Lambda, Map, Filter ve Dizi Tamamlama

Prof.Dr. Bahadır AKTUĞ
JFM212 Python ile Mühendislik Uygulamaları

*Kaynakça bölümünde verilen kaynaklardan derlenmiştir.

Lambda

- Lambda fonksiyonu veya Lambda operatörü kısaca isimsiz fonksiyonlar tanımlamak için kullanılırlar.
- Lambda fonksiyonu Lisp, Scheme gibi diğer bazı programlama dillerinden Python'a geçmiştir.
- Lamdba fonksiyonları kısa ve sınırlı işlevi olan bu nedenle özel bir isim verilmesi ihtiyacı bulunmayan fonksiyonların tanımlanmasında kullanılmaktadır.
- Lambda fonksiyonu, kendi başına kullanılabildiği gibi, özellikle parametre olarak bir fonksiyon alan diğer fonksiyonlar (map, filter vb.) için kullanılmaktadır.
- Lambda fonksiyonları temel olarak isimsiz (anonymous) olmakla birlikte istenirse lambda fonksiyonlarına da isim verilebilir.

Lambda

- Lambda fonksiyonlarının tanımlanması:

```
lambda a, b : a * b
```

- Burada a ve b fonksiyon parametreleri, iki noktadan sonraki kısım ise bu parametreler ile yapılacak ve alacağı değer fonksiyon sonucu olarak döndürülecek işlemdir.
- Yukarıdaki fonksiyona isim verilmek istenirse:

```
>>> carpim = lambda a, b : a * b
```

şeklinde yazılabilir.

- Bundan sonra "carpim" isimli fonksiyon doğrudan çağrılarak kullanılabilir:

```
>>> carpim(5,4)
>>> 20
```

Lambda

- Yukarıda verilen çarpım islemi normal fonksiyonlar yardımıyla yapılmak istenseydi:

```
def carpim(a, b) :
    return a * b
carpim(5,4)
```

şeklinde yazılabilirdi.

- Her iki yaklaşım da aynı sonucu vermekle birlikte, lambda fonksiyonları daha sonra göreceğimiz ve girdi olarak fonksiyon alan diğer fonksiyonlarla birlikte kullanıldığında önemli kolaylık sağlamaktadır.

Map

- "map" fonksiyonu, bir fonksiyonu ve bir diziyi girdi olarak alan ve dizinin her bir elemanına bu fonksiyonu uygulamaktadır
- "map" fonksiyonunun genel biçimi:

```
map(fonksiyon, dizi)
```

şeklindedir.

- "map" fonksiyonu versiyon 3 öncesinde bir liste döndürürken, 3. versiyondan itibaren bir "yineleyici (iterator)" döndürmektedir.
- Yineleyiciler "list()" komutuyla kolaylıkla liste haline getirilebilir

Map

- "map" fonksiyonu bir örnekle açıklamak gerekirse:

```
def kare(a):
    return a*a

x = range(5)

c = list(map(kare,x))
print(c)
[0, 1, 4, 9, 16]
```

"map" tarafından yineleyici döndürülüyor. Bunu listeye çevirmek için "list()" fonksiyonu kullanılıyor.

Kare fonksiyonunu, x'in herbir elemanına uyguluyor.

Map

- Aynı işlemi "lambda" ile yapalım:

```
x = range(5)
c = list(map(lambda a: a*a,x))
print(c)
[0, 1, 4, 9, 16]
```

Fonksiyon "inline" olarak tanımlanmış oluyor.

- "map" fonksiyonu aynı fonksiyonu birden fazla listeye de uygulayabilir:

```
x = [1, 5, 9, 15]; y = [2, 3, 7, 10]; z = [6, 4, 5, 20]
c = list(map(lambda a,b,c: a*b+c, x, y, z))
print(c)
[8, 19, 68, 170]
```

Filter

- "filter" fonksiyonu da birçok yönüyle "map" fonksiyonuna benzemektedir ve genel kullanımı

```
filter (fonksiyon, dizi)
```

şeklindedir.

- Bununla birlikte "filter" içinde kullanılan "fonksiyon" True/False şeklinde boolean bir ifade döndürmek zorundadır.
- Bu şekilde "filter" fonksiyonu, verilen dizinin sadece "True" döndüren elemanlarını ayıklayabilmektedir.
- Python'da True ve False değerlerinin neler olduğu daha önceki derslerde verilmişti. Buna uygun olarak farklı fonksiyonlar ile örnekler üretilebilir.

