**PYTHON**

### [Introduction to Python](https://lms.clarusway.com/course/view.php?id=2&section=1)

### What is Python?

As you can imagine, Python is a programming language (such as Java, C ++, R, Ruby, and so on). Like other programming languages, it allows you to control the machine in front of you, the computer.

The IT industry is booming with Data science applications using,

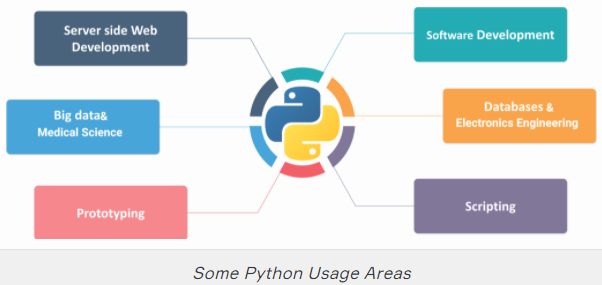
* Artificial intelligence,
* Deep learning and
* Machine learning algorithms.

**Python is the most widely used technologies** in this domain. With the new-age applications, demand for a Python developer has also increased.

\* According to the [Google search trends](https://trends.google.com/trends/explore?q=%2Fm%2F05z1_,%2Fm%2F07sbkfb,%2Fm%2F02p97,%2Fm%2F0jgqg), in 2019, Python is the most popular searched term, among all programming languages.

\* According to Tiobe.com's index of programming languages, Python is the fastest-growing language.

\* Python has become the second-most-popular language in GitHub, overtaking Java for the first time, according to GitHub’s 2019 State of the [Octoverse report](https://octoverse.github.com/) on the usage of the popular code-sharing site.



Programs can be developed very quickly with this language. In addition, the simple and clean syntax of the Python programming language has made it a preferred language by many programmers. It's easy to write programs and read a program written by others. So, it has been widely used - especially in Data Science - and has received lots of demands in recent years.

### Programming with Python

Software quality is a vital ingredient to success in industry and science. Ubiquitous IT systems control the business processes of the global economy. Increasingly powerful computers and sophisticated algorithms provide the platform for new scientific discoveries. And global communication is inconceivable without intelligent software. In the race for customers, the pole position belongs to those who get to market faster than their competitors. Better and more creative solutions combined with the ability to respond instantly to new challenges drive the race. Writing secure and reliable programs in a fraction of the time normally required gets you first across the finish line.

Compared with other modern programming languages such as Java or C, Python achieves superior results in significantly shorter timescales for a number of different reasons.

For example, **Python is a very lean programming language. Python programs are a great deal shorter than code written in other modern programming languages. As a result, both development times and maintenance costs are drastically reduced. Less code means fewer errors, meaning the cost of identifying and eliminating these errors is also reduced.**

**A comprehensive standard library and thousands of additional libraries in the Python Package Index provide developers with high-quality solutions that they can easily integrate into their applications to meet virtually any requirement**.

In this way, Python frees up vast resources, which can be earmarked for more productive use elsewhere.

Python offers unique benefits for system integration. On the one hand, there are huge numbers of Python libraries, with which virtually any third-party system can be integrated. On the other hand, the libraries of many other programming languages can also be used in Python.

**Once they have been programmed, Python applications can run on all operating systems for which a Python interpreter exists, significantly reducing the cost of operating-system-specific applications.**

The Python programming language is easy to learn. It has blurred the boundaries between users and developers. Increasing numbers of scientists, engineers, financial experts, and others with little programming experience are using Python to solve specific complex technical problems.

### Historical Development of Python

This programming language was developed in the early 90s by a Dutch programmer called Guido van Rossum. Most people think that this programming language is named after the python snake, assuming its name is Python.

However, contrary to the assumption, the name of this programming language does not come from the python snake. Guido van Rossum named this programming language inspired by the show of **Monty Python’s Flying Circus** from an English comedy group called **The Monty Python**.

This language has a huge group of developers around the world. If you have any problem, you can always ask other Python users/developers for help or find a suitable answer on several sites like [**stackoverflow.com**](https://stackoverflow.com/)

## First Steps into Coding

* Surrounding the expression with triple quotes: **"""..."""** or **'''...'''** ensures that the code returns no error, especially in long texts.
* Use double quotes if your string includes the single one:e.g. "It's my pleasure!"
* Use single quotes if your string includes the double one:e.g. 'He said: "I am done" and fell down.'
* Use triple quotes if your string is too long which composed of multiple lines : e.g. :
* If you have noticed we used 👉🏻**.** not 👉🏻**,** for the decimal number: 3.14.
* Surrounding the expression with quotes makes it in string type. We will see immutable types (int, string, tuple, etc.) in the next lessons.
* If you need an empty line, you can use only print() function.

### [Format & Style of Coding](https://lms.clarusway.com/course/view.php?id=2&section=2)

## PEP 8 Conventions

### What is PEP 8 ?

**PEP** stands for Python Enhancement Proposal. PEP 8 is a coding convention, a set of recommendations, about how to write your Python code more readable.

In other words, PEP 8 is a document that gives coding conventions for the Python code comprising the standard library in the main Python distribution. One of Guido's (author of Python) key insights is that code is read much more often than it is written.

The guidelines (PEP 8) provided here are intended to improve the readability of code and make it consistent across the wide spectrum of Python code.

However, know when to be inconsistent. Sometimes style guide recommendations aren't just applicable. When in doubt, use your best judgment. Look at other examples and decide what looks best. And don't hesitate to ask someone else.

The main idea of PEP 8 is to use the same code style for all Python projects as if they were written by the same programmer. PEP 8, even for beginners, assures that it will easily understand the code written by any other developer.

### Some Important PEP 8 Rules

* Limit all lines to a maximum of **79 characters**. For flowing long blocks of text with fewer structural restrictions (docstrings or comments), the line length should be limited to **72 characters**. During this course, we will learn some ways of reducing the length of lines.
* **Spaces**are the preferred indentation method. **Tabs** should be used solely to remain consistent with code that is already indented with tabs. Python 3 disallows mixing the use of tabs and spaces for indentation.
* Avoid extraneous **whitespaces** in the following situations:
* Immediately inside parentheses, brackets or braces :



* Between a trailing comma and a following close parenthesis :



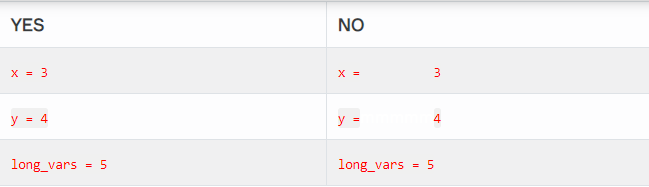
* Immediately before a comma, semicolon, or colon :



* Immediately before the open parenthesis that starts the argument list of a function call:



* More than one space around an assignment (or other) operator to align it with another:



* Avoid trailing whitespace anywhere. Because it's usually invisible, it can be confusing: e.g. a backslash followed by a space and a newline does not count as a line continuation marker.
* Always surround these binary operators with a single space on either side: assignment (=), augmented assignment (+=, -=, etc.), comparisons (==, <, >, !=, <>, <=, >=, in, not in, is, is not), Booleans (and, or, not).

Failure to follow the basic rules of PEP 8 does not make your program wrong or unable to work. In the near future, you will learn a lot about Python and become a more skilled programmer, but it will always be important to follow the code style.

There's nothing to worry about following PEP 8. You don't need to learn the traditional PEP 8 rules all at once right away. When you need it, you can open and read it now and then. We will also show you some PEP 8 conventions throughout this course.

### Comments and Docstrings

### Docstrings

We have to say at the beginning that you will not learn to create and write docstrings in this course. Only what we will show you is what docstrings are and how we will call and display it.

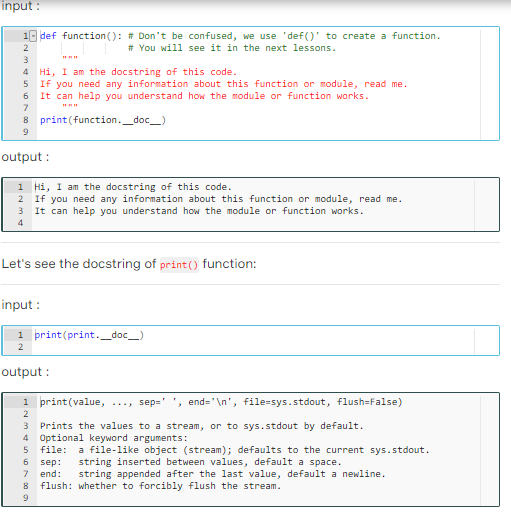
**Docstrings** are - unlike regular comments - stored as an attribute of the function or the module they document, meaning that you can access them programmatically. Docstring runs as an explanatory text of codes and it should be written between triple quotes. Like: """docstring""".

### Some Definitions

A **function** is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing. As you already know, Python gives you many built-in functions like print(), etc. but you can also create your own functions.

A **module** is a Python object with arbitrarily named attributes that you can bind and reference. Simply, a module is a file consisting of Python code. A module can define functions, classes, and variables. A module can also include runnable code.

Normally, when we want to call docstring of a function or module to read, we will use **\_\_doc\_\_** (the keyword doc enclosed by double underscores) syntax. See the example below :



## Naming Variables

### General Description

As you know, each variable has a unique name that distinguishes it from others. Giving a good name to a variable may not be as simple as it sounds.

A Python **variable** is a reserved memory location to store values. In other words, a variable gives data to the computer for processing. We will discuss variables in detail in the next lessons.

Remember, a nice and meaningful naming of variables is a skill that can be gained over time. Of course, you also need to be familiar with PEP 8 traditional rules.

If variables have poor names, even your own code may seem unclear to you in a couple of months. Now let's learn how to choose good names for our variables in accordance with PEP 8 rules:

* Choose lowercase words and use underscore to split the words:

**price = [22, 44, 66]**,

**low\_price = 12.00**

* Do not use the characters 'l' (lowercase letter el), 'O' (uppercase letter oh), or 'I' (uppercase letter eye) as single-character variable names. In some fonts, these characters are indistinguishable from the numerals one and zero. If you want to use 'l', use 'L' instead.

**l = 'It is not correct use'**,

**O = "It's also incorrect use"**

* Do not use specific Python keywords (name of a function or phrase) as a name, like sum, max, min, in, or, for, etc.
* Use a sensible name. The variable name needs to be legible and meaningful and explain to the reader what types of values will be stored in it.

**figures = 'this is better'**,

**f = 'it is not meaningful'**

* Don't choose too common names. Use a name to describe the meaning of the variable. However, try to limit it to no more than 3 words.
* If the word you intend to choose is long, try to find the most common and expected short form to make it easy to predict later.

### [Data Types and Useful Operations](https://lms.clarusway.com/course/view.php?id=2&section=3)

## Basic Data Types

### Introduction to Data Types

Each data has a type, whether constant or variable. This type of data defines how you store it in memory and it also describes which process can be applied to it.

In fact, data types are nothing but variables you use to reserve some space in memory. Python variables do not need an explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable.

* String, str
* Signed Integer, int
* Floating Point, float
* Complex,
* Boolean. bool

Strings are immutable sequence data type, i.e each time one makes any changes to a string, a completely new string object is created. You will be able to better understand **immutability** with examples that will continue in the next lessons.

### Numeric Types

For any programmer, using numbers is the most important issue. You can hardly write a serious program without using numbers, so let's talk about some basic numeric types. There are three distinct numeric types: **signed integer numbers**, **floating point numbers** and **complex numbers**.

* **Signed Integer** type is called int, they are whole numbers (positive, negative or zero), including no decimal point. For example: 71, -122, 0
* **Floating point** type is called float and they stand for real numbers with a decimal point. For example: 71.0, -33.03
* **Complex** type is called complex and they are written in the form, **x + yj** , where x is the real part and y is the imaginary part. For example: 3.14j. Imaginary numbers, also called complex numbers, are used in real-life applications, such as electricity, as well as quadratic equations. In quadratic planes, imaginary numbers show up in equations that don't touch the x-axis. Imaginary numbers become particularly useful in advanced calculus. We will not use this type much.

### Boolean

Boolean types are called bool and their values are the two constant objects **False** and **True**. They are used to represent truth values (other values can also be considered false or true). In numeric contexts (for example, when used as the argument to an arithmetic operator), they behave like the integers 0 and 1, respectively.

Bools are important data types that are widely used in Python as they can find use in every aspect of our daily lives. For example, imagine, whether the TV is turned on or off in your home or if the weather is rainy can be explained easily with bools.

Bools are mostly used in **conditional operations** which we will discuss in the next lessons.

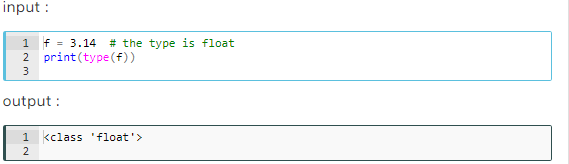
### Type Conversion

We can convert the types of data to each other if the type allows to be converted. There are some functions to convert the types:

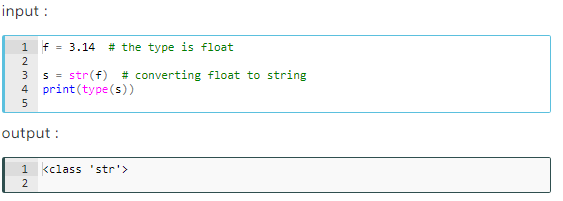
* str() converts to **string** type
* int() converts to **signed integer** type
* float() converts to **floating point** type

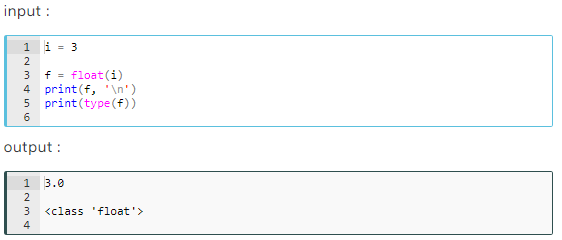
We can print the types of data using type() function.

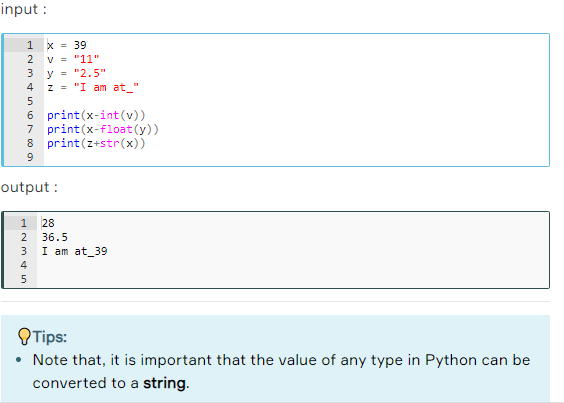
**Örnekler**

****

Note that we write the first letter of True in **uppercase**. This is the rule of Python that we must write like this : True, False.







### Variables

**Variable** is a location designated where a value can be stored and accessed later. Imagine a box where you store something. That's a variable.

Let's create a box (variable) in which contains basketball balls. Let's name it ball\_box. It is also the name of the variable.

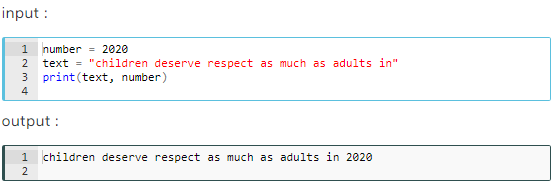
Creating, naming the variable and assigning a value to it happen simultaneously by this syntax : ball\_box = 20 basketball balls

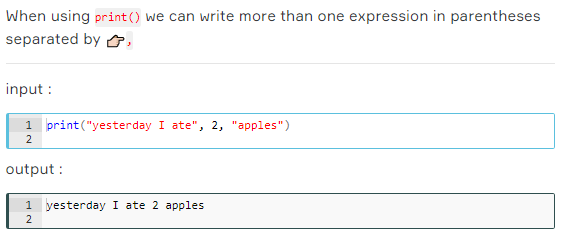
Python variables do not need an explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable.

## Simple Operations

### 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.





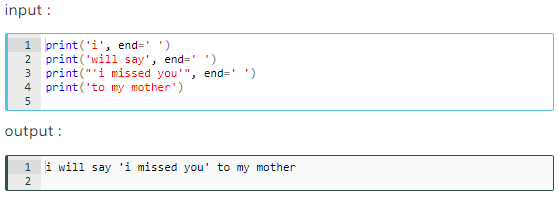
When you type more than one expression in print(), you notice that the expressions are joined to each other by spaces. This is due to the default value of keyword argument **sep** in the print() function. This argument, which is defined as a **space** 👉🏻" " by default, is not visible in the background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

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

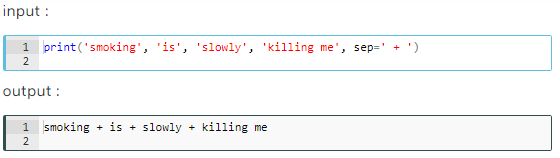
Here are the keyword arguments that run in the background of the print() function :

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

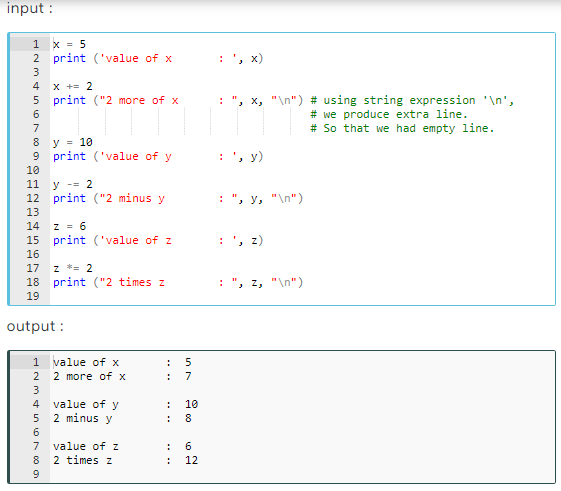
* 👉🏻\n represents next line.
* Let's focus on the arguments sep= and end= we discussed above:



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.

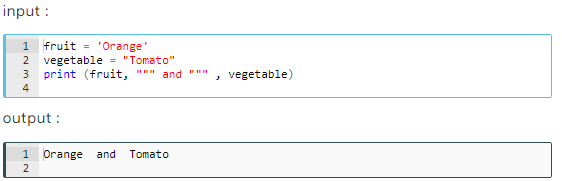


Some other useful operations that covers arithmetic and print() function are as follows. Carefully examine the examples:



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.

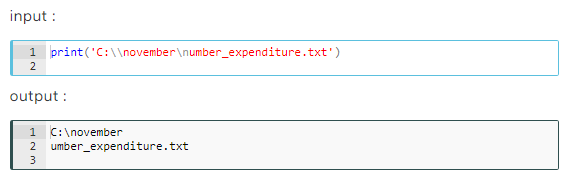


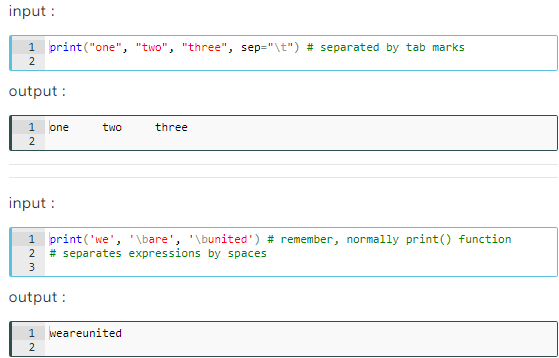
### Escape Sequences

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:

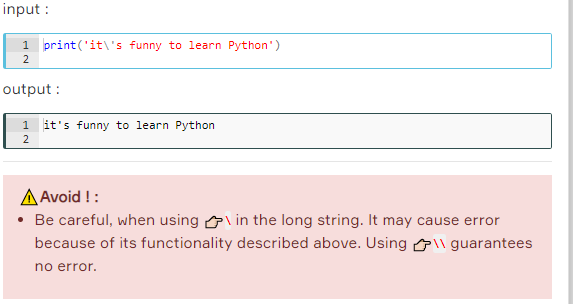
* **\n** : means new line,
* **\t** : means tab mark,
* **\b** : means backspace. It moves the cursor one character to the left.

Look at these examples carefully:





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.



* Use **double** quotes if your string includes **single** one,
* Use **single** quotes if string contains **double** one,
* Use **triple** quotes if string is too long and comprise of both single and double,
* Do not mix two quotes style in one string.
* Use . not , for decimal numbers ,
* Her satırı maksimum 79 karakter ile sınırla,
* Şu durumlarda gereksiz boşluklardan kaçınılmalı:

\* Parantezlerin, köşeli parantezlerin veya ayraçların içerisinde,

* \* Virgül ile kapanan köşeli veya yuvarlak parantez arasında,
* \* Virgül, noktalı virgül veya iki nokta üst üste’nin hemen öncesinde,
* \* Fonksiyondan sonra gelen parantezin hemen öncesinde,

We will show you some important PEP 8 traditional rules that you can follow.

* Limit all lines to a maximum of **79 characters**. For flowing long blocks of text with fewer structural restrictions (docstrings or comments), the line length should be limited to **72 characters**. During this course, we will learn some ways of reducing the length of lines.
* **Spaces**are the preferred indentation method. **Tabs** should be used solely to remain consistent with code that is already indented with tabs. Python 3 disallows mixing the use of tabs and spaces for indentation.
* Avoid extraneous **whitespaces** in the following situations:

Immediately inside parentheses, brackets or braces :

**YES** : spam(meat[1], {milk: 2}) , **NO** : spam( meat[ 1 ], { milk: 2 } )

Between a trailing comma and a following close parenthesis :

**YES** : df[0,] or foo = (2,) , **NO** : df[0, ] or foo = (2, )

Immediately before a comma, semicolon, or colon :

**YES** : if y == 3: print x, y; x, y = y, x , **NO** : if y == 3 : print x , y ; x , y = y , x

Immediately before the open parenthesis that starts the argument list of a function call:

**YES** : print('peace') , **NO** : print ('peace')

More than one space around an assignment (or other) operator to align it with another:

| **YES** | **NO** |
| --- | --- |
| x = 3 | x =mmmmm3 |
| y = 4 | y =mmmmm4 |
| long\_vars = 5 | long\_vars = 5 |

* Avoid trailing whitespace anywhere. Because it's usually invisible, it can be confusing: e.g. a backslash followed by a space and a newline does not count as a line continuation marker.
* Always surround these binary operators with a single space on either side: assignment (=), augmented assignment (+=, -=, etc.), comparisons (==, <, >, !=, <>, <=, >=, in, not in, is, is not), Booleans (and, or, not).

Naming Variables

Conventional (PEP 8) Naming Rules

If variables have poor names, even your own code may seem unclear to you in a couple of months. Now let's learn how to choose good names for our variables in accordance with PEP 8 rules:

* Choose lowercase words and use underscore to split the words:
* **price = [22, 44, 66]**,
* **low\_price = 12.00**
* Do not use the characters 'l' (lowercase letter el), 'O' (uppercase letter oh), or 'I' (uppercase letter eye) as single-character variable names. In some fonts, these characters are indistinguishable from the numerals one and zero. If you want to use 'l', use 'L' instead.
* **l = 'It is not correct use'**,
* **O = "It's also incorrect use"**

**⚠️Avoid ! :**

* Do not use specific Python keywords (name of a function or phrase) as a name, like sum, max, min, in, or, for, etc.
* Use a sensible name. The variable name needs to be legible and meaningful and explain to the reader what types of values will be stored in it.
* **figures = 'this is better'**,
* **f = 'it is not meaningful'**
* Don't choose too common names. Use a name to describe the meaning of the variable. However, try to limit it to no more than 3 words.
* If the word you intend to choose is long, try to find the most common and expected short form to make it easy to predict later.

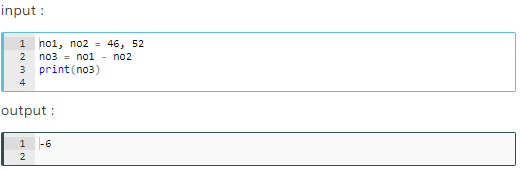
| **Variable to be named** | **Sample of Good name** | **Sample of Bad name** | **Why bad?** |
| --- | --- | --- | --- |
| Cleaned Data | cleaned\_data | cdat | it doesn't make sense enough. |
| Indexes of the Clear Application Syntaxes | clr\_app\_syntx | ix\_app\_syntax | it doesn't make sense enough. |
| Customer Information of the Bank Accounts | customer\_bank\_info | costomer\_info\_bank\_account | it's too wordy. |

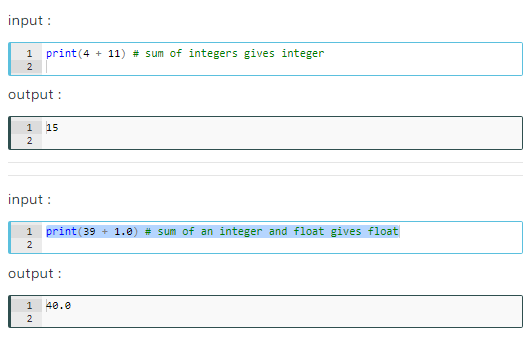
## Simple Operations

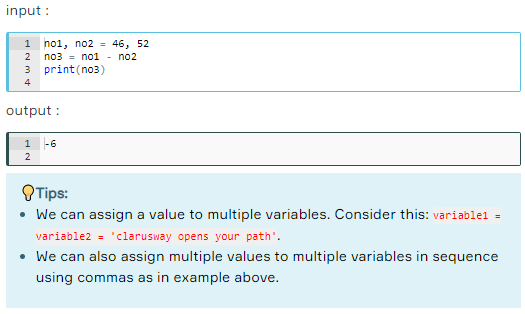
### Arithmetic Operations

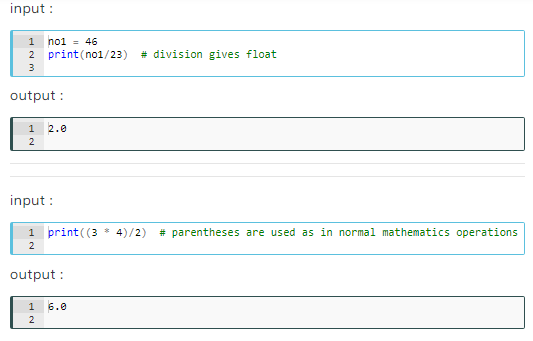
In Python, there are almost all of the arithmetic operations we use in mathematics. They are so simple to use and we can also use these operations on almost all data types, including string.

Basic Arithmetic operators are as follows :









Basic Data Types

Numeric Types

For any programmer, using numbers is the most important issue. You can hardly write a serious program without using numbers, so let's talk about some basic numeric types. There are three distinct numeric types: **signed integer numbers**, **floating point numbers** and **complex numbers**.

* **Signed Integer** type is called int, they are whole numbers (positive, negative or zero), including no decimal point. For example: 71, -122, 0
* **Floating point** type is called float and they stand for real numbers with a decimal point. For example: 71.0, -33.03
* **Complex** type is called complex and they are written in the form, **x + yj** , where x is the real part and y is the imaginary part. For example: 3.14j. Imaginary numbers, also called complex numbers, are used in real-life applications, such as electricity, as well as quadratic equations. In quadratic planes, imaginary numbers show up in equations that don't touch the x-axis. Imaginary numbers become particularly useful in advanced calculus. We will not use this type much.

**💡Tips:**

* 71 and 71.0 have the same numerical value. But they differ in terms of numeric type. The types of these numbers are int and float respectively.

## Basic Data Types

### Boolean

Boolean types are called bool and their values are the two constant objects **False** and **True**. They are used to represent truth values (other values can also be considered false or true). In numeric contexts (for example, when used as the argument to an arithmetic operator), they behave like the integers 0 and 1, respectively.

Bools are important data types that are widely used in Python as they can find use in every aspect of our daily lives. For example, imagine, whether the TV is turned on or off in your home or if the weather is rainy can be explained easily with bools.

Bools are mostly used in **conditional operations** which we will discuss in the next lessons.

Basic Data Types

Type Conversion

We can convert the types of data to each other if the type allows to be converted. There are some functions to convert the types:

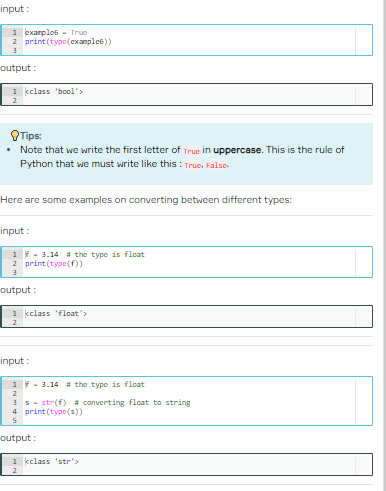
* str() converts to **string** type
* int() converts to **signed integer** type
* float() converts to **floating point** type

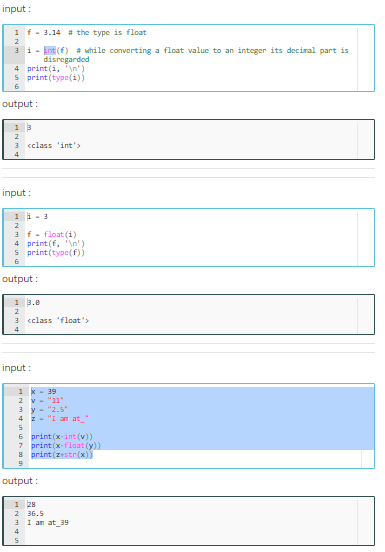
**💡Tips:**

* We can print the types of data using type() function.

Look at the examples below to how we learn the types of data.







## Simple 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 :

* print(4 + 11) # sum of integers gives integer
* 15
* print(39 + 1.0) # sum of an integer and float gives float
* 40.0
* input

no1, no2 = 46, 52

no3 = no1 - no2

print(no3)

* -6
* 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.
* no1 = 46
* print(no1/23) # division gives float **BÖLME İŞLEMİ FOLAT VERİRİ**
* 2.0
* print((3 \* 4)/2) # parentheses are used as in normal mathematics operations
* 6.0
* print(7 // 2) # it gives integer part of division **BÖLÜM’ü verir**
* 3
* print(9 % 2) # remainder of this division is 1 MODULUS **KALAN’ı verir**
* # it means 9 is an odd number
* 1
* print(3\*\*2) EXPONTETION **ÜS**
* 9
* print(64\*\*0.5) # square root **KAREKÖK**
* 8.0
* print('Result of this (12+7) sum :', 12 + 7)
* 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 : +, -

**Boolean Operations**

**Definitions**

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;

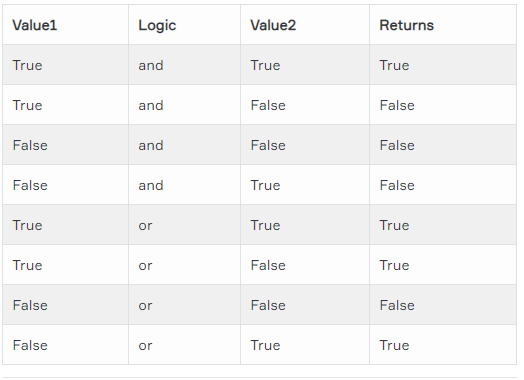
If you pass the course : is\_pass = True,  
If you did not pass the course : is\_pass = False

**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.

**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).



**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 :

**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.

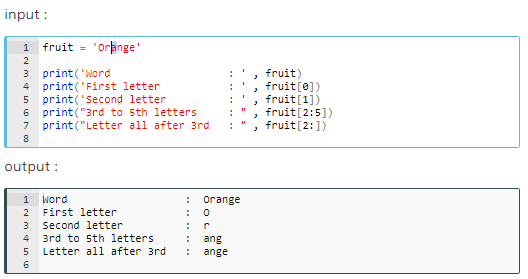
Here are some and operations :

## The Strength of Strings in Python

### 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:

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



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

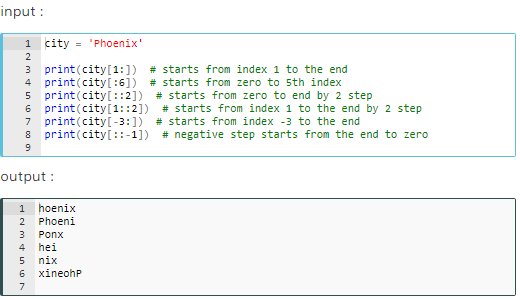
**string[:]** : returns the full copy of the sequence

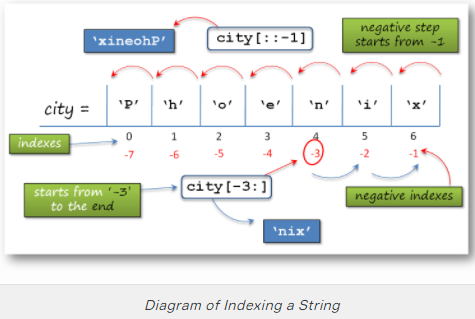
**string[start:]** : returns elements from start to the end element

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

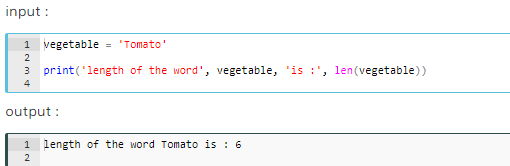
**string[::step]** : returns each element with a given step

Let's see it in an example :





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



### String Formatting with Arithmetic Syntax

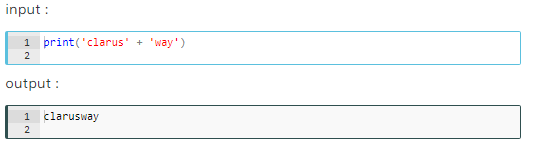
There are several ways in Python that we use when processing and using string data structures. The most important of these are:

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

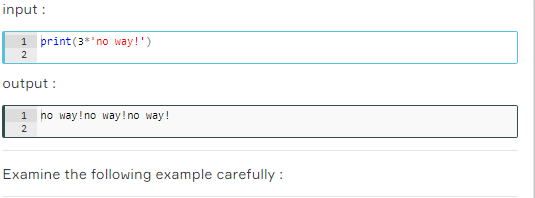
We have stated to you what the **function** is? in the previous lessons. At this point, let us give the definition of the term **method**. A **method** is like a function, except it is attached to an **object**. We call a method on an object, and it possibly makes changes to that object (like string.format()).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

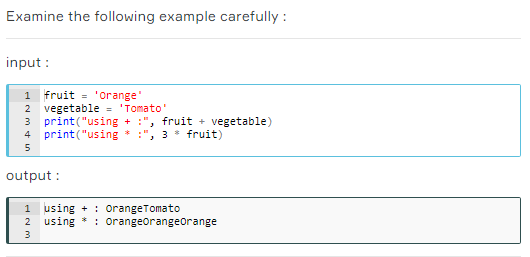
##### **Arithmetic syntax (**+**,**=**,**\***) :**

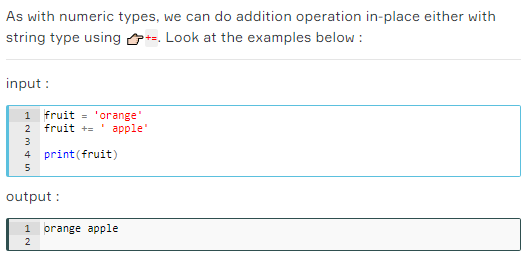
We can use + operator for combining the two string together without any spaces. For example :

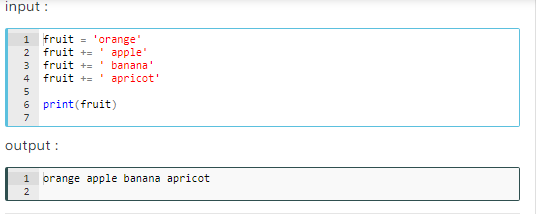


We can also use \* operator for repeating the string without any spaces. For example :









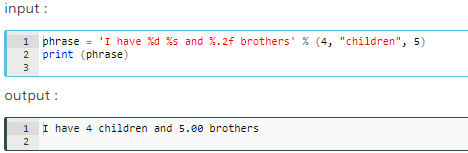
### String Formatting with '%' Operator

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.

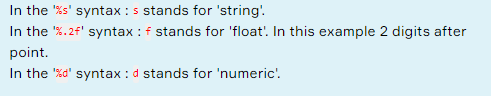
##### **'**%**' operator formatting :**

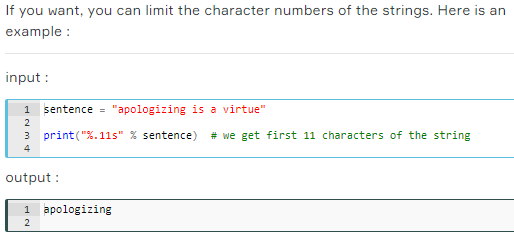
👉🏻% operator gets the values in order and prints them in order using several characters accordingly. Look at the example :

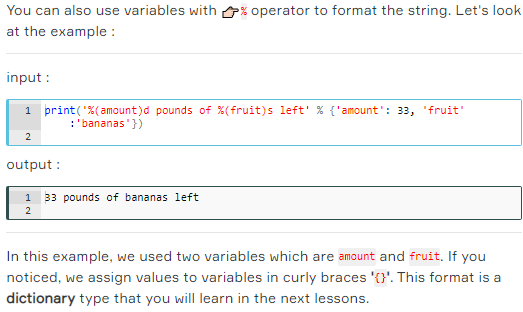
For now, we used only s, d and f characters to specify the data type in a string.



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.







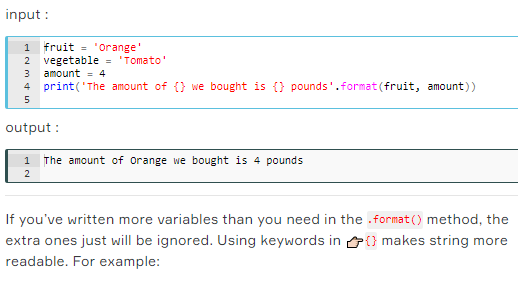
### String Formatting with 'string.format()' Method

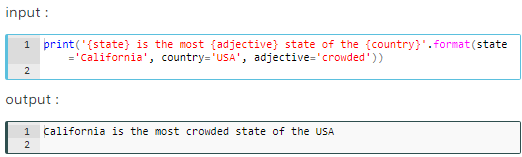
You can make strings change depending on the value of a variable or an expression. The main methods of Python to format the output are :

##### **'**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().

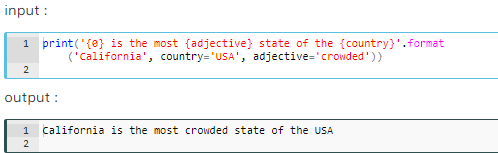




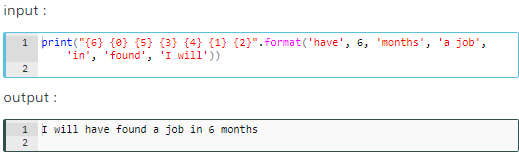
* If you have noticed, we do not have to write the keywords in .format() method in order.

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.



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.



* 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 Formatting with 'f-string'

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

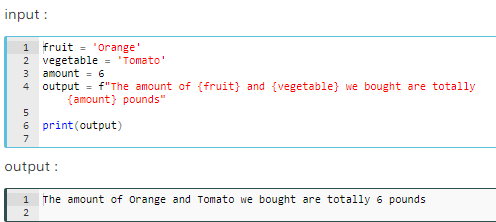
##### **'**f-string**' formatting :**

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

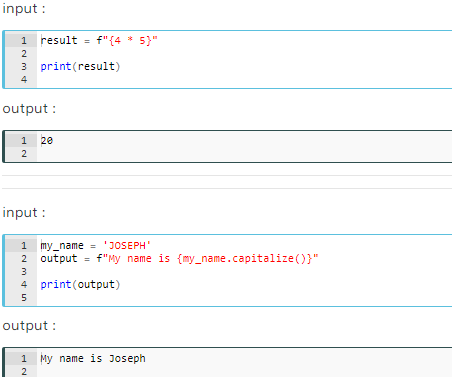
**f-string** is the string syntax that is enclosed in quotes with a letter **f** at the beginning. Curly braces 👉🏻{} that contain variable names or expressions are used to replace with their values.

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

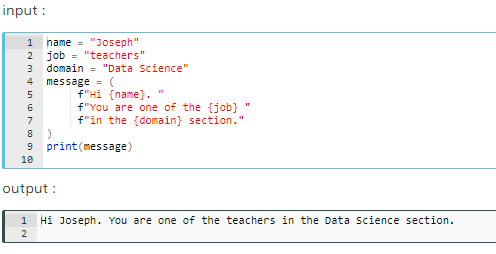
Let's look at the example below on how the syntax is simple and readable.



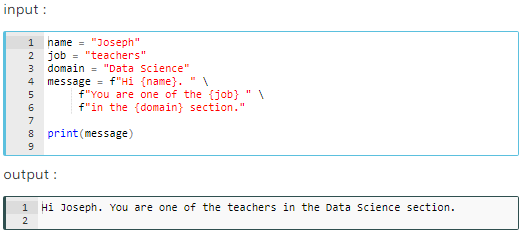
You can use all valid expressions, variables, and even methods in curly braces. Look at the examples:



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



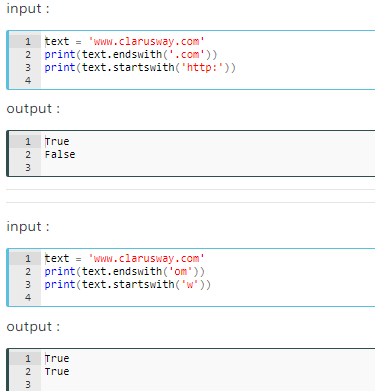
If you want to use multiple f-string formatting lines without parentheses, you will have the other option that you can use backslash 👉🏻**\** between lines.



## Main String Operations

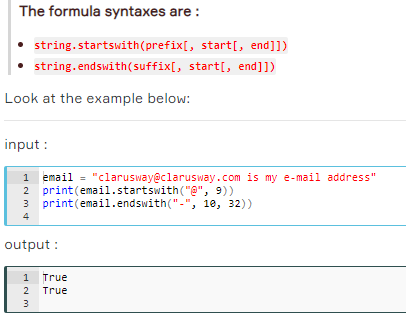
### Searching a String

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.



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.

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



### 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.

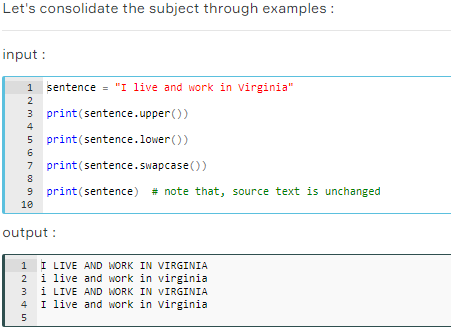
**The formula syntax is : string.method()**

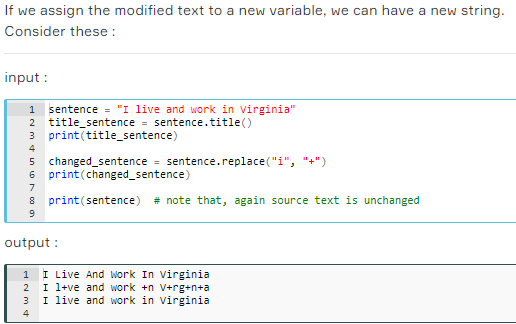
Let's examine some common and the most important methods of string changing :

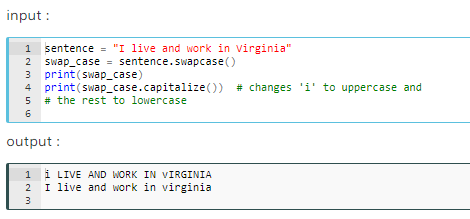
* **str.replace(old, new[, count])** replaces all occurrences of old with the new.

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.







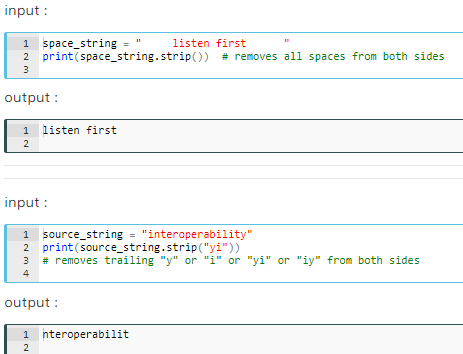
### Editing a String

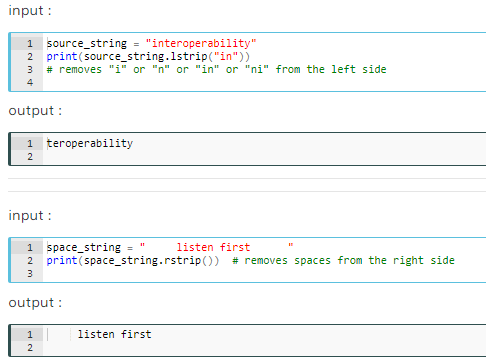
The methods described below remove the trailing characters (i.e. characters from the right side). The default for the argument chars is also whitespace. If the argument chars aren’t specified, trailing whitespaces are removed.

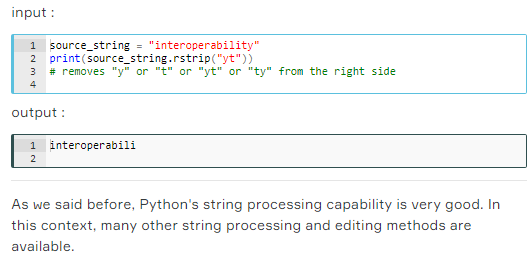
**The formula syntax is : string.method()**

* **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? :







## Lists

### Introduction

There are various collection types in Python. While types such as int and str hold a single value, collection types hold multiple values.

In your programs, you usually need to group several items to render as a single object. We use collection types of data to do this job.

One of the most useful collections in Python is a list. In Python, a list is only an **ordered collection** of valid Python values.

The list type is probably the most commonly used collection type in Python. In spite of its name, a list is more like an array in some other languages (e.g. JavaScript).

### Creating a List

A list can be created by enclosing values, separated by commas, in square brackets 👉🏻[].

Let's create a simple list that includes some country names.

**Koşullu İfadeler (Conditional Statements)**

### If Statement Yapıları

* In some cases, your program needs to execute some part of the code only if a specific condition is true. The simple structure of an if statement is :

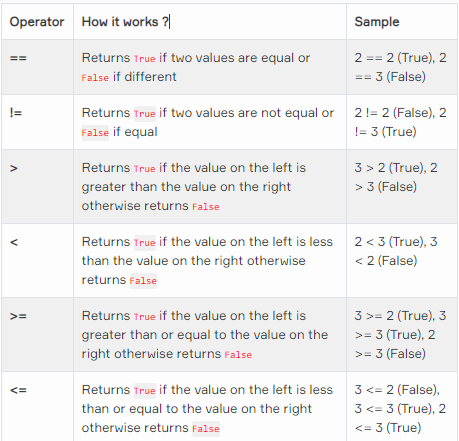


* The **if** statements check the **condition**. The **condition** is always a **Boolean**  expression, that is, its value equals either True or False. Remmber truthy or falsy (None, 0 and [], (), {})
* If it evaluates to True, it executes the **body** of the **if** statement. If it evaluates to False, it skips the **body**. This logic works like the English language.

### Comparison Operators

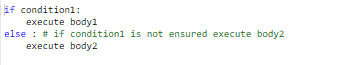
Boolean values basically make it clear if a piece of code needs to be executed. Because comparisons result in bool, it's always best to use them as a condition.

Therefore, it is time to examine **comparison operators** :



### 'if-else' Statements

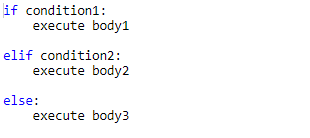
* An **if-else** statement is another kind of conditional statements in Python. It is used with an additional keyword: else.
* else works like an if statement. If none of the conditions in if are ensured, "else" will be used to specify all remaining conditions. The simple structure of an if-else looks like



* else doesn't require any condition and the body2 requires 4-space indentation

### if-elif-else' Statements

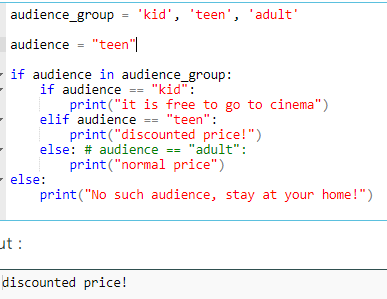
* The elif statement is used when it requires to specify several conditions in our program.
* In Python you can deﬁne a series of conditionals using :
* if for the **ﬁrst** one,
* elif for the **rest**, up until the ﬁnal (optional),
* else for **anything not caught by the other conditionals**.
* The basic structure of these statements looks like :



* We can use as many elif statements as we need, so your conditions can be varied.

### Nested 'if-elif-else' Statements

* Both if-else and if-elif statements can be nested. Let's see the nested structure on the same movie ticket example.



* Nested if-elif-else structures may seem a bit complicated to you, the best way to overcome this is to examine and practice plenty of samples.

**Not:** If yapılarında kullanıcıdan Input fonksiyonu ile “string” bir veri istendiğinde, case sensitive bir dil olması sebebiyle **değişken\_adı.lower () == “XXXX”** formatı kullanılabilir.

**ülke = USA**

**if ülke.**lower() == “usa”

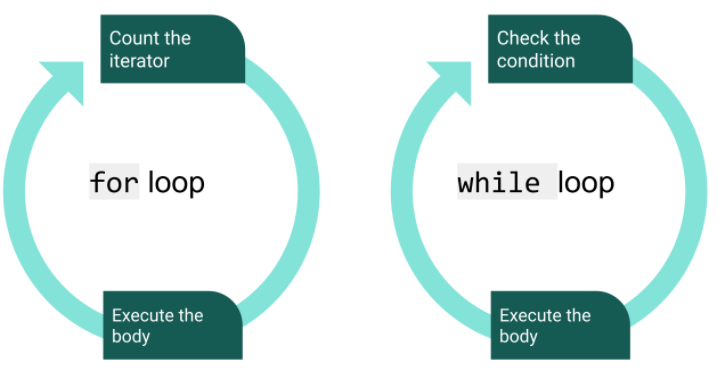
**print(“”)**

## Loops

* When writing programs in Python, in some cases it is not enough to execute our block of code only once. **The loops are used to repeat (iterate) the execution of a block of code.**
* As one of the most main functions in programming, loops are an important part of almost every programming language. Loops enable programmers to set certain sections of their code to repeat through a number of loops which are referred to as iterations.

