The easy way to deal with file paths on Windows, Mac and Linux

One of programming's little annoyances is that Microsoft Windows uses a backslash character between folder names while almost every other computer uses a forward slash:

Windows filenames:

C:\some_folder\some_file.txt

Most other operating systems:

/some_folder/some_file.txt

This is an accident of early 1980's computer history. The first version of MS-DOS used the forward slash character for specifying command-line options. When Microsoft added support for folders in MS-DOS 2.0, the forward slash character was already taken so they used a backslash instead. Thirty-five years later, we are still stuck with this incompatibility.

If you want your Python code to work on both Windows and Mac/Linux, you'll need to deal with these kinds of platform-specific issues. Luckily, Python 3 has a new module called **pathlib** that makes working with files nearly painless.

The Wrong Solution: Building File Paths by Hand

Let's say you have a data folder that contains a file that you want to open in your Python program:

This is the *wrong way* to code it in Python:

```
data_folder = "source_data/text_files/"

file_to_open = data_folder + "raw_data.txt"

f = open(file_to_open)

print(f.read())
```

Notice that I've hardcoded the path using Unix-style forward slashes since I'm on a Mac. This will make Windows users angry.

Technically this code will still work on Windows because Python has a hack where it will recognize either kind of slash when you call **open()** on Windows. But even still, you shouldn't depend on that.

Not all Python libraries will work if you use wrong kind of slash on the wrong operating system — especially if they interface with external programs or libraries.

And Python's support for mixing slash types is a Windows-only hack that doesn't work in reverse. Using backslashes in code will totally fail on a Mac:

```
data_folder = "source_data\\text_files\\"

file_to_open = data_folder + "raw_data.txt"

f = open(file_to_open)

print(f.read())

# On a Mac, this code will throw an exception:
# FileNotFoundError: [Errno 2] No such file or directory:
'source_data\\text_files\\raw_data.txt'
```

For all these reasons and more, writing code with hardcoded path strings is the kind of thing that will make other programmers look at you with great suspicion. In general, you should try to avoid it.

The Old Solution: Python's os.path module

Python's **os.path** module has lots of tools for working around these kinds of operating system-specific file system issues.

You can use **os.path.join()** to build a path string using the right kind of slash for the current operating system:

```
import os.path

data_folder = os.path.join("source_data", "text_files")

file_to_open = os.path.join(data_folder, "raw_data.txt")

f = open(file_to_open)

print(f.read())
```

This code will work perfectly on both Windows or Mac. The problem is that it's a pain to use. Writing out **os.path.join()** and passing in each part of the path as a separate string is wordy and unintuitive.

Since most of the functions in the **os.path** module are similarly annoying to use, developers often "forget" to use them even when they know better. This leads to a lot of cross-platform bugs and angry users.

The Better Solution: Python 3's pathlib!

Python 3.4 introduced a new standard library for dealing with files and paths called **pathlib**

To use it, you just pass a path or filename into a new Path() object using forward slashes and it handles the rest:

```
from pathlib import Path

data_folder = Path("source_data/text_files/")

file_to_open = data_folder / "raw_data.txt"

f = open(file_to_open)

print(f.read())
```

Notice two things here:

1. You should use forward slashes with **pathlib** functions. The **Path()** object will convert forward slashes into the correct kind of slash for the current operating system. Nice!
2. If you want to add on to the path, you can use the **/** operator directly in your code. Say goodbye to typing out **os.path.join(a, b)** over and over.

And if that's all **pathlib** did, it would be a nice addition to Python — but it does a lot more!

For example, we can read the contents of a text file without having to mess with opening and closing the file:

```
from pathlib import Path

data_folder = Path("source_data/text_files/")

file_to_open = data_folder / "raw_data.txt"

print(file_to_open.read_text())
```

Pro-tip: The previous examples were buggy because the opened file was never closed. This syntax avoids that bug entirely.

In fact, **pathlib** makes most standard file operations quick and easy:

```
from pathlib import Path

filename = Path("source_data/text_files/raw_data.txt")

print(filename.name)
# prints "raw_data.txt"

print(filename.suffix)
# prints "txt"

print(filename.stem)
# prints "raw_data"

if not filename.exists():
    print("Oops, file doesn't exist!")
else:
    print("Yay, the file exists!")
```

You can even use **pathlib** to explicitly convert a Unix path into a Windows-formatted path:

```
from pathlib import Path, PureWindowsPath

filename = Path("source_data/text_files/raw_data.txt")

# Convert path to Windows format
path_on_windows = PureWindowsPath(filename)

print(path_on_windows)
# prints "source_data\text_files\raw_data.txt"
```

And if you *REALLY* want to use backslashes in your code safely, you can declare your path as Windows-formatted and **pathlib** can convert it to work on the current operating system:

```
from pathlib import Path, PureWindowsPath

# I've explicitly declared my path as being in Windows format, so I can use
forward slashes in it.
filename = PureWindowsPath("source_data\\text_files\\raw_data.txt")

# Convert path to the right format for the current operating system
correct_path = Path(filename)

print(correct_path)
# prints "source_data/text_files/raw_data.txt" on Mac and Linux
# prints "source_data\text_files\raw_data.txt" on Windows
```

If you want to get fancy, you can even use **pathlib** to do things like resolve relative file paths, parse network share paths and generate file:// urls. Here's an example that will open a local file in your web browser with just two lines a code:

```
from pathlib import Path
import webbrowser

filename = Path("source_data/text_files/raw_data.txt")

webbrowser.open(filename.absolute().as_uri())
```