

Get the path of running file (.py) in Python: `__file__`

In Python, you can get the location (path) of the running script file `.py` with `__file__`. `__file__` is useful for reading other files based on the location of the running file.

In Python 3.8 and earlier, `__file__` returns the path specified when executing the `python` (or `python3`) command. If you specify a relative path, a relative path is returned. If you specify an absolute path, an absolute path is returned.

In Python 3.9 and later, `__file__` always returns an absolute path, regardless of whether the path specified with the `python` command is relative or absolute.

This article describes the following contents.

- `os.getcwd()` and `__file__`
- Get the file name and the directory name of the running file
- Get the absolute path of the running file
- Read other files based on the location of the running file
- Change the current directory to the directory of the running file
- Read the same file regardless of the current working directory

Refer to the following article for how to get and change the current working directory.

- [Get and change the current working directory in Python](#)

Note that `__file__` cannot be used in Jupyter Notebook (`.ipynb`). Regardless of the directory where Jupyter Notebook is started, the current directory is the directory where `.ipynb` is located. It is

possible to change the current directory using `os.chdir ()` in the code.

Sponsored Link

os.getcwd() and `__file__`

Suppose you work in the following directory. On Windows, you can check the current directory with the `dir` command instead of `pwd`.

```
pwd
# /Users/mbp/Documents/my-project/python-snippets/notebook
```


Create a Python script file (`file_path.py`) with the following code in the lower directory (`data/src`).

```
import os

print('getcwd:      ', os.getcwd())
print('__file__':    ', __file__')
```

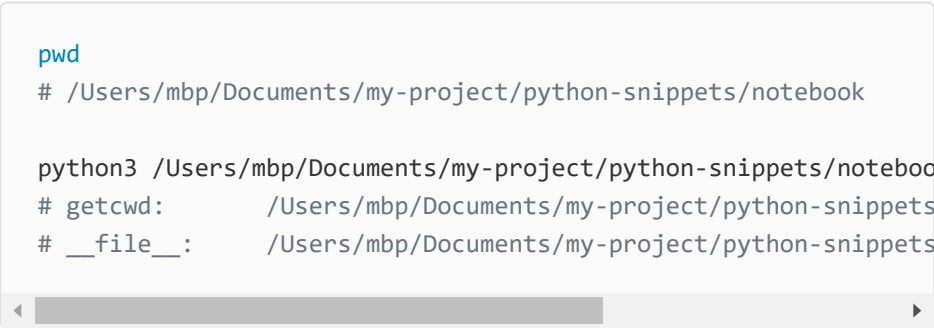
Run it with the `python` (or `python3`) command.

```
python3 data/src/file_path.py
# cwd:      /Users/mbp/Documents/my-project/python-snippets
# __file__: data/src/file_path.py
```



You can get the absolute path of the current working directory with `os.getcwd()` and the path specified with the `python3` command with `__file__`.

In Python 3.8 and earlier, the path specified by the `python` (or `python3`) command is stored in `__file__`. In the above example, a relative path is specified, so a relative path is returned, but if an absolute path is specified, an absolute path is returned.



```
pwd
# /Users/mbp/Documents/my-project/python-snippets/notebook

python3 /Users/mbp/Documents/my-project/python-snippets/notebook
# getcwd:      /Users/mbp/Documents/my-project/python-snippets
# __file__:    /Users/mbp/Documents/my-project/python-snippets
```

In Python 3.9 and later, `__file__` always returns an absolute path, regardless of whether the path specified with the `python` command is relative or absolute.

In the following examples, add code to the same script file (`file_path.py`), and execute from the same directory as the above example in Python3.7.

Note that if `__file__` is an absolute path (when Python 3.9 or later, or when specifying with the absolute path in Python 3.8 or earlier), you can use the same code as described below to read other files based on the location of the running script file. The results of executing by specifying the absolute path in Python3.7 are shown at the end.

Get the file name and the directory name of the running file

Use `os.path.basename()`, `os.path.dirname()` to get the file name and the directory name of the running file.

```
print('basename:      ', os.path.basename(__file__))
print('dirname:       ', os.path.dirname(__file__))
```

The result is as follows.

```
# basename:      file_path.py
# dirname:       data/src
```

See the following article for details

on `os.path.basename()`, `os.path.dirname()`, etc.

