**Backslash on Windows and Forward Slash on OS X and Linux**

On Windows, paths are written using backslashes (*\*) as the separator between folder names. OS X and Linux, however, use the forward slash (*/*) as their path separator. If you want your programs to work on all operating systems, you will have to write your Python scripts to handle both cases.

Fortunately, this is simple to do with the os.path.join() function. If you pass it the string values of individual file and folder names in your path, os.path.join() will return a string with a file path using the correct path separators. Enter the following into the interactive shell:

>>> **import os**

>>> **os.path.join('usr', 'bin', 'spam')**

'usr\\bin\\spam'

I’m running these interactive shell examples on Windows, so os.path.join('usr', 'bin', 'spam') returned 'usr\\bin\\spam'. (Notice that the backslashes are doubled because each backslash needs to be escaped by another backslash character.) If I had called this function on OS X or Linux, the string would have been 'usr/bin/spam'.

The os.path.join() function is helpful if you need to create strings for filenames. These strings will be passed to several of the file-related functions introduced in this chapter. For example, the following example joins names from a list of filenames to the end of a folder’s name:

>>> **myFiles = ['accounts.txt', 'details.csv', 'invite.docx']**

>>> **for filename in myFiles:**

print(os.path.join('C:\\Users\\asweigart', filename))

C:\Users\asweigart\accounts.txt

C:\Users\asweigart\details.csv

C:\Users\asweigart\invite.docx

**The Current Working Directory**

Every program that runs on your computer has a *current working directory*, or *cwd*. Any filenames or paths that do not begin with the root folder are assumed to be under the current working directory. You can get the current working directory as a string value with the os.getcwd() function and change it with os.chdir(). Enter the following into the interactive shell:

>>> **import os**

>>> **os.getcwd()**

'C:\\Python34'

>>> **os.chdir('C:\\Windows\\System32')**

>>> **os.getcwd()**

'C:\\Windows\\System32'

Here, the current working directory is set to *C:\Python34*, so the filename *project.docx* refers to *C:\Python34\project.docx*. When we change the current working directory to *C:\Windows*, *project.docx* is interpreted as *C:\ Windows\project.docx*.

Python will display an error if you try to change to a directory that does not exist.

>>> **os.chdir('C:\\ThisFolderDoesNotExist')**

Traceback (most recent call last):

File "<pyshell#18>", line 1, in <module>

os.chdir('C:\\ThisFolderDoesNotExist')

FileNotFoundError: [WinError 2] The system cannot find the file specified:

'C:\\ThisFolderDoesNotExist'

**Note**

*While folder is the more modern name for directory, note that* current working directory *(or just* working directory*) is the standard term, not current working folder.*

**Absolute vs. Relative Paths**

There are two ways to specify a file path.

* An *absolute path*, which always begins with the root folder
* A *relative path*, which is relative to the program’s current working directory

There are also the *dot* (.) and *dot-dot* (..) folders. These are not real folders but special names that can be used in a path. A single period (“dot”) for a folder name is shorthand for “this directory.” Two periods (“dot-dot”) means “the parent folder.”

[Figure 8-2](https://automatetheboringstuff.com/chapter8/#calibre_link-83) is an example of some folders and files. When the current working directory is set to *C:\bacon*, the relative paths for the other folders and files are set as they are in the figure.

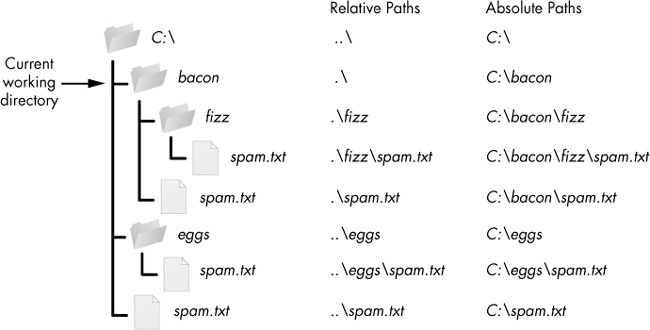


Figure 8-2. The relative paths for folders and files in the working directory *C:\bacon*

The *.\* at the start of a relative path is optional. For example, *.\spam.txt* and *spam.txt* refer to the same file.

**Creating New Folders with os.makedirs()**

Your programs can create new folders (directories) with the os.makedirs() function. Enter the following into the interactive shell:

>>> **import os**

>>> **os.makedirs('C:\\delicious\\walnut\\waffles')**

This will create not just the *C:\delicious* folder but also a *walnut* folder inside *C:\delicious* and a *waffles* folder inside *C:\delicious\walnut*. That is, os.makedirs() will create any necessary intermediate folders in order to ensure that the full path exists. [Figure 8-3](https://automatetheboringstuff.com/chapter8/#calibre_link-84) shows this hierarchy of folders.

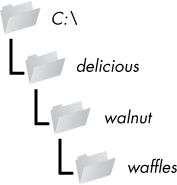


Figure 8-3. The result of os.makedirs('C:\\delicious \\walnut\\waffles')

**The os.path Module**

The os.path module contains many helpful functions related to filenames and file paths. For instance, you’ve already used os.path.join() to build paths in a way that will work on any operating system. Since os.path is a module inside the os module, you can import it by simply running import os. Whenever your programs need to work with files, folders, or file paths, you can refer to the short examples in this section. The full documentation for the os.path module is on the Python website at [*http://docs.python.org/3/library/os.path.html*](http://docs.python.org/3/library/os.path.html).

