Introduction

Python'da bir değişken oluşturduğunuzda, bu değişkenin karşılık gelen bir veri türüne sahip bir değeri olur. Tam sayılar, ondalık sayılar, boolean'lar ve dizeler gibi birçok farklı veri türü vardır ve bunların hepsini bu derste ele alacağız. (Bu, mevcut veri türlerinin yalnızca küçük bir alt kümesidir; ayrıca sözlükler, kümeler, listeler, tuple'lar ve çok daha fazlası da vardır.)

Veri türleri önemlidir, çünkü bunlarla ne tür eylemler yapabileceğinizi belirlerler. Örneğin, iki ondalık sayıyı bölebilirsiniz, ancak iki dizeyi bölemezsiniz. Örneğin, 12.0/2.0 mantıklıdır, ancak "kedi"/"köpek" mantıklı değildir.

Hatalardan kaçınmak için, eylemlerin sahip olduğumuz veri türleriyle eşleştiğinden emin olmamız gerekir.

# Integers

Tam sayılar, kesirli kısmı olmayan sayılardır ve pozitif (1, 2, 3, ...), negatif (-1, -2, -3, ...) veya sıfır (0) olabilir.

Aşağıdaki kod hücresinde, x değişkenini bir tam sayıya ayarlıyoruz. Ardından type() ile veri türünü doğruluyor ve değişken adını parantez içine geçirmemiz yeterli oluyor.

x = 14

print(x)

print(type(x))

14

<class 'int'>

Yukarıdaki çıktıda, <class 'int'> tamsayı veri türünü ifade eder.

# Floats

Kayan noktalı sayılar, kesirli kısımlara sahip sayılardır. Ondalık basamaktan sonra birden fazla sayı içerebilirler.

nearly\_pi = 3.141592653589793238462643383279502884197169399375105820974944

print(nearly\_pi)

print(type(nearly\_pi))

3.141592653589793

<class 'float'>

Bir float'ı kesirli olarak da belirtebiliriz.

almost\_pi = 22/7

print(almost\_pi)

print(type(almost\_pi))

3.142857142857143

<class 'float'>

Kesirler için özellikle kullanışlı olan fonksiyonlardan biri round() fonksiyonudur. Bu fonksiyon, bir sayıyı belirtilen ondalık basamak sayısına yuvarlamanıza olanak tanır.

*# Round to 5 decimal places*

rounded\_pi = round(almost\_pi, 5)

print(rounded\_pi)

print(type(rounded\_pi))

3.14286

<class 'float'>

Ondalık noktalı bir sayı yazdığınızda, Python bunu bir float veri türü olarak tanır.

Örneğin, 1. (veya 1.0, 1.00 vb.) bir float olarak tanınır. Bu, bu sayıların teknik olarak kesirli kısmı olmamasına rağmen böyledir!

y\_float = 1.

print(y\_float)

print(type(y\_float))

1.0

<class 'float'>

# Booleans

Boole değerleri iki değerden birini temsil eder: Doğru veya Yanlış. Aşağıdaki kod hücresinde, z\_one, Doğru değerine sahip bir Boole değerine ayarlanmıştır.

z\_one = True

print(z\_one)

print(type(z\_one))

True

<class 'bool'>

Daha sonra z\_two, değeri False olan bir boole değerine ayarlanır.

z\_two = False

print(z\_two)

print(type(z\_two))

False

<class 'bool'>

Boole değerleri, bir ifadenin doğruluk değerini temsil etmek için kullanılır. 1 < 2 doğru bir ifade olduğundan, z\_three Doğru değerini alır.

z\_three = (1 < 2)

print(z\_three)

print(type(z\_three))

True

<class 'bool'>

Benzer şekilde, 5 < 3 yanlış bir ifade olduğundan, z\_four False değerini alır.

z\_four = (5 < 3)

print(z\_four)

print(type(z\_four))

False

<class 'bool'>

Bir boolean değişkeninin değerini not kullanarak değiştirebiliriz. Yani, not True, False'a eşdeğerdir ve not False, True olur.

z\_five = **not** z\_four

print(z\_five)

print(type(z\_five))

True

<class 'bool'>

Bir sonraki derste, koşullar ve koşullu ifadeleri öğrendiğinizde Boole'lar önemli hale gelecektir.

# Strings

Dize veri türü, tırnak işaretleri içinde yer alan karakterlerden (alfabe harfleri, noktalama işaretleri, sayısal rakamlar veya semboller gibi) oluşan bir koleksiyondur. Dizeler genellikle metni temsil etmek için kullanılır.

w = "Hello, Python!"

print(w)

print(type(w))

Hello, Python!

<class 'str'>

Bir dizenin uzunluğunu len() ile bulabilirsiniz. "Merhaba, Python!" ifadesinin uzunluğu 14'tür, çünkü boşluk, virgül ve ünlem işareti dahil 14 karakter içerir. Uzunluk hesaplanırken tırnak işaretlerinin dahil edilmediğini unutmayın.

print(len(w))

14

Özel bir dize türü, uzunluğu sıfır olan boş dizedir.

shortest\_string = ""

print(type(shortest\_string))

print(len(shortest\_string))

<class 'str'>

0

Bir sayıyı tırnak işareti içine alırsanız, bu sayının veri tipi string olur.

my\_number = "1.12321"

print(my\_number)

print(type(my\_number))

1.12321

<class 'str'>

Float'a dönüştürülebilen bir dizemiz varsa, float() kullanabiliriz.

Bu her zaman işe yaramaz! Örneğin, "10.43430" ve "3"ü float'a dönüştürebiliriz, ancak "Hello, Python!"ı float'a dönüştüremeyiz.

also\_my\_number = float(my\_number)

print(also\_my\_number)

print(type(also\_my\_number))

1.12321

<class 'float'>

Tıpkı iki sayıyı (kayan noktalı sayılar veya tam sayılar) toplayabildiğiniz gibi, iki dizeyi de toplayabilirsiniz. Bu, iki orijinal dizeyi birleştirerek daha uzun bir dize elde etmenizi sağlar.

new\_string = "abc" + "def"

print(new\_string)

print(type(new\_string))

abcdef

<class 'str'>

İki dizeyle çıkarma veya bölme işlemi yapamayacağınızı unutmayın. Ayrıca iki dizeyi çarpamazsınız, ancak bir dizeyi bir tam sayıyla çarpabilirsiniz. Bu da yine, orijinal dizenin belirli sayıda kendisiyle birleştirilmiş hali olan bir dizeyle sonuçlanır.

newest\_string = "abc" \* 3

print(newest\_string)

print(type(newest\_string))

abcabcabc

<class 'str'>

Bir string'i bir float ile çarpamayacağınızı unutmayın! Bunu yapmaya çalışmak bir hata döndürecektir.

will\_not\_work = "abc" \* 3.

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

TypeError Traceback (most recent call last)

/tmp/ipykernel\_19/2386798361.py in <module>

----> 1 will\_not\_work = "abc" \* 3.

TypeError: can't multiply sequence by non-int of type 'float'

Hatada, "dizi" "abc" dizesi ve "'float' türündeki tam sayı olmayan" ise float'tır (3.). Dolayısıyla, hata mesajı "dize ile float çarpılamaz" şeklinde yeniden ifade edilebilir.