**Basic Data Types And Useful Operations**

**Arithmetic Operations**

In Python, there are almost all of the arithmetic operations we use in mathematics. They are so simple to use and we can also use these operations on almost all data types, including string.

Basic Arithmetic operators are as follows :

|  |
| --- |
|  |

Let's now grasp the arithmetic operations with several examples:

input :

print(4 + 11) # sum of integers gives integer

output :

15

input :

print(39 + 1.0) # sum of an integer and float gives float

output :

40.0

input :

no1, no2 = 46, 52

no3 = no1 - no2

print(no3)

output :

-6

**💡Tips:**

* We can assign a value to multiple variables. Consider this: variable1 = variable2 = 'clarusway opens your path'.
* We can also assign multiple values to multiple variables in sequence using commas as in example above.

input :

no1 = 46

print(no1/23) # division gives float

output :

2.0

input :

print((3 \* 4)/2) # parentheses are used as in normal mathematics operations

output :

6.0

input :

print(7 // 2) # it gives integer part of division

output :

3

input :

print(9 % 2) # remainder of this division is 1

# it means 9 is an odd number

output :

1

input :

print(3\*\*2)

output :

9

input :

print(2\*\*3)

output :

8

input :

print(64\*\*0.5) # square root

output :

8.0

input :

print('Result of this (12+7) sum :', 12 + 7)

output :

Result of this (12+7) sum : 19

There is a list of priorities for all considered operations: it is worth keeping this priority in your mind.

1. parentheses : ()
2. power : \*\*
3. unary minus : -3
4. multiplication and division : \*, /
5. addition and subtraction : +, -

**Aritmetik işlemler**

Python'da matematikte kullandığımız aritmetik işlemlerin neredeyse tamamı vardır. Kullanımı çok basittir ve bu işlemleri dize dahil hemen hemen tüm veri türlerinde de kullanabiliriz.  
  
Temel Aritmetik operatörler aşağıdaki gibidir:  
  
  
Python'da Aritmetik Operatörler  
Şimdi aritmetik işlemleri birkaç örnekle kavrayalım:  
  
Giriş:  
  
Print(4 + 11) # tam sayıların toplamı tamsayı verir  
Çıktı :  
  
15  
Giriş:  
  
Print(39 + 1.0) # bir tamsayı ve şamandıranın toplamı şamandıra verir  
Çıktı :  
  
40.0  
Giriş:  
  
No1, no2 = 46, 52  
No3 = no1 - no2  
Baskı(no3)  
Çıktı :  
  
-6  
💡İpuçları:  
Birden çok değişkene değer atayabiliriz. Şunu düşünün: değişken1 = değişken2 = 'clarusway yolunuzu açar'.  
Ayrıca yukarıdaki örnekte olduğu gibi virgül kullanarak sırayla birden çok değişkene birden çok değer atayabiliriz.  
Giriş:  
  
1 numara = 46  
Print(no1/23) # bölme şamandıra verir  
Çıktı :  
  
2.0  
Giriş:  
  
Print(3 \* 4)/2) # parantez normal matematik işlemlerinde olduğu gibi kullanılır  
Çıktı :  
  
6.0  
Giriş:  
  
Print(7 // 2) # bölmenin tamsayı kısmını verir  
Çıktı :  
  
3  
Giriş:  
  
Print(9 % 2) # bu bölümün kalanı 1  
# 9'un tek bir sayı olduğu anlamına gelir  
Çıktı :  
  
1  
Giriş:  
  
Yazdır(3\*\*2)  
Çıktı :  
  
9  
Giriş:  
  
Yazdır(2\*\*3)  
Çıktı :  
  
8  
Giriş:  
  
Print(64\*\*0.5) # karekök  
Çıktı :  
  
8.0  
Giriş:  
  
Print('Bunun (12+7) toplamının sonucu:', 12 + 7)  
Çıktı :  
  
Bu (12+7) toplamının sonucu : 19  
Dikkate alınan tüm işlemler için bir öncelikler listesi vardır: bu önceliği aklınızda tutmaya değer.  
  
Parantez : ()  
Güç : \*\*  
Tekli eksi : -3  
Çarpma ve bölme : \*, /  
Toplama ve çıkarma : +, -

**Operations with 'print( )' Function**

Let's open a title here and take a closer look at print(), which is the most frequently used function. Since the need to make constant changes and see the results frequently occurs when writing code, printing directly on the screen can be the choice of most developers.

input :

number = 2020

text = "children deserve respect as much as adults in"

print(text, number)

output :

children deserve respect as much as adults in 2020

When using print() we can write more than one expression in parentheses separated by 👉🏻,

input :

print("yesterday I ate", 2, "apples")

output :

yesterday I ate 2 apples

When you type more than one expression in print(), you notice that the expressions are joined to each other by spaces. This is due to the default value of keyword argument **sep** in the print() function. This argument, which is defined as a **space** 👉🏻" " by default, is not visible in the background in the print().

The print() command automatically switches to the next line. This is due to the keyword argument end = "\n"

Here are the keyword arguments that run in the background of the print() function :

print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

* 👉🏻\n represents next line.
* Let's focus on the arguments sep= and end= we discussed above:

input :

print('i', end=' ')

print('will say', end=' ')

print("'i missed you'", end=' ')

print('to my mother')

output :

i will say 'i missed you' to my mother

If you noticed, in the example above we have combined all expressions with space in a single line using the 👉🏻end= If we didn't use end=' ', we would normally get 4 lines of output.

input :

print('smoking', 'is', 'slowly', 'killing me', sep=' + ')

output :

smoking + is + slowly + killing me

Some other useful operations that covers arithmetic and print() function are as follows. Carefully examine the examples:

input :

x = 5

print ('value of x : ', x)

x += 2

print ("2 more of x : ", x, "\n") # using string expression '\n',

# we produce extra line.

# So that we had empty line.

y = 10

print ('value of y : ', y)

y -= 2

print ("2 minus y : ", y, "\n")

z = 6

print ('value of z : ', z)

z \*= 2

print ("2 times z : ", z, "\n")

output :

value of x : 5

2 more of x : 7

value of y : 10

2 minus y : 8

value of z : 6

2 times z : 12

**💡Tips:**

* Variable math operator = number gives the same result as Variable = Variable math operator number.
* Variable += number gives the same result as Variable = Variable + number.

Any mathematics operator can be used before the **=** character to make an in-place operation:

* **-=** decrements the variable in place,
* **+=** increment the variable in place,
* **\*=** multiply the variable in place,
* **/=** divide the variable in place,
* **//=** ﬂoor divide the variable in place,
* **%=** returns the modulus of the variable in place,
* **\*\*=** raise to power in place.

input :

fruit = 'Orange'

vegetable = "Tomato"

print (fruit, """ and """ , vegetable)

output :

Orange and Tomato

'print()' Fonksiyonlu İşlemler  
Burada bir başlık açalım ve en sık kullanılan fonksiyon olan print()'e daha yakından bakalım. Kod yazarken sürekli değişiklik yapma ve sonuçları sık sık görme ihtiyacı ortaya çıktığı için, doğrudan ekrana yazdırmak çoğu geliştiricinin tercihi olabilir.  
  
