ADDITIONAL READING

Project Setup Guide

Visit our website

WELCOME TO THE PROJECT SETUP GUIDE!

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

To enhance the development process, we introduce the concept of linting and demonstrate how to integrate **PEP8 linting** into your project using **Flake8**. