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.
* 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

print("Hello, World!")

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

## 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)

### 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))

## 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

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"

## 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)

## 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)

## 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)

## 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)

## 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 |  |

## 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)

## 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)

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.")

## 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("No, 'expensive' is NOT present.")

## 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])

## 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])

## 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!']

## 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)

## String Format

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

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))

## 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:

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 |  |

## 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 |

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

## 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)

## 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.

## 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])

### 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])

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")

## 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)

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)

## 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)

## 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)

## 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)

## 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 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

## 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:

### 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]

## 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)

## 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)

## 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)

## 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 |

## Tuple

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

Tuple is one of 4 built-in data types in Python used to store collections of data, the other 3 are [List](https://www.w3schools.com/python/python_lists.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.

A tuple is a collection which is ordered and **unchangeable**.

Tuples are written with round brackets.

### Example

Create a Tuple:

thistuple = ("apple", "banana", "cherry")  
print(thistuple)

## Tuple Items

Tuple items are ordered, unchangeable, and allow duplicate values.

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

## Ordered

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

## Unchangeable

Tuples are unchangeable, meaning that we cannot change, add or remove items after the tuple has been created.

## Allow Duplicates

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

### Example

Tuples allow duplicate values:

thistuple = ("apple", "banana", "cherry", "apple", "cherry")  
print(thistuple)

## Tuple Length

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

### Example

Print the number of items in the tuple:

thistuple = ("apple", "banana", "cherry")  
print(len(thistuple))

## Create Tuple With One Item

To create a tuple with only one item, you have to add a comma after the item, otherwise Python will not recognize it as a tuple.

### Example

One item tuple, remember the comma:

thistuple = ("apple",)  
print(type(thistuple))  
  
#NOT a tuple  
thistuple = ("apple")  
print(type(thistuple))

## Tuple Items - Data Types

Tuple items can be of any data type:

### Example

String, int and boolean data types:

tuple1 = ("apple", "banana", "cherry")  
tuple2 = (1, 5, 7, 9, 3)  
tuple3 = (True, False, False)

A tuple can contain different data types:

### Example

A tuple with strings, integers and boolean values:

tuple1 = ("abc", 34, True, 40, "male")

## type()

From Python's perspective, tuples are defined as objects with the data type 'tuple':

<class 'tuple'>

### Example

What is the data type of a tuple?

mytuple = ("apple", "banana", "cherry")  
print(type(mytuple)

## The tuple() Constructor

It is also possible to use the tuple() constructor to make a tuple.

### Example

Using the tuple() method to make a tuple:

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

Python Collections (Arrays)

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

* [**List**](https://www.w3schools.com/python/python_lists.asp) is a collection which is ordered and changeable. Allows duplicate members.
* **Tuple** 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.

## Access Tuple Items

You can access tuple items by referring to the index number, inside square brackets:

### Example

Print the second item in the tuple:

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

## 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 tuple:

thistuple = ("apple", "banana", "cherry")  
print(thistuple[-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 tuple with the specified items.

### Example

Return the third, fourth, and fifth item:

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

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 included, "kiwi":

thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")  
print(thistuple[: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" and to the end:

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

Tuples are unchangeable, meaning that you cannot change, add, or remove items once the tuple is created.

But there are some workarounds.

## Change Tuple Values

Once a tuple is created, you cannot change its values. Tuples are **unchangeable**, or **immutable** as it also is called.

But there is a workaround. You can convert the tuple into a list, change the list, and convert the list back into a tuple.

### Example

Convert the tuple into a list to be able to change it:

x = ("apple", "banana", "cherry")  
y = list(x)  
y[1] = "kiwi"  
x = tuple(y)  
  
print(x)

## Add Items

Since tuples are immutable, they do not have a build-in append() method, but there are other ways to add items to a tuple.

1. **Convert into a list**: Just like the workaround for changing a tuple, you can convert it into a list, add your item(s), and convert it back into a tuple.