Giriş:  
  
Sayı = 2020  
Text = 'çocuklar yetişkinler kadar saygıyı hak ediyor'  
Yazdır(metin, sayı)  
Çıktı :  
  
2020'de çocuklar da yetişkinler kadar saygıyı hak ediyor  
Print() kullanırken, 👉🏻 ile ayırarak parantez içinde birden fazla ifade yazabiliriz,  
  
Giriş:  
  
Print('dün yedim', 2, 'elma')  
Çıktı :  
  
Dün 2 elma yedim  
Print()'de birden fazla ifade yazdığınızda, ifadelerin boşluklarla birbirine bağlandığını fark edeceksiniz. Bunun nedeni, print() işlevindeki sep anahtar kelime argümanının varsayılan değeridir. Varsayılan olarak boşluk 👉🏻' ' olarak tanımlanan bu argüman print()'de arka planda görünmez.  
  
Print() komutu otomatik olarak bir sonraki satıra geçer. Bunun nedeni end = '  
' anahtar sözcüğü bağımsız değişkenidir.  
  
Print() fonksiyonunun arka planında çalışan anahtar kelime argümanları şunlardır:  
  
Print(değer, ..., sep=' ', bitiş='  
', dosya=sys.stdout, flush=Yanlış)  
  
👉🏻  
sonraki satırı temsil eder.  
  
Yukarıda tartıştığımız sep= ve end= argümanlarına odaklanalım:  
  
Giriş:  
  
Print('i', bitiş=' ')  
Print('diyecek', bitiş=' ')  
Print(''seni özledim'', bitiş=' ')  
Print('anneme')  
Çıktı :  
  
Anneme seni özledim diyeceğim  
Dikkat ettiyseniz, yukarıdaki örnekte boşluklu tüm ifadeleri 👉🏻end= kullanarak tek satırda birleştirdik end=' ' kullanmasaydık normalde 4 satır çıktı alırdık.  
  
Giriş:  
  
Print('sigara içiyor', 'olur', 'yavaşça', 'beni öldürüyor', eylül=' + ')  
Çıktı :  
  
Sigara içmek + yavaş yavaş + beni öldürüyor  
Aritmetik ve print() işlevini kapsayan diğer bazı yararlı işlemler aşağıdaki gibidir. Örnekleri dikkatlice inceleyin:  
  
Giriş:  
  
X = 5  
Print ('x'in değeri:', x)  
  
X += 2  
Print ('2 tane daha x : ', x, '  
') # '  
' dize ifadesini kullanarak,  
#ekstra hat üretiyoruz.  
# Böylece boş satırımız oldu.  
Y = 10  
Print ('y'nin değeri:', y)  
  
Y -= 2  
Yazdır ('2 eksi y : ', y, '  
')  
  
Z = 6  
Print ('z'nin değeri:', z)  
  
Z \*= 2  
Yazdır ('2 kez z : ', z, '  
')  
Çıktı :  
  
X'in değeri: 5  
2 tane daha x : 7  
  
Y'nin değeri: 10  
2 eksi y : 8  
  
Z'nin değeri : 6  
2 kere z : 12  
💡İpuçları:  
Değişken matematik operatörü = sayı, Değişken = Değişken matematik operatör numarası ile aynı sonucu verir.  
Değişken += sayı, Değişken = Değişken + sayı ile aynı sonucu verir.  
Yerinde bir işlem yapmak için = karakterinden önce herhangi bir matematik operatörü kullanılabilir:  
  
-= değişkeni yerinde azaltır,  
+= değişkeni yerinde artır,  
\*= değişkeni yerinde çarpın,  
/= değişkeni yerine böl,  
//= kat değişkeni yerine böl,  
%= yerinde değişkenin modülünü döndürür,  
\*\*= yerinde güce yükseltin.  
Giriş:  
  
Meyve = 'Turuncu'  
Sebze = 'Domates'  
Print (meyve, ''' ve ''' , sebze)  
Çıktı :  
  
Portakal ve Domates

**Escape Sequences**

Actually, the examples in the previous lesson show us how backslash 👉🏻\ works. 👉🏻\ is a special sign used in expressions called **escape sequences**, which behaves according to the character immediately after 👉🏻\. Here are basic escape sequences in Python:

* **\n** : means new line,
* **\t** : means tab mark,
* **\b** : means backspace. It moves the cursor one character to the left.

Look at these examples carefully:

input :

print('C:\\november\number\_expenditure.txt')

output :

C:\november

umber\_expenditure.txt

input :

print("one", "two", "three", sep="\t") # separated by tab marks

output :

one two three

input :

print('we', '\bare', '\bunited') # remember, normally print() function

# separates expressions by spaces

output :

weareunited

Normally when we use 👉🏻**'** inside the 👉🏻**' '**, Python will give error. Because single-quote in single quotes gives an error. But here, in the example below, 👉🏻**\** allows single-quote 👉🏻**'** to be ignored. So it gives no error.

input :

print('it\'s funny to learn Python')

output :

it's funny to learn Python

Kaçış dizileri  
Aslında bir önceki dersteki örnekler bize ters eğik çizginin 👉🏻 nasıl çalıştığını gösteriyor. 👉🏻, 👉🏻'den hemen sonra gelen karaktere göre davranan, kaçış dizileri adı verilen ifadelerde kullanılan özel bir işarettir. Python'daki temel kaçış dizileri şunlardır:  
  
  
: yeni satır anlamına gelir,  
  
T : sekme işareti anlamına gelir,  
  
B : geri al anlamına gelir. İmleci bir karakter sola hareket ettirir.  
  
Bu örneklere dikkatlice bakın:  
  
Giriş:  
  
Print('C:  
ovember  
umber\_expenditure.txt')  
Çıktı :  
  
C:kasım  
Umber\_expenditure.txt  
Giriş:  
  
Print('bir', 'iki', 'üç', sep='t') # sekme işaretleriyle ayrılmış  
Çıktı :  
  
Bir, iki, üç  
Giriş:  
  
Print('we', 'bare', 'bunited') # hatırla, normalde print() işlevi  
# ifadeleri boşluklarla ayırır  
Çıktı :  
  
Yıpranmış  
Normalde 👉🏻' içinde 👉🏻' kullandığımızda Python hata verir. Çünkü tek tırnak içinde tek tırnak hata veriyor. Ancak burada, aşağıdaki örnekte 👉🏻 tek tırnaklı 👉🏻' yoksayılmasına izin verir. Yani hata vermiyor.  
  
Giriş:  
  
Print('Python öğrenmek komik')  
Çıktı :  
  
Python öğrenmek eğlencelidir

**Definitions Boolean**

As we learned in the previous lesson boolean or bool can only have two values. True and False.

To put it easily, we can say that bool represent 1 and 0. In other words, yes & no or exist & nonexistent can be expressed by bool type.

For example, let's define a variable as to whether students have passed a course. Let the variable be called is\_pass. Then;

If you pass the course : is\_pass = True,  
If you did not pass the course : is\_pass = False

Q: What is a boolean in Python?  
A: Boolean is one of the built-in data types in Python, it mainly contains two values, and they are **True** and **False**.

- Interview Q&A

**Tanımlar**  
Önceki derste öğrendiğimiz gibi boolean veya bool sadece iki değere sahip olabilir. Doğru ve yanlış.  
  
Basitçe söylemek gerekirse, bool'un 1 ve 0'ı temsil ettiğini söyleyebiliriz. Başka bir deyişle, evet ve hayır veya var ve yok, bool türü ile ifade edilebilir.  
  
Örneğin öğrencilerin bir dersi geçip geçmediğine dair bir değişken tanımlayalım. Değişken is\_pass olarak adlandırılsın. Sonra;  
  
Kursu geçerseniz: is\_pass = True,  
Kursu geçmediyseniz: is\_pass = False  
S: Python'da boole nedir?  
C: Boolean, Python'daki yerleşik veri türlerinden biridir, esas olarak iki değer içerir ve bunlar Doğru ve Yanlış'tır.

**Boolean Logic Expressions**

Python has three built-in boolean operators: and, or and not. Except not, all are binary operators, which means two arguments are required.

**And** operator : The and operator evaluates all expressions and returns the last expression if all expressions are evaluated True. Otherwise, it returns the ﬁrst value that evaluated False.

d

**Or** operator : The or operator evaluates the expressions left to right and returns the ﬁrst value that evaluated True or the last value (if none is True).

| **Value1** | **Logic** | **Value2** | **Returns** |
| --- | --- | --- | --- |
| True | and | True | True |
| True | and | False | False |
| False | and | False | False |
| False | and | True | False |
| True | or | True | True |
| True | or | False | True |
| False | or | False | False |
| False | or | True | True |

Q: Python has three built-in Boolean operators. What are they?  
A: They are and, or, not.

**Boole Mantık İfadeleri**

Python'un üç yerleşik boole operatörü vardır: and, or and not. Değilse, tümü ikili operatörlerdir, bu da iki bağımsız değişkenin gerekli olduğu anlamına gelir.  
  
Ve operatörü : and operatörü tüm ifadeleri değerlendirir ve tüm ifadeler True olarak değerlendirilirse son ifadeyi döndürür. Aksi takdirde, False olarak değerlendirilen ilk değeri döndürür.  
  
NS  
  
Veya operatörü : veya operatörü ifadeleri soldan sağa değerlendirir ve True olarak değerlendiren ilk değeri veya son değeri (hiçbiri True değilse) döndürür.  
  
Değer1MantıkDeğer2İadeler  
TrueandTrueTrue  
DoğruveYanlışYanlış  
YanlışveYanlışYanlış  
YanlışveDoğruYanlış  
TrueorTrueTrue  
TrueorYanlışTrue  
YanlışYanlışYanlışYanlış  
FalseorTrueTrue  
S: Python'un üç yerleşik Boole işleci vardır. Onlar neler?  
C: Onlar ve, ya da değiller.

**Order of Priority**

It is important to remember that, logical operators have a different priority and it has an effect on the order of evaluation. Here are the operators in order of their priorities:

1. **not**
2. **and**
3. **or**

For example : x = True and not True, the value of x returns False.

It evaluates not True first and gives False. It becomes x = True and False and gives False.

Let's consider one more example :

input :

logic = True and False or not False or False

print(logic)

output :

True

You can follow the logic priority flow in the diagram.

| **Tiobe_Index** |
| --- |
| *Bool Logic Priority* |

**💡Tips:**

* Note that and and or return one of its operands, not necessarily a bool type. But not always returns bool type.

Q: What is the order of priority of the logical operators?  
A:

1. not
2. and
3. or

**Öncelik sırası**

Mantıksal operatörlerin farklı bir önceliğe sahip olduğunu ve değerlendirme sırasını etkilediğini hatırlamak önemlidir. Operatörler önceliklerine göre sıralanmıştır:  
  
Olumsuz  
Ve  
Veya  
Örneğin: x = Doğru ve Doğru değil, x'in değeri False değerini döndürür.  
  
Önce True değil değerini değerlendirir ve False değerini verir. X = Doğru ve Yanlış olur ve Yanlış verir.  
  
Bir örnek daha ele alalım:  
  
Giriş:  
  
Mantık = Doğru ve Yanlış veya Yanlış veya Yanlış değil  
Yazdır(mantık)  
Çıktı :  
  
NS  
Diyagramda mantık önceliği akışını takip edebilirsiniz.  
  
  
Bool Mantık Önceliği  
💡İpuçları:  
Unutmayın ve ve/veya işlenenlerinden birini döndürün, mutlaka bir bool türü değil. Ancak her zaman bool türünü döndürmez.  
S: Mantıksal operatörlerin öncelik sırası nedir?  
A:  
  
Olumsuz  
Ve  
Veya

**Truth Values of Logic Statements**

Although Python has its own boolean data type, we often use non-boolean values in logical operations.

The values of non-boolean types (integers, strings, etc.) are considered truthy or falsy when used with logical operations, depending on whether they are seen as True or False.

The following values are considered False, in that they evaluate to False when applied to a boolean operator:

* None.
* Zero of any numeric type: 0, 0.0, 0j
* Empty sequences and collections: '', [], {}.
* Other than above values, any remaining value is evaluated as True.

Here are some and operations :

input :

print(2 and 3)

output :

3

input :

print(1 and 0)

output :

0

input :

print(1 and "I am doing good!")

output :

I am doing good!

input :

print([] and "Hello World!")

output :

[]

Here are some or operations :

input :

print(2 or 3)

output :

2

input :

print(None or 1)

output :

1

input :

print(0 or {})

output :

{}

input :

print([] or "Hello World!")

output :

Hello World!

Q: What are the values evaluated to False when applied to a Boolean operator?  
A:

* None and False.
* Zero of any numeric type: 0, 0.0, 0j.
* Empty sequences and collections: '', [], {}.
* Any remaining value is evaluated as True.

**Mantık İfadelerinin Doğruluk Değerleri**

Python'un kendi boole veri türü olmasına rağmen, mantıksal işlemlerde genellikle boole olmayan değerler kullanırız.  
  
Boole olmayan türlerin (tamsayılar, dizeler vb.) değerleri, mantıksal işlemlerle kullanıldığında True veya False olarak görülmelerine bağlı olarak doğru veya yanlış olarak kabul edilir.  
  
Aşağıdaki değerler, bir boole işlecine uygulandığında False olarak değerlendirdikleri için False olarak kabul edilir:  
  
Hiçbiri.  
Herhangi bir sayısal türden sıfır: 0, 0.0, 0j  
Boş diziler ve koleksiyonlar: '', [], {}.  
Yukarıdaki değerler dışında kalan tüm değerler True olarak değerlendirilir.  
İşte bazıları ve işlemler:  
  
Giriş:  
  
Yazdır(2 ve 3)  
Çıktı :  
  
3  
Giriş:  
  
Yazdır(1 ve 0)  
Çıktı :  
  
0  
Giriş:  
  
Print(1 ve 'İyiyim!')  
Çıktı :  
  
İyi yapıyorum!  
Giriş:  
  
Print([] ve 'Merhaba Dünya!')  
Çıktı :  
  
[]  
İşte bazı veya işlemler:  
  
Giriş:  
  
Yazdır(2 veya 3)  
Çıktı :  
  
2  
Giriş:  
  
Yazdır(Yok veya 1)  
Çıktı :  
  
1  
Giriş:  
  
Yazdır(0 veya {})  
Çıktı :  
  
{}  
Giriş:  
  
Print([] veya 'Merhaba Dünya!')  
Çıktı :  
  
Selam Dünya!  
S: Bir Boole işlecine uygulandığında False olarak değerlendirilen değerler nelerdir?  
A:  
  
Yok ve Yanlış.  
Herhangi bir sayısal türden sıfır: 0, 0.0, 0j.  
Boş diziler ve koleksiyonlar: '', [], {}.  
Kalan herhangi bir değer True olarak değerlendirilir.

**Indexing&Slicing Strings**

As we mentioned earlier, one of the most powerful aspects of Python is its string processing capability. You can access all elements of a string type data very easily. Accordance with the sequence of string letters, you can specify them from left to right in brackets, as follows:

input :

fruit = 'Orange'

print('Word : ' , fruit)

print('First letter : ' , fruit[0])

print('Second letter : ' , fruit[1])

print("3rd to 5th letters : " , fruit[2:5])

print("Letter all after 3rd : " , fruit[2:])

output :

Word : Orange

First letter : O

Second letter : r

3rd to 5th letters : ang

Letter all after 3rd : ange

**💡Tips:**

* Remember, the enumeration of a string starts from **zero**.

**The formula syntax of string indexing is : string[start:stop:step]**.

**string[:]** : returns the full copy of the sequence

**string[start:]** : returns elements from start to the end element

**string[:stop]** : returns element from the 1st element to stop-1

**string[::step]** : returns each element with a given step

Let's see it in an example :

input :

city = 'Phoenix'

print(city[1:]) # starts from index 1 to the end

print(city[:6]) # starts from zero to 5th index

print(city[::2]) # starts from zero to end by 2 step

print(city[1::2]) # starts from index 1 to the end by 2 step

print(city[-3:]) # starts from index -3 to the end

print(city[::-1]) # negative step starts from the end to zero

output :

hoenix

Phoeni

Ponx

hei

nix

xineohP

| **indexing_String** |
| --- |
| *Diagram of Indexing a String* |

You can use the len() function to find out the length (number of characters) of a text or a variable of any type.

input :

vegetable = 'Tomato'

print('length of the word', vegetable, 'is :', len(vegetable))

output :

length of the word Tomato is : 6

**Q**: What is the output of print(str[4:]) if str = 'Python Language' ?  
**A**: on Language

**Dizin Oluşturma ve Dilimleme**

Daha önce de belirttiğimiz gibi, Python'un en güçlü yönlerinden biri, dize işleme yeteneğidir. Bir dize tipi verinin tüm öğelerine çok kolay bir şekilde erişebilirsiniz. Dize harflerinin sırasına göre, bunları aşağıdaki gibi parantez içinde soldan sağa belirtebilirsiniz:  
  
Giriş:  
  
Meyve = 'Turuncu'  
  
Print('Kelime:', meyve)  
Print('İlk harf :' , meyve[0])  
Print('İkinci harf :' , meyve[1])  
Print('3. ila 5. harf : ' , meyve[2:5])  
Print('3'ten sonraki harf : ' , meyve[2:])  
Çıktı :  
  
Kelime: turuncu  
İlk harf : O  
İkinci harf : r  
3. ila 5. harfler: ang  
3'ten sonraki harf : öfke  
💡İpuçları:  
Unutmayın, bir dizenin numaralandırılması sıfırdan başlar.  
Dize indekslemenin formül sözdizimi şöyledir: string[start:stop:step].  
  
String[:] : dizinin tam kopyasını döndürür  
  
String[start:] : başlangıçtan bitiş öğesine kadar öğeleri döndürür  
  
String[:stop] : 1. elemandan stop-1'e eleman döndürür  
  
String[::step] : her öğeyi belirli bir adımla döndürür  
  
Bir örnekte görelim:  
  
Giriş:  
  
Şehir = 'Phoenix'  
  
Print(city[1:]) # dizin 1'den sonuna kadar başlar  
Print(city[:6]) # sıfırdan 5. dizine kadar başlar  
Print(city[::2]) # sıfırdan 2 adıma kadar başlar  
Print(city[1::2]) # dizin 1'den 2 adım sonuna kadar başlar  
Print(city[-3:]) # -3 dizininden sonuna kadar başlar  
Print(city[::-1]) # negatif adım sondan sıfıra başlar  
Çıktı :  
  
Hoenix  
Fenike  
Ponks  
Merhaba  
Nix  
XineohP  
  
Bir Dizeyi İndeksleme Şeması  
Bir metnin veya herhangi bir türdeki bir değişkenin uzunluğunu (karakter sayısını) bulmak için len() işlevini kullanabilirsiniz.  
  
Giriş:  
  
Sebze = 'Domates'  
  
Print('kelimenin uzunluğu', sebze, 'is :', len(sebze))  
Çıktı :  
  
Domates kelimesinin uzunluğu : 6  
S: str = 'Python Language' ise print(str[4:]) çıktısı nedir?  
A: Dil hakkında

**String Formatting with Arithmetic Syntax**

There are several ways in Python that we use when processing and using string data structures. The most important of these are:

* Arithmetic syntax (**+**, **\***, and **=**),
* **%** operator formatting,
* **string.format()** method,
* **f-string** formatting.

We have stated to you what the **function** is? in the previous lessons. At this point, let us give the definition of the term **method**. A **method** is like a function, except it is attached to an **object**. We call a method on an object, and it possibly makes changes to that object (like string.format()). A method, then, belongs to a class.

Arithmetic syntax (**+**, **=**, **\***) :

We can use + operator for combining the two string together without any spaces. For example :

input :

print('clarus' + 'way')

output :

clarusway

We can also use \* operator for repeating the string without any spaces. For example :

input :

print(3\*'no way!')

output :

no way!no way!no way!

Examine the following example carefully :

input :

fruit = 'Orange'

vegetable = 'Tomato'

print("using + :", fruit + vegetable)

print("using \* :", 3 \* fruit)

output :

using + : OrangeTomato

using \* : OrangeOrangeOrange

As with numeric types, we can do addition operation in-place either with string type using 👉🏻+=. Look at the examples below :

input :

fruit = 'orange'

fruit += ' apple'

print(fruit)

output :

orange apple

input :

fruit = 'orange'

fruit += ' apple'

fruit += ' banana'

fruit += ' apricot'

print(fruit)

output :

orange apple banana apricot

**Q**: There are several ways in Python that we use when processing and using string data structures. What are the most important of these:  
**A**:

* Arithmetic syntax (**+**, **\*** and **=**),
* **%** operator formatting,
* **string.format()** method,
* **f-string** formatting.

**Aritmetik Sözdizimi ile Dize Biçimlendirme**  
Python'da string veri yapılarını işlerken ve kullanırken kullandığımız birkaç yol vardır. Bunlardan en önemlileri:  
  
Aritmetik sözdizimi (+, \* ve =),  
  
% operatör biçimlendirmesi,  
  
String.format() yöntemi,  
  
F-string biçimlendirme.  
  
Size işlevin ne olduğunu belirttik? Önceki derslerde. Bu noktada yöntem teriminin tanımını verelim. Bir yöntem, bir nesneye eklenmiş olması dışında bir işlev gibidir. Bir nesne üzerinde bir yöntem çağırırız ve muhtemelen o nesnede değişiklikler yapar (string.format() gibi). O halde bir metot bir sınıfa aittir.  
  
Aritmetik sözdizimi (+, =, \*):  
İki dizeyi boşluk bırakmadan birleştirmek için + operatörünü kullanabiliriz. Örneğin :  
  
Giriş:  
  
Print('clarus' + 'yol')  
Çıktı :  
  
Clarusway  
\* operatörünü de boşluk bırakmadan dizeyi tekrarlamak için kullanabiliriz. Örneğin :  
  
Giriş:  
  
Print(3\*'olamaz!')  
Çıktı :  
  
Olmaz! Olmaz! Olmaz!  
Aşağıdaki örneği dikkatlice inceleyin:  
  
Giriş:  
  
Meyve = 'Turuncu'  
Sebze = 'Domates'  
Print('+ kullanarak:', meyve + sebze)  
Print('\* kullanarak:', 3 \* meyve)  
Çıktı :  
  
+ kullanarak: PortakalDomates  
\* kullanarak: TuruncuTuruncuTuruncu  
Sayısal türlerde olduğu gibi, yerinde toplama işlemini ya string türü ile 👉🏻+= kullanarak yapabiliriz. Aşağıdaki örneklere bakın:  
  
Giriş:  
  
Meyve = 'portakal'  
Meyve += 'elma'  
  
Baskı(meyve)  
Çıktı :  
  
Portakal elma  
Giriş:  
  
Meyve = 'portakal'  
Meyve += 'elma'  
Meyve += 'muz'  
Meyve += 'kayısı'  
  
Baskı(meyve)  
Çıktı :  
  
Portakal elma muz kayısı  
S: Python'da string veri yapılarını işlerken ve kullanırken kullandığımız birkaç yol vardır. Bunlardan en önemlileri nelerdir:  
A:  
  
Aritmetik sözdizimi (+, \* ve =),  
% operatör biçimlendirmesi,  
String.format() yöntemi,  
F-string biçimlendirme.

**String Formatting with '%' Operator**

The other way that you will learn to format the strings is % operator. This one is not a frequently used way, but it's worth learning.

**' % '** operator formatting :

👉🏻% operator gets the values in order and prints them in order using several characters accordingly. Look at the example :

For now, we used only s, d and f characters to specify the data type in a string.

input :

phrase = 'I have %d %s and %.2f brothers' % (4, "children", 5)

print (phrase)

output :

I have 4 children and 5.00 brothers

Here in the example, the % operator first takes '4' and puts it in the first % operator, then takes 'children' secondly and puts it in the second % operator and finally takes '5' and puts it in the third % operator.

**💡Tips:**

* In the '%s' syntax : s stands for 'string'.
* In the '%.2f' syntax : f stands for 'float'. In this example 2 digits after point.
* In the '%d' syntax : d stands for 'numeric'.

If you want, you can limit the character numbers of the strings. Here is an example :

input :

sentence = "apologizing is a virtue"

print("%.11s" % sentence) # we get first 11 characters of the string

output :

apologizing

You can also use variables with 👉🏻% operator to format the string. Let's look at the example :

input :

print('%(amount)d pounds of %(fruit)s left' % {'amount': 33, 'fruit':'bananas'})

output :

33 pounds of bananas left

In this example, we used two variables which are amount and fruit. If you noticed, we assign values to variables in curly braces **'{}'**. This format is a **dictionary** type that you will learn in the next lessons.

**Q**: What is the output of print('%.5s' % x) if x = "HelloWorld!" ?  
**A**: Hello

**'%' Operatörü ile Dize Biçimlendirme**

Dizeleri biçimlendirmeyi öğreneceğiniz diğer yol ise % operatörüdür. Bu sık kullanılan bir yol değil ama öğrenmeye değer.  
  
' % ' operatör biçimlendirmesi :  
👉🏻% operatörü, değerleri sırayla alır ve buna göre birkaç karakter kullanarak sırayla yazdırır. Örneğe bakınız :  
  
Şimdilik bir dizgedeki veri tipini belirtmek için sadece s, d ve f karakterlerini kullandık.  
  
Giriş:  
  
Ifade = '%d %s ve %.2f erkek kardeşim var' % (4, 'çocuklar', 5)  
Yazdır (ifade)  
Çıktı :  
  
4 çocuğum ve 5.00 erkek kardeşim var  
Burada örnekte, % operatörü önce '4'ü alır ve ilk % operatörüne koyar, sonra ikinci olarak 'çocukları' alır ve ikinci % operatörüne koyar ve son olarak '5' alır ve üçüncü % operatörüne koyar. .  
  
💡İpuçları:  
'%s' sözdiziminde : s, 'dize' anlamına gelir.  
'%.2f' sözdiziminde : f 'kayan' anlamına gelir. Bu örnekte noktadan sonra 2 basamak.  
'%d' sözdiziminde : d 'sayısal' anlamına gelir.  
İsterseniz dizelerin karakter numaralarını sınırlayabilirsiniz. İşte bir örnek :  
  
Giriş:  
  
Cümle = 'özür dilemek bir erdemdir'  
  
Print('%.11s' % cümle) # stringin ilk 11 karakterini alıyoruz  
Çıktı :  
  
Özür dilemek  
Dizeyi biçimlendirmek için 👉🏻% operatörüne sahip değişkenleri de kullanabilirsiniz. Örneğe bakalım:  
  
Giriş:  
  
Print('%(amount)d pound %(meyve)s kaldı' % {'amount': 33, 'meyve':'muz'})  
Çıktı :  
  
33 kilo muz kaldı  
Bu örnekte miktar ve meyve olmak üzere iki değişken kullandık. Fark ettiyseniz, küme parantezleri '{}' içindeki değişkenlere değerler atarız. Bu format sonraki derslerde öğreneceğiniz bir sözlük türüdür.  
  
S: x = 'MerhabaDünya!' ise print('%.5s' % x) çıktısı nedir? ?  
Bir merhaba

**String Formatting with 'string.format()' Method**

You can make strings change depending on the value of a variable or an expression. The main methods of Python to format the output are :

**'string.format()'** method :

string.format() method is the improved form of % operator formatting.

As in this example below, the value of expression comes from .format() method in order. Curly braces 👉🏻{} receives values from .format().

input :

fruit = 'Orange'

vegetable = 'Tomato'

amount = 4

print('The amount of {} we bought is {} pounds'.format(fruit, amount))

output :

The amount of Orange we bought is 4 pounds

If you’ve written more variables than you need in the .format() method, the extra ones just will be ignored. Using keywords in 👉🏻{} makes string more readable. For example:

input :

print('{state} is the most {adjective} state of the {country}'.format(state='California', country='USA', adjective='crowded'))

output :

California is the most crowded state of the USA

**💡Tips:**

* If you have noticed, we do not have to write the keywords in .format() method in order.

There is no limit in Python language! You can combine both positional and keyword arguments in the same .format() method :

At this point, let us give you some explanations : **Positional arguments** are arguments that can be called by their position in the function or method definition. **Keyword arguments** are arguments that can be called by their names.

input :

print('{0} is the most {adjective} state of the {country}'.format('California', country='USA', adjective='crowded'))

output :

California is the most crowded state of the USA

You can use the same variable in a string more than once if you need it. Also, you can select the objects by referring to their positions in brackets.

input :

print("{6} {0} {5} {3} {4} {1} {2}".format('have', 6, 'months', 'a job', 'in', 'found', 'I will'))

output :

I will have found a job in 6 months

**⚠️Avoid ! :**

* Be careful not to write keyword arguments before positional arguments.

Using str.format() method is much more readable and useful than using %-operator formatting in our codes, but str.format() method can still be too wordy if you are dealing with multiple parameters and longer strings. At this point, the f-string formatting which you will learn in the next lesson suffices.

**'string.format()' Yöntemiyle Dize Biçimlendirme**

Bir değişkenin veya bir ifadenin değerine bağlı olarak dizelerin değişmesini sağlayabilirsiniz. Çıktıyı biçimlendirmek için Python'un ana yöntemleri şunlardır:  
  
'string.format()' yöntemi:  
String.format() yöntemi, % operatör biçimlendirmesinin geliştirilmiş biçimidir.  
  
Aşağıdaki bu örnekte olduğu gibi, ifadenin değeri sırayla .format() yönteminden gelir. Kıvrımlı parantezler 👉🏻{} değerleri .format()'tan alır.  
  