**Note**

*Most of the examples that follow in this section will require the* os *module, so remember to import it at the beginning of any script you write and any time you restart IDLE. Otherwise, you’ll get a* NameError: name 'os' is not defined *error message.*

**Handling Absolute and Relative Paths**

The os.path module provides functions for returning the absolute path of a relative path and for checking whether a given path is an absolute path.

* Calling os.path.abspath(*path*) will return a string of the absolute path of the argument. This is an easy way to convert a relative path into an absolute one.
* Calling os.path.isabs(*path*) will return True if the argument is an absolute path and False if it is a relative path.
* Calling os.path.relpath(*path, start*) will return a string of a relative path from the *start* path to *path*. If *start* is not provided, the current working directory is used as the start path.

Try these functions in the interactive shell:

>>> **os.path.abspath('.')**

'C:\\Python34'

>>> **os.path.abspath('.\\Scripts')**

'C:\\Python34\\Scripts'

>>> **os.path.isabs('.')**

False

>>> **os.path.isabs(os.path.abspath('.'))**

True

Since *C:\Python34* was the working directory when os.path.abspath() was called, the “single-dot” folder represents the absolute path 'C:\\Python34'.

**Note**

*Since your system probably has different files and folders on it than mine, you won’t be able to follow every example in this chapter exactly. Still, try to follow along using folders that exist on your computer.*

Enter the following calls to os.path.relpath() into the interactive shell:

>>> **os.path.relpath('C:\\Windows', 'C:\\')**

'Windows'

>>> **os.path.relpath('C:\\Windows', 'C:\\spam\\eggs')**

'..\\..\\Windows'

>>> **os.getcwd()** 'C:\\Python34'

Calling os.path.dirname(*path*) will return a string of everything that comes before the last slash in the path argument. Calling os.path.basename(*path*) will return a string of everything that comes after the last slash in the path argument. The dir name and base name of a path are outlined in [Figure 8-4](https://automatetheboringstuff.com/chapter8/#calibre_link-85).

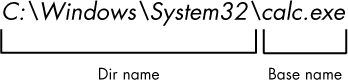


Figure 8-4. The base name follows the last slash in a path and is the same as the filename. The dir name is everything before the last slash.

For example, enter the following into the interactive shell:

>>> **path = 'C:\\Windows\\System32\\calc.exe'**

>>> **os.path.basename(path)**

'calc.exe'

>>> **os.path.dirname(path)**

'C:\\Windows\\System32'

If you need a path’s dir name and base name together, you can just call os.path.split() to get a tuple value with these two strings, like so:

>>> **calcFilePath = 'C:\\Windows\\System32\\calc.exe'**

>>> **os.path.split(calcFilePath)**

('C:\\Windows\\System32', 'calc.exe')

Notice that you could create the same tuple by calling os.path.dirname() and os.path.basename() and placing their return values in a tuple.

>>> **(os.path.dirname(calcFilePath), os.path.basename(calcFilePath))**

('C:\\Windows\\System32', 'calc.exe')

But os.path.split() is a nice shortcut if you need both values.

Also, note that os.path.split() does *not* take a file path and return a list of strings of each folder. For that, use the split() string method and split on the string in os.sep. Recall from earlier that the os.sep variable is set to the correct folder-separating slash for the computer running the program.

For example, enter the following into the interactive shell:

>>> **calcFilePath.split(os.path.sep)**

['C:', 'Windows', 'System32', 'calc.exe']

On OS X and Linux systems, there will be a blank string at the start of the returned list:

>>> **'/usr/bin'.split(os.path.sep)**

['', 'usr', 'bin']

The split() string method will work to return a list of each part of the path. It will work on any operating system if you pass it os.path.sep.

## Checking Path Validity

Many Python functions will crash with an error if you supply them with a path that does not exist. The os.path module provides functions to check whether a given path exists and whether it is a file or folder.

* Calling os.path.exists(*path*) will return True if the file or folder referred to in the argument exists and will return False if it does not exist.
* Calling os.path.isfile(*path*) will return True if the path argument exists and is a file and will return False otherwise.
* Calling os.path.isdir(*path*) will return True if the path argument exists and is a folder and will return False otherwise.

Here’s what I get when I try these functions in the interactive shell:

>>> **os.path.exists('C:\\Windows')**

True

>>> **os.path.exists('C:\\some\_made\_up\_folder')**

False

>>> **os.path.isdir('C:\\Windows\\System32')**

True

>>> **os.path.isfile('C:\\Windows\\System32')**

False

>>> **os.path.isdir('C:\\Windows\\System32\\calc.exe')**

False

>>> **os.path.isfile('C:\\Windows\\System32\\calc.exe')**

True

You can determine whether there is a DVD or flash drive currently attached to the computer by checking for it with the os.path.exists() function. For instance, if I wanted to check for a flash drive with the volume named D:\ on my Windows computer, I could do that with the following:

>>> **os.path.exists('D:\\')**

False

Oops! It looks like I forgot to plug in my flash drive.