**Python Tutorial**

Python is a programming language.

Python can be used on a server to create web applications.

Learning by Examples

With our "Try it Yourself" editor, you can edit Python code and view the result.

**Example**

print("Hello, World!")

**Python File Handling**

In our File Handling section you will learn how to open, read, write, and delete files.

[Python File Handling](https://www.w3schools.com/python/python_file_handling.asp)

**Python Database Handling**

In our database section you will learn how to access and work with MySQL and MongoDB databases:

[Python MySQL Tutorial](https://www.w3schools.com/python/python_mysql_getstarted.asp)

[Python MongoDB Tutorial](https://www.w3schools.com/python/python_mongodb_getstarted.asp)

**Python ExercisesTop of Form**

Exercise: Insert the missing part of the code below to output "Hello World".

("Hello World")

Bottom of Form **Python Examples**

Learn by examples! This tutorial supplements all explanations with clarifying examples.

**Python Quiz**

Test your Python skills with a quiz.

[Python Quiz](https://www.w3schools.com/python/python_quiz.asp)

**Python Reference**

You will also find complete function and method references:

[Reference Overview](https://www.w3schools.com/python/python_reference.asp)

[Built-in Functions](https://www.w3schools.com/python/python_ref_functions.asp)

[String Methods](https://www.w3schools.com/python/python_ref_string.asp)

[List/Array Methods](https://www.w3schools.com/python/python_ref_list.asp)

[Dictionary Methods](https://www.w3schools.com/python/python_ref_dictionary.asp)

[Tuple Methods](https://www.w3schools.com/python/python_ref_tuple.asp)

[Set Methods](https://www.w3schools.com/python/python_ref_set.asp)

[File Methods](https://www.w3schools.com/python/python_ref_file.asp)

[Python Keywords](https://www.w3schools.com/python/python_ref_keywords.asp)

[Python Exceptions](https://www.w3schools.com/python/python_ref_exceptions.asp)

[Python Glossary](https://www.w3schools.com/python/python_ref_glossary.asp)

[Random Module](https://www.w3schools.com/python/module_random.asp)

[Requests Module](https://www.w3schools.com/python/module_requests.asp)

[Math Module](https://www.w3schools.com/python/module_math.asp)

[CMath Module](https://www.w3schools.com/python/module_cmath.asp)

Download Python from the official Python web site: [https://python.org](https://python.org/)

# Python Introduction

## **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.

### Example

print("Hello, World!")

# Python Getting Started

## **Python Install**

## Many **PCs** and **Macs** will have python already installed.

To check if you have python installed on a Windows PC, search in the start bar for Python or run the following on the Command Line (cmd.exe):

**C:\Users\Your Name>python --version**

To check if you have python installed on a Linux or Mac, then on linux open the command line or on Mac open the Terminal and type:

*python --version*

## **Python Quickstart**

## Python is an interpreted programming language, this means that as a developer you write Python (.py) files in a text editor and then put those files into the python interpreter to be executed.

The way to run a python file is like this on the command line:

**C:\Users\Your Name>python helloworld.py**

Where "helloworld.py" is the name of your python file.

Let's write our first Python file, called helloworld.py, which can be done in any text editor.

helloworld.py

print("Hello, World!")

Simple as that. Save your file. Open your command line, navigate to the directory where you saved your file, and run:

**C:\Users\Your Name>python helloworld.py**

The output should read:

Hello, World!

Congratulations, you have written and executed your first Python program.

## **The Python Command Line**

To test a short amount of code in python sometimes it is quickest and easiest not to write the code in a file. This is made possible because Python can be run as a command line itself.

Type the following on the Windows, Mac or Linux command line:

C:\Users\Your Name>python

Or, if the "python" command did not work, you can try "py":

C:\Users\Your Name>py

From there you can write any python, including our hello world example from earlier in the tutorial:

C:\Users\Your Name>python  
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)] on win32  
Type "help", "copyright", "credits" or "license" for more information.  
>>> print("Hello, World!")

Which will write "Hello, World!" in the command line:

C:\Users\Your Name>python  
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)] on win32  
Type "help", "copyright", "credits" or "license" for more information.  
>>> print("Hello, World!")  
Hello, World!

Whenever you are done in the python command line, you can simply type the following to quit the python command line interface:

exit()

# Python Syntax

[Execute Python Syntax](https://www.w3schools.com/python/python_syntax.asp#execute_python_syntax)[Python Indentation](https://www.w3schools.com/python/python_syntax.asp#python_indentation)[Python Variables](https://www.w3schools.com/python/python_syntax.asp#python_variables)[Python Comments](https://www.w3schools.com/python/python_syntax.asp#python_comments)[Exercises](https://www.w3schools.com/python/python_syntax.asp#exercises)

## **Execute Python Syntax**

## As we learned in the previous page, Python syntax can be executed by writing directly in the Command Line:

>>> print("Hello, World!")  
Hello, World!

