5. Veri Yapıları

Bu bölümde daha önce öğrendiğiniz bazı şeyler daha detaylı olarak açıklanmakta ve bunlara bazı yeni şeyler eklenmektedir.

5.1. Listeler Hakkında Daha Fazla Bilgi

Liste veri tipinin birkaç yöntemi daha vardır. İşte liste nesnelerinin tüm yöntemleri:

liste. **ekle** ( *x* )

Listenin sonuna bir öğe ekleyin. Benzeri .a[len(a):] = [x]

liste.genişlet ( **yinelenebilir** )*​*

Listeyi yinelemeli öğenin tüm öğelerini ekleyerek genişletin. 'e benzer .a[len(a):] = iterable

liste. **ekle** ( *i* ,*x* )

Belirli bir konuma bir öğe ekler. İlk argüman, eklenecek öğenin dizinidir, bu nedenle listenin önüne eklenir ve ile eşdeğerdir .a.insert(0, x)a.insert(len(a), x)a.append(x)

liste. **kaldır** ( *x* )

*Değeri x'e* eşit olan ilk öğeyi listeden siler . [ValueError](https://docs.python.org/3/library/exceptions.html#ValueError)Eğer böyle bir öğe yoksa a'yı yükseltir .

liste. **pop** ( [ *i* ] )

Listedeki belirtilen konumdaki öğeyi kaldırın ve döndürün. Hiçbir dizin belirtilmemişse, a.pop()listedeki son öğeyi kaldırır ve döndürür. [IndexError](https://docs.python.org/3/library/exceptions.html#IndexError)Liste boşsa veya dizin liste aralığının dışındaysa bir yükseltir.

liste. **temizle** ( )

Listeden tüm öğeleri kaldır. Benzeri .del a[:]

liste.indeks ( *x* [ ,*başlangıç* ​​[ ,*bitiş* ] ] )**​**

*Değeri x'e* eşit olan ilk öğenin listesinde sıfır tabanlı indeks döndürür . [ValueError](https://docs.python.org/3/library/exceptions.html#ValueError)Böyle bir öğe yoksa a yükseltir.

*İsteğe bağlı başlangıç* ​​ve *bitiş* argümanları dilim gösteriminde olduğu gibi yorumlanır ve aramayı listenin belirli bir alt dizisiyle sınırlamak için kullanılır. Döndürülen dizin, *başlangıç* ​​argümanı yerine tam dizinin başlangıcına göre hesaplanır.

liste.say ( *x* )**​**

*x'in* listede kaç kez göründüğünü döndür .

liste.sıralama ( *\** ,*anahtar = Hiçbiri* ,*ters = Yanlış* )**​**

Listedeki öğeleri yerinde sıralayın (argümanlar sıralama özelleştirmesi için kullanılabilir, [sorted()](https://docs.python.org/3/library/functions.html#sorted)açıklamaları için bkz.).

liste. **ters** ( )

Listenin elemanlarını yerinde ters çevirin.

liste.kopyala ( )**​**

Listenin yüzeysel bir kopyasını döndürür. 'e benzer a[:].

Liste yöntemlerinin çoğunu kullanan bir örnek:

>>>

**>>>** fruits = ['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']

**>>>** fruits.count('apple')

2

**>>>** fruits.count('tangerine')

0

**>>>** fruits.index('banana')

3

**>>>** fruits.index('banana', 4) *# Find next banana starting at position 4*

6

**>>>** fruits.reverse()

**>>>** fruits

['banana', 'apple', 'kiwi', 'banana', 'pear', 'apple', 'orange']

**>>>** fruits.append('grape')

**>>>** fruits

['banana', 'apple', 'kiwi', 'banana', 'pear', 'apple', 'orange', 'grape']

**>>>** fruits.sort()

**>>>** fruits

['apple', 'apple', 'banana', 'banana', 'grape', 'kiwi', 'orange', 'pear']

**>>>** fruits.pop()

'pear'

insert, removeveya gibi yalnızca listeyi değiştiren yöntemlerin hiçbir dönüş değerinin yazdırılmadığını fark etmiş olabilirsiniz sort; varsayılan değerini döndürürler None. [[ 1 ]](https://docs.python.org/3/tutorial/datastructures.html#id2) Bu, Python'daki tüm değiştirilebilir veri yapıları için bir tasarım ilkesidir.