### Example

Convert the tuple into a list, add "orange", and convert it back into a tuple:

thistuple = ("apple", "banana", "cherry")  
y = list(thistuple)  
y.append("orange")  
thistuple = tuple(y)

2. **Add tuple to a tuple**. You are allowed to add tuples to tuples, so if you want to add one item, (or many), create a new tuple with the item(s), and add it to the existing tuple:

### Example

Create a new tuple with the value "orange", and add that tuple:

thistuple = ("apple", "banana", "cherry")  
y = ("orange",)  
thistuple += y  
  
print(thistuple)

Tuples are **unchangeable**, so you cannot remove items from it, but you can use the same workaround as we used for changing and adding tuple items:

### Example

Convert the tuple into a list, remove "apple", and convert it back into a tuple:

thistuple = ("apple", "banana", "cherry")  
y = list(thistuple)  
y.remove("apple")  
thistuple = tuple(y)

Or you can delete the tuple completely:

### Example

The del keyword can delete the tuple completely:

thistuple = ("apple", "banana", "cherry")  
del thistuple  
print(thistuple) #this will raise an error because the tuple no longer exists

## Unpacking a Tuple

When we create a tuple, we normally assign values to it. This is called "packing" a tuple:

### Example

Packing a tuple:

fruits = ("apple", "banana", "cherry")

But, in Python, we are also allowed to extract the values back into variables. This is called "unpacking":

### Example

Unpacking a tuple:

fruits = ("apple", "banana", "cherry")  
  
(green, yellow, red) = fruits  
  
print(green)  
print(yellow)  
print(red)

## Using Asterisk\*

If the number of variables is less than the number of values, you can add an \* to the variable name and the values will be assigned to the variable as a list:

### Example

Assign the rest of the values as a list called "red":

fruits = ("apple", "banana", "cherry", "strawberry", "raspberry")  
  
(green, yellow, \*red) = fruits  
  
print(green)  
print(yellow)  
print(red)

If the asterisk is added to another variable name than the last, Python will assign values to the variable until the number of values left matches the number of variables left.

### Example

Add a list of values the "tropic" variable:

fruits = ("apple", "mango", "papaya", "pineapple", "cherry")  
  
(green, \*tropic, red) = fruits  
  
print(green)  
print(tropic)  
print(red)

## Loop Through a Tuple

You can loop through the tuple items by using a for loop.

### Example

Iterate through the items and print the values:

thistuple = ("apple", "banana", "cherry")  
for x in thistuple:  
  print(x)

## Loop Through the Index Numbers

You can also loop through the tuple 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:

thistuple = ("apple", "banana", "cherry")  
for i in range(len(thistuple)):  
  print(thistuple[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 tuple, then start at 0 and loop your way through the tuple 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:

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

## Join Two Tuples

To join two or more tuples you can use the + operator:

### Example

Join two tuples:

tuple1 = ("a", "b" , "c")  
tuple2 = (1, 2, 3)  
  
tuple3 = tuple1 + tuple2  
print(tuple3)

## Multiply Tuples

If you want to multiply the content of a tuple a given number of times, you can use the \* operator:

### Example

Multiply the fruits tuple by 2:

fruits = ("apple", "banana", "cherry")  
mytuple = fruits \* 2  
  
print(mytuple)

## Tuple Methods

Python has two built-in methods that you can use on tuples.

|  |  |
| --- | --- |
| **Method** | **Description** |
| [count()](https://www.w3schools.com/python/ref_tuple_count.asp) | Returns the number of times a specified value occurs in a tuple |
| [index()](https://www.w3schools.com/python/ref_tuple_index.asp) | Searches the tuple for a specified value and returns the position of where it was found |

Python Conditions and If statements

Python supports the usual logical conditions from mathematics:

* Equals: a == b
* Not Equals: a != b
* Less than: a < b
* Less than or equal to: a <= b
* Greater than: a > b
* Greater than or equal to: a >= b

These conditions can be used in several ways, most commonly in "if statements" and loops.

An "if statement" is written by using the if keyword.

Example

If statement:

a = 33  
b = 200  
if b > a:  
  print("b is greater than a")

In this example we use two variables, a and b, which are used as part of the if statement to test whether b is greater than a. As a is 33, and b is 200, we know that 200 is greater than 33, and so we print to screen that "b is greater than a".

## Elif

The elif keyword is pythons way of saying "if the previous conditions were not true, then try this condition".

### Example

a = 33  
b = 33  
if b > a:  
  print("b is greater than a")  
elif a == b:  
  print("a and b are equal")

In this example a is equal to b, so the first condition is not true, but the elif condition is true, so we print to screen that "a and b are equal".

## Else

The else keyword catches anything which isn't caught by the preceding conditions.

### Example

a = 200  
b = 33  
if b > a:  
  print("b is greater than a")  
elif a == b:  
  print("a and b are equal")  
else:  
  print("a is greater than b")

In this example a is greater than b, so the first condition is not true, also the elif condition is not true, so we go to the else condition and print to screen that "a is greater than b".