Or by creating a python file on the server, using the .py file extension, and running it in the Command Line:

C:\Users\Your Name>python myfile.py

## **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.

### Example

if 5 > 2:  
  print("Five is greater than two!")

Python will give you an error if you skip the indentation:

### Example

Syntax Error:

if 5 > 2:  
print("Five is greater than two!")

The number of spaces is up to you as a programmer, but it has to be at least one.

### Example

if 5 > 2:  
 print("Five is greater than two!")   
if 5 > 2:  
        print("Five is greater than two!")

You have to use the same number of spaces in the same block of code, otherwise Python will give you an error:

### Example

Syntax Error:

if 5 > 2:  
 print("Five is greater than two!")  
        print("Five is greater than two!")

## **Python Variables**

## In Python, variables are created when you assign a value to it:

### Example

Variables in Python:

x = 5  
y = "Hello, World!"

Python has no command for declaring a variable.

You will learn more about variables in the [Python Variables](https://www.w3schools.com/python/python_variables.asp) chapter.

**Comments**

Python has commenting capability for the purpose of in-code documentation.

Comments start with a #, and Python will render the rest of the line as a comment:

### Example

Comments in Python:

#This is a comment.  
print("Hello, World!")

Top of Form

## **Exercise**

## Insert the missing part of the code below to output "Hello World".

…………………………("Hello World")

# Python Comments

Comments can be used to explain Python code.

Comments can be used to make the code more readable.

Comments can be used to prevent execution when testing code.

## **Creating a Comment**

## Comments starts with a #, and Python will ignore them:

### Example

#This is a comment  
print("Hello, World!")

Comments can be placed at the end of a line, and Python will ignore the rest of the line:

### Example

print("Hello, World!") #This is a comment

A comment does not have to be text that explains the code, it can also be used to prevent Python from executing code:

### Example

#print("Hello, World!")  
print("Cheers, Mate!")

## **Multi Line Comments**

## Python does not really have a syntax for multi line comments.

To add a multiline comment you could insert a # for each line:

### Example

#This is a comment  
#written in  
#more than just one line  
print("Hello, World!")

Or, not quite as intended, you can use a multiline string.

Since Python will ignore string literals that are not assigned to a variable, you can add a multiline string (triple quotes) in your code, and place your comment inside it:

### Example

""”  
This is a comment  
written in  
more than just one line  
"""  
print("Hello, World!")

As long as the string is not assigned to a variable, Python will read the code, but then ignore it, and you have made a multiline comment.

**Exercise:**

Comments in Python are written with a special character, which one?

…………This is a comment

# Python Variables

## **Variables**

Variables are containers for storing data values.

## **Creating Variables**

Python has no command for declaring a variable.

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

### Example

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.

### Example

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))

You will learn more about [data types](https://www.w3schools.com/python/python_datatypes.asp) and [casting](https://www.w3schools.com/python/python_casting.asp) later in this tutorial.

## **Single or Double Quotes?**

## String variables can be declared either by using single or double quotes:

### Example

x = "John"  
# is the same as  
x = 'John'

## **Case-Sensitive**

## Variable names are case-sensitive.

### Example

This will create two variables:

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

# Python - Variable Names

## **(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)

### 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"

Remember that variable names are case-sensitive

## **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:

## **1.Camel Case**

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

myVariableName = "John"

**2.Pascal Case**

Each word starts with a capital letter:

MyVariableName = "John"

## **3.Snake Case**

## Each word is separated by an underscore character:

my\_variable\_name = "John"

# (b). Python Variables - Assign Multiple Values

## **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 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)

Learn more about unpacking in our [Unpack Tuples](https://www.w3schools.com/python/python_tuples_unpack.asp) Chapter.

# (c). Python - Output Variables

# Output Variables

The Python print statement is often used to output variables.

To combine both text and a variable, Python uses the + character:

### Example

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

You can also use the + character to add a variable to another variable:

### Example

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

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

### Example

x = 5  
y = 10  
print(x + y)

If you try to combine a string and a number, Python will give you an error:

**Example**

x = 5  
y = "John"  
print(x + y)

# (d). Python - Global Variables

# Global Variables

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

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

### Example

Create a variable outside of a function, and use it inside the function

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

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.

### Example

Create a variable inside a function, with the same name as the global variable

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

## **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)

# (e). Python - Variable Exercises

## **Test Yourself With Exercises**

Now you have learned a lot about variables, and how to use them in Python.

Are you ready for a test?

Try to insert the missing part to make the code work as expected:

Top of Form

## **Exercise:**

Create a variable named carname and assign the value Volvo to it.