Giriş:  
  
Meyve = 'Turuncu'  
Sebze = 'Domates'  
Miktar = 4  
Print('Satın aldığımız {} miktarı {} pound'.format(meyve, miktar))  
Çıktı :  
  
Aldığımız portakal miktarı 4 pound  
.format() yönteminde ihtiyacınız olandan daha fazla değişken yazdıysanız, fazla olanlar yok sayılır. 👉🏻{} içinde anahtar kelimeler kullanmak, dizeyi daha okunaklı hale getirir. Örneğin:  
  
Giriş:  
  
Print('{state}, {country}'in en {sıfat} eyaletidir.format(state='California', country='ABD', sıfat='kalabalık'))  
Çıktı :  
  
Kaliforniya, ABD'nin en kalabalık eyaletidir  
💡İpuçları:  
Dikkat ettiyseniz .format() yöntemindeki anahtar kelimeleri sırayla yazmamıza gerek yok.  
Python dilinde sınır yoktur! Hem konumsal hem de anahtar sözcük bağımsız değişkenlerini aynı .format() yönteminde birleştirebilirsiniz:  
  
Bu noktada size bazı açıklamalar yapalım: Konumsal argümanlar, fonksiyon veya metot tanımındaki konumlarına göre çağrılabilen argümanlardır. Anahtar kelime argümanları, isimleriyle çağrılabilen argümanlardır.  
  
Giriş:  
  
Print('{0}, {country}'nin en {sıfat} eyaletidir.'.format('California', country='ABD', sıfat='kalabalık'))  
Çıktı :  
  
Kaliforniya, ABD'nin en kalabalık eyaletidir  
Gerekirse aynı değişkeni bir dizgede birden fazla kullanabilirsiniz. Ayrıca nesneleri parantez içindeki konumlarına bakarak da seçebilirsiniz.  
  
Giriş:  
  
Print('{6} {0} {5} {3} {4} {1} {2}'.format('var', 6, 'ay', 'bir iş', 'in', 'bulundu' , 'NS'))  
Çıktı :  
  
6 ay sonra iş bulmuş olacağım  
⚠️ Kaçının ! :  
Konumsal argümanlardan önce anahtar kelime argümanları yazmamaya dikkat edin.  
Str.format() yöntemini kullanmak, kodlarımızda %-operatör biçimlendirmesini kullanmaktan çok daha okunabilir ve kullanışlıdır, ancak birden fazla parametre ve daha uzun dizelerle uğraşıyorsanız str.format() yöntemi yine de çok endişeli olabilir. Bu noktada bir sonraki derste öğreneceğiniz f-string biçimlendirmesi yeterlidir.

**String Formatting with 'f-string'**

It is the easiest and useful formatting method of the strings.

**'f-string'** formatting :

It makes string formatting easier. This method was introduced in 2015 with Python 3.6.

**f-string** is the string syntax that is enclosed in quotes with a letter **f** at the beginning. Curly braces 👉🏻{} that contain variable names or expressions are used to replace with their values.

**Sample of a formula syntax is : f"strings {variable1} {variable2} string {variable3}"**

Let's look at the example below on how the syntax is simple and readable.

input :

fruit = 'Orange'

vegetable = 'Tomato'

amount = 6

output = f"The amount of {fruit} and {vegetable} we bought are totally {amount} pounds"

print(output)

output :

The amount of Orange and Tomato we bought are totally 6 pounds

You can use all valid expressions, variables, and even methods in curly braces. Look at the examples:

input :

result = f"{4 \* 5}"

print(result)

output :

20

input :

my\_name = 'JOSEPH'

output = f"My name is {my\_name.capitalize()}"

print(output)

output :

My name is Joseph

There is also a multiline f-string formatting style. Follow the example :

input :

name = "Joseph"

job = "teachers"

domain = "Data Science"

message = (

f"Hi {name}. "

f"You are one of the {job} "

f"in the {domain} section."

)

print(message)

output :

Hi Joseph. You are one of the teachers in the Data Science section.

If you want to use multiple f-string formatting lines without parentheses, you will have the other option that you can use backslash 👉🏻**\** between lines.

input :

name = "Joseph"

job = "teachers"

domain = "Data Science"

message = f"Hi {name}. " \

f"You are one of the {job} " \

f"in the {domain} section."

print(message)

output :

Hi Joseph. You are one of the teachers in the Data Science section.

**Q**: If you want to use multiple 'f-string formatting' lines without parentheses, what will be the other option that you can use?  
**A**: You can use backslashes 👉\ between f-lines.

**'f-string' ile Dize Biçimlendirme**

Dizelerin en kolay ve kullanışlı biçimlendirme yöntemidir.  
  
'f-string' biçimlendirmesi:  
Dize biçimlendirmeyi kolaylaştırır. Bu yöntem, 2015 yılında Python 3.6 ile tanıtıldı.  
  
F-string, başında f harfi ile tırnak içine alınmış dize sözdizimidir. Değişken adları veya ifadeleri içeren küme parantezleri 👉🏻{} değerleriyle değiştirmek için kullanılır.  
  
Formül sözdizimi örneği: f'dizeler {değişken1} {değişken2} dize {değişken3}'  
  
Sözdiziminin nasıl basit ve okunabilir olduğuna dair aşağıdaki örneğe bakalım.  
  
Giriş:  
  
Meyve = 'Turuncu'  
Sebze = 'Domates'  
Miktar = 6  
Çıktı = f'Aldığımız {meyve} ve {sebze} miktarı tamamen {amount} pound'  
  
Yazdır(çıktı)  
Çıktı :  
  
Aldığımız Portakal ve Domates miktarı toplam 6 lira.  
Tüm geçerli ifadeleri, değişkenleri ve hatta yöntemleri kaşlı ayraçlar içinde kullanabilirsiniz. Örneklere bak:  
  
Giriş:  
  
Sonuç = f'{4 \* 5}'  
  
Yazdır(sonuç)  
Çıktı :  
  
20  
Giriş:  
  
My\_name = 'JOSEPH'  
Çıktı = f'Benim adım {my\_name.capitalize()}'  
  
Yazdır(çıktı)  
Çıktı :  
  
Benim adım Joseph  
Ayrıca çok satırlı bir f-string biçimlendirme stili de vardır. Örneği takip edin :  
  
Giriş:  
  
Isim = 'Yusuf'  
Iş = 'öğretmenler'  
Alan = 'Veri Bilimi'  
Mesaj = (  
F'Merhaba {isim}.'  
F'Siz {iş}'den birisiniz'  
F'{domain} bölümünde.'  
)  
Yazdır(mesaj)  
Çıktı :  
  
Merhaba Yusuf. Veri Bilimi bölümündeki öğretmenlerden birisiniz.  
Parantez olmadan birden fazla f-string biçimlendirme satırı kullanmak istiyorsanız, satırlar arasında ters eğik çizgi 👉🏻 kullanabileceğiniz diğer seçeneğiniz olacaktır.  
  
Giriş:  
  
Isim = 'Yusuf'  
Iş = 'öğretmenler'  
Alan = 'Veri Bilimi'  
Mesaj = f'Merhaba {isim}. '  
F'Siz {iş}'den birisiniz '  
F'{domain} bölümünde.'  
  