Fark edebileceğiniz bir diğer şey de tüm verilerin sıralanamayacağı veya karşılaştırılamayacağıdır. Örneğin, sıralanmaz çünkü tam sayılar dizelerle karşılaştırılamaz ve diğer türlerle karşılaştırılamaz. Ayrıca, tanımlanmış bir sıralama ilişkisi olmayan bazı türler vardır. Örneğin, geçerli bir karşılaştırma değildir.[None, 'hello', 10]None3+4j < 5+7j

5.1.1. Listeleri Yığın Olarak Kullanma

Liste yöntemleri, bir listeyi yığın olarak kullanmayı çok kolaylaştırır; burada eklenen son öğe, alınan ilk öğedir ("son giren, ilk çıkan"). Yığının en üstüne bir öğe eklemek için kullanın ~list.append(). Yığının en üstünden bir öğe almak için ~list.pop()açık bir dizin olmadan kullanın. Örneğin:

>>>

**>>>** stack = [3, 4, 5]

**>>>** stack.append(6)

**>>>** stack.append(7)

**>>>** stack

[3, 4, 5, 6, 7]

**>>>** stack.pop()

7

**>>>** stack

[3, 4, 5, 6]

**>>>** stack.pop()

6

**>>>** stack.pop()

5

**>>>** stack

[3, 4]

5.1.2. Listeleri Kuyruk Olarak Kullanma

Bir listeyi, eklenen ilk öğenin alınan ilk öğe olduğu bir kuyruk olarak kullanmak da mümkündür ("ilk giren, ilk çıkar"); ancak, listeler bu amaç için verimli değildir. Listenin sonundan eklemeler ve çıkarmalar hızlı olsa da, listenin başından eklemeler veya çıkarmalar yapmak yavaştır (çünkü diğer tüm öğeler birer birer kaydırılmalıdır).

Bir kuyruğu uygulamak için, [collections.deque](https://docs.python.org/3/library/collections.html#collections.deque)her iki uçtan hızlı eklemeler ve çıkarmalar için tasarlanmış which'i kullanın. Örneğin:

>>>

**>>> from** **collections** **import** deque

**>>>** queue = deque(["Eric", "John", "Michael"])

**>>>** queue.append("Terry") *# Terry arrives*

**>>>** queue.append("Graham") *# Graham arrives*

**>>>** queue.popleft() *# The first to arrive now leaves*

'Eric'

**>>>** queue.popleft() *# The second to arrive now leaves*

'John'

**>>>** queue *# Remaining queue in order of arrival*

deque(['Michael', 'Terry', 'Graham'])

5.1.3. Liste Kapsamları

Liste kavrayışları, listeler oluşturmanın özlü bir yolunu sağlar. Yaygın uygulamalar, her bir öğenin başka bir dizinin veya yinelemenin her bir üyesine uygulanan bazı işlemlerin sonucu olduğu yeni listeler oluşturmak veya belirli bir koşulu karşılayan bu öğelerin bir alt dizisini oluşturmaktır.

Örneğin, şu şekilde bir kareler listesi oluşturmak istediğimizi varsayalım:

>>>

**>>>** squares = []

**>>> for** x **in** range(10):

**...**  squares.append(x\*\*2)

**...**

**>>>** squares

[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

Bunun, döngü tamamlandıktan sonra hala var olan bir isimli değişkeni oluşturduğunu (veya üzerine yazdığını) unutmayın x. Kareler listesini herhangi bir yan etki olmadan şu şekilde hesaplayabiliriz:

squares = list(map(**lambda** x: x\*\*2, range(10)))

veya eşdeğer olarak:

squares = [x\*\*2 **for** x **in** range(10)]

hangisi daha öz ve okunabilir.