………………….= "…………………"

Bottom of Form

# Python Data Types

## **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:

|  |  |
| --- | --- |
| **Text Type:** | Str |
| **Numeric Types:** | int, float, complex |
| **Sequence Types:** | list, tuple, range |
| **Mapping Type:** | Dict |
| **Set Types:** | set, frozenset |
| **Boolean Type:** | Bool |
| **Binary Types:** | bytes, bytearray, memoryview |

## **Getting the Data Type**

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

### Example

Print the data type of the variable x:

x = 5  
print(type(x))

## **Setting the Data Type**

In Python, the data type is set when you assign a value to a variable:

|  |  |  |
| --- | --- | --- |
| **Example** | **Data Type** | **Try it** |
| x = "Hello World" | str | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_str) |
| x = 20 | int | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_int) |
| x = 20.5 | float | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_float) |
| x = 1j | complex | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_complex) |
| x = ["apple", "banana", "cherry"] | list | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_list) |
| x = ("apple", "banana", "cherry") | tuple | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_tuple) |
| x = range(6) | range | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_range) |
| x = {"name" : "John", "age" : 36} | dict | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_dict) |
| x = {"apple", "banana", "cherry"} | set | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_set) |
| x = frozenset({"apple", "banana", "cherry"}) | frozenset | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_frozenset) |
| x = True | bool | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_bool) |
| x = b"Hello" | bytes | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_bytes) |
| x = bytearray(5) | bytearray | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_bytearray) |
| x = memoryview(bytes(5)) | memoryview | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_memoryview) |

## **Setting the Specific Data Type**

## If you want to specify the data type, you can use the following constructor functions:

|  |  |  |
| --- | --- | --- |
| **Example** | **Data Type** | **Try it** |
| x = str("Hello World") | str | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_str2) |
| x = int(20) | int | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_int2) |
| x = float(20.5) | float | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_float2) |
| x = complex(1j) | complex | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_complex2) |
| x = list(("apple", "banana", "cherry")) | list | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_list2) |
| x = tuple(("apple", "banana", "cherry")) | tuple | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_tuple2) |
| x = range(6) | range | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_range2) |
| x = dict(name="John", age=36) | dict | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_dict2) |
| x = set(("apple", "banana", "cherry")) | set | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_set2) |
| x = frozenset(("apple", "banana", "cherry")) | frozenset | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_frozenset2) |
| x = bool(5) | bool | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_bool2) |
| x = bytes(5) | bytes | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_bytes2) |
| x = bytearray(5) | bytearray | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_bytearray2) |
| x = memoryview(bytes(5)) | memoryview | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_memoryview2) |

Top of Form

## **Exercise:**

The following code example would print the data type of x, what data type would that be?

x = 5

print(type(x))

…………………

Bottom of Form

# Python Numbers

**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:

### Example

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

To verify the type of any object in Python, use the type() function:

### Example

print(type(x))  
print(type(y))  
print(type(z))  
**Int**

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

### Example

Integers:

x = 1  
y = 35656222554887711  
z = -3255522  
  
print(type(x))  
print(type(y))  
print(type(z))

## **Float**

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

### Example

Floats:

x = 1.10  
y = 1.0  
z = -35.59  
  
print(type(x))  
print(type(y))  
print(type(z))

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

### Example

Floats:

x = 35e3  
y = 12E4  
z = -87.7e100  
  
print(type(x))  
print(type(y))  
print(type(z))

## **Complex**

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

### Example

Complex:

x = 3+5j  
y = 5j  
z = -5j  
  
print(type(x))  
print(type(y))  
print(type(z))

## **Type Conversion**

You can convert from one type to another with the int(), float(), and complex() methods:

### Example

Convert from one type to another:

x = 1    # int  
y = 2.8  # float  
z = 1j   # complex  
  
#convert from int to float:  
a = float(x)  
  
#convert from float to int:  
b = int(y)  
  
#convert from int to complex:  
c = complex(x)  
  
print(a)  
print(b)  
print(c)  
  
print(type(a))  
print(type(b))  
print(type(c))

**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 between 1 and 9:

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

In our [Random Module Reference](https://www.w3schools.com/python/module_random.asp) you will learn more about the Random module.

Top of Form

## **Exercise:**

Insert the correct syntax to convert x into a floating point number.

x = 5

x = …………………(x)

# Python Casting

## **Specify a Variable Type**

There may be times when you want to specify a type on to a variable. This can be done with casting. Python is an object-orientated language, and as such it uses classes to define data types, including its primitive types.

Casting in python is therefore done using constructor functions:

* int() - constructs an integer number from an integer literal, a float literal (by removing all decimals), or a string literal (providing the string represents a whole number)
* float() - constructs a float number from an integer literal, a float literal or a string literal (providing the string represents a float or an integer)
* str() - constructs a string from a wide variety of data types, including strings, integer literals and float literals

### Example

Integers:

x = int(1)   # x will be 1  
y = int(2.8) # y will be 2  
z = int("3") # z will be 3

### Example

Floats:

x = float(1)     # x will be 1.0  
y = float(2.8)   # y will be 2.8  
z = float("3")   # z will be 3.0  
w = float("4.2") # w will be 4.2

### Example

Strings:

x = str("s1") # x will be 's1'  
y = str(2)    # y will be '2'  
z = str(3.0)  # z will be '3.0'

# Python Strings

## **(a). 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:

### Example

print("Hello")  
print('Hello')

## **Assign String to a Variable**

Assigning a string to a variable is done with the variable name followed by an equal sign and the string:

### Example

a = "Hello"  
print(a)

## **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)

Or three single quotes:

### Example

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

**Note:** in the result, the line breaks are inserted at the same position as in the code.

## **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)

Learn more about For Loops in our [Python For Loops](https://www.w3schools.com/python/python_for_loops.asp) chapter.

