ITU ACM Student Chapter Course Program

Introduction to Python

Week 1

Instructor
Zafer Yıldız

Assistants
Serra Bozkurt
Hüseyin Averbek

İçindekiler

Week 1

Print() Fonksiyonu.....3

Commentler.....3

Değişkenler 4

Veri Tipleri.....8

Input() Fonksiyonu.....8

Type Casting.....9

Python'a Giriş

1 Print() Fonksiyonu

Bir programın en temel fonksiyonlarından birisi ekrana çıktı vermektir. Python'da ekrana output bastırmak için print() fonksiyonu kullanılır:

print() fonksiyonu aracılığıyla ekrana farklı veri tipleri bastırılabilir. Bunlardan en sık kullanılanı stringlerdir. print() fonksiyonu ile ekrana string bastırmak için bastırılacak stringin " " işaretleri arasına yazılması gerekmektedir.

```
print("Hello World!")
```

```
Hello World!
```

print() fonksiyonu ile aynı zamanda matematiksel işlemlerin sonucu da bastırılabilir:

```
print(3+5)
```

```
8
```

2 Commentler

Yazdığımız kod satırları arasında çalışmasını istemediğimiz satırlar olabilir. Bu durumda der-leyici tarafından görülmesini istemediğimiz satırın başına '#' işareti koyduğumuz takdirde bu satır çalıştırılmayacaktır:

```
#print("Python Programlama Kursu")
print("Hoşgeldiniz!")
```

```
Hoşgeldiniz!
```

Yukarıda da görüldüğü gibi commente alınan ilk satırda yazan "Python Programlama Kursu" ifadesi print() fonksiyonu tarafından ekrana bastırılmazken, alt satırdaki "Hoşgeldiniz!" ifadesi program tarafından çalıştırıldı.

```
print("Hoşgeldiniz!") #Ekrana "Hoşgeldiniz!" ifadesi bastırılacak.
```

Commentlerin bir diğer işlevi ise yazılan kodun anlaşılabilirliğini artırmak amacıyla programdaki satırların işlevlerini yanına not almaktır.

3 Değişkenler

Değişkenler programlamada en sık kullanılan elemanlar arasında yer almaktadır. Değişkenler program içerisinde veri tutma görevini üstlenirler. İçlerinde tuttukları değer sonradan değiştirilebilir veya bu değer başka bir amaçla kullanılabilir.

3.0.1 Değişken Tanımlama

Değişken tanımlarken dikkat etmemiz gereken birtakım kurallar var. Öncelikle basit bir değişken tanımlama örneği gösterelim:

```
a = 15    # Bu satırda a değişkenine 15 değerini atadık.
```

```
print(a) # Bu satırda ise a değişkeninin değerini yazdırdık.
```

```
15
```

Değişkenin adı = Değişkenin alacağı değer

Değişken tanımlamayı yukarıdaki gibi gösterebiliriz. Ancak buradaki '=' işareti matematikteki eşitlik anlamından ziyade atama anlamı taşımaktadır.

Tek satırda birden fazla değişkene değer ataması yapılabilir:

```
a, b = 3,5
print(a)
print(b)
```

```
3
5
```

Bu aşamada değişkenimize vereceğimiz ad konusunda dikkatli ve açık olmalıyız. Bu konuda da bazı kurallar mevcut:

1) Değişken adının içinde boşluk bulunamaz.

```
hava sıcaklığı = 34

print(hava sıcaklığı)
```

```
  File "<ipython-input-3-ad46b0ee513e>", line 1
    hava sıcaklığı = 34

SyntaxError: invalid syntax
```

Bu şekilde adlandırılmış bir değişken çalışmayıp hata verecektir.

