Handling file and directory Paths

There are two main modules in Python that deals with path manipulation. One is the `os.path` module and the other is the `pathlib` module.

Linux and Windows Paths

On Windows, paths are written using backslashes (`\`) as the separator between folder names. On Unix based operating system such as macOS, Linux, and BSDs, the forward slash (`/`) is used as the path separator. Joining paths can be a headache if your code needs to work on different platforms.

Fortunately, Python provides easy ways to handle this. We will showcase how to deal with both, `os.path.join` and `pathlib.Path.joinpath`

Using `os.path.join` on Windows:

```
>>> import os

>>> os.path.join('usr', 'bin', 'spam')
# 'usr\\bin\\spam'
```

And using `pathlib` on *nix:

```
>>> from pathlib import Path

>>> print(Path('usr').joinpath('bin').joinpath('spam'))
# usr/bin/spam
```

`pathlib` also provides a shortcut to joinpath using the `/` operator:

```
>>> from pathlib import Path

>>> print(Path('usr') / 'bin' / 'spam')
# usr/bin/spam
```

Notice the path separator is different between Windows and Unix based operating system, that's why you want to use one of the above methods instead of adding strings together to join paths together.

Joining paths is helpful if you need to create different file paths under the same directory.

Using `os.path.join` on Windows:

```
>>> my_files = ['accounts.txt', 'details.csv', 'invite.docx']

>>> for filename in my_files:
...     print(os.path.join('C:\\Users\\asweigart', filename))
...
# C:\Users\asweigart\accounts.txt
# C:\Users\asweigart\details.csv
# C:\Users\asweigart\invite.docx
```

Using `pathlib` on *nix:

```
>>> my_files = ['accounts.txt', 'details.csv', 'invite.docx']
>>> home = Path.home()
>>> for filename in my_files:
...     print(home / filename)
...
# /home/asweigart/accounts.txt
# /home/asweigart/details.csv
# /home/asweigart/invite.docx
```

The current working directory

Using `os` on Windows:

```
>>> import os

>>> os.getcwd()
# 'C:\\Python34'
>>> os.chdir('C:\\Windows\\System32')

>>> os.getcwd()
# 'C:\\Windows\\System32'
```

Using `pathlib` on *nix:

```
>>> from pathlib import Path
>>> from os import chdir

>>> print(Path.cwd())
# /home/asweigart

>>> chdir('/usr/lib/python3.6')
>>> print(Path.cwd())
# /usr/lib/python3.6
```

Creating new folders

Using `os` on Windows:

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

Using `pathlib` on *nix:

```
>>> from pathlib import Path
>>> cwd = Path.cwd()
>>> (cwd / 'delicious' / 'walnut' / 'waffles').mkdir()
# Traceback (most recent call last):
#   File "<stdin>", line 1, in <module>
#   File "/usr/lib/python3.6/pathlib.py", line 1226, in mkdir
#     self._accessor.mkdir(self, mode)
#   File "/usr/lib/python3.6/pathlib.py", line 387, in wrapped
#     return strfunc(str(pathobj), *args)
# FileNotFoundError: [Errno 2] No such file or directory:
'/home/asweigart/delicious/walnut/waffles'
```

Oh no, we got a nasty error! The reason is that the 'delicious' directory does not exist, so we cannot make the 'walnut' and the 'waffles' directories under it. To fix this, do:

```
>>> from pathlib import Path
>>> cwd = Path.cwd()
>>> (cwd / 'delicious' / 'walnut' / 'waffles').mkdir(parents=True)
```

And all is good :)

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.”

Handling Absolute paths

To see if a path is an absolute path:

Using `os.path` on *nix:

```
>>> import os
>>> os.path.isabs('/')
# True

>>> os.path.isabs('..')
# False
```

Using `pathlib` on *nix:

```
>>> from pathlib import Path
>>> Path('/').is_absolute()
# True

>>> Path('..').is_absolute()
# False
```

You can extract an absolute path with both `os.path` and `pathlib`

Using `os.path` on *nix:

```
>>> import os
>>> os.getcwd()
```

```
'/home/asweigart'

>>> os.path.abspath('..')
'/home'
```

Using `pathlib` on *nix:

```
from pathlib import Path
print(Path.cwd())
# /home/asweigart

print(Path('..').resolve())
# /home
```

Handling Relative paths

You can get a relative path from a starting path to another path.

Using `os.path` on *nix:

```
>>> import os
>>> os.path.relpath('/etc/passwd', '/')
# 'etc/passwd'
```

Using `pathlib` on *nix:

```
>>> from pathlib import Path
>>> print(Path('/etc/passwd').relative_to('/'))
# etc/passwd
```

Path and File validity

Checking if a file/directory exists

Using `os.path` on *nix:

```
>>> import os

>>> os.path.exists('.')
# True

>>> os.path.exists('setup.py')
# True

>>> os.path.exists('/etc')
# True

>>> os.path.exists('nonexistentfile')
# False
```

Using `pathlib` on *nix:

```
from pathlib import Path
```

```
>>> Path('.').exists()
# True

>>> Path('setup.py').exists()
# True

>>> Path('/etc').exists()
# True

>>> Path('nonexistentfile').exists()
# False
```

Checking if a path is a file

Using `os.path` on *nix:

```
>>> import os

>>> os.path.isfile('setup.py')
# True

>>> os.path.isfile('/home')
# False

>>> os.path.isfile('nonexistentfile')
# False
```