## **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)  
Use it in an if statement:

### Example

Print only if "free" is present:

txt = "The best things in life are free!"  
if "free" in txt:  
  print("Yes, 'free' is present.")

Learn more about If statements in our [Python If...Else](https://www.w3schools.com/python/python_conditions.asp) chapter.

## **Check if NOT**

To check if a certain phrase or character is NOT present in a string, we can use the keyword not in.

### Example

Check if "expensive" is NOT present in the following text:

txt = "The best things in life are free!"  
print("expensive" not in txt)

Use it in an if statement:

### Example

print only if "expensive" is NOT present:

txt = "The best things in life are free!"  
if "expensive" not in txt:  
  print("Yes, 'expensive' is NOT present.")

# (b). Python - Slicing Strings

# Slicing

You can return a range of characters by using the slice syntax.

Specify the start index and the end index, separated by a colon, to return a part of the string.

### Example

Get the characters from position 2 to position 5 (not included):

b = "Hello, World!"  
print(b[2:5])

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

## **Slice From the Start**

By leaving out the start index, the range will start at the first character:

### Example

Get the characters from the start to position 5 (not included):

b = "Hello, World!"  
print(b[:5])

**Slice To the End**

By leaving out the end index, the range will go to the end:

### Example

Get the characters from position 2, and all the way to the end:

b = "Hello, World!"  
print(b[2:])

## **Negative Indexing**

Use negative indexes to start the slice from the end of the string:

### Example

Get the characters:

From: "o" in "World!" (position -5)

To, but not included: "d" in "World!" (position -2):

b = "Hello, World!"  
print(b[-5:-2])

# (c). Python - Modify Strings

Python has a set of built-in methods that you can use on strings.

## **Upper Case**

### Example

The upper() method returns the string in upper case:

a = "Hello, World!"  
print(a.upper())

## **Lower Case**

### Example

The lower() method returns the string in lower case:

a = "Hello, World!"  
print(a.lower())

**Remove Whitespace**

Whitespace is the space before and/or after the actual text, and very often you want to remove this space.

### Example

The strip() method removes any whitespace from the beginning or the end:

a = " Hello, World! "  
print(a.strip()) # returns "Hello, World!"

## **Replace String**

### Example

The replace() method replaces a string with another string:

a = "Hello, World!"  
print(a.replace("H", "J"))

## **Split String**

The split() method returns a list where the text between the specified separator becomes the list items.

### Example

The split() method splits the string into substrings if it finds instances of the separator:

a = "Hello, World!"  
print(a.split(",")) # returns ['Hello', ' World!']

Learn more about Lists in our [Python Lists](https://www.w3schools.com/python/python_lists.asp) chapter.

## **String Methods**

# (d). Python - String Concatenation

## **String Concatenation**

To concatenate, or combine, two strings you can use the + operator.

### Example

Merge variable a with variable b into variable c:

a = "Hello"  
b = "World"  
c = a + b  
print(c)

### Example

To add a space between them, add a " ":

a = "Hello"  
b = "World"  
c = a + " " + b  
print(c)

# (e). Python - Format - Strings

## **String Format**

As we learned in the Python Variables chapter, we cannot combine strings and numbers like this:

### Example

age = 36  
txt = "My name is John, I am " + age  
print(txt)

But we can combine strings and numbers by using the format() method!

The format() method takes the passed arguments, formats them, and places them in the string where the placeholders {} are:

### Example

Use the format() method to insert numbers into strings:

age = 36  
txt = "My name is John, and I am {}"  
print(txt.format(age))

The format() method takes unlimited number of arguments, and are placed into the respective placeholders:

### Example

quantity = 3  
itemno = 567  
price = 49.95  
myorder = "I want {} pieces of item {} for {} dollars."  
print(myorder.format(quantity, itemno, price))

You can use index numbers {0} to be sure the arguments are placed in the correct placeholders:

### Example

quantity = 3  
itemno = 567  
price = 49.95  
myorder = "I want to pay {2} dollars for {0} pieces of item {1}."  
print(myorder.format(quantity, itemno, price))

# (f). Python - Escape Characters

## **Escape Character**

To insert characters that are illegal in a string, use an escape character.

An escape character is a backslash \ followed by the character you want to insert.

An example of an illegal character is a double quote inside a string that is surrounded by double quotes:

### Example

You will get an error if you use double quotes inside a string that is surrounded by double quotes:

txt = "We are the so-called "Vikings" from the north."

