرد یا یک تابع به عنوان خروجی بدهد. اما چون به مباحث این دوره خیلی ربطی ندارد، ما در پرداختن به نحوه‌ی ساخت آن‌ها به یک مثال بسنده می‌کنیم و بیشتر استفاده از توابع مرتبه بالای معروف را می‌بینیم.

```
>>> def create_adder(x):
...     def adder(y):
...         return x + y
...     return adder
...
>>> add_10 = create_adder(10)
>>> add_10(5)
15
```

- در کد بالا تابع `create_adder` تابع دیگری تولید می‌کند و آن را خروجی می‌دهد. تابع تولید شده، عدد y را می‌گیرد و آن را با عدد ثابتی که توسط تابع مرتبه بالا تنظیم شده (`x`)، جمع می‌کند. در این مثال، تابعی خروجی داده شده که آرگومان اول خود را همواره با ۱۰ جمع می‌کند و خروجی می‌دهد.

توابع ناشناس (*anonymous functions*)

اگر بخواهیم تابعی تعریف کنیم که نیازی به اسم‌گذاری نداشته باشد و مثلا به عنوان آرگومان به تابع دیگری داده شود، از توابع ناشناس استفاده می‌کنیم. نوشتار آنها هم به شکل زیر است.

```
lambda arg1,arg2,... : expression
```

- جواب `expression` به عنوان خروجی برگردانده می‌شود.

```
>>> (lambda x: x > 2)(3)
True
>>> (lambda x, y: x ** 2 + y ** 2)(2, 1)
5
```

- در مثال بالا، استفاده از توابع ناشناس را می‌بینید.

map `map(fucntion,arg1_sequence,arg2_sequence , ...)`

یک تابع را می‌گیرد و عنصر اول `arg1_sequence` را به عنوان آرگومان اول آن و عنصر اول `arg2_sequence` را به عنوان آرگومان دوم آن و همین طور به ترتیب به تابع می‌دهد و خروجی تابع را می‌گیرد. سپس همین کار را برای عناصر دوم `sequence` ها تکرار می‌کند و خروجی را می‌گیرد و به همین شکل ادامه می‌دهد تا `sequence` ها تمام شوند. این تابع، یک شی `map` برمی‌گرداند. این شی قابل

پیمایش (*iterable*) است اما اگر بخواهید همه خروجی‌ها را یک جا داشته باشید، می‌توانید با استفاده از سازنده `list` ، آن را به لیست تبدیل کنید.

```
>>> map(lambda x:x+2 , [1,2,3])
<map object at 0x000001DD91A01048>
>>> list(map(lambda x:x+2 , [1,2,3]))
[3, 4, 5]
>>> list(map(max, [1, 2, 3], [4, 2, 1]))
[4, 2, 3]
>>> list(map(lambda x,y,z: x+y-z , [1,2,3],[4,5,6],[7,8,9]))
[-2, -1, 0]
```

filter `filter(condition_function , sequence)`

یک تابع با خروجی `True/False` و یک `sequence` را می گیرد و اعضای آن را به عنوان آرگومان به تابع می‌دهد. اعضایی را که خروجی تابع به ازای آنها `False` است، حذف می‌کند. در واقع این تابع مانند شرط فیلتر شدن عمل می‌کند.

```
>>> list(filter(lambda x: x > 5, [3, 4, 5, 6, 7]))
[6, 7]
```

reduce `reduce(function, sequence[, initial])`

دو عنصر اول `sequence` (اگر `initial` به آن داده شود، `initial` و عنصر اول `sequence`) را به تابع می‌دهد و خروجی می‌گیرد. سپس خروجی را به همراه عنصر بعدی `sequence` به تابع می‌دهد و همین طور ادامه می‌دهد تا عناصر تمام شوند. سپس جواب را خروجی می‌هد.

```
>>> from functools import reduce
>>> reduce(lambda a,b:a+b , [1,2,3,4,5])
15
```

در واقع این مثال معادل محاسبه زیر است.

$$((((1+2)+3)+4)+5)=15$$

معرف‌ها (*comprehensions*)

در بعضی مواقع می‌توان به جای توابع گفته‌شده در بالا از معرف‌ها استفاده کرد. (در مواقعی که توابع ورودی داده شده پیچیده نیستند) نحوه استفاده از معرف به شکل زیر است:

```
[expression for item in old_sequence if condition]
{expression for item in old_sequence if condition}
{key_expression:value_expression for item in old_list if condition}
```

نمونه‌ی استفاده از معرف:

```
>>> [i*2 + 10 for i in [1, 2, 3]]
[12, 14, 16]
>>> [x for x in [3, 4, 5, 6, 7] if x > 5]
[6, 7]
>>> {x for x in 'abcddeef' if x not in 'abc'}
{'e', 'f', 'd'}
>>> {x: x**2 for x in range(5)}
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
```