You can also have an else without the elif:

### Example

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

## Short Hand If

If you have only one statement to execute, you can put it on the same line as the if statement.

### Example

One line if statement:

if a > b: print("a is greater than b")

## Short Hand If ... Else

If you have only one statement to execute, one for if, and one for else, you can put it all on the same line:

### Example

One line if else statement:

a = 2  
b = 330  
print("A") if a > b else print("B")

You can also have multiple else statements on the same line:

### Example

One line if else statement, with 3 conditions:

a = 330  
b = 330  
print("A") if a > b else print("=") if a == b else print("B")

## And

The and keyword is a logical operator, and is used to combine conditional statements:

### Example

Test if a is greater than b, AND if c is greater than a:

a = 200  
b = 33  
c = 500  
if a > b and c > a:  
  print("Both conditions are True")

## Or

The or keyword is a logical operator, and is used to combine conditional statements:

### Example

Test if a is greater than b, OR if a is greater than c:

a = 200  
b = 33  
c = 500  
if a > b or a > c:  
  print("At least one of the conditions is True")

## Nested If

You can have if statements inside if statements, this is called nested if statements.

### Example

x = 41  
  
if x > 10:  
  print("Above ten,")  
  if x > 20:  
    print("and also above 20!")  
  else:  
    print("but not above 20.")

## The pass Statement

if statements cannot be empty, but if you for some reason have an if statement with no content, put in the pass statement to avoid getting an error.

### Example

a = 33  
b = 200  
  
if b > a:  
  pass

Python Loops

Python has two primitive loop commands:

* while loops
* for loops

The while Loop

With the while loop we can execute a set of statements as long as a condition is true.

Example

Print i as long as i is less than 6:

i = 1  
while i < 6:  
  print(i)  
  i += 1

The while loop requires relevant variables to be ready, in this example we need to define an indexing variable, i, which we set to 1.

## The break Statement

With the break statement we can stop the loop even if the while condition is true:

### Example

Exit the loop when i is 3:

i = 1  
while i < 6:  
  print(i)  
  if i == 3:  
    break  
  i += 1

## The continue Statement

With the continue statement we can stop the current iteration, and continue with the next:

### Example

Continue to the next iteration if i is 3:

i = 0  
while i < 6:  
  i += 1  
  if i == 3:  
    continue  
  print(i)

## The else Statement

With the else statement we can run a block of code once when the condition no longer is true:

### Example

Print a message once the condition is false:

i = 1  
while i < 6:  
  print(i)  
  i += 1  
else:  
  print("i is no longer less than 6")

## Python For Loops

A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

This is less like the for keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.

With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.

### Example

Print each fruit in a fruit list:

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

## Looping Through a String

Even strings are iterable objects, they contain a sequence of characters:

### Example

Loop through the letters in the word "banana":

for x in "banana":  
  print(x)

## The break Statement

With the break statement we can stop the loop before it has looped through all the items:

### Example

Exit the loop when x is "banana":

fruits = ["apple", "banana", "cherry"]  
for x in fruits:  
  print(x)  
  if x == "banana":  
    break

### Example

Exit the loop when x is "banana", but this time the break comes before the print:

fruits = ["apple", "banana", "cherry"]  
for x in fruits:  
  if x == "banana":  
    break  
  print(x)

## The continue Statement

With the continue statement we can stop the current iteration of the loop, and continue with the next:

### Example

Do not print banana:

fruits = ["apple", "banana", "cherry"]  
for x in fruits:  
  if x == "banana":  
    continue  
  print(x)

## The range() Function

To loop through a set of code a specified number of times, we can use the range() function,

The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

### Example

Using the range() function:

for x in range(6):  
  print(x)

The range() function defaults to 0 as a starting value, however it is possible to specify the starting value by adding a parameter: range(2, 6), which means values from 2 to 6 (but not including 6):

### Example

Using the start parameter:

for x in range(2, 6):  
  print(x)

The range() function defaults to increment the sequence by 1, however it is possible to specify the increment value by adding a third parameter: range(2, 30, **3**):

### Example

Increment the sequence with 3 (default is 1):

for x in range(2, 30, 3):  
  print(x)

## Else in For Loop

The else keyword in a for loop specifies a block of code to be executed when the loop is finished:

### Example

Print all numbers from 0 to 5, and print a message when the loop has ended:

for x in range(6):  
  print(x)  
else:  
  print("Finally finished!")

