HyperionDev

Setting Up a Virtual Environment

Additional reading

Visit our website

Introduction

In this guide, we will walk you through the process of setting up a virtual environment for your Python project using either the built-in venv module or the external virtualenv package. Virtual environments isolate your project's dependencies, providing a clean and self-contained environment.

We will explore the generation of a **requirements.txt** file to document and manage your project's dependencies. Keeping this file up to date is crucial for ensuring a reproducible development environment. By following these steps, you will be well-equipped to initiate and maintain a Python project with best practices in mind, fostering readability, consistency, and ease of collaboration.

Setting up a virtual environment

A virtual environment is a self-contained space in a computer system that isolates software, thereby managing dependencies independently. Widely used in software development, it ensures consistent and reproducible application behaviour across different computing environments.

Using venv

Follow these steps to use the venv module (built into Python 3.3 and newer versions) to create a virtual environment for your project:

1. Open a terminal or command prompt on your computer or in your IDE (such as VS Code).
2. Use the cd command to go to the directory where you want to create your virtual environment, for example:

```
cd your_path/to/your/project
```

3. Run the following command to create a virtual environment. Replace the second `venv` with the name you want to give your virtual environment.

```
python -m venv venv
```

Note: If you're using macOS, the command python typically refers to Python 2, which is no longer recommended. Instead, use python3 to ensure that your virtual environment is created with Python 3, which is the latest, actively maintained, and supported version.

```
python3 -m venv venv
```

4. Activate the virtual environment.
    a. On Windows, use:

    ```
    venv\Scripts\activate
    ```

    b. On Linux or macOS, use:

    ```
    source venv/bin/activate
    ```

    After activation, your terminal prompt should change to show that you are now in the virtual environment. You should see something like this:

    ```
    (.venv) C:\Users\your_path\to\your\project>
    ```

5. Use pip to install Python packages within your virtual environment.

    ```
    pip install pillow
    ```

6. When you are done working in your virtual environment, you can deactivate it.
    a. On Windows, use:

    ```
    deactivate
    ```

b. On Linux or macOS, use:

```
source venv/bin/deactivate
```

Using virtualenv

The `virtualenv` package is an external package we can use to create a virtual environment for your project.

If you prefer using `virtualenv`, you need to install it globally.

1. Install `virtualenv` using pip. This is a one-time step.

```
pip install virtualenv
```

2. Use the cd command to go to the directory where you want to create your virtual environment.

```
cd your_path/to/your/project
```

3. Run the following command to create a virtual environment. Replace `venv` with the name you want to give your virtual environment.

```
virtualenv venv
```

4. Activate the virtual environment.

   a. On Windows, use:

```
venv\Scripts\activate
```

   b. On Linux or macOS, use:

```
source venv/bin/activate
```

After activation, your terminal prompt should change to show that you are now in the virtual environment. You should see something like this:

```
(.venv) C:\Users\your_path\to\your\project>
```

5. Use pip to install Python packages within your virtual environment.

```
pip install pillow
```

6. When you are done working in your virtual environment, you can deactivate it.
    a. On Windows, use:

```
deactivate
```

    b. On Linux or macOS, use:

```
source venv/bin/deactivate
```

Take note

Remember to activate your virtual environment whenever you work on your project. This ensures that the dependencies you install are specific to your project and do not interfere with other projects or the system-wide Python installation.

Generating a requirements file

A **requirements.txt** file in a project serves as a crucial document outlining the specific Python packages and their corresponding versions required for the project to run successfully. This file captures the dependencies, allowing for easy replication of the project's environment on different systems. Below, you can see how to create a **requirements.txt** file.

1. Activate your virtual environment (if you are using one):

```
source venv/bin/activate  # On Linux/macOS
venv\Scripts\activate   # On Windows
```

2. Run the `pip freeze` command to generate the **requirements.txt** file:

```
pip freeze > requirements.txt
```

This command lists all installed packages and their versions.

Here's an example of a requirements.txt file:

```
pillow==11.0.0
```

You can customise the **requirements.txt** file based on your project's dependencies. It's good practice to regularly update this file as you add or remove dependencies in your project.

3. If you want to include only the packages needed for your project (excluding development dependencies), you can use the pip freeze --exclude-editable option:

```
pip freeze --exclude-editable > requirements.txt
```

This command will exclude packages installed in editable mode from the **requirements.txt** file. Editable installations are often used during development with packages installed in 'editable' mode, e.g., using the command:

```
pip install -e .
```

The “.” at the end of the command refers to the current directory, indicating that the package in editable mode should be installed from the current project directory. Excluding it ensures that only production dependencies are listed.

Summary

In conclusion, this guide has equipped you with the knowledge to kick-start your Python project development in a structured and efficient manner. By embracing virtual environments and the generation of a **requirements.txt** file, you will establish a solid foundation for building scalable, maintainable, and organised applications. Incorporating these best practices not only enhances the clarity and readability of your code, but also streamlines collaboration and ensures a smooth development experience. As you continue on your coding journey, the skills and techniques outlined here will contribute to the success of your Python projects, fostering a robust and sustainable development workflow. Remember that, if you get stuck, you can contact your mentor for help.