Loops

Döngüler, bazı kodları tekrar tekrar çalıştırmanın bir yoludur. İşte bir örnek:

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

for planet **in** planets:

print(planet, end=' ') *# print all on same line*

Mercury Venus Earth Mars Jupiter Saturn Uranus Neptune

For döngüsü şunları belirtir:

Kullanılacak değişken adı (bu durumda gezegen)

Döngü oluşturulacak değerler kümesi (bu durumda gezegenler)

Bunları birbirine bağlamak için "içinde" kelimesini kullanırsınız.

"İçinde" ifadesinin sağındaki nesne, yinelemeyi destekleyen herhangi bir nesne olabilir. Temel olarak, bir grup nesne olarak düşünülebiliyorsa, muhtemelen üzerinde döngü oluşturabilirsiniz. Listelere ek olarak, bir tuple'ın öğeleri üzerinde de yineleme yapabiliriz:

multiplicands = (2, 2, 2, 3, 3, 5)

product = 1

for mult **in** multiplicands:

product = product \* mult

product

360

Hatta bir dizedeki her karakterin üzerinden bile geçebilirsiniz:

s = 'steganograpHy is the practicE of conceaLing a file, message, image, or video within another fiLe, message, image, Or video.'

msg = ''

*# print all the uppercase letters in s, one at a time*

for char **in** s:

if char.isupper():

print(char, end='')

HELLO

### **range()**

range(), bir sayı dizisi döndüren bir fonksiyondur. Döngüler yazmak için oldukça kullanışlıdır.

Örneğin, bir eylemi 5 kez tekrarlamak istiyorsak:

for i **in** range(5):

print("Doing important work. i =", i)

Doing important work. i = 0

Doing important work. i = 1

Doing important work. i = 2

Doing important work. i = 3

Doing important work. i = 4

## **while loops**

Python'daki diğer döngü türü, bir koşul karşılanana kadar yineleme yapan while döngüsüdür:

i = 0

while i < 10:

print(i, end=' ')

i += 1 *# increase the value of i by 1*

0 1 2 3 4 5 6 7 8 9

While döngüsünün argümanı bir boole ifadesi olarak değerlendirilir ve ifade False olarak değerlendirilene kadar döngü yürütülür.

# List comprehensions

Liste kavramaları, Python'ın en sevilen ve benzersiz özelliklerinden biridir. Bunları anlamanın en kolay yolu muhtemelen birkaç örneğe bakmaktır:

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

squares

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

Liste kavrayışı olmadan aynı şeyi nasıl yapacağımızı görelim:

squares = []

for n **in** range(10):

squares.append(n\*\*2)

squares

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

Bir if koşulu da ekleyebiliriz:

short\_planets = [planet for planet **in** planets if len(planet) < 6]

short\_planets

['Venus', 'Earth', 'Mars']

(SQL'e aşinaysanız, bunu bir "WHERE" ifadesi gibi düşünebilirsiniz.)

İşte bir if koşuluyla filtreleme ve döngü değişkenine bazı dönüşümler uygulama örneği:

*# str.upper() returns an all-caps version of a string*

loud\_short\_planets = [planet.upper() + '!' for planet **in** planets if len(planet) < 6]

loud\_short\_planets

['VENUS!', 'EARTH!', 'MARS!']

İnsanlar bunları genellikle tek bir satıra yazarlar, ancak 3 satıra bölündüğünde yapının daha net anlaşılabileceğini görebilirsiniz:

[

planet.upper() + '!'

for planet **in** planets

if len(planet) < 6

]

['VENUS!', 'EARTH!', 'MARS!']

(SQL benzetmesine devam edersek, bu üç satırı SELECT, FROM ve WHERE olarak düşünebilirsiniz.)

Soldaki ifadenin teknik olarak döngü değişkenini içermesi gerekmez (gerçi içermemesi oldukça sıra dışı olurdu). Aşağıdaki ifadenin ne şekilde değerlendirileceğini düşünüyorsunuz? Kontrol etmek için 'çıktı' düğmesine basın.

[32 for planet **in** planets]

[32, 32, 32, 32, 32, 32, 32, 32]

Liste kavramaları, min, max ve sum gibi fonksiyonlarla birleştirildiğinde, aksi takdirde birkaç satır kod gerektirecek problemler için etkileyici tek satırlık çözümlere yol açabilir.

Örneğin, aynı işi yapan aşağıdaki iki hücre kodunu karşılaştırın.

def count\_negatives(nums):

*"""Return the number of negative numbers in the given list.*

*>>> count\_negatives([5, -1, -2, 0, 3])*

*2*

*"""*

n\_negative = 0

for num **in** nums:

if num < 0:

n\_negative = n\_negative + 1

return n\_negative

İşte liste kavrayışını kullanan bir çözüm:

def count\_negatives(nums):

return len([num for num **in** nums if num < 0])

Çok daha iyi, değil mi?

Eğer tek derdimiz kodumuzun uzunluğunu en aza indirmekse, bu üçüncü çözüm daha da iyi!

def count\_negatives(nums):

*# Reminder: in the "booleans and conditionals" exercises, we learned about a quirk of*

*# Python where it calculates something like True + True + False + True to be equal to 3.*

return sum([num < 0 for num **in** nums])