Birden fazla kelimeden oluşmasını istediğiniz değişken adlarını bitişik olacak biçimde yazmak durumundasınız.

```
havasıcaklığı = 34

print(havasıcaklığı)
```

```
34
```

Burda kelimeleri bitişik yazdığımızda oluşabilecek kargaşayı önlemek için iki farklı yönteme başvurabiliriz

```
havaSıcaklığı = 34 # Python'da büyük-küçük harf duyarlılığı vardır.

print(havaSıcaklığı)
```

```
34
```

```
hava_sıcaklığı = 34 # Kelimelerin arasına noktalama işareti koyularak␣
↪karışıklık azaltılabilir.

print(hava_sıcaklığı)
```

```
34
```

2) Değişken adı konarken anahtar kelimeler kullanamazsınız.

```
if = "merhaba" #Burada Python'da kullanılan bir anahtar kelime olan if'e␣
↪"merhaba" string değerine atamaya çalışıyoruz.
```

```
  File "<ipython-input-15-acea8714d593>", line 1
    if = "merhaba" #Burada Python'da kullanılan bir anahtar kelime olan if'e␣
↪"merhaba" string değerine atamaya çalışıyoruz.
       ^
SyntaxError: invalid syntax
```

Burada "if" Python dilinde kullanılan anahtar kelimelerden biri olduğundan hata aldık.

3) Değişkenler adlarının içinde sayı bulunabilir fakat değişken adları sayı ile başlayamaz.

```
1sayı = 18
```

```
  File "<ipython-input-8-096078785a38>", line 1
    1sayı = 18
     ^
SyntaxError: invalid syntax
```

Değişkenler tanımlanırken işlemler de yapılabilir. Örneğin bir değişkene iki sayının toplamını atayabiliriz.

```
toplam = 3+5

print(toplam)
```

8

Aynı zamanda diğer değişkenler ile yaptığımız işlemlerin sonucunu da bir başka değişkene atayabiliriz.

```
a = 12
b = 13

c = a+b

print(c)
```

25

Daha kompleks bir örnek vermemiz gerekirse:

```
a = 4
b = 6

c = 3*(a+b) - a*b

print(c)
```

6

Değişkenlere atayabildiğimiz bir diğer veri tipi ise stringlerdir:

```
isim = "Hüseyin"

print(isim)
```

Hüseyin

Stringleri birleştirmek için toplama işaretini kullanabiliriz:

```
isim = "Hüseyin"
soyisim = "Averbek"

kişi = isim + soyisim

print(kişi)
```

HüseyinAverbek

Burada "Hüseyin" stringinin sonunda veya "Averbek" stringinin başında boşluk bulunmadığı için program, ismi ve soyismi bitişik yazdırdı. Bu durumun önüne geçmek için string birleştirme işlemimize " " (boşluk) elemanını ekleyebiliriz:

```
isim = "Hüseyin"
soyisim = "Averbek"

kişi = isim + " " + soyisim

print(kişi)
```

```
Hüseyin Averbek
```

Uyarı: Tırnak işareti kullanılmadan yazılan sayılar ile stringler toplama işaretiyle birleştirilemez!

```
isim = "Hüseyin"
yaş = 21

isim_yaş = isim + " " + yaş

print(isim_yaş)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-15-44cd13e20d0a> in <module>
      2 yaş = 21
      3
----> 4 isim_yaş = isim + " " + yaş
      5
      6 print(isim_yaş)

TypeError: can only concatenate str (not "int") to str
```

Yukarıdaki hatayı almamızın sebebi [isim_yaş = isim + " " + yaş] satırında bir stringle bir tam sayıyı toplamaya çalışmamız. Bu durumun önüne geçmek için tamsayı değerimiz tırnak içine alınabilir. Böylelikle 2 stringi birleştirebiliriz:

```
isim = "Hüseyin"
yaş = "21"

isim_yaş = isim + " " + yaş

print(isim_yaş)
```

```
Hüseyin 21
```

4 Veri Tipleri

Python'da farklı verileri tutmak için farklı veri tiplerinde değişkenler kullanılır. En sık kullanılanları harfleri tutmak için string (str), tamsayıları tutmak için integer (int), ve rasyonel sayıları tutmak için float'tır (float).

