# 2. Using Python on Unix platforms

## 2.1. Getting and installing the latest version of Python

### 2.1.1. On Linux

Python comes preinstalled on most Linux distributions, and is available as a package on all others. However there are certain features you might want to use that are not available on your distro's package. You can easily compile the latest version of Python from source.

In the event that Python doesn't come preinstalled and isn't in the repositories as well, you can easily make packages for your own distro. Have a look at the following links:

**See also:**

**https://www.debian.org/doc/manuals/maint-guide/first.en.html**
for Debian users

**https://en.opensuse.org/Portal:Packaging**
for OpenSuse users

**https://docs.fedoraproject.org/en-US/Fedora_Draft_Documentation/0.1/html/RPM_Guide/ch-creating-rpms.html**
for Fedora users

**http://www.slackbook.org/html/package-management-making-packages.html**
for Slackware users

### 2.1.2. On FreeBSD and OpenBSD

- FreeBSD users, to add the package use:

```
pkg install python3
```

- OpenBSD users, to add the package use:

```
pkg_add -r python

pkg_add ftp://ftp.openbsd.org/pub/OpenBSD/4.2/packages/<insert yo
```

For example i386 users get the 2.5.1 version of Python using:

```
pkg_add ftp://ftp.openbsd.org/pub/OpenBSD/4.2/packages/i386/pytho
```

### 2.1.3. On OpenSolaris

You can get Python from OpenCSW. Various versions of Python are available and can be installed with e.g. `pkgutil -i python27`.

## 2.2. Building Python

If you want to compile CPython yourself, first thing you should do is get the source. You can download either the latest release's source or just grab a fresh clone. (If you want to contribute patches, you will need a clone.)

The build process consists in the usual

```
./configure
make
make install
```

invocations. Configuration options and caveats for specific Unix platforms are extensively documented in the README.rst file in the root of the Python source tree.

**Warning:** `make install` can overwrite or masquerade the `python3` binary. `make altinstall` is therefore recommended instead of `make install` since it only installs *`exec_prefix`*`/bin/python`*`version`*.

## 2.3. Python-related paths and files

These are subject to difference depending on local installation conventions; `prefix` (`${prefix}`) and `exec_prefix` (`${exec_prefix}`) are installation-dependent and should be interpreted as for GNU software; they may be the same.

For example, on most Linux systems, the default for both is `/usr`.

| File/directory | Meaning |
| --- | --- |
| *`exec_prefix`*`/bin/python3` | Recommended location of the interpreter. |
| *`prefix`*`/lib/python`*`version`*, *`exec_prefix`*`/lib/python`*`version` | Recommended locations of the directories containing the standard modules. |

| File/directory | Meaning |
| --- | --- |
| *prefix*/include/python*version*, *exec_prefix*/include/python*version* | Recommended locations of the directories containing the include files needed for developing Python extensions and embedding the interpreter. |

## 2.4. Miscellaneous

To easily use Python scripts on Unix, you need to make them executable, e.g. with

```
$ chmod +x script
```

and put an appropriate Shebang line at the top of the script. A good choice is usually

```
#!/usr/bin/env python3
```

which searches for the Python interpreter in the whole `PATH`. However, some Unices may not have the **env** command, so you may need to hardcode `/usr/bin/python3` as the interpreter path.

To use shell commands in your Python scripts, look at the `subprocess` module.

## 2.5. Editors and IDEs

There are a number of IDEs that support Python programming language. Many editors and IDEs provide syntax highlighting, debugging tools, and PEP-8 checks.

Please go to Python Editors and Integrated Development Environments for a comprehensive list.