

# Current Working Directory in Python

Contributed by Larry Gomez  
 Wednesday, 07 March 2007  
 Last Updated Friday, 11 January 2008

The notion of the current working directory (CWD) turns out to be a key concept in some scripts' execution: it's always the implicit place where files processed by the script are assumed to reside unless their names have absolute directory paths.

The `os.getcwd` lets a script fetch the CWD name explicitly, and `os.chdir` allows a script to move to a new CWD.

{mosgoogle}

## CWD, Files, and Import Paths

When you run a Python script by typing a shell command line such as `python dir1dir2file.py`, the CWD is the directory you were in when you typed this command, not `dir1dir2`.

On the other hand, Python automatically adds the identity of the script's home directory to the front of the module search path such that `file.py` can always import other files in `dir1dir2` no matter where it is run from.

To illustrate, let's write a simple script to echo both its CWD and its module search path:

```
C:PP3rdEdExamplesPP3ESystem>type whereami.py
import os, sys
print 'my os.getcwd =>', os.getcwd( )           # show my cwd execution dir
print 'my sys.path =>', sys.path[:6]           # show first 6 import paths
raw_input( )                                     # wait for keypress if clicked
```

Now, running this script in the directory in which it resides sets the CWD as expected and adds an empty string ("") to the front of the module search path in order to designate the CWD:

```
C:PP3rdEdExamplesPP3ESystem>set PYTHONPATH=C:PP3rdEdExamples
C:PP3rdEdExamplesPP3ESystem>python whereami.py
my os.getcwd => C:PP3rdEdExamplesPP3ESystem
my sys.path => ['', 'C:\PP3rdEd\Examples', 'C:\Program Files\Python
\lib\plat-win', 'C:\Program Files\Python\Lib', 'C:\Program Files\
Python\DLLs', 'C:\Program Files\Python\Lib\lib-tk']
```

But if we run this script from other places, the CWD moves with us (it's the directory where we type commands), and Python adds a directory to the front of the module search path that allows the script to still see files in its own home directory.

For instance, when running from one level up (`..`), the System name added to the front of `sys.path` will be the first directory that Python searches for imports within `whereami.py`; it points imports back to the directory containing the script that was run.

Filenames without complete paths, though, will be mapped to the CWD (`C:PP3rdEdExamplesPP3E`), not the System subdirectory nested there:

```
C:PP3rdEdExamplesPP3ESystem>cd ..
C:PP3rdEdExamplesPP3E>python Systemwhereami.py
my os.getcwd => C:PP3rdEdExamplesPP3E
my sys.path => ['System', 'C:\PP3rdEd\Examples', ...rest same... ]
```

```
C:PP3rdEdExamplesPP3E>cd ..
C:PP3rdEdExamples>python PP3ESystemwhereami.py
my os.getcwd => C:PP3rdEdExamples
my sys.path => ['PP3E\System', 'C:\PP3rdEd\Examples', ...rest same... ]
```

```
C:PP3rdEdExamplesPP3ESystem>cd PP3ESystemApp
C:PP3rdEdExamplesPP3ESystemApp>python ..whereami.py
my os.getcwd => C:PP3rdEdExamplesPP3ESystemApp
my sys.path => ['..', 'C:\PP3rdEd\Examples', ...rest same... ]
```

The net effect is that filenames without directory paths in a script will be mapped to the place where the command was typed (`os.getcwd`), but imports still have access to the directory of the script being run (via the front of `sys.path`).

Finally, when a file is launched by clicking its icon, the CWD is just the directory that contains the clicked file. The following output, for example, appears in a new DOS console box when `whereami.py` is double-clicked in Windows Explorer:

```
my os.getcwd => C:PP3rdEdExamplesPP3ESystem
my sys.path => ['C:\PP3RDED\EXAMPLES\PP3E\SYSTEM', 'C:\PP3rdEd\Examples',
'C:\Program Files\Python\Lib\plat-win', 'C:\Program Files\Python\Lib',
'C:\Program Files\Python\DLLs']
```

In this case, both the CWD used for filenames and the first import search directory are the directory containing the script file. This all usually works out just as you expect, but there are two pitfalls to avoid:

- Filenames might need to include complete directory paths if scripts cannot be sure from where they will be run.
- Command-line scripts cannot use the CWD to gain import visibility to files that are not in their own directories; instead, use `PYTHONPATH` settings and package import paths to access modules in other directories.

As usual for modules, the `PP3Edir1dir2` directory name could also be added to `PYTHONPATH` to make files there visible everywhere without package path imports (though adding more directories to `PYTHONPATH` increases the likelihood of name clashes).

In either case, though, imports are always resolved to the script's home directory or other Python search path settings, not to the CWD.

## CWD and Command Lines

This distinction between the CWD and import search paths explains why many scripts operate in the current working directory are run with command lines such as this one:

```
C:temp>python %X%PyToolscleanpyc-py.py          process cwd
```

In this example, the Python script file itself lives in the directory `C:PP3rdEdExamplesPP3EPyTools`, but because it is run from `C:temp`, it processes the files located in `C:temp` (i.e., in the CWD, not in the script's home directory). To process files elsewhere with such a script, simply `cd` to the directory to be processed to change the CWD:

```
C:temp>cd C:PP2nEdexamples
C:PP3rdEdexamples>python %X%PyToolscleanpyc-py.py  process cwd
```

Because the CWD is always implied, a `cd` command tells the script which directory to process in no less certain terms than passing a directory name to the script explicitly, like this:

```
C:...PP3EPyTools>python find.py *.py C:temp      process named dir
```

In this command line, the CWD is the directory containing the script to be run (notice that the script filename has no directory path prefix); but since this script processes a directory named explicitly on the command line (`C:temp`), the CWD is irrelevant. Finally, if we want to run such a script located in some other directory in order to process files located in yet another directory, we can simply give directory paths to both:

```
C:temp>python %X%PyToolsfind.py *.cxx C:PP3rdEdExamplesPP3E
```

Here, the script has import visibility to files in its `PP3EPyTools` home directory and processes files in the `PP3E` root, but the CWD is something else entirely (`C:temp`). This last form is more to type, of course, but watch for a variety of CWD and explicit script-path command lines.

