Lists

Python'da listeler, sıralı değer dizilerini temsil eder. Listelerin nasıl oluşturulacağına dair bir örnek aşağıdadır:

primes = [2, 3, 5, 7]

Listelere başka türden şeyler de koyabiliriz:

planets = ['Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn', 'Uranus', 'Neptune']

Hatta listelerin listesini bile yapabiliriz:

hands = [

['J', 'Q', 'K'],

['2', '2', '2'],

['6', 'A', 'K'], *# (Comma after the last element is optional)*

]

*# (I could also have written this on one line, but it can get hard to read)*

hands = [['J', 'Q', 'K'], ['2', '2', '2'], ['6', 'A', 'K']]

Bir liste farklı tipteki değişkenlerin bir karışımını içerebilir:

my\_favourite\_things = [32, 'raindrops on roses', help]

*# (Yes, Python's help function is \*definitely\* one of my favourite things)*

## **Indexing**

Köşeli parantez kullanarak tek tek liste öğelerine erişebilirsiniz.

Hangi gezegen Güneş'e en yakındır? Python sıfır tabanlı indeksleme kullanır, bu nedenle ilk öğenin indeksi 0'dır.

planets[0]

'Mercury'

En yakın gezegen hangisi?

planets[1]

'Venus'

Which planet is furthest from the sun?

Listenin sonundaki elemanlara -1'den başlayarak negatif sayılarla erişilebilir:

planets[-1]

planets[-1]

'Neptune'

planets[-2]

'Uranus'

## **Slicing**

İlk üç gezegen hangileridir? Bu soruyu dilimleme yöntemiyle cevaplayabiliriz:

planets[0:3]

['Mercury', 'Venus', 'Earth']

planets[0:3], 0 indeksinden başlayıp 3 indeksini içermeyene kadar devam eden gezegenlerin elemanlarını sorma yöntemimizdir.

Başlangıç ve bitiş indeksleri isteğe bağlıdır. Başlangıç indeksini çıkarırsam, 0 olduğu varsayılır. Dolayısıyla yukarıdaki ifadeyi şu şekilde yeniden yazabilirim:

planets[:3]

['Mercury', 'Venus', 'Earth']

Son indeksi dışarıda bırakırsam, listenin uzunluğu olduğu varsayılır.

planets[3:]

['Mars', 'Jupiter', 'Saturn', 'Uranus', 'Neptune']

Yani yukarıdaki ifade "3. indeksten itibaren tüm gezegenleri bana ver" anlamına geliyor.

Dilimleme yaparken negatif indeksleri de kullanabiliriz:

*# All the planets except the first and last*

planets[1:-1]

['Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn', 'Uranus']

*# The last 3 planets*

planets[-3:]

['Saturn', 'Uranus', 'Neptune']

## **Changing lists**

Listeler "değiştirilebilir"dir, yani "yerinde" değiştirilebilirler.

Bir listeyi değiştirmenin bir yolu, onu bir indeks veya dilim ifadesine atamaktır.

Örneğin, Mars'ın adını değiştirmek istediğimizi varsayalım:

planets[3] = 'Malacandra'

planets

['Mercury',

'Venus',

'Earth',

'Malacandra',

'Jupiter',

'Saturn',

'Uranus',

'Neptune']

Hmm, bu epey uzun bir kelime. İlk 3 gezegenin isimlerini kısaltarak telafi edelim.

planets[:3] = ['Mur', 'Vee', 'Ur']

print(planets)

*# That was silly. Let's give them back their old names*

planets[:4] = ['Mercury', 'Venus', 'Earth', 'Mars',]

['Mur', 'Vee', 'Ur', 'Malacandra', 'Jupiter', 'Saturn', 'Uranus', 'Neptune']

## **List functions**

Python'da listelerle çalışmak için çeşitli kullanışlı işlevler bulunur.

len, bir listenin uzunluğunu verir:

*# How many planets are there?*

len(planets)

8

sorted, bir listenin sıralanmış halini döndürür:

*# The planets sorted in alphabetical order*

sorted(planets)

['Earth', 'Jupiter', 'Mars', 'Mercury', 'Neptune', 'Saturn', 'Uranus', 'Venus']

sum beklediğiniz şeyi yapar:

primes = [2, 3, 5, 7]

sum(primes)

17

Daha önce birkaç argümanın minimum veya maksimumunu elde etmek için min ve max değerlerini kullanmıştık. Ancak tek bir liste argümanı da geçirebiliriz.

max(primes)

7

## **Interlude: objects**

Şimdiye kadar "nesne" terimini çok kullandım; hatta Python'da her şeyin bir nesne olduğunu bile okumuş olabilirsiniz. Peki bu ne anlama geliyor?

Kısacası, nesneler yanlarında bazı şeyler taşırlar. Bu şeylere Python'ın nokta sözdizimini kullanarak erişirsiniz.

Örneğin, Python'daki sayılar, sanal kısımlarını temsil eden imag adlı ilişkili bir değişken taşırlar. (Çok tuhaf matematik işlemleri yapmıyorsanız muhtemelen bunu kullanmanıza hiç gerek kalmayacaktır.)

x = 12

*# x is a real number, so its imaginary part is 0.*

print(x.imag)

*# Here's how to make a complex number, in case you've ever been curious:*

c = 12 + 3j

print(c.imag)

0

3.0

Bir nesnenin taşıdığı şeyler, işlevleri de içerebilir. Bir nesneye bağlı bir işleve yöntem denir. (Bir nesneye bağlı işlev dışı öğelere, örneğin imag'e, öznitelik denir).