To fix this problem, use the escape character \":

### Example

The escape character allows you to use double quotes when you normally would not be allowed:

txt = "We are the so-called \"Vikings\" from the north."

## **Escape Characters**

Other escape characters used in Python:

|  |  |  |
| --- | --- | --- |
| **Code** | **Result** | **Try it** |
| \' | Single Quote | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_string_escape2) |
| \\ | Backslash | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_string_backslash) |
| \n | New Line | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_string_newline) |
| \r | Carriage Return | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_string_r) |
| \t | Tab | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_string_t) |
| \b | Backspace | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_string_b) |
| \f | Form Feed |  |
| \ooo | Octal value | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_string_octal) |
| \xhh | Hex value | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_string_hex) |

# (g). Python - String Methods

## **String Methods**

Python has a set of built-in methods that you can use on strings.

**Note:** All string methods returns new values. They do not change the original string.

|  |  |
| --- | --- |
| **Method** | **Description** |
| [capitalize()](https://www.w3schools.com/python/ref_string_capitalize.asp) | Converts the first character to upper case |
| [casefold()](https://www.w3schools.com/python/ref_string_casefold.asp) | Converts string into lower case |
| [center()](https://www.w3schools.com/python/ref_string_center.asp) | Returns a centered string |
| [count()](https://www.w3schools.com/python/ref_string_count.asp) | Returns the number of times a specified value occurs in a string |
| [encode()](https://www.w3schools.com/python/ref_string_encode.asp) | Returns an encoded version of the string |
| [endswith()](https://www.w3schools.com/python/ref_string_endswith.asp) | Returns true if the string ends with the specified value |
| [expandtabs()](https://www.w3schools.com/python/ref_string_expandtabs.asp) | Sets the tab size of the string |
| [find()](https://www.w3schools.com/python/ref_string_find.asp) | Searches the string for a specified value and returns the position of where it was found |
| [format()](https://www.w3schools.com/python/ref_string_format.asp) | Formats specified values in a string |
| format\_map() | Formats specified values in a string |
| [index()](https://www.w3schools.com/python/ref_string_index.asp) | Searches the string for a specified value and returns the position of where it was found |
| [isalnum()](https://www.w3schools.com/python/ref_string_isalnum.asp) | Returns True if all characters in the string are alphanumeric |
| [isalpha()](https://www.w3schools.com/python/ref_string_isalpha.asp) | Returns True if all characters in the string are in the alphabet |
| [isdecimal()](https://www.w3schools.com/python/ref_string_isdecimal.asp) | Returns True if all characters in the string are decimals |
| [isdigit()](https://www.w3schools.com/python/ref_string_isdigit.asp) | Returns True if all characters in the string are digits |
| [isidentifier()](https://www.w3schools.com/python/ref_string_isidentifier.asp) | Returns True if the string is an identifier |
| [islower()](https://www.w3schools.com/python/ref_string_islower.asp) | Returns True if all characters in the string are lower case |
| [isnumeric()](https://www.w3schools.com/python/ref_string_isnumeric.asp) | Returns True if all characters in the string are numeric |
| [isprintable()](https://www.w3schools.com/python/ref_string_isprintable.asp) | Returns True if all characters in the string are printable |
| [isspace()](https://www.w3schools.com/python/ref_string_isspace.asp) | Returns True if all characters in the string are whitespaces |
| [istitle()](https://www.w3schools.com/python/ref_string_istitle.asp) | Returns True if the string follows the rules of a title |
| [isupper()](https://www.w3schools.com/python/ref_string_isupper.asp) | Returns True if all characters in the string are upper case |
| [join()](https://www.w3schools.com/python/ref_string_join.asp) | Joins the elements of an iterable to the end of the string |
| [ljust()](https://www.w3schools.com/python/ref_string_ljust.asp) | Returns a left justified version of the string |
| [lower()](https://www.w3schools.com/python/ref_string_lower.asp) | Converts a string into lower case |
| [lstrip()](https://www.w3schools.com/python/ref_string_lstrip.asp) | Returns a left trim version of the string |
| [maketrans()](https://www.w3schools.com/python/ref_string_maketrans.asp) | Returns a translation table to be used in translations |
| [partition()](https://www.w3schools.com/python/ref_string_partition.asp) | Returns a tuple where the string is parted into three parts |
| [replace()](https://www.w3schools.com/python/ref_string_replace.asp) | Returns a string where a specified value is replaced with a specified value |
| [rfind()](https://www.w3schools.com/python/ref_string_rfind.asp) | Searches the string for a specified value and returns the last position of where it was found |
| [rindex()](https://www.w3schools.com/python/ref_string_rindex.asp) | Searches the string for a specified value and returns the last position of where it was found |
| [rjust()](https://www.w3schools.com/python/ref_string_rjust.asp) | Returns a right justified version of the string |
| [rpartition()](https://www.w3schools.com/python/ref_string_rpartition.asp) | Returns a tuple where the string is parted into three parts |
| [rsplit()](https://www.w3schools.com/python/ref_string_rsplit.asp) | Splits the string at the specified separator, and returns a list |
| [rstrip()](https://www.w3schools.com/python/ref_string_rstrip.asp) | Returns a right trim version of the string |
| [split()](https://www.w3schools.com/python/ref_string_split.asp) | Splits the string at the specified separator, and returns a list |
| [splitlines()](https://www.w3schools.com/python/ref_string_splitlines.asp) | Splits the string at line breaks and returns a list |
| [startswith()](https://www.w3schools.com/python/ref_string_startswith.asp) | Returns true if the string starts with the specified value |
| [strip()](https://www.w3schools.com/python/ref_string_strip.asp) | Returns a trimmed version of the string |
| [swapcase()](https://www.w3schools.com/python/ref_string_swapcase.asp) | Swaps cases, lower case becomes upper case and vice versa |
| [title()](https://www.w3schools.com/python/ref_string_title.asp) | Converts the first character of each word to upper case |
| [translate()](https://www.w3schools.com/python/ref_string_translate.asp) | Returns a translated string |
| [upper()](https://www.w3schools.com/python/ref_string_upper.asp) | Converts a string into upper case |
| [zfill()](https://www.w3schools.com/python/ref_string_zfill.asp) | Fills the string with a specified number of 0 values at the beginning |