Bir liste kavrayışı, bir ifadeyi içeren parantezlerden ve ardından bir cümleden for, ardından sıfır veya daha fazla foror cümlesinden oluşur. Sonuç, ifadeyi onu takip eden ve cümleleri if bağlamında değerlendirmekten kaynaklanan yeni bir liste olacaktır . Örneğin, bu listcomp, eşit değillerse iki listenin öğelerini birleştirir:forif

>>>

**>>>** [(x, y) **for** x **in** [1,2,3] **for** y **in** [3,1,4] **if** x != y]

[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]

ve şuna eşdeğerdir:

>>>

**>>>** combs = []

**>>> for** x **in** [1,2,3]:

**...**  **for** y **in** [3,1,4]:

**...**  **if** x != y:

**...**  combs.append((x, y))

**...**

**>>>** combs

[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]

[for](https://docs.python.org/3/reference/compound_stmts.html#for)Bu iki kod parçacığında ve ifadelerinin sırasının [if](https://docs.python.org/3/reference/compound_stmts.html#if)aynı olduğuna dikkat edin.

Eğer ifade bir tuple ise (örneğin önceki örnekteki gibi), parantez içine alınmalıdır.(x, y)

>>>

**>>>** vec = [-4, -2, 0, 2, 4]

**>>>** *# create a new list with the values doubled*

**>>>** [x\*2 **for** x **in** vec]

[-8, -4, 0, 4, 8]

**>>>** *# filter the list to exclude negative numbers*

**>>>** [x **for** x **in** vec **if** x >= 0]

[0, 2, 4]

**>>>** *# apply a function to all the elements*

**>>>** [abs(x) **for** x **in** vec]

[4, 2, 0, 2, 4]

**>>>** *# call a method on each element*

**>>>** freshfruit = [' banana', ' loganberry ', 'passion fruit ']

**>>>** [weapon.strip() **for** weapon **in** freshfruit]

['banana', 'loganberry', 'passion fruit']

**>>>** *# create a list of 2-tuples like (number, square)*

**>>>** [(x, x\*\*2) **for** x **in** range(6)]

[(0, 0), (1, 1), (2, 4), (3, 9), (4, 16), (5, 25)]

**>>>** *# the tuple must be parenthesized, otherwise an error is raised*

**>>>** [x, x\*\*2 **for** x **in** range(6)]

File "<stdin>", line 1

[x, x\*\*2 **for** x **in** range(6)]

^^^^^^^

SyntaxError: did you forget parentheses around the comprehension target?

**>>>** *# flatten a list using a listcomp with two 'for'*

**>>>** vec = [[1,2,3], [4,5,6], [7,8,9]]

**>>>** [num **for** elem **in** vec **for** num **in** elem]

[1, 2, 3, 4, 5, 6, 7, 8, 9]

Liste kavrayışları karmaşık ifadeler ve iç içe geçmiş işlevler içerebilir:

>>>

**>>> from** **math** **import** pi

**>>>** [str(round(pi, i)) **for** i **in** range(1, 6)]

['3.1', '3.14', '3.142', '3.1416', '3.14159']

5.1.4. İç İçe Liste Kavramları

Bir liste kavrayışındaki başlangıç ​​ifadesi, başka bir liste kavrayışı da dahil olmak üzere herhangi bir keyfi ifade olabilir.

3x4'lük bir matrisin, uzunluğu 4 olan 3 listeden oluşan bir liste olarak uygulandığı aşağıdaki örneği ele alalım:

>>>

**>>>** matrix = [

**...**  [1, 2, 3, 4],

**...**  [5, 6, 7, 8],

**...**  [9, 10, 11, 12],

**...** ]

Aşağıdaki liste kavrayışı satır ve sütunları yer değiştirecektir:

>>>

**>>>** [[row[i] **for** row **in** matrix] **for** i **in** range(4)]

[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]

Önceki bölümde gördüğümüz gibi, iç liste kavrayışı [for](https://docs.python.org/3/reference/compound_stmts.html#for)onu takip eden bağlam içinde değerlendirilir, dolayısıyla bu örnek şuna eşdeğerdir:

>>>

**>>>** transposed = []

**>>> for** i **in** range(4):

**...**  transposed.append([row[i] **for** row **in** matrix])

**...**

**>>>** transposed

[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]

ki bu da şu anlama gelir:

>>>

**>>>** transposed = []

**>>> for** i **in** range(4):

**...**  *# the following 3 lines implement the nested listcomp*

**...**  transposed\_row = []

**...**  **for** row **in** matrix:

**...**  transposed\_row.append(row[i])

**...**  transposed.append(transposed\_row)

**...**

**>>>** transposed

[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]

Gerçek dünyada, karmaşık akış ifadeleri yerine yerleşik işlevleri tercih etmelisiniz. [zip()](https://docs.python.org/3/library/functions.html#zip)İşlev bu kullanım durumu için harika bir iş çıkaracaktır:

>>>

**>>>** list(zip(\*matrix))

[(1, 5, 9), (2, 6, 10), (3, 7, 11), (4, 8, 12)]

Bu satırdaki yıldız işaretinin ayrıntıları için [Argüman Listelerinin Açılması bölümüne](https://docs.python.org/3/tutorial/controlflow.html#tut-unpacking-arguments) bakın .

. Beyan del¶

Bir öğeyi değeri yerine indeksi verildiğinde listeden kaldırmanın bir yolu vardır: [del](https://docs.python.org/3/reference/simple_stmts.html#del)ifade. Bu, ~list.pop()bir değer döndüren yöntemden farklıdır. delİfade ayrıca bir listeden dilimleri kaldırmak veya tüm listeyi temizlemek için de kullanılabilir (bunu daha önce dilime boş bir liste atayarak yapmıştık). Örneğin:

>>>

**>>>** a = [-1, 1, 66.25, 333, 333, 1234.5]

**>>> del** a[0]

**>>>** a

[1, 66.25, 333, 333, 1234.5]

**>>> del** a[2:4]

**>>>** a

[1, 66.25, 1234.5]

**>>> del** a[:]

**>>>** a

[]

[del](https://docs.python.org/3/reference/simple_stmts.html#del)tüm değişkenleri silmek için de kullanılabilir:

>>>

**>>> del** a

Bundan sonra isme atıfta bulunmak abir hatadır (en azından ona başka bir değer atanana kadar). Daha sonra başka kullanımlar bulacağız [del](https://docs.python.org/3/reference/simple_stmts.html#del).

5.3. Demetler ve Diziler

Listelerin ve dizelerin indeksleme ve dilimleme işlemleri gibi birçok ortak özelliğe sahip olduğunu gördük. Bunlar *dizi* veri türlerinin iki örneğidir (bkz. [Dizi Türleri — liste, tuple, aralık](https://docs.python.org/3/library/stdtypes.html#typesseq) ). Python gelişen bir dil olduğundan, diğer dizi veri türleri eklenebilir. Ayrıca başka bir standart dizi veri türü daha vardır: *tuple* .

Bir tuple, virgülle ayrılmış bir dizi değerden oluşur, örneğin:

>>>

**>>>** t = 12345, 54321, 'hello!'

**>>>** t[0]

12345

**>>>** t

(12345, 54321, 'hello!')

**>>>** *# Tuples may be nested:*

**>>>** u = t, (1, 2, 3, 4, 5)

**>>>** u

((12345, 54321, 'hello!'), (1, 2, 3, 4, 5))

**>>>** *# Tuples are immutable:*

**>>>** t[0] = 88888

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: 'tuple' object does not support item assignment

**>>>** *# but they can contain mutable objects:*

**>>>** v = ([1, 2, 3], [3, 2, 1])

**>>>** v

([1, 2, 3], [3, 2, 1])

Gördüğünüz gibi, çıktıda tuple'lar her zaman parantez içindedir, böylece iç içe geçmiş tuple'lar doğru şekilde yorumlanır; parantezli veya parantezsiz girdi olabilirler, ancak genellikle parantezler her durumda gereklidir (eğer tuple daha büyük bir ifadenin parçasıysa). Bir tuple'ın bireysel öğelerine atama yapmak mümkün değildir, ancak listeler gibi değiştirilebilir nesneler içeren tuple'lar oluşturmak mümkündür.

