

**What is Python?**

Python is a popular programming language. It was created by *Guido van Rossum*, and released in *1991*.

**It is used for:**

* web development (server-side),
* software development,
* mathematics,
* system scripting.

**What can Python do?**

* Python can be used on a server to create web applications.
* Python can be used alongside software to create workflows.
* Python can connect to database systems. It can also read and modify files.
* Python can be used to handle big data and perform complex mathematics.
* Python can be used for rapid prototyping, or for production-ready software development.

**Why Python?**

* Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
* Python has a simple syntax similar to the English language.
* Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
* Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
* Python can be treated in a procedural way, an object-oriented way or a functional way.

**Good to know**

* The most recent major version of Python is Python 3, which we shall be using in this tutorial. However, Python 2, although not being updated with anything other than security updates, is still quite popular.
* In this tutorial Python will be written in a text editor. It is possible to write Python in an Integrated Development Environment, such as Thonny, Pycharm, Netbeans or Eclipse which are particularly useful when managing larger collections of Python files.

**Python Syntax compared to other programming languages**

* Python was designed for readability, and has some similarities to the English language with influence from mathematics.
* Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.
* Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.

print("Hello, World!")

**Python Indentation:**

Indentation refers to the spaces at the beginning of a code line.

Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important.

Python uses indentation to indicate a block of code.

**Python Variables:**

Variables are containers for storing data values.

Python has no command for declaring a variable.

A variable is created the moment you first assign a value to it.

e.g.

x = 5  
y = "John"  
print(x)  
print(y)

Variables do not need to be declared with any particular *type*, and can even change type after they have been set.

x = 4        # x is of type int  
x = "Sally"  # x is now of type str  
print(x)

**Casting**

If you want to specify the data type of a variable, this can be done with casting.

Example

x = str(3)    # x will be '3'  
y = int(3)    # y will be 3  
z = float(3)  # z will be 3.0

**Get the Type**

You can get the data type of a variable with the type() function.

Example

x = 5  
y = "John"  
print(type(x))  
print(type(y))

**Case-Sensitive**

Variable names are case-sensitive.

Example

This will create two variables:

a = 4  
A = "Sally"  
#A will not overwrite a

**Variable Names**

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total\_volume).

**Rules for Python variables:**

* A variable name must start with a letter or the underscore character
* A variable name cannot start with a number
* A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and \_ )
* Variable names are case-sensitive (age, Age and AGE are three different variables)
* A variable name cannot be any of the [Python keywords](https://www.w3schools.com/python/python_ref_keywords.asp).

Example

Legal variable names:

myvar = "John"  
my\_var = "John"  
\_my\_var = "John"  
myVar = "John"  
MYVAR = "John"  
myvar2 = "John"

Example

**Illegal variable names:**

2myvar = "John"  
my-var = "John"  
my var = "John"

**Multi Words Variable Names**

Variable names with more than one word can be difficult to read.

There are several techniques you can use to make them more readable:

**Camel Case**

Each word, except the first, starts with a capital letter:

myVariableName = "John"

**Pascal Case**

Each word starts with a capital letter:

MyVariableName = "John"

**Snake Case**

Each word is separated by an underscore character:

my\_variable\_name = "John"

**Many Values to Multiple Variables**

Python allows you to assign values to multiple variables in one line:

**Example**

x, y, z = "Orange", "Banana", "Cherry"  
print(x)  
print(y)  
print(z)

**Note: Make sure the number of variables matches the number of values, or else you will get an error.**

**One Value to Multiple Variables**

And you can assign the *same* value to multiple variables in one line:

Example

x = y = z = "Orange"  
print(x)  
print(y)  
print(z)

**Unpack a Collection**

If you have a collection of values in a list, tuple etc. Python allows you to extract the values into variables. This is called *unpacking*.

Example

Unpack a list:

fruits = ["apple", "banana", "cherry"]  
x, y, z = fruits  
print(x)  
print(y)  
print(z)

**Output Variables**

The Python print() function is often used to output variables.

In the print() function, you output multiple variables, separated by a comma:

x = "Python"  
y = "is"  
z = "awesome"  
print(x, y, z)

You can also use the + operator to output multiple variables:

x = "Python "  
y = "is "  
z = "awesome"  
print(x + y + z)

For numbers, the + character works as a mathematical operator:

Example

x = 5  
y = 10  
print(x + y)

In the print() function, when you try to combine a string and a number with the + operator, Python will give you an error:

Example

x = 5  
y = "John"  
print(x + y)

The best way to output multiple variables in the print() function is to separate them with commas, which even support different data types:

Example

x = 5  
y = "John"  
print(x, y)

**Global Variables**

Variables that are created outside of a function (as in all of the examples in the previous pages) are known as global variables.

Global variables can be used by everyone, both inside of functions and outside.

If you create a variable with the same name inside a function, this variable will be local, and can only be used inside the function. The global variable with the same name will remain as it was, global and with the original value.

**The global Keyword**

Normally, when you create a variable inside a function, that variable is local, and can only be used inside that function.

To create a global variable inside a function, you can use the global keyword.

Example

If you use the global keyword, the variable belongs to the global scope:

def myfunc():  
  global x  
  x = "fantastic"  
  
myfunc()  
  
print("Python is " + x)

Also, use the global keyword if you want to change a global variable inside a function.

Example

To change the value of a global variable inside a function, refer to the variable by using the global keyword:

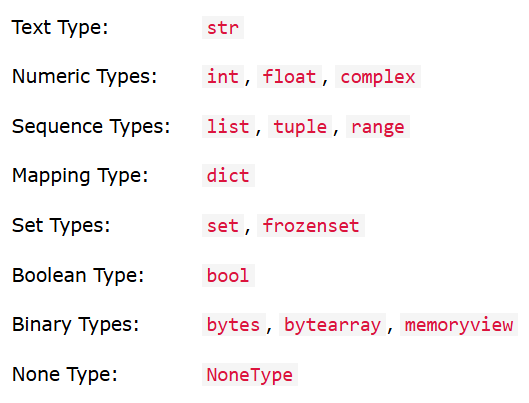
x = "awesome"  
  
def myfunc():  
  global x  
  x = "fantastic"  
  
myfunc()  
  
print("Python is " + x)

**Built-in Data Types:**

In programming, data type is an important concept.

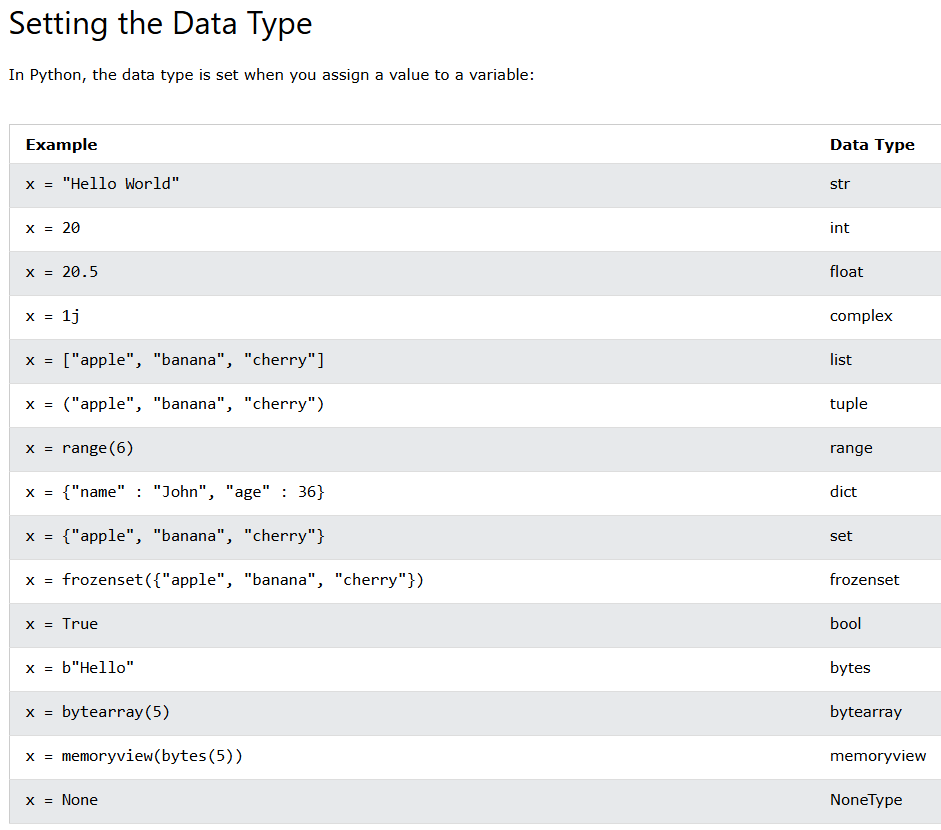
Variables can store data of different types, and different types can do different things.

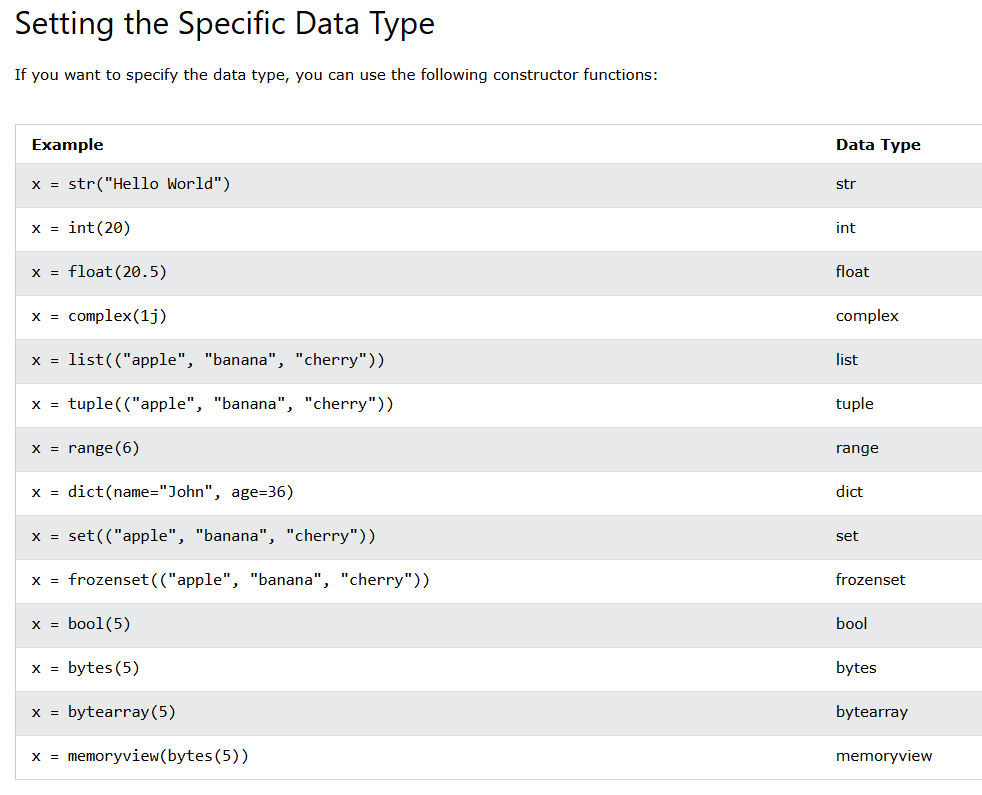
Python has the following data types built-in by default, in these categories:



Getting the Data Type

You can get the data type of any object by using the type() function





Python Numbers

There are three numeric types in Python:

* int
* float
* complex

Variables of numeric types are created when you assign a value to them

x = 1    # int  
y = 2.8  # float  
z = 1j   # complex

*Int*

Int, or integer, is a whole number, positive or negative, without decimals, of unlimited length.

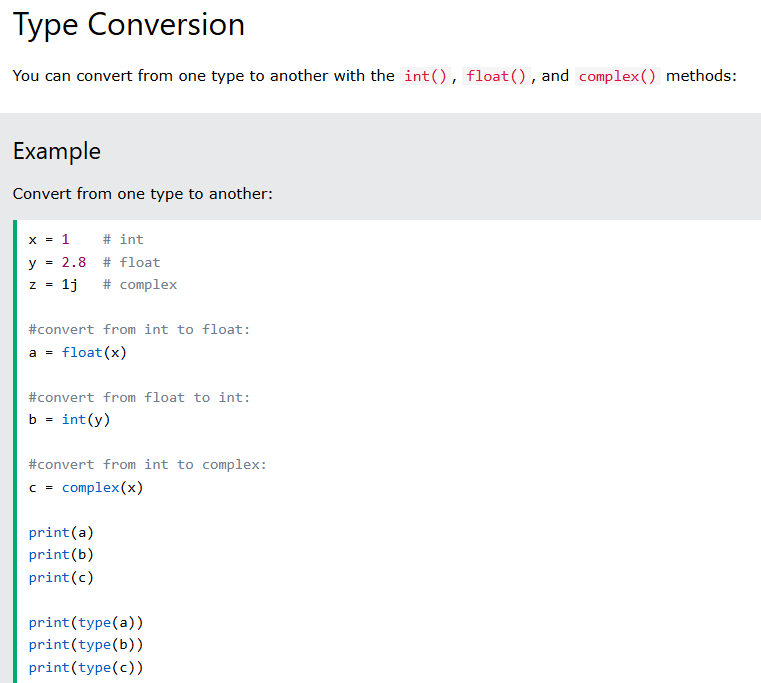
*Float*

Float, or "floating point number" is a number, positive or negative, containing one or more decimals.

Float can also be scientific numbers with an "e" to indicate the power of 10.

*Complex*

Complex numbers are written with a "j" as the imaginary part.



**Note:** You cannot convert complex numbers into another number type.

*Random Number:*

Python does not have a random() function to make a random number, but Python has a built-in module called random that can be used to make random numbers:

Example

Import the random module, and display a random number from 1 to 9:

import random  
  
print(random.randrange(1, 10))

**Strings:**

Strings in python are surrounded by either single quotation marks, or double quotation marks.

'hello' is the same as "hello".

You can display a string literal with the print() function.

*Quotes Inside Quotes*

You can use quotes inside a string, as long as they don't match the quotes surrounding the string:

Example

print("It's alright")  
print("He is called 'Johnny'")  
print('He is called "Johnny"')

*Multiline Strings*

You can assign a multiline string to a variable by using three quotes:

Example

You can use three double quotes:

a = """Lorem ipsum dolor sit amet,  
consectetur adipiscing elit,  
sed do eiusmod tempor incididunt  
ut labore et dolore magna aliqua."""  
print(a)

*Strings are Arrays*

Like many other popular programming languages, strings in Python are arrays of bytes representing unicode characters.

However, Python does not have a character data type, a single character is simply a string with a length of 1.

Square brackets can be used to access elements of the string.

Example

Get the character at position 1 (remember that the first character has the position 0):

a = "Hello, World!"  
print(a[1])

*Looping Through a String*

Since strings are arrays, we can loop through the characters in a string, with a for loop.

Example

Loop through the letters in the word "banana":

for x in "banana":  
  print(x)

*String Length*

To get the length of a string, use the len() function.

Example

The len() function returns the length of a string:

a = "Hello, World!"  
print(len(a))

Check String

To check if a certain phrase or character is present in a string, we can use the keyword in.

Example

Check if "free" is present in the following text:

txt = "The best things in life are free!"  
print("free" in txt)

print only if "expensive" is NOT present:

txt = "The best things in life are free!"  
if "expensive" not in txt:  
  print("No, 'expensive' is NOT present.")

***Collections:***

List

Lists are used to store multiple items in a single variable.

Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are [Tuple](https://www.w3schools.com/python/python_tuples.asp), [Set](https://www.w3schools.com/python/python_sets.asp), and [Dictionary](https://www.w3schools.com/python/python_dictionaries.asp), all with different qualities and usage.

