Bölüm 1: Program Elementleri

Haluk Tanrıkulu - www.haluktanrikulu.com

Örnek : Listedeki çift sayıların ortalamasını bulmak

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

```
numbers
```

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

```
count = 0
total = 0

for number in numbers:
    if number % 2 == 0:  # sadece çift sayıları bulalım
        count = count + 1
        total = total + number

average = total / count
```

Henüz bir çıktı görmüyoruz ama tekrar referans alarak ortalama değerini elde ediyoruz.

```
print(average)
```

```
7.0
```

Jupyter Notebook Doğrudan Çıktı Almak

```
"I am invisible!";
```

Önceki dört kod hücresinden yalnızca ikisi bir çıktı üretirken, ikisi "sessiz" kalır (yani, çalıştırıldıktan sonra hücrenin altında hiçbir şey görünmez). Varsayılan olarak, Jupyter not defterleri yalnızca bir kod hücresinin son satırındaki ifadenin değerini gösterir.

```
"Hello, World!"
"I am feeling great :-)"
```

```
print("Hello, World!")
print("I am feeling great :-)")
```

```
Hello, World!
I am feeling great :-)
```

```
print("Hello, World!"); print("I am feeling great :-)")
```

```
Hello, World!
I am feeling great :-)
```

```
-1
```

Aritmetik Operatörler

```
77 + 13
```

```
90
```

```
101 - 93
```

```
8
```

```
-1
```

```
2 * 21
```

```
84 / 2
```

```
42.0
```

```
85 % 2
```

```
     1
```

```
84 // 2
```

```
     42
```

```
85 // 2
```

```
     42
```

```
-85 // 2
```

```
     -43
```

Kalan Bul

```
49 % 7
```

```
     0
```

```
789 % 10
```

```
     9
```

```
789 % 100
```

```
     89
```

divmod kodunun çıktısında --- birinci // operatörünün işini yapar, ikinci ise kalanı (% operatörünün işini yapar) veren bir fonksiyon

```
divmod(42, 10)
```

```
     (4, 2)
```

```
2 ** 3
```

```
     8
```

```
3 ** 2 * 2
```

```
     18
```

```
greeting = "Merhaba!  "
audience = "YZ Sınıfı"
```

```
greeting + audience
```

```
(3 ** 2) * 2
```

```
     18
```

```
3 ** (2 * 2)
```

```
     81
```

```
3**2 * 2  # bu şekilde kullanmanızı tavsiye etmem. Yukarıdaki gibi hangi işin önce yapılma
```

```
     18
```

String (cümlecik) ile operatörleri birlikte kullanmak

```
10 * greeting
```

Nesneler, Tipler, Değerler

Python, bir programın belleğini düzenleme paradigması olan nesne yönelimli bir dildir. Bir nesne, belirli bir bellek konumunda 0'lar ve 1'lerden oluşan bir "çanta" olarak görülebilir. Bir torbadaki 0'lar ve 1'ler, nesnenin değerini oluşturur. Farklı çanta türleri vardır ve her türün 0'lar ve 1'lerin nasıl yorumlandığı ve çalışılabileceği kendi kuralları vardır. Dolayısıyla, bir nesnenin her zaman üç ana özelliği vardır. Aşağıdaki örneklere bakalım ve bunları çözelim.

```
a = 42
b = 42.0
c = "Python rocks"
```

Tanımlar : id() Fonksiyonu ile "Hafızadaki Konumu" bulalım.

id() fonksiyonu built-in yani halihazırda bizim kullanacağımız python ile gelen hazır fonksiyonlardan biridir. id() fonksiyonu bir nesnenin bellekteki "adresini" gösterir.

```
id(a)
```

```
94172275441440
```

```
id(b)
```

```
139623131254544
```

```
id(c)
```

```
139623068426160
```

Bu adresler, iki değişkenin aynı nesneye referansta bulunup bulunmadığını kontrol etmekten başka bir şey için anlamlı değildir. Açıkçası, a ve b eşitlik operatörü == tarafından ortaya konan değerle aynı değere sahiptir: a ve b'ye "eşit değerleri mi içerir" deriz. Ortaya çıkan **True** - veya aşağıdaki gibi **False** - mantıksal işlemlerin sonuçları boole adı verilen başka bir veri türüdür.

```
a == b  # gerçektende  42, 42.0 eşit mi?
```

```
True
```