### Example

Break the loop when x is 3, and see what happens with the else block:

for x in range(6):  
  if x == 3: break  
  print(x)  
else:  
  print("Finally finished!")

## Nested Loops

A nested loop is a loop inside a loop.

The "inner loop" will be executed one time for each iteration of the "outer loop":

### Example

Print each adjective for every fruit:

adj = ["red", "big", "tasty"]  
fruits = ["apple", "banana", "cherry"]  
  
for x in adj:  
  for y in fruits:  
    print(x, y)

## The pass Statement

for loops cannot be empty, but if you for some reason have a for loop with no content, put in the pass statement to avoid getting an error.

### Example

for x in [0, 1, 2]:  
  pass

A function is a block of code which only runs when it is called.

You can pass data, known as parameters, into a function.

A function can return data as a result.

## Creating a Function

In Python a function is defined using the def keyword:

### Example

def my\_function():  
  print("Hello from a function")

## Calling a Function

To call a function, use the function name followed by parenthesis:

### Example

def my\_function():  
  print("Hello from a function")  
  
**my\_function()**

## Arguments

Information can be passed into functions as arguments.

Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

The following example has a function with one argument (fname). When the function is called, we pass along a first name, which is used inside the function to print the full name:

### Example

def my\_function(**fname**):  
  print(fname + " Refsnes")  
  
my\_function(**"Emil"**)  
my\_function(**"Tobias"**)  
my\_function(**"Linus"**)

## Number of Arguments

By default, a function must be called with the correct number of arguments. Meaning that if your function expects 2 arguments, you have to call the function with 2 arguments, not more, and not less.

### Example

This function expects 2 arguments, and gets 2 arguments:

def my\_function(fname, lname):  
  print(fname + " " + lname)  
  
my\_function("Emil", "Refsnes")

## Arbitrary Arguments, \*args

If you do not know how many arguments that will be passed into your function, add a \* before the parameter name in the function definition.

This way the function will receive a tuple of arguments, and can access the items accordingly:

### Example

If the number of arguments is unknown, add a \* before the parameter name:

def my\_function(\*kids):  
  print("The youngest child is " + kids[2])  
  
my\_function("Emil", "Tobias", "Linus")

## Keyword Arguments

You can also send arguments with the key = value syntax.

This way the order of the arguments does not matter.

### Example

def my\_function(child3, child2, child1):  
  print("The youngest child is " + child3)  
  
my\_function(child1 = "Emil", child2 = "Tobias", child3 = "Linus")

## Arbitrary Keyword Arguments, \*\*kwargs

If you do not know how many keyword arguments that will be passed into your function, add two asterisk: \*\* before the parameter name in the function definition.

This way the function will receive a dictionary of arguments, and can access the items accordingly:

### Example

If the number of keyword arguments is unknown, add a double \*\* before the parameter name:

def my\_function(\*\*kid):  
  print("His last name is " + kid["lname"])  
  
my\_function(fname = "Tobias", lname = "Refsnes")

## Default Parameter Value

The following example shows how to use a default parameter value.

If we call the function without argument, it uses the default value:

### Example

def my\_function(**country = "Norway"**):  
  print("I am from " + country)  
  
my\_function("Sweden")  
my\_function("India")  
my\_function()  
my\_function("Brazil")

## Passing a List as an Argument

You can send any data types of argument to a function (string, number, list, dictionary etc.), and it will be treated as the same data type inside the function.

E.g. if you send a List as an argument, it will still be a List when it reaches the function:

### Example

## def my\_function(food):   for x in food:     print(x) fruits = ["apple", "banana", "cherry"] my\_function(fruits) Return Values

To let a function return a value, use the return statement:

### Example

def my\_function(x):  
  **return 5 \* x**  
print(my\_function(3))  
print(my\_function(5))  
print(my\_function(9))

## The pass Statement

function definitions cannot be empty, but if you for some reason have a function definition with no content, put in the pass statement to avoid getting an error.

### Example

def myfunction():  
  pass

## Recursion

Python also accepts function recursion, which means a defined function can call itself.

Recursion is a common mathematical and programming concept. It means that a function calls itself. This has the benefit of meaning that you can loop through data to reach a result.

The developer should be very careful with recursion as it can be quite easy to slip into writing a function which never terminates, or one that uses excess amounts of memory or processor power. However, when written correctly recursion can be a very efficient and mathematically-elegant approach to programming.

In this example, tri\_recursion() is a function that we have defined to call itself ("recurse"). We use the k variable as the data, which decrements (-1) every time we recurse. The recursion ends when the condition is not greater than 0 (i.e. when it is 0).

