# Python Functions

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 + " Welcome")  
  
my\_function(**"Dear Student"**)  
my\_function(**"Ahmad"**)  
my\_function(**"Alizay"**)

Arguments are often shortened to args in Python documentations.

## Parameters or Arguments?

The terms parameter and argument can be used for the same thing: information that are passed into a function.

From a function's perspective:

A parameter is the variable listed inside the parentheses in the function definition.

An argument is the value that is sent to the function when it is called.

## 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("Ahmad", "Alizay")

If you try to call the function with 1 or 3 arguments, you will get an error:

### Example

This function expects 2 arguments, but gets only 1:

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

## 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("Rasheed", "Minhas", "Ambreen")

Arbitrary Arguments are often shortened to \*args in Python documentations.

## 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 = " Rasheed ", child2 = " Minhas ", child3 = " Ambreen ")

The phrase Keyword Arguments are often shortened to kwargs in Python documentations.

## 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 = "Ambreen", lname = "Ahmad")

Arbitrary Kword Arguments are often shortened to \*\*kwargs in Python documentations.

## 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(**city = "Islamabad"**):  
  print("I am from " + city)  
  
my\_function("Swat")  
my\_function("Karachi")  
my\_function()  
my\_function("Lahore")

## 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

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")

# Python File Write

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

### Example

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

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

**Note:** the "w" method will overwrite the entire file.

# Python File Open

## Open a File on the Server

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

myfile.txt

Hello! Welcome to myfile.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("myfile.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("myfile.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("myfile.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("myfile.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("myfile.txt", "r")  
print(f.readline())  
f.close()

**Note:** You should always close your files, in some cases, due to buffering, changes made to a file may not show until you close the file.

# Python File Open

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.

**Note:** Make sure the file exists, or else you will get an error.

# Python Delete File

[❮ Previous](https://www.w3schools.com/python/python_file_write.asp)[Next ❯](https://www.w3schools.com/python/numpy_intro.asp)

## 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")

**Note:** You can only remove empty folders