

Using the Filesystem

Chapter 12

This chapter covers

- Managing paths and pathnames
- Getting information about files
- Performing filesystem operations
- Processing all files in a directory subtree

os and os.path vs. pathlib

- The traditional way that file paths and filesystem operations have been handled in Python is by using functions included in the os and os.path modules.
- These functions have worked well enough but often resulted in more verbose code than necessary.
- Since Python 3.5, a new library, pathlib, has been added; it offers a more object-oriented and more unified way of doing the same operations.

Paths and pathnames

- All operating systems refer to files and directories with strings naming a given file or directory.
- Strings used in this manner are usually called pathnames (or sometimes just paths).
- Pathname semantics across operating systems are very similar because the filesystem on almost all operating systems is modeled as a tree structure, with a disk being the root and folders, subfolders, and so on being branches, subbranches, and so on.
- Different operating systems have different conventions regarding the precise syntax of pathnames.

Absolute and relative paths

- These operating systems allow two types of pathnames:
 - Absolute pathnames specify the exact location of a file in a filesystem without any ambiguity; they do this by listing the entire path to that file, starting from the root of the filesystem.
 - Relative pathnames specify the position of a file relative to some other point in the filesystem, and that other point isn't specified in the relative pathname itself; instead, the absolute starting point for relative pathnames is provided by the context in which they're used.
- As examples, here are two Windows absolute pathnames:
 - `C:\Program Files\Doom`
 - `D:\backup\June`

Absolute and relative paths

- and here are two Linux absolute pathnames and a Mac absolute pathname:

`/bin/Doom`

`/floppy/backup/June`

`/Applications/Utilities`

- and here are two Windows relative pathnames:

`mydata\project1\readme.txt`

`games\tetris`

- and these are Linux/UNIX/Mac relative pathnames:

`mydata/project1/readme.txt`

`games/tetris`

`Utilities/Java`

The current working directory

- The directory that a Python program is in is called the current working directory for that program.
- This directory may be different from the directory the program resides in.

```
1  import os
2
3  # get current working directory
4  print(os.getcwd())
5
6  # get listing
7  print(os.listdir(os.curdir))
8
9  # change directory
10 os.chdir("foldername")
11 print(os.getcwd())
```

Accessing directories with pathlib

- To get the current directory with pathlib, you could do the following:

```
import pathlib  
cur_path = pathlib.Path()  
cur_path.cwd()
```


Manipulating pathnames

- To start, construct a few pathnames on different operating systems, using the `os.path.join` function.
- Note that importing `os` is sufficient to bring in the `os.path` submodule also; there's no need for an explicit `import os.path` statement.

```
import os
print(os.path.join('bin', 'utils', 'disktools'))
```

Manipulating pathnames with pathlib

- Start by constructing a few pathnames on different operating systems, using the path object's methods.

```
from pathlib import Path
cur_path = Path()
print(cur_path.joinpath('bin', 'utils', 'disktools'))
```

Useful constants and functions

- Checks whether the parent of the parent of path is a directory.

```
os.path.isdir(os.path.join(path, os.pardir, os.curdir))
```

- Returns a list of filenames in the current working directory.

```
os.listdir(os.curdir)
```

- The `os.name` constant returns the name of the Python module imported to handle the operating system-specific details.

```
os.name
```

Getting information about files

- The most commonly used Python path-information functions are
 - `os.path.exists`
 - `os.path.isfile`
 - `os.path.isdir`
 - `os.path.islink`
 - `os.path.ismount`
 - `os.path.samefile(path1, path2)`
 - `os.path.isabs(path)`
 - `os.path.getsize(path)`
 - `os.path.getmtime(path)`
 - `os.path.getatime(path)`

More filesystem operations

- `glob.glob("*")`
 - The `glob` function from the `glob` module (named after an old UNIX function that did pattern matching) expands Linux/UNIX shell-style wildcard characters and character sequences in a pathname, returning the files in the current working directory that match.
- `os.rename`
 - To rename or move a file or directory.
- `os.remove`
 - To remove or delete a data file.
- `os.makedirs` or `os.mkdir`
- `os.rmdir`

Lab

More File Operations

Processing all files in a directory subtree

- Finally, a highly useful function for traversing recursive directory structures is the `os.walk` function.
- You can use it to walk through an entire directory tree, returning three things for each directory it traverses: the root, or path, of that directory; a list of its subdirectories; and a list of its files.
- When called, `os.walk` creates an iterator that recursively applies itself to all the directories contained in the top parameter. In other words, for each subdirectory `subdir` in `names`, `os.walk` recursively invokes a call to itself, of the form `os.walk(subdir, ...)`.

```
1 import os
2
3 for root, dirs, files in os.walk(os.curdir):
4     print("{0} has {1} files".format(root, len(files)))
5     if ".git" in dirs:
6         |     dirs.remove(".git")
```

Summary

- Python provides a group of functions and constants that handle filesystem references (pathnames) and filesystem operations in a manner independent of the underlying operating system.
- For more advanced and specialized filesystem operations that typically are tied to a certain operating system or systems, look at the main Python documentation for the `os`, `pathlib`, and `posix` modules.