

**Python Directory & File Management**

# 

# Python Directory & File Management

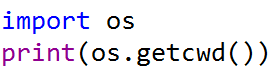
In a computer system, files are organized into directories. These may contain subdirectories and files. Indeed, this makes a vital part of a user-friendly UI. But don’t be confused; a dictionary is simply what you call a folder.

In this Python Directory section, we will import the **OS module** to be able to access the methods we will apply.

* **import os**

## Getting Current Python Directory (CWD)

To find out which directory in python you are currently in, use the **getcwd()** method. This returns the path of the current python directory as a **String in Python**.



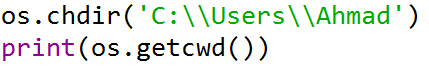
To check the type

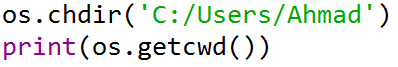
* **print(type(os.getcwd()))**

**<class 'str'>**

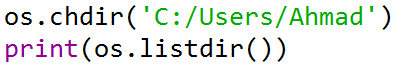
## Changing Current Python Directory

To change our current working directories in python, we use the **chdir()** method. This takes one argument- the path to the directory to which to change.



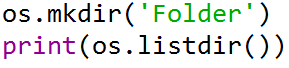
You can also use forward slashes for the path. This way, you don’t have to use backward slashes to escape.

## Python List Directories and Files

To get the contents of a directory into a **python list**, we use the **listdir()** method. Note that this includes the hidden and system files as well.

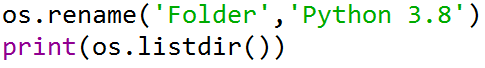
## How to Create Python Directory?

We can also create new python directories with the **mkdir()** method. It takes one argument, that is, the path of the new python directory to create.



## How to Rename Python Directory?

To rename directories in python, we use the **rename()** method. It takes two arguments- the python directory to rename, and the new name for it.



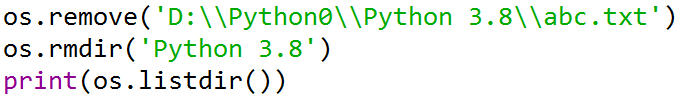
## How to Remove Python Directory/File?

We made a file named **‘abc.txt’** inside our folder **Python3.8**. To delete this file, we use the method **remove().**

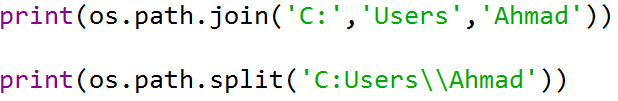


**OSError: [WinError 145] The directory is not empty: 'Python 3.8'**

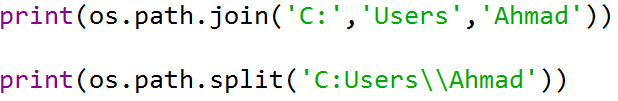
As you can see, it raised a **python exception** called OSError. So, let’s first remove the file and then delete the python directory.



## Joining and Splitting Path

We must use platform-independent file and directory in python paths, so our program runs on every platform. We use the submodule **os.path** for this.  
**join()** in python joins path components and returns a path as a string. It adds appropriate separators (\ for Windows and / for Unix)

Conversely, **split()** splits the path into components, removing the separator.



## Checking if Python Directory Exists

It is possible to check whether a path exists. We use the **exists()** function for this. Also, this is in the os.path submodule.

* print(os.path.exists('C:\\Users\\Ahmad\\Desktop'))

**True**

* print(os.path.exists('C:\\Users\\Ahmad\\Desktop\\Myfolder'))

**False**

* print(os.path.exists('C:\\Users\\Ahmad\\Desktop\\topics.txt'))

**True**

Then, to check whether that path leads us to a directory, we use the isdir() function.

* print(os.path.isdir('C:\\Users\\Ahmad\\Desktop'))

**True**

* print(os.path.isdir('C:\\Users\\Ahmad\\Desktop\\topics.txt'))

**False**

## Recursively Traversing a Directory in Python

The **walk()** function lets us recursively traverse a directory. This means that it returns the roots, subdirectories, and files in a directory. You can traverse it using for **loops in Python.**