# Python Booleans

Booleans represent one of two values: True or False.

## **Boolean Values**

In programming you often need to know if an expression is True or False.

You can evaluate any expression in Python, and get one of two answers, True or False.

When you compare two values, the expression is evaluated and Python returns the Boolean answer:

### Example

print(10 > 9)  
print(10 == 9)  
print(10 < 9)

When you run a condition in an if statement, Python returns True or False:

### Example

Print a message based on whether the condition is True or False:

a = 200  
b = 33  
  
if b > a:  
  print("b is greater than a")  
else:  
  print("b is not greater than a")

## **Evaluate Values and Variables**

The bool() function allows you to evaluate any value, and give you True or False in return,

### Example

Evaluate a string and a number:

print(bool("Hello"))  
print(bool(15))

# Python Operators

## **Python Operators**

Operators are used to perform operations on variables and values.

In the example below, we use the + operator to add together two values:

### Example

print(10 + 5)

Python divides the operators in the following groups:

* Arithmetic operators
* Assignment operators
* Comparison operators
* Logical operators
* Identity operators
* Membership operators
* Bitwise operators

## **Python Arithmetic Operators**

Arithmetic operators are used with numeric values to perform common mathematical operations:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Name** | **Example** | **Try it** |
| + | Addition | x + y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_add) |
| - | Subtraction | x - y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_sub) |
| \* | Multiplication | x \* y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_mult) |
| / | Division | x / y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_div) |
| % | Modulus | x % y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_mod) |
| \*\* | Exponentiation | x \*\* y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_exp) |
| // | Floor division | x // y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_floordiv) |

## **Python Assignment Operators**

Assignment operators are used to assign values to variables:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Example** | **Same As** | **Try it** |
| = | x = 5 | x = 5 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_ass1) |
| += | x += 3 | x = x + 3 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_ass2) |
| -= | x -= 3 | x = x - 3 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_ass3) |
| \*= | x \*= 3 | x = x \* 3 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_ass4) |
| /= | x /= 3 | x = x / 3 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_ass5) |
| %= | x %= 3 | x = x % 3 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_ass6) |
| //= | x //= 3 | x = x // 3 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_ass7) |
| \*\*= | x \*\*= 3 | x = x \*\* 3 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_ass8) |
| &= | x &= 3 | x = x & 3 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_ass9) |
| |= | x |= 3 | x = x | 3 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_ass10) |
| ^= | x ^= 3 | x = x ^ 3 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_ass11) |
| >>= | x >>= 3 | x = x >> 3 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_ass12) |
| <<= | x <<= 3 | x = x << 3 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_ass13) |

## **Python Comparison Operators**

Comparison operators are used to compare two values:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Name** | **Example** | **Try it** |
| == | Equal | x == y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_compare1) |
| != | Not equal | x != y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_compare2) |
| > | Greater than | x > y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_compare4) |
| < | Less than | x < y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_compare5) |
| >= | Greater than or equal to | x >= y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_compare6) |
| <= | Less than or equal to | x <= y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_compare7) |

## **Python Logical Operators**

Logical operators are used to combine conditional statements:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Description** | **Example** | **Try it** |
| and | Returns True if both statements are true | x < 5 and  x < 10 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_logical1) |
| Or | Returns True if one of the statements is true | x < 5 or x < 4 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_logical2) |
| not | Reverse the result, returns False if the result is true | not(x < 5 and x < 10) | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_logical3) |

## **Python Identity Operators**

Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Description** | **Example** | **Try it** |
| is | Returns True if both variables are the same object | x is y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_identity1) |
| is not | Returns True if both variables are not the same object | x is not y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_identity2) |

