Installing Python 2 on Windows

First, download the latest version of Python 2.7 from the official website. If you want to be sure you
are installing a fully up-to-date version, click the Downloads > Windows link from the home page of the
Python.org web site .

TheWindows version is provided as an MSI package. To install it manually, just double-click the file. The
MSI package format allows Windows administrators to automate installation with their standard tools.
By design, Python installs to a directory with the version number embedded, e.g. Python version 2.7 will
install at C:\Python27\, so that you can have multiple versions of Python on the same system without
conflicts. Of course, only one interpreter can be the default application for Python file types.

It also does not automatically modify the PATH environment variable, so that you always have control over which copy of Python is run.

Typing the full path name for a Python interpreter each time quickly gets tedious, so add the directories for your default Python version to the PATH. Assuming that your Python installation is in C:\Python27\,
add this to your PATH:

```
C:\Python27\;C:\Python27\Scripts\
```

You can do this easily by running the following in powershell:

```
[Environment]::SetEnvironmentVariable("Path",
"$env:Path;C:\Python27\;C:\Python27\Scripts\", "User")
```

This is also an option during the installation process.
The second (Scripts) directory receives command files when certain packages are installed, so it is a
very useful addition. You do not need to install or configure anything else to use Python. Having said
that, I would strongly recommend that you install the tools and libraries described in the next section

before you start building Python applications for real-world use. In particular, you should always install
Setuptools, as it makes it much easier for you to use other third-party Python libraries.

Setting environment variables

Windows has a built-in dialog for changing environment variables (following guide applies to XP classical view): Right-click the icon for your machine (usually located on your Desktop and called "My Computer") and choose Properties there. Then, open the Advanced tab and click the Environment Variables button.

In short, your path is:

My Computer ‣ Properties ‣ Advanced ‣ Environment Variables

In this dialog, you can add or modify User and System variables. To change System variables, you need non-restricted access to your machine (i.e. Administrator rights).

Another way of adding variables to your environment is using the **set** command:

```
set PYTHONPATH=%PYTHONPATH%;C:\My_python_lib
```

To make this setting permanent, you could add the corresponding command line to your `autoexec.bat`. **msconfig** is a graphical interface to this file.

Viewing environment variables can also be done more straight-forward: The command prompt will expand strings wrapped into percent signs automatically:

```
echo %PATH%
```

Consult **set /?** for details on this behaviour.

Finding the Python executable

Besides using the automatically created start menu entry for the Python interpreter, you might want to start Python in the DOS prompt. To make this work, you need to set your `%PATH%` environment variable to include

the directory of your Python distribution, delimited by a semicolon from other entries. An example variable could look like this (assuming the first two entries are Windows' default):

```
C:\WINDOWS\system32;C:\WINDOWS;C:\Python25
```

Typing **python** on your command prompt will now fire up the Python interpreter. Thus, you can also execute your scripts with command line options, see Command line documentation.

Finding modules

Python usually stores its library (and thereby your site-packages folder) in the installation directory. So, if you had installed Python to `C:\Python\`, the default library would reside in `C:\Python\Lib\` and third-party modules should be stored in `C:\Python\Lib\site-packages\`.

This is how `sys.path` is populated on Windows:

- An empty entry is added at the start, which corresponds to the current directory.
- If the environment variable PYTHONPATH exists, as described in Environment variables, its entries are added next. Note that on Windows, paths in this variable must be separated by semicolons, to distinguish them from the colon used in drive identifiers (C:\ etc.).
- Additional "application paths" can be added in the registry as subkeys of \SOFTWARE\Python\PythonCore*version*\PythonPath under both the HKEY_CURRENT_USER and HKEY_LOCAL_MACHINE hives. Subkeys which have semicolon-delimited path strings as their default value will cause each path to be added to sys.path. (Note that all known installers only use HKLM, so HKCU is typically empty.)
- If the environment variable PYTHONHOME is set, it is assumed as "Python Home". Otherwise, the path of the main Python

executable is used to locate a “landmark file” (`Lib\os.py`) to deduce the “Python Home”. If a Python home is found, the relevant sub-directories added to `sys.path` (`Lib`, `plat-win`, etc) are based on that folder. Otherwise, the core Python path is constructed from the PythonPath stored in the registry.

- If the Python Home cannot be located, no `PYTHONPATH` is specified in the environment, and no registry entries can be found, a default path with relative entries is used (e.g. `.\Lib;.\plat-win`, etc).

The end result of all this is:

- When running `python.exe`, or any other .exe in the main Python directory (either an installed version, or directly from the PCbuild directory), the core path is deduced, and the core paths in the registry are ignored. Other “application paths” in the registry are always read.

- When Python is hosted in another .exe (different directory, embedded via COM, etc), the “Python Home” will not be deduced, so the core path from the registry is used. Other “application paths” in the registry are always read.
- If Python can’t find its home and there is no registry (eg, frozen .exe, some very strange installation setup) you get a path with some default, but relative, paths.