Python

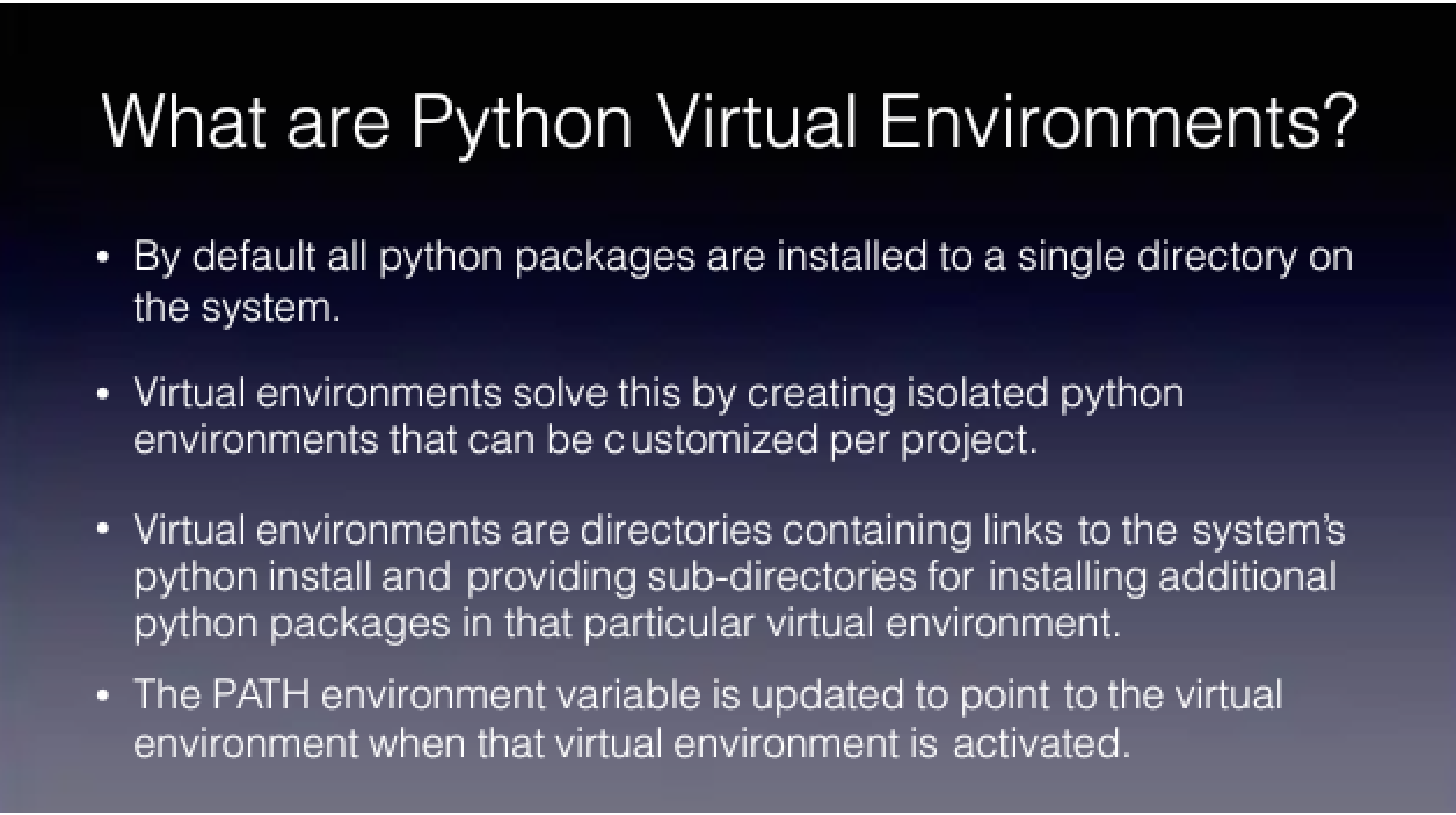
Overview of Python Virtual Environments

Python applications will often use packages and modules that don’t come as part of the standard library. Applications will sometimes need a specific version of a library, because the application may require that a particular bug has been fixed or the application may be written using an obsolete version of the library’s interface.

This means it may not be possible for one Python installation to meet the requirements of every application. If application A needs version 1.0 of a particular module but application B needs version 2.0, then the requirements are in conflict and installing either version 1.0 or 2.0 will leave one application unable to run.

The solution for this problem is to create a virtual environment, a self-contained directory tree that contains a Python installation for a particular version of Python, plus a number of additional packages.

Different applications can then use different virtual environments. To resolve the earlier example of conflicting requirements, application A can have its own virtual environment with version 1.0 installed while application B has another virtual environment with version 2.0. If application B requires a library be upgraded to version 3.0, this will not affect application A's environment.



What are Python Virtual Environments?

- By default all python packages are installed to a single directory on the system.
- Virtual environments solve this by creating isolated python environments that can be customized per project.
- Virtual environments are directories containing links to the system's python install and providing sub-directories for installing additional python packages in that particular virtual environment.
- The PATH environment variable is updated to point to the virtual environment when that virtual environment is activated.

Setting Up a Python Virtual Environment in Python 2.7

- Install virtualenv utility via “pip install virtualenv”.
- Create a new virtual environment with the command “virtualenv <NameOfVirtualEnv>.
- Activate your virtual environment by sourcing the activate script in the virtual environments bin directory (i.e. source ./<NameOfVirtualEnv>/bin/activate).
- Deactivate your virtual environment with the “deactivate” command.
- Delete your virtual environment by deleting its directory.

Setting Up a Python Virtual Environment in Python 3

- Python 3 comes with a virtual environment module built-in called venv.
- Virtualenv can also be used with the python 3 but venv is what's recommended by the python community as it is built-in to python 3, creates smaller virtual environments and is extendable.
- The only difference with creating, activating, deactivating, or deleting virtual environments with venv versus virtualenv is the creation command.
- To create a virtual environment with venv you run the command "python3 -m venv <VirtualEnvironmentName>.

12.2. Creating Virtual Environments

The module used to create and manage virtual environments is called `venv`. `venv` will usually install the most recent version of Python that you have available. If you have multiple versions of Python on your system, you can select a specific Python version by running `python3` or whichever version you want.

To create a virtual environment, decide upon a directory where you want to place it, and run the `venv` module as a script with the directory path:

```
python -m venv tutorial-env
```

This will create the `tutorial-env` directory if it doesn't exist, and also create directories inside it containing a copy of the Python interpreter and various supporting files.

A common directory location for a virtual environment is `.venv`. This name keeps the directory typically hidden in your shell and thus out of the way while giving it a name that explains why the directory exists. It also prevents clashing with `.env` environment variable definition files that some tooling supports.

Once you've created a virtual environment, you may activate it.

On Windows, run:

```
tutorial-env\Scripts\activate.bat
```

On Unix or MacOS, run:

```
source tutorial-env/bin/activate
```

(This script is written for the bash shell. If you use the **csh** or **fish** shells, there are alternate `activate.csh` and `activate.fish` scripts you should use instead.)

Activating the virtual environment will change your shell's prompt to show what virtual environment you're using, and modify the environment so that running `python` will get you that particular version and installation of Python. For example:

```
$ source ~/envs/tutorial-env/bin/activate
(tutorial-env) $ python
Python 3.5.1 (default, May  6 2016, 10:59:36)
  ...
>>> import sys
>>> sys.path
['', '/usr/local/lib/python35.zip', ...,
'~/envs/tutorial-env/lib/python3.5/site-package
>>>
```

To deactivate a virtual environment, type:

```
deactivate
```

into the terminal.