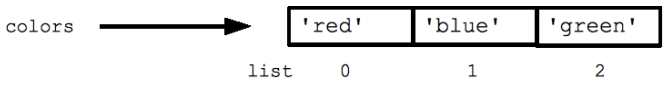
Python Listeleri

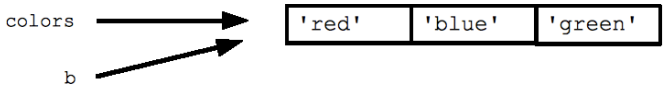
Python'da "list" adlı harika bir yerleşik liste türü vardır. Liste değişmez değerleri köşeli parantez [ ] içinde yazılır. Listeler dizelere benzer şekilde çalışır. Verilere erişmek için len() işlevini ve köşeli parantezleri [ ] kullanın. İlk öğe dizin 0'dır. (Resmi python.org listesi belgelerine (http://docs.python.org/tut/node7.html) bakın.)

```
colors = ['red', 'blue', 'green']
print(colors[0])    ## red
print(colors[2])    ## green
print(len(colors))  ## 3
```



Listelerin üzerinde = işaretiyle yapılan atamalar kopya oluşturmaz. Bunun yerine atama, iki değişkeni bellekteki tek bir listeye yönlendirir.

```
b = colors   ## Does not copy the list
```



"Boş liste" yalnızca boş bir köşeli parantez [ ]'dir. "+" iki liste eklemek için kullanılabilir. Bu nedenle [1, 2] + [3, 4], [1, 2, 3, 4] sonucunu verir (bu, dizelerle + gibidir).

İÇİN VE İÇİNDE

Python'un *for* ve *in* yapıları çok yararlıdır ve bunların ilk kullanımı listelerle olacaktır. *için* yapısı -- **`for var in list`**, bir listedeki (veya başka bir koleksiyondaki) her öğeye bakmanın kolay bir yoludur. Yineleme sırasında listeye eklemeyin veya listeden kaldırmayın.

```
squares = [1, 4, 9, 16]
sum = 0
for num in squares:
  sum += num
print(sum)  ## 30
```

Listede ne tür bir şey olduğunu biliyorsanız, döngüde "num", "name" veya "url" gibi bu bilgileri yakalayan bir değişken adı kullanın. Python kodunda, türleri hatırlatmak için başka bir söz dizimi bulunmadığından, değişken adlarınız olup biteni doğrudan yönetmenin önemli bir yoludur. (Bu biraz yanıltıcı olabilir. Piton ile daha fazla tanındıkça işlev tanımlarınıza yazma bilgileri eklemenize olanak tanıyan tür ipuçları (https://docs.python.org/3/library/typing.html) için referanslar görürsünüz. Python, programlarınızı çalıştırırken bu tür ipuçlarını kullanmaz. IDE'ler (entegre geliştirme ortamları) gibi diğer programlar ve işlevlerinizin uyumlu bağımsız değişkenlerle çağrıldığını doğrulamak için linter/tür denetleyiciler gibi statik analiz araçları tarafından kullanılır.

in yapısı kendiliğinden bir öğenin listede (veya başka bir koleksiyonda) görünüp görünmediğini kolayca test edebilir. **`value in collection`**, değerin koleksiyonda olup olmadığını test ederek Doğru/Yanlış değerini döndürür.

```
list = ['larry', 'curly', 'moe']
if 'curly' in list:
  print('yay') ## yay
```

For/in derlemeleri, Python kodunda çok yaygın şekilde kullanıldığından liste dışındaki veri türleri üzerinde çalıştığından yalnızca söz dizimini ezberlemeniz gerekir. Başka dillerde alışkanlıklarınız olabilir. Bu durumda, bir koleksiyon üzerinde manuel olarak iterasyon yapmaya başlayabilirsiniz. Python'da bu kullanımları yalnızca içine dahil etmeniz gerekir.

Bir dize üzerinde çalışmak için for/in ifadesini de kullanabilirsiniz. Dize, karakterlerinin listesi gibi çalışır. Bu nedenle `for ch in s: print(ch)`, tüm karakterleri bir dizeye yazdırır.

Aralık

aralık(n) işlevi 0, 1, ... n-1 sayısını ve aralık(a, b) a, a+1, ... b-1 -- sonucunu verir, ancak son sayıyı içermez. for-loop ve range() işlevinin kombinasyonu, döngü için geleneksel bir sayısal değer oluşturmanıza olanak tanır:

```
## print the numbers from 0 through 99
for i in range(100):
  print(i)
```

xrange() varyantı, performansa duyarlı destek kayıtları için tüm listeyi oluşturma maliyetini ortadan kaldıran bir varyant içerir (Python 3'te range() iyi performans gösterir ve xrange()'i unutabilirsiniz).