Aksine, 'a' ve 'b', **identity operator** olarak kullandığımız **'is'** fonksiyonun gösterdiği gibi *farklı* nesnelerdir: Yani gerçektede bellekte *farklı* adreslerde saklanırlar.

```
a is b  # a, b midir? yani 42, 42.0 mıdır? Sayısal değerleri aynı ancak tipleri farklıdır.
```

```
False
```

```
a is not b
```

```
True
```

Veri Tipleri - Tipler / "İçerikleri" (Type / "Behavior")

a ve b değişkenlerinin tiplerine bakalım.

```
a.is_float()
```

```
type(a)
```

```
     int
```

```
type(b)
```

```
     float
```

a ve b bir birlerinden farklı tiplerde sayıları ifade eder. int, **tamsayı** olduğunu, float daha *akılada kalsın* diye **ondalıklı sayıları** ifade eder diyebiliriz.

```
b.is_integer()
```

```
     True
```

Bir **int** nesnesi için, bu **.is_integer()** denetimi, bir int olduğunu bildiğimiz bir tamsayıya kullanamayız. Çünkü bu float ama .0 şeklinde sayısal değeri tamsayı değeri ile eşit olan sayıları kıyaslamak için kullanılır. Daha açık söylersek, **.is_integer()** foksiyonunu sadece float tipi değerler için kullanırız. Aşağıdaki örneğe bakalım ve tamsayı bir değer alan a değişkeninde kullanalım : a is_integer()'in ne anlama geldiğini bile bilmediğinden, aşağıdaki AttributeError'ı görüyoruz.

```
type(c)
```

```
     str
```

```
c.lower()
```

```
a
```

```
b
```

```
c.upper()
```

```
c
```

Veri Tipleri Değeri / Anlamsal Anlamı (Value / (Semantic) "Meaning")

```
1 / 0
```

Neredeyse önemsiz bir şekilde, her nesnenin, başvurulduğunda değerlendirdiği bir değeri vardır. Değeri, çantadaki 0'lar ve 1'lerin insanlar için ne anlama geldiğine dair kavramsal fikir olarak düşünüyoruz. Başka bir deyişle, bir nesnenin değeri onun anlamsal anlamı ile ilgilidir. Yerleşik veri türleri için Python, bir nesnenin değerini sözde değişmez [literal] olarak yazdırır: Bu, değeri kopyalayıp bir kod hücresine geri yapıştırabileceğimiz ve aynı değere sahip yeni bir nesne oluşturabileceğimiz anlamına gelir.

```
42
```

```
42.0
```

- Çalışma Zamanı Hataları (Runtime Errors)
- Anlamsal Hatalar (Semantic Errors)

```
num1 = input('Bir sayı girin :')
num2 = input('İkinci bir sayı daha girin:')
sum = num1 + num2

print( num1 ,'ve', num2, ' sayılarının toplamı :', sum)
```

```
Bir sayı girin :2
İkinci bir sayı daha girin:6
2 ve 6  sayılarının toplamı : 26
```

```
sum  # Çok ilginç bir durumla karşılaşacağız. Aslında toplama için yukarıda bir hata yaptı
```

Hataları sistematik olarak bulmaya **hata ayıklama** (debugging) denir. Terimin tarihi için bu makaleye. bakın.

- En iyi pratik çalışmalar burada!

```
from IPython.display import YouTubeVideo
YouTubeVideo("Hwckt4J96dI", width="60%")
numbers = [7, 11, 8, 5, 3, 12, 2, 6, 9, 10, 1, 4]


çift_sayılar = [n for n in numbers if n % 2 == 0]  # liste içinde for döngüsü !


çift_sayılar
```

```
[8, 12, 2, 6, 10, 4]
```

```
average = sum(çift_sayılar) / len(çift_sayılar)  # sum() burada hazır fonksiyor
```

```
average
```

```
7.0
```

Zen of Python

```
import this

      The Zen of Python, by Tim Peters

      Beautiful is better than ugly.
      Explicit is better than implicit.
      Simple is better than complex.
      Complex is better than complicated.
      Flat is better than nested.
      Sparse is better than dense.
      Readability counts.
      Special cases aren't special enough to break the rules.
      Although practicality beats purity.
      Errors should never pass silently.
      Unless explicitly silenced.
      In the face of ambiguity, refuse the temptation to guess.
      There should be one-- and preferably only one --obvious way to do it.
      Although that way may not be obvious at first unless you're Dutch.
      Now is better than never.
      Although never is often better than *right* now.
      If the implementation is hard to explain, it's a bad idea.
      If the implementation is easy to explain, it may be a good idea.
      Namespaces are one honking great idea -- let's do more of those!
```