Tuple'lar listelerle benzer görünse de, genellikle farklı durumlarda ve farklı amaçlar için kullanılırlar. Tuple'lar değişmezdir [ve](https://docs.python.org/3/glossary.html#term-immutable) genellikle paket açma (bu bölümün ilerleyen kısımlarında bakın) veya dizinleme (veya durumunda öznitelik yoluyla [namedtuples](https://docs.python.org/3/library/collections.html#collections.namedtuple)) yoluyla erişilen heterojen bir öğe dizisi içerir. Listeler değişebilirdir [ve](https://docs.python.org/3/glossary.html#term-mutable) öğeleri genellikle homojendir ve liste üzerinde yineleme yaparak erişilir.

Özel bir sorun, 0 veya 1 öğe içeren tuple'ların oluşturulmasıdır: sözdiziminin bunlara uyum sağlamak için bazı ek tuhaflıkları vardır. Boş tuple'lar boş bir parantez çiftiyle oluşturulur; bir öğeli bir tuple, bir değerin ardından virgülle oluşturularak oluşturulur (tek bir değeri parantez içine almak yeterli değildir). Çirkin ama etkili. Örneğin:

>>>

**>>>** empty = ()

**>>>** singleton = 'hello', *# <-- note trailing comma*

**>>>** len(empty)

0

**>>>** len(singleton)

1

**>>>** singleton

('hello',)

Bu ifade, *tuple paketlemenin* bir örneğidir : , ve değerleri bir tuple içinde birlikte paketlenir. Tersi işlem de mümkündür:t = 12345, 54321, 'hello!'1234554321'hello!'

>>>

**>>>** x, y, z = t

*Buna, uygun bir şekilde, dizi açma* denir ve sağ taraftaki herhangi bir dizi için çalışır. Dizi açma, eşittir işaretinin sol tarafında dizideki eleman sayısı kadar değişken olmasını gerektirir. Çoklu atamanın aslında sadece tuple paketleme ve dizi açmanın bir kombinasyonu olduğunu unutmayın.

5.4. Kümeler

*Python ayrıca kümeler* için bir veri türü içerir . Bir küme, yinelenen öğeleri olmayan sıralanmamış bir koleksiyondur. Temel kullanımlar arasında üyelik testi ve yinelenen girişleri ortadan kaldırmak yer alır. Küme nesneleri ayrıca birleşim, kesişim, fark ve simetrik fark gibi matematiksel işlemleri de destekler.

Kıvrımlı parantezler veya [set()](https://docs.python.org/3/library/stdtypes.html#set)fonksiyon kümeler oluşturmak için kullanılabilir. Not: Boş bir küme oluşturmak için kullanmalısınız set(), kullanamazsınız {}; ikincisi boş bir sözlük oluşturur, bu veri yapısını bir sonraki bölümde ele alacağız.

İşte kısa bir gösteri:

>>>

**>>>** basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}

**>>>** print(basket) *# show that duplicates have been removed*

{'orange', 'banana', 'pear', 'apple'}

**>>>** 'orange' **in** basket *# fast membership testing*

True

**>>>** 'crabgrass' **in** basket

False

**>>>** *# Demonstrate set operations on unique letters from two words*

**>>>**

**>>>** a = set('abracadabra')

**>>>** b = set('alacazam')

**>>>** a *# unique letters in a*

{'a', 'r', 'b', 'c', 'd'}

**>>>** a - b *# letters in a but not in b*

{'r', 'd', 'b'}

**>>>** a | b *# letters in a or b or both*

{'a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'}

**>>>** a & b *# letters in both a and b*

{'a', 'c'}

**>>>** a ^ b *# letters in a or b but not both*

{'r', 'd', 'b', 'm', 'z', 'l'}

[Liste kavrayışlarına](https://docs.python.org/3/tutorial/datastructures.html#tut-listcomps) benzer şekilde , küme kavrayışları da desteklenir:

>>>

**>>>** a = {x **for** x **in** 'abracadabra' **if** x **not** **in** 'abc'}

**>>>** a

{'r', 'd'}

5.5. Sözlükler

Python'a yerleştirilmiş bir diğer yararlı veri türü sözlüktür *(* bkz. [Eşleme Türleri — dict](https://docs.python.org/3/library/stdtypes.html#typesmapping) ). Sözlükler bazen diğer dillerde "ilişkisel bellekler" veya "ilişkisel diziler" olarak bulunur. Bir sayı aralığıyla indekslenen dizilerin aksine, sözlükler herhangi bir değişmez tür olabilen *keys* tarafından indekslenir ; dizeler ve sayılar her zaman anahtar olabilir. Yalnızca dizeler, sayılar veya tuple'lar içeriyorsa tuple'lar anahtar olarak kullanılabilir; bir tuple doğrudan veya dolaylı olarak herhangi bir değiştirilebilir nesne içeriyorsa, anahtar olarak kullanılamaz. Listeleri anahtar olarak kullanamazsınız çünkü listeler dizin atamaları, dilim atamaları veya ~list.append()ve gibi yöntemler kullanılarak yerinde değiştirilebilir ~list.extend().

*Bir sözlüğü anahtar:değer* çiftleri kümesi olarak düşünmek en iyisidir , anahtarların benzersiz olması (bir sözlük içinde) şartıyla. Bir çift parantez boş bir sözlük oluşturur: {}. Parantezler içine virgülle ayrılmış bir anahtar:değer çiftleri listesi yerleştirmek sözlüğe ilk anahtar:değer çiftlerini ekler; sözlükler çıktıya bu şekilde yazılır.

Bir sözlükteki temel işlemler, bir değeri bir anahtarla depolamak ve anahtara verilen değeri çıkarmaktır. Ayrıca, bir anahtar:değer çiftini ile silmek de mümkündür del. Zaten kullanımda olan bir anahtarı kullanarak depolama yaparsanız, o anahtarla ilişkilendirilen eski değer unutulur. Varolmayan bir anahtarı kullanarak bir değer çıkarmak bir hatadır.

Bir sözlükte işlem yapmak list(d), sözlükte kullanılan tüm anahtarların ekleme sırasına göre bir listesini döndürür (sıralı olmasını istiyorsanız, bunun sorted(d)yerine kullanın). Tek bir anahtarın sözlükte olup olmadığını kontrol etmek için [in](https://docs.python.org/3/reference/expressions.html#in)anahtar sözcüğü kullanın.

İşte sözlük kullanarak yapılmış küçük bir örnek:

>>>

**>>>** tel = {'jack': 4098, 'sape': 4139}

**>>>** tel['guido'] = 4127

**>>>** tel

{'jack': 4098, 'sape': 4139, 'guido': 4127}

**>>>** tel['jack']

4098

**>>> del** tel['sape']

**>>>** tel['irv'] = 4127

**>>>** tel

{'jack': 4098, 'guido': 4127, 'irv': 4127}

**>>>** list(tel)

['jack', 'guido', 'irv']

**>>>** sorted(tel)

['guido', 'irv', 'jack']

**>>>** 'guido' **in** tel

True

**>>>** 'jack' **not** **in** tel

False

Oluşturucu [dict()](https://docs.python.org/3/library/stdtypes.html#dict), sözlükleri doğrudan anahtar-değer çiftleri dizilerinden oluşturur:

>>>

**>>>** dict([('sape', 4139), ('guido', 4127), ('jack', 4098)])

{'sape': 4139, 'guido': 4127, 'jack': 4098}

Ek olarak, sözlük kavrayışları, keyfi anahtar ve değer ifadelerinden sözlükler oluşturmak için kullanılabilir:

>>>

**>>>** {x: x\*\*2 **for** x **in** (2, 4, 6)}

{2: 4, 4: 16, 6: 36}

Anahtarlar basit dizeler olduğunda, bazen anahtar sözcük argümanlarını kullanarak çiftleri belirtmek daha kolaydır:

>>>

**>>>** dict(sape=4139, guido=4127, jack=4098)

{'sape': 4139, 'guido': 4127, 'jack': 4098}

5.6. Döngü Teknikleri

Sözlükler arasında döngü yapılırken, anahtar ve karşılık gelen değer, yöntem kullanılarak aynı anda alınabilir [items()](https://docs.python.org/3/library/stdtypes.html#dict.items).