## **Python Membership Operators**

Membership operators are used to test if a sequence is presented in an object:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Description** | **Example** | **Try it** |
| in | Returns True if a sequence with the specified value is present in the object | x in y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_membership1) |
| not in | Returns True if a sequence with the specified value is not present in the object | x not in y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_membership2) |

## **Python Bitwise Operators**

Bitwise operators are used to compare (binary) numbers:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Name** | **Description** |
| & | AND | Sets each bit to 1 if both bits are 1 |
| | | OR | Sets each bit to 1 if one of two bits is 1 |
| ^ | XOR | Sets each bit to 1 if only one of two bits is 1 |
| ~ | NOT | Inverts all the bits |
| << | Zero fill left shift | Shift left by pushing zeros in from the right and let the leftmost bits fall off |
| >> | Signed right shift | Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off |

Top of Form

## **Test Yourself With Exercises**

## **Exercise:**

## Multiply 10 with 5, and print the result.

print(10 5)

# Python Lists

mylist = ["apple", "banana", "cherry"]

## **(a). 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:

### Example

Create a List:

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)

ADVERTISEMENT

## **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]

A list can contain different data types:

### Example

A list with strings, integers and boolean values:

list1 = ["abc", 34, True, 40, "male"]

## **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 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.

\*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.

# (b). Python - Access List Items

## **Access Items**

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

### Example

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")

# (c). Python - Change List Items

## **Change Item Value**

To change the value of a specific item, refer to the index number:

### Example

Change the second item:

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

## **Change a Range of Item Values**

To change the value of items within a specific range, define a list with the new values, and refer to the range of index numbers where you want to insert the new values:

### Example

Change the values "banana" and "cherry" with the values "blackcurrant" and "watermelon":

thislist = ["apple", "banana", "cherry", "orange", "kiwi", "mango"]  
thislist[1:3] = ["blackcurrant", "watermelon"]  
print(thislist)

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

### Example

Change the second value by replacing it with two new values:

thislist = ["apple", "banana", "cherry"]  
thislist[1:2] = ["blackcurrant", "watermelon"]  
print(thislist)

**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:

### Example

Change the second and third value by replacing it with one value:

thislist = ["apple", "banana", "cherry"]  
thislist[1:3] = ["watermelon"]  
print(thislist)

## **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)

**Note:** As a result of the example above, the list will now contain 4 items.

# (d). Python - Add List Items

## **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)

**Note:** As a result of the examples above, the lists will now contain 4 items.

## **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)

The elements will be added to the end of the list.

## **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)

# (e). Python - Remove List Items

## **Remove Specified Item**

The remove() method removes the specified item.

### Example

Remove "banana":

thislist = ["apple", "banana", "cherry"]  
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.

### Example

Remove the last item:

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

The del keyword also removes the specified index:

### Example

Remove the first item:

thislist = ["apple", "banana", "cherry"]  
del thislist[0]  
print(thislist)

The del keyword can also delete the list completely.

### Example

Delete the entire list:

thislist = ["apple", "banana", "cherry"]  
del thislist

## **Clear the List**

The clear() method empties the list.

The list still remains, but it has no content.

### Example

Clear the list content:

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

# (f). Python - Loop Lists

## **Loop Through a List**

You can loop through the list items by using a for loop:

### Example

Print all items in the list, one by one:

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

Learn more about for loops in our [Python For Loops](https://www.w3schools.com/python/python_for_loops.asp) Chapter.

## **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])

The iterable created in the example above is [0, 1, 2].

## **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 refering 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

Learn more about while loops in our [Python While Loops](https://www.w3schools.com/python/python_while_loops.asp) Chapter.

## **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]

# (g) Python - List Comprehension

## **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:

### Example

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

With list comprehension you can do all that with only one line of code:

### Example

fruits = ["apple", "banana", "cherry", "kiwi", "mango"]  
  
newlist = [x for x in fruits if "a" in 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 valuate to True.

### Example

Only accept items that are not "apple":

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

The condition if x != "apple"  will return True for all elements other than "apple", making the new list contain all fruits except "apple".

The condition is optional and can be omitted:

### Example

With no if statement:

newlist = [x for x in fruits]

### Iterable

The iterable can be any iterable object, like a list, tuple, set etc.

### Example

You can use the range() function to create an iterable:

newlist = [x for x in range(10)]

Same example, but with a condition:

### Example

Accept only numbers lower than 5:

newlist = [x for x in range(10) if x < 5]

### Expression

The expression is the current item in the iteration, but it is also the outcome, which you can manipulate before it ends up like a list item in the new list:

### Example

Set the values in the new list to upper case:

newlist = [x.upper() for x in fruits]

You can set the outcome to whatever you like:

### Example

Set all values in the new list to 'hello':

newlist = ['hello' for x in fruits]

The expression can also contain conditions, not like a filter, but as a way to manipulate the outcome:

### Example

Return "orange" instead of "banana":

newlist = [x if x != "banana" else "orange" for x in fruits]

The expression in the example above says:

"Return the item if it is not banana, if it is banana return orange".