To a new developer it can take some time to work out how exactly this works, best way to find out is by testing and modifying it.

### Example

Recursion Example

def tri\_recursion(k):  
  if(k > 0):  
    result = k + tri\_recursion(k - 1)  
    print(result)  
  else:  
    result = 0  
  return result  
  
print("\n\nRecursion Example Results")  
tri\_recursion(6)

File handling is an important part of any web application.

Python has several functions for creating, reading, updating, and deleting files.

## File Handling

The key function for working with files in Python is the open() function.

The open() function takes two parameters; filename, and mode.

There are four different methods (modes) for opening a file:

"r" - Read - Default value. Opens a file for reading, error if the file does not exist

"a" - Append - Opens a file for appending, creates the file if it does not exist

"w" - Write - Opens a file for writing, creates the file if it does not exist

"x" - Create - Creates the specified file, returns an error if the file exists

In addition you can specify if the file should be handled as binary or text mode

"t" - Text - Default value. Text mode

"b" - Binary - Binary mode (e.g. images)

## Syntax

To open a file for reading it is enough to specify the name of the file:

f = open("demofile.txt")

The code above is the same as:

f = open("demofile.txt", "rt")

Because "r" for read, and "t" for text are the default values, you do not need to specify them.

## Open a File on the Server

Assume we have the following file, located in the same folder as Python:

demofile.txt

Hello! Welcome to demofile.txt  
This file is for testing purposes.  
Good Luck!

To open the file, use the built-in open() function.

The open() function returns a file object, which has a read() method for reading the content of the file:

### Example

f = open("demofile.txt", "r")  
print(f.read())

If the file is located in a different location, you will have to specify the file path, like this:

### Example

Open a file on a different location:

f = open("D:\\myfiles\welcome.txt", "r")  
print(f.read())

## Read Only Parts of the File

By default the read() method returns the whole text, but you can also specify how many characters you want to return:

### Example

Return the 5 first characters of the file:

f = open("demofile.txt", "r")  
print(f.read(**5**))

## Read Lines

You can return one line by using the readline() method:

### Example

Read one line of the file:

f = open("demofile.txt", "r")  
print(f.readline())

By calling readline() two times, you can read the two first lines:

### Example

Read two lines of the file:

f = open("demofile.txt", "r")  
print(f.readline())  
print(f.readline())

By looping through the lines of the file, you can read the whole file, line by line:

### Example

Loop through the file line by line:

f = open("demofile.txt", "r")  
for x in f:  
  print(x)

## Close Files

It is a good practice to always close the file when you are done with it.

### Example

Close the file when you are finish with it:

f = open("demofile.txt", "r")  
print(f.readline())  
f.close()

## Write to an Existing File

To write to an existing file, you must add a parameter to the open() function:

"a" - Append - will append to the end of the file

"w" - Write - will overwrite any existing content

### Example

Open the file "demofile2.txt" and append content to the file:

f = open("demofile2.txt", "a")  
f.write("Now the file has more content!")  
f.close()  
  
#open and read the file after the appending:  
f = open("demofile2.txt", "r")  
print(f.read())

### Example

Open the file "demofile3.txt" and overwrite the content:

f = open("demofile3.txt", "w")  
f.write("Woops! I have deleted the content!")  
f.close()  
  
#open and read the file after the appending:  
f = open("demofile3.txt", "r")  
print(f.read())

## Create a New File

To create a new file in Python, use the open() method, with one of the following parameters:

"x" - Create - will create a file, returns an error if the file exist

"a" - Append - will create a file if the specified file does not exist

"w" - Write - will create a file if the specified file does not exist

### Example

Create a file called "myfile.txt":

f = open("myfile.txt", "x")

Result: a new empty file is created!

### Example

Create a new file if it does not exist:

f = open("myfile.txt", "w")

## Delete a File

To delete a file, you must import the OS module, and run its os.remove() function:

### Example

Remove the file "demofile.txt":

import os  
os.remove("demofile.txt")

## Check if File exist:

To avoid getting an error, you might want to check if the file exists before you try to delete it:

### Example

Check if file exists, then delete it:

import os  
if os.path.exists("demofile.txt"):  
  os.remove("demofile.txt")  
else:  
  print("The file does not exist")

## Delete Folder

To delete an entire folder, use the os.rmdir() method:

### Example

Remove the folder "myfolder":

import os  
os.rmdir("myfolder")