Installing Python modules

- Key terms
- Basic usage
- How do I ...?
  - ... install packages just for the current user?
  - ... install scientific Python packages?
  - ... work with multiple versions of Python installed in parallel?
- Common installation issues
  - Installing into the system Python on Linux
  - Pip not installed
  - Installing binary extensions

As a popular open source development project, Python has an active supporting community of contributors and users that also make their software available for other Python developers to use under open-source license terms.

This allows Python users to share and collaborate effectively, benefiting from the solutions others have already created to common (and sometimes even rare!) problems, as well as potentially contributing their own solutions to the common pool.

This guide covers the installation part of the process. For a guide to creating and sharing your own Python projects, refer to the Python packaging user guide.

> [!NOTE] For corporate and other institutional users, be aware that many organisations have their own policies around using and contributing to open source software. Please take such policies into account when making use of the distribution and installation tools provided with Python.

Key terms

- `pip` is the preferred installer program. It is included by default with the Python binary installers.
- A *virtual environment* is a semi-isolated Python environment that allows packages to be installed for use by a particular application, rather than being installed system wide.
- `venv` is the standard tool for creating virtual environments. It defaults to installing `pip` into all created virtual environments.
- `virtualenv` is a third-party alternative (and predecessor) to `venv`.
- The Python Package Index (PyPI) is a public repository of open source licensed packages made available for use by other Python users.
- The Python Packaging Authority is the group of developers and documentation authors responsible for the maintenance and evolution of the standard packaging tools and the associated metadata and file format standards. They maintain a variety of tools, documentation, and issue trackers on GitHub.

3.5 The use of `venv` is now recommended for creating virtual environments.

Python Packaging User Guide: Creating and using virtual environments

Basic usage

The standard packaging tools are all designed to be used from the command line.

The following command will install the latest version of a module and its dependencies from PyPI:

```
python -m pip install SomePackage
```

> [!NOTE] For POSIX users (including macOS and Linux users), the examples in this guide assume the use of a `virtual environment`.
>
> For Windows users, the examples in this guide assume that the option to adjust the system PATH environment variable was selected when installing Python.

It's also possible to specify an exact or minimum version directly on the command line. When using comparator operators such as `>`, `<` or some other special character which get interpreted by shell, the package name and the version should be enclosed within double quotes:

```
python -m pip install SomePackage==1.0.4    # specific version
python -m pip install "SomePackage>=1.0.4"  # minimum version
```

Normally, if a suitable module is already installed, attempting to install it again will have no effect. Upgrading existing modules must be requested explicitly:

```
python -m pip install --upgrade SomePackage
```

More information and resources regarding `pip` and its capabilities can be found in the Python Packaging User Guide.

Creation of virtual environments is done through the `venv` module. Installing packages into an active virtual environment uses the commands shown above.

Python Packaging User Guide: Installing Python Distribution Packages

How do I ...?

These are quick answers or links for some common tasks.

... install packages just for the current user?

Passing the `--user` option to `python -m pip install` will install a package just for the current user, rather than for all users of the system.

... install scientific Python packages?

A number of scientific Python packages have complex binary dependencies, and aren't currently easy to install using `pip` directly. It will often be easier for users to install these packages by other means rather than attempting to install them with `pip`.

Python Packaging User Guide: Installing Scientific Packages

... work with multiple versions of Python installed in parallel?

On Linux, macOS, and other POSIX systems, use the versioned Python commands in combination with the `-m` switch to run the appropriate copy of `pip`:

```
python3    -m pip install SomePackage  # default Python 3
python3.14 -m pip install SomePackage  # specifically Python 3.14
```

Appropriately versioned `pip` commands may also be available.

On Windows, use the `py` Python launcher in combination with the `-m` switch:

```
py -3    -m pip install SomePackage  # default Python 3
py -3.14 -m pip install SomePackage  # specifically Python 3.14
```

> Once the Development & Deployment part of PPUG is fleshed out, some of those sections should be linked from new questions here (most notably, we should have a question about avoiding depending on PyPI that links to https://packaging.python.org/en/latest/guides/index-mirrors-and-caches/)

Common installation issues

Installing into the system Python on Linux

On Linux systems, a Python installation will typically be included as part of the distribution. Installing into this Python installation requires root access to the system, and may interfere with the operation of the system package manager and other components of the system if a component is unexpectedly upgraded using `pip`.

On such systems, it is often better to use a virtual environment or a per-user installation when installing packages with `pip`.

Pip not installed

It is possible that `pip` does not get installed by default. One potential fix is:

```
python -m ensurepip --default-pip
```

There are also additional resources for installing pip.

Installing binary extensions

Python once relied heavily on source-based distribution, with end users being expected to compile extension modules from source as part of the installation process.

With the introduction of the binary wheel format, and the ability to publish wheels through PyPI, this problem is diminishing, as users are more regularly able to install pre-built extensions rather than needing to build them themselves.

Some of the solutions for installing scientific software that are not yet available as pre-built wheel files may also help with obtaining other binary extensions without needing to build them locally.

Python Packaging User Guide: Binary Extensions

□□□ #2
□ □□□ □□□ 2026-10-31 17:01:23 UTC
□ □□□ □□□ 2026-10-31 17:01:25 UTC