- [Extract the file, dir, extension name from a path string in Python](#)

Get the absolute path of the running file

If you get the relative path with `__file__`, you can convert it to an absolute path with `os.path.abspath()`.

```
print('abspath:      ', os.path.abspath(__file__))  
print('abs dirname: ', os.path.dirname(os.path.abspath(__file__)))
```

The result is as follows.

```
# abspath:      /Users/mbp/Documents/my-project/python-snippets  
# abs dirname:  /Users/mbp/Documents/my-project/python-snippets
```

If you specify an absolute path in `os.path.abspath()`, it will be returned as it is, so if `__file__` is an absolute path, no error will occur even if you set `os.path.abspath(__file__)`.

Read other files based on the location of the running file

If you want to read other files based on the location (path) of the running file, join the directory path of the running file and the relative path (from the running file) of the file you want to read with `os.path.join()`.

Note that files in the same directory as the running file can be read by specifying only the file name.

```
print('[set target path 1]')
target_path_1 = os.path.join(os.path.dirname(__file__), 'target

print('target_path_1: ', target_path_1)

print('read target file:')
with open(target_path_1) as f:
    print(f.read())
```

The result is as follows.

```
# [set target path 1]
# target_path_1:  data/src/target_1.txt
# read target file:
# !! This is "target_1.txt" !!
```

The upper directory is represented by `../`. There is no problem as it is, but if you use `os.path.normpath()`, you can normalize the path and remove extra `../`, etc.

- `os.path.normpath()` — Common pathname manipulations — Python 3.8.5 documentation

```
print('[set target path 2]')
target_path_2 = os.path.join(os.path.dirname(__file__), '../dst

print('target_path_2: ', target_path_2)
print('normalize      : ', os.path.normpath(target_path_2))
```

```
print('read target file:')
with open(target_path_2) as f:
    print(f.read())
```

The result is as follows.

```
# [set target path 2]
# target_path_2:  data/src/../../dst/target_2.txt
# normalize      :  data/dst/target_2.txt
# read target file:
# !! This is "target_2.txt" !!
```

Change the current directory to the directory of the running file

Use `os.chdir()` to change the current working directory to the directory of the running file.

- Get and change the current working directory in Python

You can confirm that it has been changed with `os.getcwd()` .

```
print('[change directory]')
os.chdir(os.path.dirname(os.path.abspath(__file__)))
print('getcwd:      ', os.getcwd())
```

The result is as follows.

```
# [change directory]
# getcwd:          /Users/mbp/Documents/my-project/python-snippets
```



If the current working directory is the same as the directory of the running file, you can specify a relative path from the running file to read other files.

```
print('[set target path 1 (after chdir)]')
target_path_1 = 'target_1.txt'

print('target_path_1: ', target_path_1)

print('read target file:')
with open(target_path_1) as f:
    print(f.read())

print()
print('[set target path 2 (after chdir)]')
target_path_2 = '../dst/target_2.txt'

print('target_path_2: ', target_path_2)

print('read target file:')
with open(target_path_2) as f:
    print(f.read())
```

The result is as follows.


```
# [set target path 1 (after chdir)]
# target_path_1: target_1.txt
# read target file:
# !! This is "target_1.txt" !!
#
# [set target path 2 (after chdir)]
# target_path_2: ../dst/target_2.txt
# read target file:
# !! This is "target_2.txt" !!
```

Read the same file regardless of the current working directory

By using `__file__` to get the path of the running script file, the same file can be read regardless of the current working directory.

As shown so far, there are two ways:

- Join the `__file__` directory and the relative path (from `__file__`) of the file you want to read with `os.path.join()`
- Change the current working directory to `__file__` directory

It is easier to change the current directory, but of course, if you read or write files after that, you need to consider that the current directory has been changed.