>>>

**>>>** knights = {'gallahad': 'the pure', 'robin': 'the brave'}

**>>> for** k, v **in** knights.items():

**...**  print(k, v)

**...**

gallahad the pure

robin the brave

Bir dizi üzerinde döngü yapıldığında, pozisyon indeksi ve karşılık gelen değer, fonksiyon kullanılarak aynı anda alınabilir [enumerate()](https://docs.python.org/3/library/functions.html#enumerate).

>>>

**>>> for** i, v **in** enumerate(['tic', 'tac', 'toe']):

**...**  print(i, v)

**...**

0 tic

1 tac

2 toe

Aynı anda iki veya daha fazla dizi üzerinde döngü oluşturmak için, girdiler fonksiyonla eşleştirilebilir [zip()](https://docs.python.org/3/library/functions.html#zip).

>>>

**>>>** questions = ['name', 'quest', 'favorite color']

**>>>** answers = ['lancelot', 'the holy grail', 'blue']

**>>> for** q, a **in** zip(questions, answers):

**...**  print('What is your **{0}**? It is **{1}**.'.format(q, a))

**...**

What is your name? It is lancelot.

What is your quest? It is the holy grail.

What is your favorite color? It is blue.

Bir diziyi ters yönde döngüye almak için, önce diziyi ileri yönde belirtin ve ardından [reversed()](https://docs.python.org/3/library/functions.html#reversed)fonksiyonu çağırın.

>>>

**>>> for** i **in** reversed(range(1, 10, 2)):

**...**  print(i)

**...**

9

7

5

3

1

Sıralı düzende bir dizi üzerinde döngü oluşturmak için, [sorted()](https://docs.python.org/3/library/functions.html#sorted)kaynağı değiştirmeden yeni bir sıralı liste döndüren işlevi kullanın.

>>>

**>>>** basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']

**>>> for** i **in** sorted(basket):

**...**  print(i)

**...**

apple

apple

banana

orange

orange

pear

on a sequence kullanımı yinelenen öğeleri ortadan kaldırır. in over a sequence ile birlikte [set()](https://docs.python.org/3/library/stdtypes.html#set)kullanımı, sıralı düzende dizinin benzersiz öğeleri üzerinde döngü kurmanın deyimsel bir yoludur.[sorted()](https://docs.python.org/3/library/functions.html#sorted)[set()](https://docs.python.org/3/library/stdtypes.html#set)

>>>

**>>>** basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']

**>>> for** f **in** sorted(set(basket)):

**...**  print(f)

**...**

apple

banana

orange

pear

Bazen bir liste üzerinde döngüye girerken onu değiştirmek cazip gelebilir; ancak bunun yerine yeni bir liste oluşturmak genellikle daha basit ve daha güvenlidir.

>>>

**>>> import** **math**

**>>>** raw\_data = [56.2, float('NaN'), 51.7, 55.3, 52.5, float('NaN'), 47.8]

**>>>** filtered\_data = []

**>>> for** value **in** raw\_data:

**...**  **if** **not** math.isnan(value):

**...**  filtered\_data.append(value)

**...**

**>>>** filtered\_data

[56.2, 51.7, 55.3, 52.5, 47.8]

5.7. Koşullar Hakkında Daha Fazla Bilgi

whileve ifadelerinde kullanılan koşullar ifyalnızca karşılaştırmaları değil, herhangi bir işleci de içerebilir.

Karşılaştırma operatörleri inve bir değerin bir kapsayıcıda olup olmadığını (veya olmadığını) belirleyen üyelik testleridir. Operatörler ve iki nesnenin gerçekten aynı nesne olup olmadığını karşılaştırır. Tüm karşılaştırma operatörleri aynı önceliğe sahiptir ve bu, tüm sayısal operatörlerinkinden daha düşüktür.not inisis not