# (h). Python - Sort Lists

## **Sort List Alphanumerically**

List objects have a sort() method that will sort the list alphanumerically, ascending, by default:

### Example

Sort the list alphabetically:

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

### Example

Sort the list numerically:

thislist = [100, 50, 65, 82, 23]  
thislist.sort()  
print(thislist)

## **Sort Descending**

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)

### Example

Sort the list descending:

thislist = [100, 50, 65, 82, 23]  
thislist.sort(reverse = True)  
print(thislist)

## **Customize Sort Function**

You can also customize your own function by using the keyword argument key = function.

The function will return a number that will be used to sort the list (the lowest number first):

### Example

Sort the list based on how close the number is to 50:

def myfunc(n):  
  return abs(n - 50)  
  
thislist = [100, 50, 65, 82, 23]  
thislist.sort(key = myfunc)  
print(thislist)

## **Case Insensitive Sort**

By default the sort() method is case sensitive, resulting in all capital letters being sorted before lower case letters:

### Example

Case sensitive sorting can give an unexpected result:

thislist = ["banana", "Orange", "Kiwi", "cherry"]  
thislist.sort()  
print(thislist)

Luckily we can use built-in functions as key functions when sorting a list.

So if you want a case-insensitive sort function, use str.lower as a key function:

### Example

Perform a case-insensitive sort of the list:

thislist = ["banana", "Orange", "Kiwi", "cherry"]  
thislist.sort(key = str.lower)  
print(thislist)

## **Reverse Order**

What if you want to reverse the order of a list, regardless of the alphabet?

The reverse() method reverses the current sorting order of the elements.

### Example

Reverse the order of the list items:

thislist = ["banana", "Orange", "Kiwi", "cherry"]  
thislist.reverse()  
print(thislist)

# (i). Python - Copy Lists

## **Copy a List**

You cannot copy a list simply by typing list2 = list1, because: list2 will only be a reference to list1, and changes made in list1 will automatically also be made in list2.

There are ways to make a copy, one way is to use the built-in List method copy().

### Example

Make a copy of a list with the copy() method:

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

Another way to make a copy is to use the built-in method list().

### Example

Make a copy of a list with the list() method:

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

# (j). Python - Join Lists

## **Join Two Lists**

There are several ways to join, or concatenate, two or more lists in Python.

One of the easiest ways are by using the + operator.

### Example

Join two list:

list1 = ["a", "b", "c"]  
list2 = [1, 2, 3]  
  
list3 = list1 + list2  
print(list3)

Another way to join two lists is by appending all the items from list2 into list1, one by one:

### Example

Append list2 into list1:

list1 = ["a", "b" , "c"]  
list2 = [1, 2, 3]  
  
for x in list2:  
  list1.append(x)  
  
print(list1)

Or you can use the extend() method, which purpose is to add elements from one list to another list:

### Example

Use the extend() method to add list2 at the end of list1:

list1 = ["a", "b" , "c"]  
list2 = [1, 2, 3]  
  
list1.extend(list2)  
print(list1)

# (k). Python - List Methods

## **List Methods**

Python has a set of built-in methods that you can use on lists.

|  |  |
| --- | --- |
| **Method** | **Description** |
| [append()](https://www.w3schools.com/python/ref_list_append.asp) | Adds an element at the end of the list |
| [clear()](https://www.w3schools.com/python/ref_list_clear.asp) | Removes all the elements from the list |
| [copy()](https://www.w3schools.com/python/ref_list_copy.asp) | Returns a copy of the list |
| [count()](https://www.w3schools.com/python/ref_list_count.asp) | Returns the number of elements with the specified value |
| [extend()](https://www.w3schools.com/python/ref_list_extend.asp) | Add the elements of a list (or any iterable), to the end of the current list |
| [index()](https://www.w3schools.com/python/ref_list_index.asp) | Returns the index of the first element with the specified value |
| [insert()](https://www.w3schools.com/python/ref_list_insert.asp) | Adds an element at the specified position |
| [pop()](https://www.w3schools.com/python/ref_list_pop.asp) | Removes the element at the specified position |
| [remove()](https://www.w3schools.com/python/ref_list_remove.asp) | Removes the item with the specified value |
| [reverse()](https://www.w3schools.com/python/ref_list_reverse.asp) | Reverses the order of the list |
| [sort()](https://www.w3schools.com/python/ref_list_sort.asp) | Sorts the list |

# (l). Python List Exercises

## **Test Yourself With Exercises**

Now you have learned a lot about lists, and how to use them in Python.

Are you ready for a test?

Try to insert the missing part to make the code work as expected:

Top of Form

## **Exercise:**

Print the second item in the fruits list.

fruits = ["apple", "banana", "cherry"]

print()

Bottom of Form

Go to the Exercise section and test all of our Python List Exercises:

Bottom of Form