The results of the examples so far are as follows.

pwd

```
# /Users/mbp/Documents/my-project/python-snippets/notebook
```

```
python3 data/src/file_path.py
```

```
# getcwd:          /Users/mbp/Documents/my-project/python-snippets
# __file__:        data/src/file_path.py
# basename:        file_path.py
# dirname:         data/src
# abspath:         /Users/mbp/Documents/my-project/python-snippets
# abs dirname:     /Users/mbp/Documents/my-project/python-snippets
#
# [set target path 1]
# target_path_1:   data/src/target_1.txt
# read target file:
# !! This is "target_1.txt" !!
#
# [set target path 2]
# target_path_2:   data/src/../dst/target_2.txt
# normalize       : data/dst/target_2.txt
# read target file:
# !! This is "target_2.txt" !!
#
# [change directory]
# getcwd:          /Users/mbp/Documents/my-project/python-snippets
#
# [set target path 1 (after chdir)]
# target_path_1:   target_1.txt
# read target file:
# !! This is "target_1.txt" !!
#
# [set target path 2 (after chdir)]
# target_path_2:   ../dst/target_2.txt
# read target file:
# !! This is "target_2.txt" !!
```

The results for specifying the absolute path with `python3` command is as follows. The same file can be read.

```
pwd
```

```
# /Users/mbp/Documents/my-project/python-snippets/notebook
```

```
python3 /Users/mbp/Documents/my-project/python-snippets/notebook
```

```
# getcwd:      /Users/mbp/Documents/my-project/python-snippets
```

```
# __file__:    /Users/mbp/Documents/my-project/python-snippets
```

```
# basename:    file_path.py
```

```
# dirname:     /Users/mbp/Documents/my-project/python-snippets
```

```
# abspath:     /Users/mbp/Documents/my-project/python-snippets
```

```
# abs dirname: /Users/mbp/Documents/my-project/python-snippets
```

```
#
```

```
# [set target path 1]
```

```
# target_path_1: /Users/mbp/Documents/my-project/python-snippets
```

```
# read target file:
```

```
# !! This is "target_1.txt" !!
```

```
#
```

```
# [set target path 2]
```

```
# target_path_2: /Users/mbp/Documents/my-project/python-snippets
```

```
# normalize    : /Users/mbp/Documents/my-project/python-snippets
```

```
# read target file:
```

```
# !! This is "target_2.txt" !!
```

```
#
```

```
# [change directory]
```

```
# getcwd:      /Users/mbp/Documents/my-project/python-snippets
```

```
#
```

```
# [set target path 1 (after chdir)]
```

```
# target_path_1: target_1.txt
```

```
# read target file:
```

```
# !! This is "target_1.txt" !!
```

```
#
```

```
# [set target path 2 (after chdir)]
```

```
# target_path_2: ../dst/target_2.txt
```

```
# read target file:
# !! This is "target_2.txt" !!
```

Change the current directory in the terminal and execute the same script file. You can see that the same file can be read from different locations.

```
cd data/src
```

```
pwd
```

```
# /Users/mbp/Documents/my-project/python-snippets/notebook/data
```

```
python3 file_path.py
```

```
# getcwd:      /Users/mbp/Documents/my-project/python-snippets
```

```
# __file__:    file_path.py
```

```
# basename:    file_path.py
```

```
# dirname:
```

```
# abspath:     /Users/mbp/Documents/my-project/python-snippets
```

```
# abs dirname: /Users/mbp/Documents/my-project/python-snippets
```

```
#
```

```
# [set target path 1]
```

```
# target_path_1: target_1.txt
```

```
# read target file:
```

```
# !! This is "target_1.txt" !!
```

```
#
```

```
# [set target path 2]
```

```
# target_path_2: ../dst/target_2.txt
```

```
# normalize    : ../dst/target_2.txt
```

```
# read target file:
```

```
# !! This is "target_2.txt" !!
```

```
#
```

```
# [change directory]
```

```
# getcwd:      /Users/mbp/Documents/my-project/python-snippets
```

```
#  
# [set target path 1 (after chdir)]  
# target_path_1: target_1.txt  
# read target file:  
# !! This is "target_1.txt" !!  
#  
# [set target path 2 (after chdir)]  
# target_path_2: ../dst/target_2.txt  
# read target file:  
# !! This is "target_2.txt" !!
```