Isolating Dependencies with Virtual Environments

1. **Welcome & Course Overview** ✅
2. **Managing Third-Party Dependencies With pip** ✅
3. **Isolating Dependencies With Virtual Environments**
4. Finding Quality Python Packages
5. Setting Up Reproducible Environments & Application Deploys
6. Course Conclusion

Isolating Dependencies with Virtual Environments

1. Introduction to Virtual Environments
2. Creating and Activating a Virtual Environment
3. Installing Packages Into a Virtual Environment
4. Deactivating Virtual Environments
5. Destroying Virtual Environments
6. My Virtual Environment Workflow

Why do we need isolation?

By default, `pip` installs all packages
in a single **shared environment**:

→ Version Conflicts 🚨

Version Conflicts: Packages

Project #1 needs: `django` 1.8, `requests` 2.0

Project #2 needs: `django` 1.10, `requests` 2.13

Version Conflicts: Interpreter

Project #1 needs Python 2.7

Project #2 needs Python 3.6

Virtual Environments to the rescue:

- Isolate Python dependencies by project
- Works for packages and interpreters

→ Individual *sandboxes* for each project

Creating and activating a virtual environment

Demo

Recap

Python 3.3+:

```
$ python3 -m venv venv
$ source ./venv/bin/activate
```

Python 2.x:

```
$ pip install virtualenv
$ virtualenv venv
$ source ./venv/bin/activate
```

Recap

Windows:

```
C:\> python -m venv venv
C:\> venv\Scripts\activate
```

Installing packages into a virtual environment

Demo

Leaving (deactivating) virtual environments

Demo

Destroying virtual environments

Demo

My virtualenv workflow

Demo

My virtualenv workflow

~/.bash_profile:

```
alias ae='deactivate &> /dev/null; source ./venv/bin/activate'
alias de='deactivate'
```

Usage:

```
$ cd project
$ ae
# (work on the project)
$ de
```

Isolating Dependencies with Virtual Environments ✅

1. **Introduction to Virtual Environments** ✅
2. **Creating and Activating a Virtual Environment** ✅
3. **Installing Packages Into a Virtual Environment** ✅
4. **Deactivating Virtual Environments** ✅
5. **Destroying Virtual Environments** ✅
6. **My Virtual Environment Workflow** ✅

Summary

- Virtual environments keep your project dependencies **isolated**.
- They help you **avoid version conflicts** between packages and different versions of the Python runtime.
- As a **best practice**, all of your Python projects should use virtual environments to store their dependencies.