Örneğin, sayıların bit\_length adlı bir yöntemi vardır. Yine, buna nokta sözdizimini kullanarak erişiriz:

x.bit\_length

<function int.bit\_length()>

Bunu gerçekten adlandırmak için parantez ekliyoruz:

x.bit\_length()

4

Not: Eğer alıştırmaları yaptıysanız, aslında metotları zaten çağırmışsınızdır. Alıştırma defterlerinde q1, q2, q3 vb. nesnelerin hepsi check, hint ve solution adlı metotlara sahiptir.

Aynı şekilde yardım fonksiyonuna fonksiyonlar geçirebildiğimiz gibi (örneğin help(max)), metotları da geçirebiliriz:

help(x.bit\_length)

Help on built-in function bit\_length:

bit\_length() method of builtins.int instance

Number of bits necessary to represent self in binary.

>>> bin(37)

'0b100101'

>>> (37).bit\_length()

6

Yukarıdaki örnekler tamamen anlaşılmazdı. Şimdiye kadar incelediğimiz nesne türlerinin (sayılar, fonksiyonlar, boolean'lar) hiçbiri, muhtemelen kullanacağınız niteliklere veya yöntemlere sahip değil.

Ancak listelerin her zaman kullanacağınız birkaç yöntemi olduğu ortaya çıktı.

## **List methods**

list.append, bir listeyi sonuna bir öğe ekleyerek değiştirir:

*# Pluto is a planet darn it!*

planets.append('Pluto')

Yukarıdaki hücrenin neden çıktısı yok? help(planets.append) fonksiyonunu çağırarak dokümantasyona bakalım.

Not: append, yalnızca planetler için değil, liste türündeki tüm nesneler tarafından kullanılan bir yöntemdir, bu yüzden help(list.append) olarak da çağırabilirdik. Ancak help(append) olarak çağırmayı denersek, Python "append" adında bir değişkenin bulunmadığından şikayet edecektir. "append" adı yalnızca listelerde mevcuttur; max veya len gibi yerleşik işlevler gibi bağımsız bir ad olarak mevcut değildir.

help(planets.append)

Help on built-in function append:

append(object, /) method of builtins.list instance

Append object to the end of the list.

-> None kısmı bize list.append'in hiçbir şey döndürmediğini söylüyor. Ancak gezegenlerin değerini kontrol edersek, metot çağrısının gezegenlerin değerini değiştirdiğini görebiliriz:

planets

['Mercury',

'Venus',

'Earth',

'Mars',

'Jupiter',

'Saturn',

'Uranus',

'Neptune',

'Pluto']

list.pop bir listenin son elemanını siler ve döndürür:

planets.pop()

'Pluto'

planets

['Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn', 'Uranus', 'Neptune']

### **Searching lists**

Dünya gezegenler sıralamasında nerede yer alıyor? Dizinini list.index metodunu kullanarak elde edebiliriz.

planets.index('Earth')

2

Üçüncü sırada yer alıyor (yani 2 - 0 indeksinde!).

Plüton hangi indekste yer alıyor?

planets.index('Pluto')

---------------------------------------------------------------------------

ValueError Traceback (most recent call last)

/tmp/ipykernel\_19/2263615293.py in <module>

----> 1 planets.index('Pluto')

ValueError: 'Pluto' is not in list

Ah, doğru ya...

Bunun gibi tatsız sürprizlerden kaçınmak için, bir listenin belirli bir değeri içerip içermediğini belirlemek için in operatörünü kullanabiliriz:

*# Is Earth a planet?*

"Earth" **in** planets

True

*# Is Calbefraques a planet?*

"Calbefraques" **in** planets

False

Ele almadığımız birkaç ilginç liste yöntemi daha var. Belirli bir nesneye bağlı tüm yöntem ve öznitelikleri öğrenmek istiyorsanız, nesnenin kendisinde help() çağırabiliriz. Örneğin, help(planets) bize tüm liste yöntemleri hakkında bilgi verecektir:

help(planets)

Yardım sayfasının tamamını görmek için "çıktı" düğmesine tıklayın. Listelerde \_\_eq\_\_ ve \_\_iadd\_\_ gibi tuhaf görünümlü adlara sahip birçok yöntem bulunur. Şimdilik bunlar hakkında fazla endişelenmeyin. (Muhtemelen bu tür yöntemleri doğrudan çağırmayacaksınız. Ancak indeksleme veya karşılaştırma operatörleri gibi sözdizimleri kullandığımızda perde arkasında çağrılırlar.) En ilginç yöntemler listenin alt kısımlarındadır (ekleme, temizleme, kopyalama vb.).

## **Tuples**

Demetler, listelerle neredeyse tamamen aynıdır. Sadece iki noktada farklılık gösterirler.

1: Demetleri oluşturmak için kullanılan sözdizimi, köşeli parantez yerine parantez kullanır.

t = (1, 2, 3)

t = 1, 2, 3 *# equivalent to above*

t

(1, 2, 3)

2: Değiştirilemezler (değiştirilemezler).

t[0] = 100

---------------------------------------------------------------------------

TypeError Traceback (most recent call last)

/tmp/ipykernel\_19/816329950.py in <module>

----> 1 t[0] = 100

TypeError: 'tuple' object does not support item assignment

Demetler genellikle birden fazla dönüş değeri olan fonksiyonlar için kullanılır.

Örneğin, float nesnelerinin as\_integer\_ratio() metodu, bir demet biçiminde bir pay ve bir payda döndürür:

x = 0.125

x.as\_integer\_ratio()

(1, 8)

Bu çoklu dönüş değerleri aşağıdaki gibi ayrı ayrı atanabilir:

numerator, denominator = x.as\_integer\_ratio()

print(numerator / denominator)

0.125

Sonunda iki değişkeni değiştirmek için kullanılan klasik Aptalca Python Hilesi™ hakkında biraz bilgi edindik!

a = 1

b = 0

a, b = b, a

print(a, b)

1. 1