Karşılaştırmalar zincirlenebilir. Örneğin, 'den küçük olup olmadığını ve ayrıca eşit olup olmadığını test eder .a < b == cabbc

andKarşılaştırmalar Boolean operatörleri ve kullanılarak birleştirilebilir orve bir karşılaştırmanın (veya herhangi bir diğer Boolean ifadesinin) sonucu ile olumsuzlanabilir not. Bunların karşılaştırma operatörlerinden daha düşük öncelikleri vardır; bunlar arasında noten yüksek ve oren düşük önceliğe sahiptir, bu nedenle ile eşdeğerdir . Her zaman olduğu gibi, istenen bileşimi ifade etmek için parantezler kullanılabilir.A and not B or C(A and (not B)) or C

Boolean operatörleri andve orsözde *kısa devre* operatörleridir: argümanları soldan sağa doğru değerlendirilir ve değerlendirme sonuç belirlendiği anda durur. Örneğin, Ave Cdoğruysa ancak Byanlışsa ifadesini değerlendirmez . Genel bir değer olarak kullanıldığında ve Boolean olarak kullanılmadığında, kısa devre operatörünün dönüş değeri son değerlendirilen argümandır.A and B and CC

Bir karşılaştırmanın veya diğer Boolean ifadesinin sonucunu bir değişkene atamak mümkündür. Örneğin,

>>>

**>>>** string1, string2, string3 = '', 'Trondheim', 'Hammer Dance'

**>>>** non\_null = string1 **or** string2 **or** string3

**>>>** non\_null

'Trondheim'

Python'da, C'den farklı olarak, ifadelerin içindeki atamaların [walrus operatörü](https://docs.python.org/3/faq/design.html#why-can-t-i-use-an-assignment-in-an-expression) := ile açıkça yapılması gerektiğini unutmayın . Bu, C programlarında karşılaşılan yaygın bir sorun sınıfından kaçınılmasını sağlar: was amaçlandığında = bir ifade yazmak ==.

5.8. Dizileri ve Diğer Türleri Karşılaştırma

Sıra nesneleri genellikle aynı sıra türüne sahip diğer nesnelerle karşılaştırılabilir. Karşılaştırma *sözlüksel* sıralamayı kullanır: önce ilk iki öğe karşılaştırılır ve eğer farklılarsa bu karşılaştırmanın sonucunu belirler; eğer eşitlerse, sonraki iki öğe karşılaştırılır ve bu şekilde her iki sıra da tükenene kadar devam eder. Karşılaştırılacak iki öğenin kendisi aynı türden sıralarsa, sözlüksel karşılaştırma yinelemeli olarak gerçekleştirilir. İki sıranın tüm öğeleri eşit olarak karşılaştırılırsa, sıralar eşit kabul edilir. Bir sıra diğerinin başlangıç ​​alt sırasıysa, daha kısa sıra daha küçük (daha az) olandır. Dizelerin sözlüksel sıralaması, tek tek karakterleri sıralamak için Unicode kod noktası numarasını kullanır. Aynı türden sıralar arasındaki karşılaştırmalara bazı örnekler:

(1, 2, 3) < (1, 2, 4)

[1, 2, 3] < [1, 2, 4]

'ABC' < 'C' < 'Pascal' < 'Python'

(1, 2, 3, 4) < (1, 2, 4)

(1, 2) < (1, 2, -1)

(1, 2, 3) == (1.0, 2.0, 3.0)

(1, 2, ('aa', 'ab')) < (1, 2, ('abc', 'a'), 4)

Farklı türlerdeki nesneleri <veya ile karşılaştırmanın >, nesnelerin uygun karşılaştırma yöntemlerine sahip olması koşuluyla yasal olduğunu unutmayın. Örneğin, karışık sayısal türler sayısal değerlerine göre karşılaştırılır, bu nedenle 0, 0.0'a eşittir, vb. Aksi takdirde, keyfi bir sıralama sağlamak yerine, yorumlayıcı bir [TypeError](https://docs.python.org/3/library/exceptions.html#TypeError)istisna oluşturacaktır.

**Dipnotlar**

[ [1](https://docs.python.org/3/tutorial/datastructures.html#id1)]

Diğer diller, yöntem zincirlemesine izin veren mutasyona uğramış nesneyi döndürebilir, örneğin d->insert("a")->remove("b")->sort();.