4.0.1 type() Fonksiyonu

Python'da bir değişkenin veya ifadenin hangi veri tipinden olduğunu görmek için type() fonksiyonu kullanılır.

```
a = 3
print(type(a))
```

```
<class 'int'>
```

type() fonksiyonunun değeri değişkene atanmadan da print() fonksiyonu içinde kullanılabilir. Python matematikteki bileşke fonksiyon mantığıyla içten dışa fonksiyonların işlevini yerine getirir. g [f(x)] = (g o f) (x)

```
b = "merhaba"
print(type(b))
```

```
<class 'str'>
```

```
c = 12.5
print(type(c))
```

```
<class 'float'>
```

5 Input() Fonksiyonu

Bir programın en önemli fonksiyonlarından bir diğeri ise kullanıcıyla iletişime geçebilmesidir. Python'da kullanıcıdan değer girmesini talep etmek için input() fonksiyonu kullanılır.

input() fonksiyonu ile programa dışarıdan veri alınır. Alınan bu veri programda daha sonra kullanılmak üzere bir değişkene atanır. Input() fonksiyonuna parametre olarak girilen değer kullanıcıdan input almadan önce ekrana bastırılır:

```
n = input("Lütfen bir sayı giriniz:")
print(n)
```

```
Lütfen bir sayı giriniz:5
5
```

```
n = input("Lütfen bir string giriniz:")
print(n)
```

```
Lütfen bir string giriniz:ITU ACM Python Kursu
ITU ACM Python Kursu
```

Input fonksiyonuna herhangi bir parametre vermeden de çalıştırabiliriz. Bu durumda kullanıcının ekranın bir bilgilendirme mesajı belirmeyecektir:

```
girilenDeger = input()
print(type(girilenDeger))
```

```
5
<class 'str'>
```

Burada görüldüğü gibi input fonksiyonu aldığı inputları her zaman string türünde alır. Yukarıdaki örnekte de görüldüğü gibi girdiğimiz değer bir tamsayı olan 5 olmasına rağmen program bu değerin tipini string olarak algıladı. Tamsayıları, rasyonel sayıları ve stringleri birbirinden ayırmanın yolu ise type casting yöntemidir.

5.1 Type Casting

Python'da input halinde gelen değerleri farklı veri tiplerine çevirmek için kullanılır. int(), float(), ve str(); input() fonksiyonundan gelen veriyi istenen veri tipine çevirmek için kullanılır. Type casting yapmadan aldığımız inputları stringten istediğimiz veri tipine dönüştüremeyiz ve bu durumda istenilen işlemleri yaparken hata alabiliriz.

```
n = int(input())
print(type(n))
```

```
5
<class 'int'>
```

Casting yapmadığımız takdirde karşılaşabileceğimiz sorunlara örnek vermek gerekirse:

```
a = input("Birinci sayı:")
b = input("İkinci sayı:")

toplam = a + b
print(toplam)
```

```
Birinci sayı:3
İkinci sayı:5
35
```

Yukarıda da görüldüğü gibi toplama işlemi yapmak istediğimiz halde aldığımız inputları int (tam sayı) veri tipine dönüştürmediğimiz için string birleştirme işlemi gerçekleştirdik. Bu durumun önüne geçmek için önce aldığımız inputları int veri tipine dönüştürmemiz gerekir:

```
a = int(input("Birinci sayı:"))
b = int(input("İkinci sayı:"))

toplam = a + b
print(toplam)
```

```
Birinci sayı:3
İkinci sayı:5
8
```

Burada dikkat edilmesi gereken önemli nokta veri tiplerini karıştırmamaktır. Aksi takdirde hatayla karşılaşabiliriz:

```
n = int(input())
print(n)
```

Python3

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-33-93ddc6c45b1a> in <module>
----> 1 n = int(input())
      2 print(n)

ValueError: invalid literal for int() with base 10: 'Python3'
```