The os.path module

This module contains functions that deal with long filenames (path names) in various ways. To use this module, import the **os** module, and access this module as **os.path**.

Working with file names

This module contains a number of functions that deal with long filenames in a platform independent way. In other words, without having to deal with forward and backward slashes, colons, and whatnot.

Example: Using the os.path module to handle filename

```
# File: os-path-example-1.py

import os

filename = "my/little/pony"

print "using", os.name, "..."
print "split", "=>", os.path.split(filename)
print "splitext", "=>", os.path.splitext(filename)
print "dirname", "=>", os.path.dirname(filename)
print "basename", "=>", os.path.basename(filename)
print "join", "=>", os.path.join(os.path.dirname(filename),
                                 os.path.basename(filename))
```

```
using nt ...
split => ('my/little', 'pony')
splitext => ('my/little/pony', '')
dirname => my/little
basename => pony
join => my/little\pony
```

Note that **split** only splits off a single item.

This module also contains a number of functions that allow you to quickly figure out what a filename represents:

Example: Using the os.path module to check what a filename represents

```
# File: os-path-example-2.py

import os

FILES = (
    os.curdir,
    "/",
    "file",
    "/file",
    "samples",
    "samples/sample.jpg",
    "directory/file",
    "../directory/file",
    "/directory/file"
    )

for file in FILES:
    print file, "=>",
    if os.path.exists(file):
        print "EXISTS",
    if os.path.isabs(file):
        print "ISABS",
    if os.path.isdir(file):
        print "ISDIR",
    if os.path.isfile(file):
        print "ISFILE",
    if os.path.islink(file):
        print "ISLINK",
    if os.path.ismount(file):
        print "ISMOUNT",
    print
```

```
. => EXISTS ISDIR
/ => EXISTS ISABS ISDIR ISMOUNT
file =>
/file => ISABS
samples => EXISTS ISDIR
samples/sample.jpg => EXISTS ISFILE
directory/file =>
../directory/file =>
/directory/file => ISABS
```

The **expanduser** function treats a user name shortcut in the same way as most modern Unix shells (it doesn’t work well on Windows).

Example: Using the os.path module to insert the user name into a filename

```
# File: os-path-expanduser-example-1.py

import os

print os.path.expanduser("~/.pythonrc")
```

```
/home/effbot/.pythonrc
```

The **expandvars** function inserts environment variables into a filename:

Example: Using the os.path module to insert variables into a filename

```
# File: os-path-expandvars-example-1.py

import os

os.environ["USER"] = "user"

print os.path.expandvars("/home/$USER/config")
print os.path.expandvars("$USER/folders")
```

```
/home/user/config
user/folders
```

Traversing a file system

The **walk** function helps you find all files in a directory tree. It takes a directory name, a callback function, and a data object that is passed on to the callback.

Example: Using the os.path module to traverse a file system

```
# File: os-path-walk-example-1.py

import os

def callback(arg, directory, files):
    for file in files:
        print os.path.join(directory, file), repr(arg)

os.path.walk(".", callback, "secret message")


./aifc-example-1.py 'secret message'
./anydbm-example-1.py 'secret message'
./array-example-1.py 'secret message'
...
./samples 'secret message'
./samples/sample.jpg 'secret message'
./samples/sample.txt 'secret message'
./samples/sample.zip 'secret message'
./samples/articles 'secret message'
./samples/articles/article-1.txt 'secret message'
./samples/articles/article-2.txt 'secret message'
...
```

The **walk** function has a somewhat obscure user interface (maybe it’s just me, but I can never remember the order of the arguments). The **index** function in the next example returns a list of filenames instead, which lets you use a straightforward **for-in** loop to process the files:

Example: Using os.listdir to traverse a file system

```
# File: os-path-walk-example-2.py

import os

def index(directory):
    # like os.listdir, but traverses directory trees
    stack = [directory]
    files = []
    while stack:
        directory = stack.pop()
        for file in os.listdir(directory):
            fullname = os.path.join(directory, file)
            files.append(fullname)
            if os.path.isdir(fullname) and not os.path.islink(fullname):
                stack.append(fullname)
    return files

for file in index("."):
    print file


.\aifc-example-1.py
.\anydbm-example-1.py
.\array-example-1.py
...
```

If you don’t want to list all files (for performance or memory reasons), the following example uses a different approach. Here, the **DirectoryWalker** class behaves like a sequence object, returning one file at a time:

Example: Using a directory walker to traverse a file system

```
# File: os-path-walk-example-3.py

import os

class DirectoryWalker:
    # a forward iterator that traverses a directory tree

    def __init__(self, directory):
        self.stack = [directory]
        self.files = []
        self.index = 0

    def __getitem__(self, index):
        while 1:
            try:
                file = self.files[self.index]
                self.index = self.index + 1
            except IndexError:
                # pop next directory from stack
                self.directory = self.stack.pop()
                self.files = os.listdir(self.directory)
                self.index = 0
            else:
                # got a filename
                fullname = os.path.join(self.directory, file)
                if os.path.isdir(fullname) and not os.path.islink(fullname):
                    self.stack.append(fullname)
                return fullname

for file in DirectoryWalker("."):
    print file
```

```
.\aifc-example-1.py
.\anydbm-example-1.py
.\array-example-1.py
...
```

Note that this class doesn’t check the index passed to the **__getitem__** method. This means that it won’t do the right thing if you access the sequence members out of order.

Finally, if you’re interested in the file sizes or timestamps, here’s a version of the class that returns both the filename and the tuple returned from **os.stat**. This version saves one or two **stat** calls for each file (both **os.path.isdir** and **os.path.islink** uses **stat**), and runs quite a bit faster on some platforms.

Example: Using a directory walker to traverse a file system, returning both the filename and additional file information

```
# File: os-path-walk-example-4.py

import os, stat

class DirectoryStatWalker:
    # a forward iterator that traverses a directory tree, and
    # returns the filename and additional file information

    def __init__(self, directory):
        self.stack = [directory]
        self.files = []
        self.index = 0

    def __getitem__(self, index):
        while 1:
            try:
                file = self.files[self.index]
                self.index = self.index + 1
            except IndexError:
                # pop next directory from stack
                self.directory = self.stack.pop()
                self.files = os.listdir(self.directory)
                self.index = 0
            else:
                # got a filename
                fullname = os.path.join(self.directory, file)
                st = os.stat(fullname)
                mode = st[stat.ST_MODE]
                if stat.S_ISDIR(mode) and not stat.S_ISLNK(mode):
                    self.stack.append(fullname)
                return fullname, st

for file, st in DirectoryStatWalker("."):
    print file, st[stat.ST_SIZE]
```

```
.\aifc-example-1.py 336
.\anydbm-example-1.py 244
.\array-example-1.py 526
```