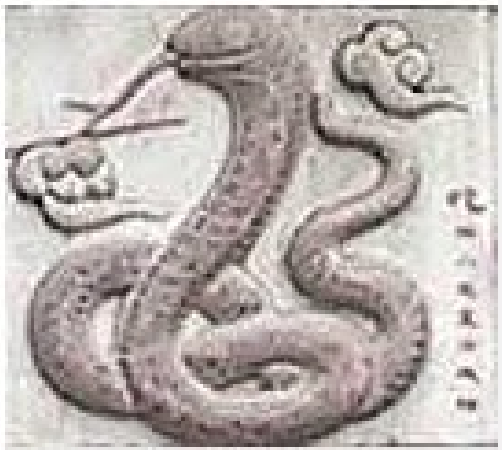
Initialization of sys.path
or
how the Python finds yummy eggs
(and other modules and packages)



Bill Freeman

PySIG NH

May 24, 2012

Disclaimers

- Much of the information below applies to most python implementations, it is most accurate for C python 2.7.3 on linux. Where “2.7” or “27” occur in file or directory names below, they change with the python version in the obvious way.
- Of course Windows is different.
- More surprisingly, Mac OS/x, particularly for the python installed by default, has noticeable differences, and more than just because the default file system’s file names are case insensitive.
- Most other *nix installations track pretty well, though Debian (and derivations) python packages mess around a bit, since they know better).
- Even for “different” OSes, the intent is close.
- Older (and newer) pythons may vary somewhat.

At start-up

At start up, python build an initial value for `sys.path`, which determines where python looks when you ‘‘`import`’’ something (other than sys and __buitins __). This begins with the calculation of values for `prefix` and `exec_prefix`. These values, when finalized, are also used to set the values of the python objects `sys.prefix` and `sys.exec_prefix`.

- `PREFIX` comes from `./configure` via the Makefile. It defaults, if unspecified, to ‘`/usr/local`’ (on most *nix), but can be changed using the `--prefix` argument to `./configure`. It is used to find the “platform independent” parts of the installation.
- `EXEC_PREFIX` usually comes from `./configure` via the Makefile. It defaults, if unspecified, to the same as `PREFIX`, but can be changed using the `--exec-prefix` argument to `./configure`. It is used to find the “platform specific” parts of the installation. If it differs `prefix`, it is probably intended to be under `prefix` in the directory hierarchy, but can work otherwise.

Finding the directory containing the executable

Python figures out the directory from which the python executable was loaded. I'll refer to this as `exec_dir` below.
It's not too hard on *nix, since the path of the executable is usually provided as the first command line argument (`argv[0]`. But even here, that might just be a symbolic link to the executable. (In fact it usually is, `python` being a symlink to `python2`, and that being a symlink to `python2.7`, but these are all in the same directory, so `exec_dir` winds up the same for all of these.)
`exec_dir` should be the ``'bin'' sub directory of `exec_prefix`. (Since `prefix` and `exec_prefix` are normally the same, you might be forgiven for thinking the it is the ``'bin'' sub directory of `prefix`.)

PYTHONHOME

If the environment variable `PYTHONHOME` is set (and if you haven't specified the ignore environment flag, `-E`) it is used to set both `prefix` and `exec_prefix`, without any of the other efforts detailed below. Python assumes that you know what you are doing.

- If `PYTHONEHOME` is a single path, it is used for both `prefix` and `exec_prefix`.
- If `PYTHONHOME` contains a delimiter (usually `':'`, but other OSes may need that as part of paths), it is taken as two paths separated by that delimiter. The first is used for `prefix`, and the second is used for `exec_prefix`.
- Either way, the `prefix` sub directory `''lib/python2.7''` is expected to contain `''os.py''` or its `''os.pyc''` or `''os.pyo''` derivatives, and `exec_prefix` sub directory `''lib/python2.7''` is expected to contain a further sub directory `''lib-dynload''`.

Running before Installation

You can run python in the build directory before you ‘‘`make install`’’ or ‘‘`make altinstall`’’, and running ‘‘`make test`’’ first may well be a good idea.
Python detects running from the build directory by looking in `exec_dir` for a file at sub path ‘‘`Modules/Setup`’’. If found:

- `exec_dir` + ‘`Modules`’ becomes `exec_prefix` with no further checking.
- Python checks for `exec_dir` + ‘`/Lib/os.py`’. If it is found, `exec_dir` + ‘`/Lib`’ becomes `prefix`. If not, `prefix` is found as detailed on subsequent slides.

Note that these values will never be used to set `sys.prefix` and `sys.exec_prefix`: the Makefile values will be used instead. These values, however, are used in constructing `sys.path`.