Yazdır(mesaj)  
Çıktı :  
  
Merhaba Yusuf. Veri Bilimi bölümündeki öğretmenlerden birisiniz.  
S: Parantez olmadan birden çok 'f-string biçimlendirme' satırı kullanmak istiyorsanız, kullanabileceğiniz diğer seçenek ne olacak?  
C: f-çizgileri arasında ters eğik çizgi 👉 kullanabilirsiniz.

**Main String Operations**

String processing is of great importance in the world of Data Science as a field of research and business in the world. Moreover, there seems to be much more to do in this field.

Good news! Python's string processing skills are very advanced that we will focus on some main parts of it.

A significant thing to keep in mind is that string is an immutable data type. This means you can't just change the string in place, so most string methods return a copy of the string. Well, how can we do that?

**💡Tips:**

* This is the way you should follow: You must create a new variable for the copy you made or assign the same name to the copy to save the changes made to the string for later use

Dize işleme, dünyada bir araştırma ve iş alanı olarak Veri Bilimi dünyasında büyük önem taşımaktadır. Üstelik bu alanda yapılacak daha çok şey var gibi görünüyor.  
  
İyi haberler! Python'un dize işleme becerileri çok ileri düzeydedir ve biz onun bazı ana bölümlerine odaklanacağız.  
  
Akılda tutulması gereken önemli bir şey, dizenin değişmez bir veri türü olmasıdır. Bu, dizeyi yerinde değiştiremeyeceğiniz anlamına gelir, bu nedenle çoğu dize yöntemi dizenin bir kopyasını döndürür. Peki, bunu nasıl yapabiliriz?  
  
💡İpuçları:  
Yapmanız gereken yol şudur: Yaptığınız kopya için yeni bir değişken oluşturmanız veya daha sonra kullanmak üzere dizede yaptığınız değişiklikleri kaydetmek için kopyaya aynı adı atamanız gerekir.

**Searching a String**

To search patterns in a string there are two useful methods called startswith() and endswith() that search for the particular pattern in the immediate beginning or end of a string and return True if the expression is found. Here are some simple examples. Examine the basic syntax of those methods carefully.

input :

text = 'www.clarusway.com'

print(text.endswith('.com'))

print(text.startswith('http:'))

output :

True

False

input :

text = 'www.clarusway.com'

print(text.endswith('om'))

print(text.startswith('w'))

output :

True

True

These methods have optional arguments start and end. We can specify the search by adding arguments so that the area of search is delimited by start and end arguments.

**💡Tips:**

* Remember! Characters of string count from left to right and start with zero.

**The formula syntaxes are :**

* **string.startswith(prefix[, start[, end]])**
* **string.endswith(suffix[, start[, end]])**

Look at the example below:

input :

email = "clarusway@clarusway.com is my e-mail address"

print(email.startswith("@", 9))

print(email.endswith("-", 10, 32))

output :

True

True

**Q**: What are the string.startswith() and string.endswith() method used for? Describe how?  
**A**: To search patterns in a string there are two useful methods called startswith() and endswith() that search for the particular pattern in the immediate beginning or end of a string and return True if the expression is found.

**Dize Arama**

Bir dizedeki kalıpları aramak için, belirli bir kalıbı bir dizenin hemen başında veya sonunda arayan ve ifade bulunursa True döndüren startwith() ve endwith() adlı iki kullanışlı yöntem vardır. İşte bazı basit örnekler. Bu yöntemlerin temel sözdizimini dikkatlice inceleyin.  
  
Giriş:  
  
Metin = 'www.clarusway.com'  
Print(text.endswith('.com'))  
Print(text.startswith('http:'))  
Çıktı :  
  
NS  
YANLIŞ  
Giriş:  
  
Metin = 'www.clarusway.com'  
Print(text.endswith('om'))  
Print(text.startswith('w'))  
Çıktı :  
  
NS  
NS  
Bu yöntemlerin başlangıç ​​ve bitiş isteğe bağlı argümanları vardır. Argümanlar ekleyerek aramayı belirtebiliriz, böylece arama alanı başlangıç ​​ve bitiş argümanlarıyla sınırlandırılır.  
  
💡İpuçları:  
Unutma! Dize karakterleri soldan sağa sayılır ve sıfır ile başlar.  
Formül sözdizimleri şunlardır:  
  
String.startswith(prefix[, start[, end]])  
String.endswith(sonek[, start[, end]])  
Aşağıdaki örneğe bakın:  
  
Giriş:  
  
Email = 'clarusway@clarusway.com benim e-posta adresim'  
Print(email.startswith('@', 9))  
Print(email.endswith('-', 10, 32))  
Çıktı :  
  
NS  
NS  
S: string.startswith() ve string.endswith() yöntemi ne için kullanılır? Nasıl anlat?  
C: Bir dizgedeki kalıpları aramak için, belirli bir kalıbı bir dizgenin hemen başında veya sonunda arayan ve ifade bulunursa True döndüren startwith() ve endwith() adlı iki kullanışlı yöntem vardır.

**Changing a String**

The methods described below return the copy of the string with some changes made.

How does the following syntax work?

A string is given first (or the name of a variable that represents a string), then comes a period followed by the method name and parentheses in which arguments are listed.

**The formula syntax is : string.method()**

Let's examine some common and the most important methods of string changing :

* **str.replace(old, new[, count])** replaces all occurrences of old with the new.

The count argument is optional, and if the optional argument count is given, only the first count occurrences are replaced. count: Maximum number of occurrences to replace. **-1** (the default value) means replace all occurrences.

* **str.swapcase()** converts upper case to lower case and vice versa.
* **str.capitalize()** changes the first character of the string to the upper case and the rest to the lower case.
* **str.upper()** converts all characters of the string to the upper case.
* **str.lower()** converts all characters of the string to the lower case.
* **str.title()** converts the first character of each word to upper case.

Let's consolidate the subject through examples :

input :

sentence = "I live and work in Virginia"

print(sentence.upper())

print(sentence.lower())

print(sentence.swapcase())

print(sentence) # note that, source text is unchanged

output :

I LIVE AND WORK IN VIRGINIA

i live and work in virginia

i LIVE AND WORK IN vIRGINIA

I live and work in Virginia

If we assign the modified text to a new variable, we can have a new string. Consider these :

input :

sentence = "I live and work in Virginia"

title\_sentence = sentence.title()

print(title\_sentence)

changed\_sentence = sentence.replace("i", "+")

print(changed\_sentence)

print(sentence) # note that, again source text is unchanged

output :

I Live And Work In Virginia

I l+ve and work +n V+rg+n+a

I live and work in Virginia

input :

sentence = "I live and work in Virginia"

swap\_case = sentence.swapcase()

print(swap\_case)

print(swap\_case.capitalize()) # changes 'i' to uppercase and

# the rest to lowercase

output :

i LIVE AND WORK IN vIRGINIA

I live and work in virginia

**Q**: print("Actions speaks louder than words".upper().swapcase().capitalize()), will this code work? If yes, what the output will be? Describe how?  
**A**: Yes it works. The syntax is : string.method(). Changing the string using these methods returns string type again. The output is :  
Actions speaks louder than words  
  
Follow the additional examples below :  
  
string.upper() # returns string type,  
string.upper().lower() # also returns string type,  
string.upper().lower().title() # returns string type again.

**Dize Değiştirme**Aşağıda açıklanan yöntemler, yapılan bazı değişikliklerle dizenin kopyasını döndürür.  
  
Aşağıdaki sözdizimi nasıl çalışır?  
  
Önce bir dize (veya bir dizeyi temsil eden bir değişkenin adı) verilir, ardından bir nokta ve ardından yöntem adı ve argümanların listelendiği parantezler gelir.  
  