Filter

- 1'den 20'ye kadar olan sayılardan tek sayıları ayıklayan bir "filter" komutu aşağıdaki şekilde yazılabilir.

```
>>> sayilar =  range(20)
>>> list(filter(lambda x: x % 2, sayilar))
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
```

- Burada, "modulo" operatörünün nasıl "True/False" döndürdüğüne dikkat ediniz. Bir başka örnek olarak eğer bir dizideki 10'dan küçük sayıları aşağıdaki şekilde filtreleyebiliriz:

```
>>> sayilar = [14, 6, 15, 17 , 9, 6 ]
>>> list(filter(lambda x: x > 10, sayilar))
[14, 15, 17]
```

Dizi Tamamlama

- Dizi tamamlama (List Comprehension), Python'ın sunduğu en esnek araçlardan biridir.
- Dizi tamamlama liste oluşturmak için en hızlı yöntemdir. Lambda fonksiyonlarından daha hızlı çalışır.
- Her ne kadar ismi "Liste tamamlama" olsa da, aslında sadece listeler için değil aynı zamanda kümeler ve sözlükler için dahi kullanılabilir.
- Genel kullanımı:

[f(x) for x in dizi] veya [f(x) for x in dizi if g(x)]

Dizi Tamamlama

- Örneğin inç cinsinden uzunluklar bulunan bir listeyi cm cinsine çevirelim:

```
>>> inch =  [1, 2, 5, 3, 6]
>>> [2.54*x for x in inch]
[2.54, 5.08, 12.7, 7.62, 15.24]
```

- Aynı dizide sadece 3 inçten büyük olanları almak istersek;

```
>>> [2.54*x for x in inch if x > 3]
[12.7, 15.24]
```

Dizi Tamamlama

- Birden fazla değişken ile çalışmak gerekirse örneğin;

```
>>> x = [5, 15, 25]
>>> y = [50, 55, 60]
```

şeklinde koordinatlar tanımlayarak Matlab'daki meshgrid'e benzer bir şekilde koordinat çiftleri üretilebilir:

```
>>> [(a,b) for a in x for b in y]
[(5, 50), (5, 55), (5, 60), (15, 50), (15, 55), (15, 60), (25, 50), (25, 55), (25, 60)]
```

Dizi Tamamlama

- Liste tamamlamayı daha açık ifade etmek gerekirse;

```
[ifade for x in dizi1 for y in dizi2]
```

şeklinde verilen bir liste tamamlama aslında aşağıdaki döngülerle oluşturulan koda karşılık gelmektedir:

```
sonuc=[]
for x in dizi1:
    for y in dizi2:
        sonuc.append(ifade)
```

Dizi Tamamlama

- Liste tamamlamayı daha açık ifade etmek gerekirse;

```
[[ifade for x in dizi1] for y in dizi2]
```

şeklinde verilen bir liste tamamlama aslında aşağıdaki döngülerle oluşturulan koda karşılık gelmektedir:

```
sonuc=[]
for y in dizi2:
  icsonuc = []
  for x in dizi1:
      icsonuc.append(ifade)
  sonuc.append(icsonuc)
```

Küme Tamamlama

- Küme tamamlama da liste tamamlamaya benzer şekilde yapılabilir;

```
{j for i in dizi1}
```

veya

```
{x for x in dizi1 for y in dizi2}
```

- Küme özelliği nedeniyle tekrarlı elemanlar yine küme tamamlamada bulunmamaktadır:

```
>>> kume = {y for x in range(5) for y in range(1,x)}
>>> print(kume)
{1, 2, 3}
```

Sözlük Tamamlama

- Sözlük tamamlama, hem anahtar hem de değer bulunacak şekilde kullanılabilir:

```
{x:2*x for x in dizi1}
```

veya

```
{x:y for x in dizi1 for y in dizi2}
```

- Sözlüklerde anahtar'ın tek olduğuna dikkat edilmelidir.

```
>>> {x:2*x for x in range(4)}
>>> {0: 0, 1: 2, 2: 4, 3: 6}
```

Sözlük Tamamlama

- Bir başka örnek ile sözlük tamamlamayı "zip" komutu gibi kullanabiliriz.

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> sozluk = {x:y for x in a for y in b}
>>> print(sozluk)
{1: 6, 2: 6, 3: 6}
```

- Şimdi yukarıdaki sözlüğü sözlük tanımlama ile ters çevirelim

```
>>> {y:x for x,y in sozluk.items()}
{6: 3}
```

Kaynakça

1 Wentworth, P., Elkner, J., Downey, A.B., Meyers, C. (2014). How to Think Like a Computer Scientist: Learning with Python (3nd edition).
2 Pilgrim, M. (2014). Dive into Python 3 by. Free online version: DiveIntoPython3.org ISBN: 978-1430224150.
3 Summerfield, M. (2014) Programming in Python 3 2nd ed (PIP3) : - Addison Wesley ISBN: 0-321-68056-1.
4 Jones E, Oliphant E, Peterson P, et al. SciPy: Open Source Scientific Tools for Python, 2001-, http://www.scipy.org/.
5 Millman, K.J., Aivazis, M. (2011). Python for Scientists and Engineers, Computing in Science & Engineering, 13, 9-12.
6 John D. Hunter (2007). Matplotlib: A 2D Graphics Environment, Computing in Science & Engineering, 9, 90-95.
7 Travis E. Oliphant (2007). Python for Scientific Computing, Computing in Science & Engineering, 9, 10-20.
8 Goodrich, M.T., Tamassia, R., Goldwasser, M.H. (2013). Data Structures and Algorithms in Python, Wiley.
9 http://www.diveintopython.net/
10 https://docs.python.org/3/tutorial/
11 http://www.python-course.eu
12 https://developers.google.com/edu/python/
13 http://learnpythonthehardway.org/book/