os.path and related functions

Python provides several methods to work with filesystem and directory paths

WE'LL COVER THE FOLLOWING

os.path

The **os.path** sub-module of the **os** module has lots of great functionality built into it. We'll be looking at the following functions:

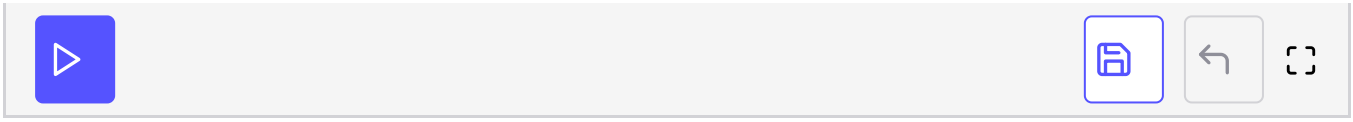
- basename
- dirname
- exists
- isdir and isfile
- join
- split

There are lots of other functions in this sub-module. You are welcome to go read about them in the Python documentation, section 10.1.

os.path.basename

The **basename** function will return just the filename of a path. Here is an example:

```
import os
print(os.path.basename(r'/usercode/test.txt'))
# 'test.txt'
```

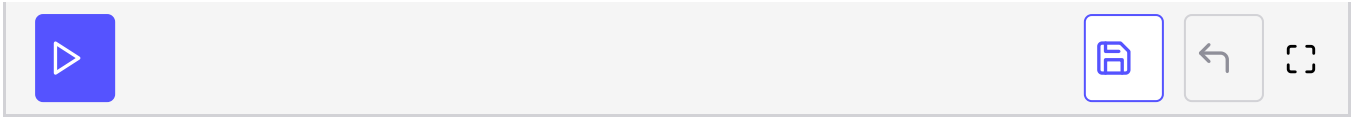


I have found this useful whenever I need to use a filename for naming some related file, such as a log file. This happens a lot when I'm processing a data file.

os.path.dirname

The **dirname** function will return just the directory portion of the path. It's easier to understand if we take a look at some code:

```
import os
print(os.path.dirname(r'/usercode/test.txt'))
```



In this example, we just get the directory path back. This is also useful when you want to store other files next to the file you're processing, like the aforementioned log file.

os.path.exists

The **exists** function will tell you if a path exists or not. All you have to do is pass it a path. Let's take a look:

```
import os

print(os.path.exists(r'/usercode/test.txt'))
# True
print(os.path.exists(r'/usercode/test_fake.txt'))
#False
```

In the first example, we pass the **exists** function a real path and it returns **True**, which means that the path exists. In the second example, we passed it a bad path and it told us that the path did not exist by returning **False**.

os.path.isdir / os.path.isfile

The **isdir** and **isfile** methods are closely related to the **exists** method in that they also test for existence. However, **isdir** only checks if the path is a directory and **isfile** only checks if the path is a file. If you want to check if a path exists regardless of whether it is a file or a directory, then you'll want to use the **exists** method. Anyway, let's study some examples:

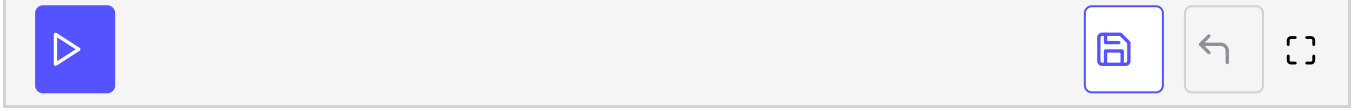
```
import os

print(os.path.isfile(r'/usercode/test.txt'))
# True

print(os.path.isdir(r'/usercode/test.txt'))
# False

print(os.path.isdir(r'/usercode'))
# True

print(os.path.isfile(r'/usercode/test.txt'))
# False
```

Take a moment to study this set of examples. In the first one we pass a path to a file and check if the path is really a file. Then the second example checks the same path to see if it's a directory. You can see for yourself how that turned out. Then in the last two examples, we switched things up a bit by passing a path to a directory to the same two functions. These examples demonstrate how these two functions work.

os.path.join

The **join** method give you the ability to join one or more path components together using the appropriate separator. For example, on Windows, the separator is the backslash, but on Linux, the separator is the forward slash. Here's how it works:

```
import os
print(os.path.join(r'/usercode', 'test.txt'))
# /usercode/test.txt
```

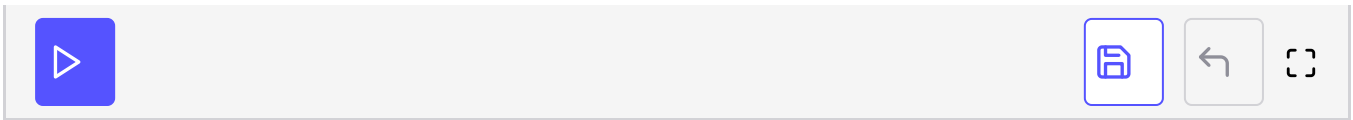


In this example, we joined a directory path and a file path together to get a fully qualified path. Note however that the **join** method does **not** check if the result actually exists!

os.path.split

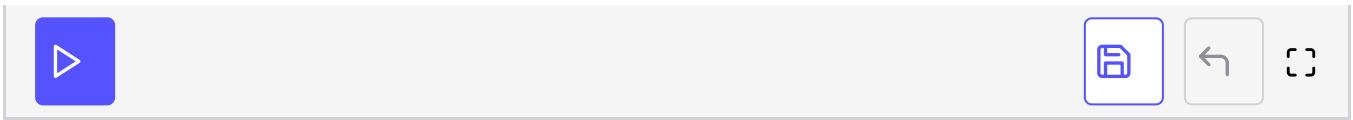
The **split** method will split a path into a tuple that contains the directory and the file. Let's take a look:

```
import os
print(os.path.split(r'/usercode/test.txt'))
# ('/usercode', 'test.txt')
```



This example shows what happens when we path in a path with a file. Let's see what happens if the path doesn't have a filename on the end:

```
import os
print(os.path.split(r'/usercode'))
```

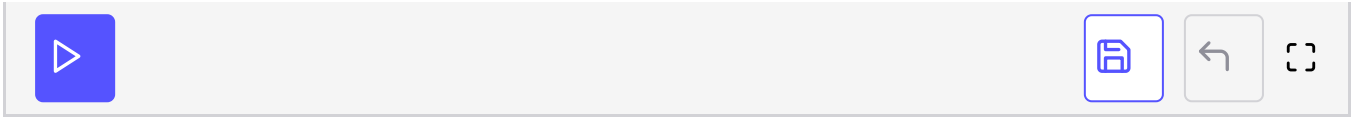


As you can see, it took the path and split it in such a way that the last sub-folder became the second element of the tuple with the rest of the path in the first element.

For our final example, I thought you might like to see a commmon use case of the **split**:

```
import os
```

```
dirname, fname = os.path.split(r'/usercode/text.data')
print(dirname)
# /usercode

print(fname)
# text.data
```



This shows how to do multiple assignment. When you split the path, it returns a two-element tuple. Since we have two variables on the left, the first element of the tuple is assigned to the first variable and the second element to the second variable.

Wrapping Up

At this point you should be pretty familiar with the **os** module. In this chapter you learned the following:

> - how to work with environment variables
> - change directories and discover your current working directory
> - create and remove folders and files
> - rename files / folders
> - start a file with its associated application
> - walk a directory
> - work with paths

There are a lot of other functions in the **os** module that are not covered here. Be sure to read the documentation to see what else you can do. In the next chapter, we will be learning about the **email** and **smtplib** modules.