Döngü modundayken

Python'da ayrıca standart atlama döngüsü vardır. *break* ve *continue* ifadeleri, C++ ve Java'da olduğu gibi çalışır. En içteki döngünün seyrini değiştirir. Yukarıdaki/giriş döngüleri, bir listedeki her öğe için tekrarlama gibi yaygın bir sorunu çözer. Diğer yandan, bu döngü, dizin numaraları üzerinde tam kontrol sağlar. Burada, bir listedeki her 3. öğeye erişen bir zaman döngüsü görebilirsiniz:

```
## Access every 3rd element in a list
i = 0
while i < len(a):
  print(a[i])
  i = i + 3
```

Yöntemleri Listeleme

Yaygın olarak kullanılan diğer bazı liste yöntemleri şunlardır.

- list.append(elem) -- Listenin sonuna tek bir öğe ekler. Genel hata: Yeni listeyi döndürmez, yalnızca orijinali değiştirir.
- list.insert(index, elem) -- Öğeyi belirtilen dizine ekler ve öğeleri sağa kaydırır.
- list.extend(list2), list2'deki öğeleri listenin sonuna ekler. Bir listede + veya += kullanımı extension() kullanmaya benzer.
- list.index(elem) -- Listenin başından itibaren belirli bir öğeyi arar ve dizinini döndürür. Öğe görünmüyorsa ValueError gönderir (ValueError olmadan kontrol etmek için "in" ifadesini kullanın).
- list.remove(elem) -- Belirtilen öğenin ilk örneğini arar ve varsa bunu kaldırır (yoksa ValueError atar)
- list.sort() -- listeyi yerine yerleştirir (döndürmez). (Daha sonra gösterilen sıralı() işlevi tercih edilir.)
- list.reverse() -- Listeyi tersine çevirir (döndürmez)
- list.pop(index) -- Belirtilen dizindeki öğeyi kaldırır ve döndürür. Dizine eklenmezse en sağdaki öğeyi döndürür (neredeyse add() ifadesinin tam tersidir).

Bunların, bir nesne nesnesinde *methods* olduğuna dikkat edin. Len() ise listeyi (veya dizeyi ya da herhangi bir şeyi) bağımsız değişken olarak alan bir işlevdir.

```
list = ['larry', 'curly', 'moe']
list.append('shemp')         ## append elem at end
list.insert(0, 'xxx')        ## insert elem at index 0
list.extend(['yyy', 'zzz'])  ## add list of elems at end
print(list)  ## ['xxx', 'larry', 'curly', 'moe', 'shemp', 'yyy', 'zzz']
print(list.index('curly'))    ## 2

list.remove('curly')         ## search and remove that element
```

```
list.pop(1)                 ## removes and returns 'larry'
print(list)  ## ['xxx', 'moe', 'shemp', 'yyy', 'zzz']
```

Genel hata: Yukarıdaki yöntemlerin değiştirilen listeyi *döndürmediğini*, yalnızca orijinal listeyi değiştirdiğini unutmayın.

```
list = [1, 2, 3]
print(list.append(4))   ## NO, does not work, append() returns None
## Correct pattern:
list.append(4)
print(list)  ## [1, 2, 3, 4]
```

Liste Derlemesi

Yaygın kullanılan yöntemlerden biri, listeyi boş liste [] olarak başlatmak, ardından listeye öğe eklemek için add() veya extension() işlevini kullanmaktır:

```
list = []          ## Start as the empty list
list.append('a')   ## Use append() to add elements
list.append('b')
```

Dilimleri Listeleme

Dilimler, dizelerde olduğu gibi listeler üzerinde de çalışır ve listenin alt bölümlerini değiştirmek için de kullanılabilir.

```
list = ['a', 'b', 'c', 'd']
print(list[1:-1])   ## ['b', 'c']
list[0:2] = 'z'    ## replace ['a', 'b'] with ['z']
print(list)         ## ['z', 'c', 'd']
```

Egzersiz: list1.py

Bu bölümdeki materyalleri uygulamak için, **list1.py**'de sıralama kullanmayan sorunları deneyin (Temel Alıştırmalar (https://developers.google.com/edu/python/exercises/basic?hl=tr)'da).