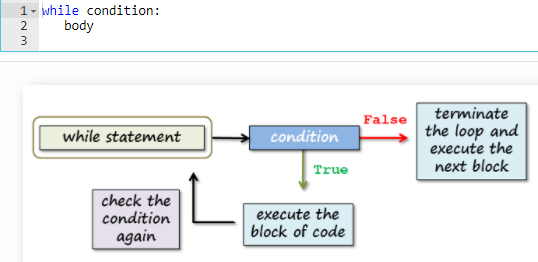
This topic covers using multiple types of loops and applications of loops in Python. You will learn two types of loops which are :

* **while Loop**,
* **for Loop**.



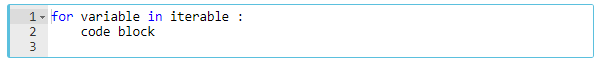
### 'while' Loop

* while loops have a boolean logic, similar to if statements.
* As long as the result of the condition returns True, the code block under while loop runs. When the condition returns to False, the loop execution is terminated and the program control moves further to the next operation.
* Here is the simple structure of a while loop :

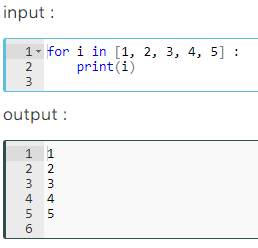


### 'for' Loop

* When you want to iterate a block of code you will use for loop. To create a for loop, you need a **variable** and an **iterable object**. Here is the simple structure of a for loop :



* We can use list as an iterable



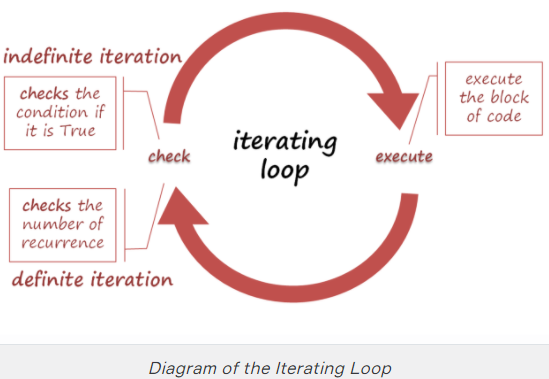
### Working with the Iterators

Let us explain the term **iteration** a little more.

**Iterable** object can be anything for which items are received one by one, forward only. In Python, the process of recurrent execution of a block of code is called an **iteration**.

We can basically classify iterations as two headings :

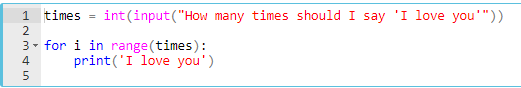
* If the number of repetitions is predetermined, it is called **definite** iteration.
* The repetition structure that makes the code block run as long as the predetermined condition generates True is called **indefinite** iteration.



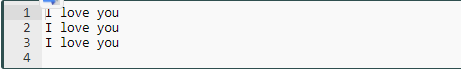
### Operations with the 'for' Loop

In this topic, you will learn about how we use the for loop using several functions and methods.

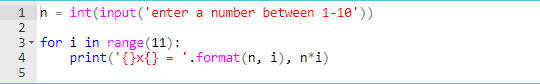
* In the example below, you'll get a number from the user and print a sentence the number of times we receive from the user.



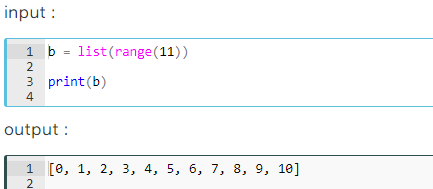
* As we stated before, input() function can get the value of different data types and assign a variable you chose. In the example above, it gets a number and assigns it to times. If the user enters 3 then the output will be:

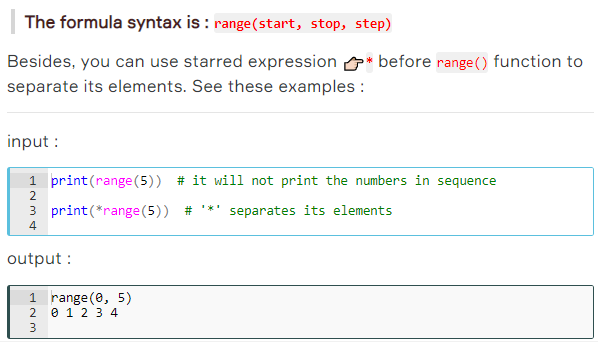


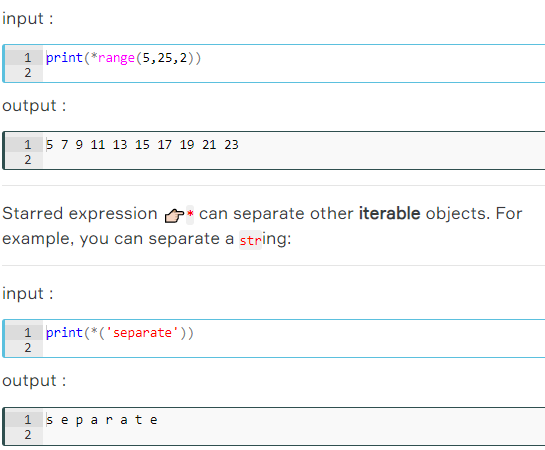
* Now, let's write a code that asks the user a number between 1 and 10 and puts that number into the **multiplication table**.

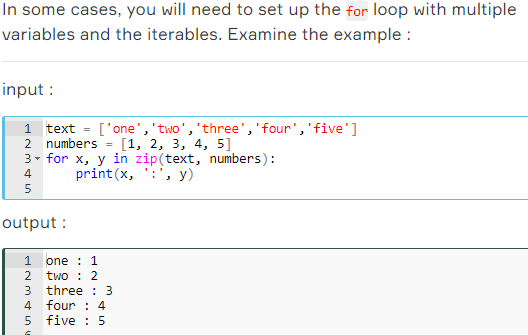


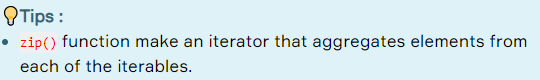
* If you want the user to input numbers, use the input() function together with the int() function. Otherwise, the value entered by the user will be in the **string** data type.
* The range() function creates an iterable sequence of numbers. And it can be simply converted into an iterable object: list, set, and tuple. For example :



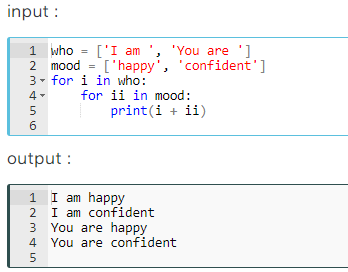








### Nested 'for' Loop



max()

* Fonksiyon, içine girilen iterable’da en yüksek değeri verir.
* Boş liste olursa default neyse onu verir.