Lists are created using square brackets:

thislist = ["apple", "banana", "cherry"]  
print(thislist)

List Items

List items are ordered, changeable, and allow duplicate values.

List items are indexed, the first item has index [0], the second item has index [1] etc.

Ordered

When we say that lists are ordered, it means that the items have a defined order, and that order will not change.

If you add new items to a list, the new items will be placed at the end of the list.

**Note:** There are some [list methods](https://www.w3schools.com/python/python_lists_methods.asp) that will change the order, but in general: the order of the items will not change.

Changeable

The list is changeable, meaning that we can change, add, and remove items in a list after it has been created.

Allow Duplicates

Since lists are indexed, lists can have items with the same value:

Example

Lists allow duplicate values:

thislist = ["apple", "banana", "cherry", "apple", "cherry"]  
print(thislist)

List Length

To determine how many items a list has, use the len() function:

Example

Print the number of items in the list:

thislist = ["apple", "banana", "cherry"]  
print(len(thislist))

List Items - Data Types

List items can be of any data type:

Example

String, int and boolean data types:

list1 = ["apple", "banana", "cherry"]  
list2 = [1, 5, 7, 9, 3]  
list3 = [True, False, False]

type()

From Python's perspective, lists are defined as objects with the data type 'list':

<class 'list'>

Example

What is the data type of a list?

mylist = ["apple", "banana", "cherry"]  
print(type(mylist))

The list() Constructor

It is also possible to use the list() constructor when creating a new list.

Example

Using the list() constructor to make a List:

thislist = list(("apple", "banana", "cherry")) # note the double round-brackets  
print(thislist)

Python Collections (Arrays)

There are four collection data types in the Python programming language:

* **List** is a collection which is ordered and changeable. Allows duplicate members.
* [**Tuple**](https://www.w3schools.com/python/python_tuples.asp) is a collection which is ordered and unchangeable. Allows duplicate members.
* [**Set**](https://www.w3schools.com/python/python_sets.asp) is a collection which is unordered, unchangeable\*, and unindexed. No duplicate members.
* [**Dictionary**](https://www.w3schools.com/python/python_dictionaries.asp) is a collection which is ordered\*\* and changeable. No duplicate members.

\*Set *items* are unchangeable, but you can remove and/or add items whenever you like.

\*\*As of Python version 3.7, dictionaries are *ordered*. In Python 3.6 and earlier, dictionaries are *unordered*.

When choosing a collection type, it is useful to understand the properties of that type. Choosing the right type for a particular data set could mean retention of meaning, and, it could mean an increase in efficiency or security.

Access Items

List items are indexed and you can access them by referring to the index number:

Print the second item of the list:

thislist = ["apple", "banana", "cherry"]  
print(thislist[1])

**Note:**The first item has index 0.

Negative Indexing

Negative indexing means start from the end

-1 refers to the last item, -2 refers to the second last item etc.

Example

Print the last item of the list:

thislist = ["apple", "banana", "cherry"]  
print(thislist[-1])

Range of Indexes

You can specify a range of indexes by specifying where to start and where to end the range.

When specifying a range, the return value will be a new list with the specified items.

Example

Return the third, fourth, and fifth item:

thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]  
print(thislist[2:5])

**Note:** The search will start at index 2 (included) and end at index 5 (not included).

Remember that the first item has index 0.

By leaving out the start value, the range will start at the first item:

Example

This example returns the items from the beginning to, but NOT including, "kiwi":

thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]  
print(thislist[:4])

By leaving out the end value, the range will go on to the end of the list:

Example

This example returns the items from "cherry" to the end:

thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]  
print(thislist[2:])

Range of Negative Indexes

Specify negative indexes if you want to start the search from the end of the list:

Example

This example returns the items from "orange" (-4) to, but NOT including "mango" (-1):

thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]  
print(thislist[-4:-1])

Check if Item Exists

To determine if a specified item is present in a list use the in keyword:

Example

Check if "apple" is present in the list:

thislist = ["apple", "banana", "cherry"]  
if "apple" in thislist:  
  print("Yes, 'apple' is in the fruits list")

**Note:** The length of the list will change when the number of items inserted does not match the number of items replaced.

