Understanding File Systems and Paths

A *file* is a collection of data stored as a single unit on a storage device such as a disk drive. On computers, files are usually organized in a *file system*. In most file systems, groups of related files are stored in a *directory* (also called a *folder* when used with a graphical interface). These file systems then allow hierarchies of directories where directories may contained both files and directories. Typically, the entire directory structure for one drive lies a single directory (the *root* directory) that contains all of the files and directories for that drive.

When working with directories, there are several common terms that we will use. The *parent* of a directory C is simply the directory that directly contains C. (Note that the root directory has no parent.) If P is the parent directory of another directory C, the C is referred to as a *sub-directory* of P. (C is also sometimes referred to as a *child* directory of P.)

Absolute paths

When working with files, one fundamental task is to specify the unique location of a file in the file system. In Python, the location is specified as string known as a *path*. For example, using the Windows File Explorer, you can extract the path of a specific file by holding down the shift key and right clicking on the desired file, and selecting "Copy as path" from the pull-down menu. Here is a typical example of the path to the Dropbox directory on my work computer: `"C:\Users\jwarren\Dropbox"`.

This path consist of a string whose first two characters `"C:"` denote which drive contains the directory. The rest of the path consists of directory names separated by backslashes `"\"`. The `"\Users"` part of the path indicates that my Dropbox directory lies in the `"Users"` sub-directory of the root directory. The `"\jwarren"` portion of the path indicates that my Dropbox directory lies in the `"jwarren"` sub-directory of the `"Users"` directory. The final `"\Dropbox"` portion of the string indices that the `"Dropbox"` directory is a sub-directory of the `"jwarren"` directory.

For Mac OS, such a path would have a form similar to something like `"/Users/jwarren/Dropbox/Python Scripting"`. The two key differences are that paths for Mac OS use slash `"/"` in place of backslash `"\"` and all files lie in a single root directory. (There are no separate logical drives).

Relative paths

While it is always possible to specify the location of a file or directory via an absolute path, this approach is not always recommended. An alternative approach is to specify the location of the desired file relative to some existing location in the file system. By default, Python maintains a *working directory* that it uses as the base location for all relative paths. In a GUI-based IDEs such as Thonny, IDLE or Atom, the working directory is usually the directory that contains the Python source code being run.

Based on this working directory, here are a couple of examples of relative paths:

- `"current_file.ext"` - relative path to the file of the same name in the current working directory.
- `"child/child_file.ext"` - relative path to the file `"child_file.ext"` in the sub-directory `"child"` of the current working directory.
- `"../parent_file.ext"` - relative path to the file `"parent_file.ext"` in parent directory of the current working directory. Here, `".."` denotes the parent directory of the current working directory.

Note that in these examples, slashes were used in constructing the specified relative paths. While paths typically use backslash in Windows, Python will accepted slashes in place of backslashes on relative paths on Windows.

Using the OS module to manipulate paths

For many applications, your Python code should work reliably independent of the particular OS for the file systems. In this case, you may wish to considering use the methods from the os module described below to create truly OS-independent Python code for manipulating paths. (Remember to `import os` before using these methods.)

- `os.getcwd()` returns the path to the current working directory for your Python code.
- `os.path.abspath(file_name)` returns to the absolute path the specified file.
- `os.path.join(path, dir1, dir2, ..., file_name)` returns the absolute path to the file `file_name` that lies in the sequence of nested directories `dir1`, `dir2`, ..., located at `path`.
- `os.pardir` returns the relative path to the parent directory of the current working directory. For most systems, this path is `".."`

For examples of these methods in action, explore this path manipulation code. Since this code manipulates your file system, the code must be run on your desktop.

Mark as completed