Using `pathlib` on *nix:

```
>>> from pathlib import Path

>>> Path('setup.py').is_file()
# True

>>> Path('/home').is_file()
# False

>>> Path('nonexistentfile').is_file()
# False
```

Checking if a path is a directory

Using `os.path` on *nix:

```
>>> import os

>>> os.path.isdir('/')
# True

>>> os.path.isdir('setup.py')
# False
```

```
>>> os.path.isdir('/spam')
# False
```

Using `pathlib` on *nix:

```
>>> from pathlib import Path

>>> Path('/').is_dir()
# True

>>> Path('setup.py').is_dir()
# False

>>> Path('/spam').is_dir()
# False
```

Getting a file's size in bytes

Using `os.path` on Windows:

```
>>> import os

>>> os.path.getsize('C:\\Windows\\System32\\calc.exe')
# 776192
```

Using `pathlib` on *nix:

```
>>> from pathlib import Path

>>> stat = Path('/bin/python3.6').stat()
>>> print(stat) # stat contains some other information about the file as well
# os.stat_result(st_mode=33261, st_ino=141087, st_dev=2051, st_nlink=2, st_uid=0,
# --snip--
# st_gid=0, st_size=10024, st_atime=1517725562, st_mtime=1515119809,
st_ctime=1517261276)

>>> print(stat.st_size) # size in bytes
# 10024
```

Listing directories

Listing directory contents using `os.listdir` on Windows:

```
>>> import os

>>> os.listdir('C:\\Windows\\System32')
# ['0409', '12520437.cpx', '12520850.cpx', '5U877.ax', 'aaclient.dll',
# --snip--
# 'xwtpdui.dll', 'xwtpw32.dll', 'zh-CN', 'zh-HK', 'zh-TW', 'zipfldr.dll']
```

Listing directory contents using `pathlib` on *nix:

```
>>> from pathlib import Path

>>> for f in Path('/usr/bin').iterdir():
...     print(f)
...
# ...
# /usr/bin/tiff2rgba
# /usr/bin/iconv
# /usr/bin/ldd
# /usr/bin/cache_restore
# /usr/bin/udiskie
# /usr/bin/unix2dos
# /usr/bin/t1reencode
# /usr/bin/epstopdf
# /usr/bin/idle3
# ...
```

Directory file sizes

Using `os.path.getsize()` and `os.listdir()` together on Windows:

```
>>> import os
>>> total_size = 0

>>> for filename in os.listdir('C:\\Windows\\System32'):
...     total_size = total_size +
os.path.getsize(os.path.join('C:\\Windows\\System32', filename))
...
>>> print(total_size)
# 1117846456
```

Using `pathlib` on *nix:

```
>>> from pathlib import Path

>>> total_size = 0
>>> for sub_path in Path('/usr/bin').iterdir():
...     total_size += sub_path.stat().st_size
...
>>> print(total_size)
# 1903178911
```

Copying files and folders

The `shutil` module provides functions for copying files, as well as entire folders.

```
>>> import shutil, os

>>> os.chdir('C:\\')
>>> shutil.copy('C:\\spam.txt', 'C:\\delicious')
# C:\\delicious\\spam.txt'
```

```
>>> shutil.copy('eggs.txt', 'C:\\delicious\\eggs2.txt')
# 'C:\\delicious\\eggs2.txt'
```

While `shutil.copy()` will copy a single file, `shutil.copytree()` will copy an entire folder and every folder and file contained in it:

```
>>> import shutil, os

>>> os.chdir('C:\\')
>>> shutil.copytree('C:\\bacon', 'C:\\bacon_backup')
# 'C:\\bacon_backup'
```

Moving and Renaming

```
>>> import shutil

>>> shutil.move('C:\\bacon.txt', 'C:\\eggs')
# 'C:\\eggs\\bacon.txt'
```

The destination path can also specify a filename. In the following example, the source file is moved and renamed:

```
>>> shutil.move('C:\\bacon.txt', 'C:\\eggs\\new_bacon.txt')
# 'C:\\eggs\\new_bacon.txt'
```

If there is no eggs folder, then `move()` will rename bacon.txt to a file named eggs:

```
>>> shutil.move('C:\\bacon.txt', 'C:\\eggs')
# 'C:\\eggs'
```

Deleting files and folders

- Calling `os.unlink(path)` or `Path.unlink()` will delete the file at path.
- Calling `os.rmdir(path)` or `Path.rmdir()` will delete the folder at path. This folder must be empty of any files or folders.
- Calling `shutil.rmtree(path)` will remove the folder at path, and all files and folders it contains will also be deleted.

Walking a Directory Tree

```
>>> import os
>>>
>>> for folder_name, subfolders, filenames in os.walk('C:\\delicious'):
...     print(f'The current folder is {folder_name}')
...     for subfolder in subfolders:
...         print('SUBFOLDER OF {folder_name}: {subfolder}')
...     for filename in filenames:
...         print('FILE INSIDE {folder_name}: filename{filename}')
...     print('')
```

```
...
# The current folder is C:\delicious
# SUBFOLDER OF C:\delicious: cats
# SUBFOLDER OF C:\delicious: walnut
# FILE INSIDE C:\delicious: spam.txt

# The current folder is C:\delicious\cats
# FILE INSIDE C:\delicious\cats: catnames.txt
# FILE INSIDE C:\delicious\cats: zophie.jpg

# The current folder is C:\delicious\walnut
# SUBFOLDER OF C:\delicious\walnut: waffles

# The current folder is C:\delicious\walnut\waffles
# FILE INSIDE C:\delicious\walnut\waffles: butter.txt
```