The Search for Landmarks

If they haven't already been determined, `prefix` and/or `exec_prefix` are sought in `exec_dir`, its parent directory, its grandparent directory, etc., until either they are found, or the root directory has been searched.

- A candidate directory is a suitable `prefix` if the sub directory ‘‘`lib/python2.7`’’ exists and contains a module ‘‘`os.py`’’. ("module" means that the directory can contain the ‘‘`os.py`’’ file itself, or the derived ‘‘`os.pyc`’’ or ‘‘`os.pyo`’’ files.)
- A candidate directory is a suitable `exec_prefix` if the sub directory ‘‘`lib/python2.7`’’ exists and contains a sub directory ‘‘`lib-dynload`’’.

This was the last gasp. If we still don't have a value for `prefix`, the Makefile version is used. Similarly, if we still don't have a value for `exec_prefix`, the Makefile value (or `prefix`, if the Makefile didn't supply `EXEC_PREFIX`) is used. If both were defaulted, python probably won't run well.

Time to make the doughnuts – er, initial sys.path

In C code:

- If there is a (non-ignored) PYTHONPATH environment variable, it is put at the front of sys.path.
- Next comes the "zip_path". Currently always based on the --prefix configuration setting (bug?), 'lib/python27.zip' is joined on. (Note the lack of a dot between the digits, probably for 8.3 file name schemes.)
- Now for each path in the configure time PYTHONPATH, if the path is absolute, it is added directly; if it is relative it is prefixed with prefix. The default would be PREFIX + 'lib/python2.7' and EXEC_PREFIX + 'lib/python2.7/lib-dynload', but the Makefile can override it. I see ''':plat-linux2:lib-tk:lib-old''' in gdb on linux. Since this starts with the delimiter (':'), the first path is an empty string, and results in just prefix being added.
- Finally, add exec_prefix.

Additions by site.py

Import of sys and __builtins__ is emmulated, then zipimport is imported. Then, unless suppressed (-S), site.py, normally found in the same directory as os.py, is imported. The sys.path related stuff is:

- If user site packages are enabled and the directory exists, append the directory ($HOME/.local/lib/python2.7/site-packages/ in my build) to sys.path, then process any .pth files there.
- In any case, for each distinct prefix (that is, just prefix if exec_prefix is the same), create some candidate site packages directory paths. For me, on linux, this means joining the prefix with ‘‘lib/python2.7/site-packages’’, and also with ‘‘lib/site-python’’.
- For each of those directories which exists, append the directory to sys.path, then process any .pth files there.

When site.py is done, the C code inserts an empty string, for the current working directory, at the front of sys.path.

Processing `.pth` files

`.pth` files in a directory are sorted by filename, and processed in that order. Each such file is processed line by line.

- Lines beginning with # are ignored.
- Lines beginning with ‘‘`import`’’ followed by space or tab get exec'ed. (Nothing is done with the result, but the module is free, in the course of being imported, to perform its own `sys.path` modifications.)
- Other lines should contain a path to a directory, either absolute, or relative to the directory containing the `.pth` file. If that directory exists, it is appended to `sys.path`.
- Anything else, including a blank line, is an error, and processing of the `.pth` file is terminated, with a message to stderr.

Debian differences

If you are using a python that came from a .deb package, its `site.py` has been modifed to suit Debian's tastes. Debian has decided that site package directories would be best named "dist-packages".

- While the user site's ``.local/lib/python2.7/site-packages'' will still be searched, so will ``.local/lib/python2.7/dist-packages'' and ``.local/local/lib/python2.7/dist-packages.
- Under the prefix directories, ``lib/python2.7/site-packages'' and ``lib/site-python'' are no longer searched. Instead, ``local/lib/python2.7/dist-packages'' and ``lib/python2.7/dist-packages'' are searched (in that order).
- All these changes can be accomplished via modifying site.py.

I'm going to always build my own pythons on Debian or Ubuntu. How about you? Adding the dist-packages directories might be OK with me, but removing those standard directories is not.

How virtualenv works

virtualenv takes advantage of the `prefix`/`exec_prefix` search starting from the directory of the executable.

- The virtualenv directory contains ‘‘`lib/python2.7/os.py`’’, ‘‘`lib/python2.7/lib-dynload/`’’, and ‘‘`bin/python`’’.
- When you “activate” the virtualenv, little more is done than to add that ‘‘`bin`’’ directory to the front of your `PATH` environment variable, such that whey you type ‘‘`python`’’, the python there is run.
- That python finds the virtualenv directory as both `prefix` and `exec_prefix`, so it sees things installed within those directories, while other pythons do not.
- The `pip` installed there uses `prefix` and `exec_prefix` to determine where to install things, so they wind up installed only for that particular python.