If you insert *less* items than you replace, the new items will be inserted where you specified, and the remaining items will move accordingly

Insert Items

To insert a new list item, without replacing any of the existing values, we can use the insert() method.

The insert() method inserts an item at the specified index:

Example

Insert "watermelon" as the third item:

thislist = ["apple", "banana", "cherry"]  
thislist.insert(2, "watermelon")  
print(thislist)

Append Items

To add an item to the end of the list, use the append() method:

Example

Using the append() method to append an item:

thislist = ["apple", "banana", "cherry"]  
thislist.append("orange")  
print(thislist)

Insert Items

To insert a list item at a specified index, use the insert() method.

The insert() method inserts an item at the specified index:

Example

Insert an item as the second position:

thislist = ["apple", "banana", "cherry"]  
thislist.insert(1, "orange")  
print(thislist)

Extend List

To append elements from *another list* to the current list, use the extend() method.

Example

Add the elements of tropical to thislist:

thislist = ["apple", "banana", "cherry"]  
tropical = ["mango", "pineapple", "papaya"]  
thislist.extend(tropical)  
print(thislist)

Add Any Iterable

The extend() method does not have to append *lists*, you can add any iterable object (tuples, sets, dictionaries etc.).

Example

Add elements of a tuple to a list:

thislist = ["apple", "banana", "cherry"]  
thistuple = ("kiwi", "orange")  
thislist.extend(thistuple)  
print(thislist)

Remove Specified Item

The remove() method removes the specified item.

Example

Remove "banana":

thislist = ["apple", "banana", "cherry"]  
thislist.remove("banana")  
print(thislist)

If there are more than one item with the specified value, the remove() method removes the first occurrence:

Example

Remove the first occurrence of "banana":

thislist = ["apple", "banana", "cherry", "banana", "kiwi"]  
thislist.remove("banana")  
print(thislist)

Remove Specified Index

The pop() method removes the specified index.

Example

Remove the second item:

thislist = ["apple", "banana", "cherry"]  
thislist.pop(1)  
print(thislist)

If you do not specify the index, the pop() method removes the last item.

The del keyword also removes the specified index:

Clear the List

The clear() method empties the list.

The list still remains, but it has no content.

Loop Through the Index Numbers

You can also loop through the list items by referring to their index number.

Use the range() and len() functions to create a suitable iterable.

Example

Print all items by referring to their index number:

thislist = ["apple", "banana", "cherry"]  
for i in range(len(thislist)):  
  print(thislist[i])

Using a While Loop

You can loop through the list items by using a while loop.

Use the len() function to determine the length of the list, then start at 0 and loop your way through the list items by referring to their indexes.

Remember to increase the index by 1 after each iteration.

Example

Print all items, using a while loop to go through all the index numbers

thislist = ["apple", "banana", "cherry"]  
i = 0  
while i < len(thislist):  
  print(thislist[i])  
  i = i + 1

Looping Using List Comprehension

List Comprehension offers the shortest syntax for looping through lists:

Example

A short hand for loop that will print all items in a list:

thislist = ["apple", "banana", "cherry"]  
[print(x) for x in thislist]

List Comprehension

List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list.

Example:

Based on a list of fruits, you want a new list, containing only the fruits with the letter "a" in the name.

Without list comprehension you will have to write a for statement with a conditional test inside:

fruits = ["apple", "banana", "cherry", "kiwi", "mango"]  
newlist = []  
  
for x in fruits:  
  if "a" in x:  
    newlist.append(x)  
  
print(newlist)

The Syntax

newlist = [*expression* for *item* in *iterable* if *condition* == True]

The return value is a new list, leaving the old list unchanged.

Condition

The *condition* is like a filter that only accepts the items that evaluate to True.

Example

Only accept items that are not "apple":

newlist = [x for x in fruits if x != "apple"]

To sort descending, use the keyword argument reverse = True:

Example

Sort the list descending:

thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]  
thislist.sort(reverse = True)  
print(thislist)

