Python - Install and Setup - Windows 10

A brief intro to installing and managing the Python programming language, including virtual environments for development, on Windows 10.

Contents

Intro

There are various install and support options for the Python programming language depending upon your chosen operating system.

Python 3.x is currently preferable for development unless there is a specific need to support legacy code from Python 2.x. There are various download and install options, again depending upon your chosen OS.

Standard downloads are available from the Python.org website. Install options are available for most OSs, including OS X, Windows, Linux &c.

Install Python 3.x

We can start our Windows 10 install by downloading the latest version of Python 3.x, for example version 3.6. Downloads are available for 32 bit and 64 bit versions of Windows 10 at the following URL,

- Python 3.6.0

After executing this install, we may choose various options including

- documentation
- pip
- tcl/tk and IDLE
- Python test suite
- py launcher...

We're then given a list of *Advanced Options*, which we may select depending upon our system setup and requirements. For example, we may select the following *advanced options*,

- Associate files with Python
- Create shortcuts for installed applications
- Add Python to environment variables
- Precompile standard library
- Download debugging symbols

After selecting these advanced options, Python 3.x will then be installed on our local host system.

We can test this install from the standard Windows console as follows,

```
python --version
```

This command will return the current installed version for Python, e.g.

```
Python 3.6.0
```

We can then use Python's `pip` to install and manage any required packages, where applicable, for development. We can check it's available using the following command, e.g.

```
> pip --version
pip 9.0.1 from c:\... (python 3.6)
```

We may then search for a package with the following example command,

```
> pip search package_name
```

and then install our chosen package, e.g.

```
> pip install package_name
```

Virtual environment

With Python, we can create programming environments with the tool `Pyenv`. This allows us to create any required virtual environments for project development.

A virtual environment allows us to isolate a project &c., thereby ensuring each project has its own set of dependencies. This helps ensure that each project remains isolated, and will not pollute or disrupt another project's development.

We can also use virtual environments to handle development of different versions, and easily handle required dependencies. This can be particularly useful for working with third-party packages.

It's often best to use a specific folder for adding such virtual environments, e.g.

```
> mkdir dev-environments
> cd dev-environments
```

Within this specific directory, we can then start to create our Python environments for application development and testing. e.g.

```
> python -m venv test_env
```

venv directory

Each virtual environment directory on Windows 10 includes the following default files and sub-directories,

- `Include` sub-directory - compiles required packages
  - `Lib` sub-directory - includes a copy of the Python version, e.g. `python3.6`, by default. It will then also save any third-party modules added to the project
- `Scripts` sub-directory - includes a copy of the Python binary along with Pip, easy_install &c.
  - `pyvenv.cfg` - config file points to Python install used to generate virtual environment

These files and sub-directories help to isolate a given project from the broader context of a host machine. In effect, the virtual environment is isolated from the system files.

activate venv

To use each virtual environment, we'll need to activate it using the following type of command, e.g.

```
> win_env\Scripts\activate.bat
```

where `win_env` is the name of the virtual environment to activate.

This command is using the provided activate script, which will then prefix the system environment with the name of the new virtual environment, e.g.

```
(win_env) C:\Users\username\development\python-dev\dev-environments>
```

This prefix simply informs us that the virtual environment is now active. Any apps developed within this environment will be isolated, and use their own installed versions, packages, &c.

deactivate venv

We can then exit the current virtual environment using the following command,

```
> deactivate
```

This will simply exit the current *venv*, and return us to the standard command line.

Install PyGame

We may install the latest version of Pygame using `pip`,

```
> pip install pygame
```

and then briefly test this install as follows,

```
> python
>>> import pygame
>>> pygame.init() #pygame will open, ready for display window &c.
>>> pygame.display.set_mode((800, 600)) # set and open display window for pygame...
>>> raise SystemExit # exit current pygame window...
```

This will simply import the installed version of the Pygame module, and then create a window with a size of `800x600` pixels. Then, we close the window and exit the Python interpreter.