Formül sözdizimi: string.method()  
  
Dize değiştirmenin bazı yaygın ve en önemli yöntemlerini inceleyelim:  
  
Str.replace(eski, yeni[, sayım]) eskinin tüm oluşumlarını yeniyle değiştirir.  
Sayma bağımsız değişkeni isteğe bağlıdır ve isteğe bağlı bağımsız değişken sayısı verilirse, yalnızca ilk sayım oluşumları değiştirilir. Count: Değiştirilecek maksimum oluşum sayısı. -1 (varsayılan değer), tüm oluşumları değiştir anlamına gelir.  
  
Str.swapcase() büyük harfi küçük harfe çevirir ve bunun tersi de geçerlidir.  
  
Str.capitalize(), dizenin ilk karakterini büyük harfe ve geri kalanını küçük harfe değiştirir.  
  
Str.upper(), dizenin tüm karakterlerini büyük harfe dönüştürür.  
  
Str.lower(), dizenin tüm karakterlerini küçük harfe dönüştürür.  
  
Str.title() her kelimenin ilk karakterini büyük harfe dönüştürür.  
  
Konuyu örneklerle pekiştirelim:  
  
Giriş:  
  
Cümle = 'Virginia'da yaşıyorum ve çalışıyorum'  
  
Yazdır(cümle.üst())  
  
Print(cümle.lower())  
  
Print(sentence.swapcase())  
  
Print(cümle) # unutmayın, kaynak metin değişmedi  
Çıktı :  
  
VİRGİNİA'DA YAŞIYOR VE ÇALIŞIYORUM  
Virginia'da yaşıyorum ve çalışıyorum  
VIRGINIA'DA YAŞIYOR VE ÇALIŞIYORUM  
Virginia'da yaşıyorum ve çalışıyorum  
Değiştirilen metni yeni bir değişkene atarsak, yeni bir dizeye sahip olabiliriz. Bunları göz önünde bulundurun:  
  
Giriş:  
  
Cümle = 'Virginia'da yaşıyorum ve çalışıyorum'  
Title\_sentence = cümle.title()  
Yazdır(başlık\_sentence)  
  
Change\_sentence = cümle.replace('i', '+')  
Yazdır(değiştirilen\_cümle)  
  
Print(cümle) # yine kaynak metnin değişmediğine dikkat edin  
Çıktı :  
  
Virginia'da Yaşıyorum ve Çalışıyorum  
Çalışıyorum ve çalışıyorum +n V+rg+n+a  
Virginia'da yaşıyorum ve çalışıyorum  
Giriş:  
  
Cümle = 'Virginia'da yaşıyorum ve çalışıyorum'  
Swap\_case = cümle.swapcase()  
Print(swap\_case)  
Print(swap\_case.capitalize()) # 'i'yi büyük harfe çevirir ve  
# geri kalanı küçük harfe  
Çıktı :  
  
VIRGINIA'DA YAŞIYOR VE ÇALIŞIYORUM  
Ben virginia'da yaşıyorum ve çalışıyorum  
S: print('Eylemler kelimelerden daha yüksek sesle konuşur'.upper().swapcase().capitalize()), bu kod işe yarayacak mı? Evet ise, çıktı ne olacak? Nasıl anlat?  
C: Evet çalışıyor. Sözdizimi şöyledir: string.method(). Bu yöntemleri kullanarak dizeyi değiştirmek, yeniden dize türünü döndürür. Çıktı:  
Eylemler kelimelerden daha yüksek sesle konuşur  
  
Aşağıdaki ek örnekleri izleyin:  
  
String.upper() # string tipini döndürür,  
String.upper().lower() # ayrıca string türünü de döndürür,  
String.upper().lower().title() #, tekrar string tipini döndürür.

**Editing a String**

The methods described below remove the trailing characters (i.e. characters from the right side). The default for the argument chars is also whitespace. If the argument chars aren’t specified, trailing whitespaces are removed.

**The formula syntax is : string.method()**

* **str.strip()** : removes all spaces (or specified characters) from both sides.
* **str.rstrip()** : removes spaces (or specified characters) from the right side.
* **str.lstrip()** : removes spaces (or specified characters) from the left side.

Now see the examples about how we implement these methods? :

input :

space\_string = " listen first "

print(space\_string.strip()) # removes all spaces from both sides

output :

listen first

input :

source\_string = "interoperability"

print(source\_string.strip("yi"))

# removes trailing "y" or "i" or "yi" or "iy" from both sides

output :

nteroperabilit

input :

source\_string = "interoperability"

print(source\_string.lstrip("in"))

# removes "i" or "n" or "in" or "ni" from the left side

output :

teroperability

input :

space\_string = " listen first "

print(space\_string.rstrip()) # removes spaces from the right side

output :

listen first

input :

source\_string = "interoperability"

print(source\_string.rstrip("yt"))

# removes "y" or "t" or "yt" or "ty" from the right side

output :

interoperabili

As we said before, Python's string processing capability is very good. In this context, many other string processing and editing methods are available.

If you want to examine them in detail, you can find resources [**here**](https://docs.python.org/3/library/stdtypes.html#string-methods).

**Dize Düzenleme**

Aşağıda açıklanan yöntemler, sondaki karakterleri kaldırır (yani, sağ taraftaki karakterler). Argüman karakterleri için varsayılan da boşluktur. Argüman karakterleri belirtilmezse, sondaki boşluklar kaldırılır.  
  
Formül sözdizimi: string.method()  
  
Str.strip() : her iki taraftaki tüm boşlukları (veya belirtilen karakterleri) kaldırır.  
  
Str.rstrip() : sağ taraftaki boşlukları (veya belirtilen karakterleri) kaldırır.  
  
Str.lstrip() : sol taraftaki boşlukları (veya belirtilen karakterleri) kaldırır.  
  
Şimdi bu yöntemleri nasıl uyguladığımızla ilgili örneklere bakın? :  
  
Giriş:  
  
Space\_string = 'önce dinleyin'  
Print(space\_string.strip()) # her iki taraftaki tüm boşlukları kaldırır  
Çıktı :  
  
Önce dinle  
Giriş:  
  
Source\_string = 'birlikte çalışabilirlik'  
Print(source\_string.strip('yi'))  
# sondaki 'y' veya 'i' veya 'yi' veya 'iy'yi her iki taraftan da kaldırır  
Çıktı :  
  
Birlikte çalışabilirlik  
Giriş:  
  
Source\_string = 'birlikte çalışabilirlik'  
Print(source\_string.lstrip('in'))  
# sol taraftan 'i' veya 'n' veya 'in' veya 'ni' yi kaldırır  
Çıktı :  
  
Birlikte çalışabilirlik  
Giriş:  
  
Space\_string = 'önce dinleyin'  
Print(space\_string.rstrip()) # sağ taraftaki boşlukları kaldırır  
Çıktı :  
  
Önce dinle  
Giriş:  
  
Source\_string = 'birlikte çalışabilirlik'  
Print(source\_string.rstrip('yt'))  
# 'y' veya 't' veya 'yt' veya 'ty' yi sağ taraftan kaldırır  
Çıktı :  
  
Birlikte çalışabilirlik  
Daha önce de söylediğimiz gibi Python'un string işleme yeteneği çok iyidir. Bu bağlamda daha birçok dizi işleme ve düzenleme yöntemi mevcuttur.  
  
Bunları detaylı incelemek isterseniz buradan kaynaklara ulaşabilirsiniz.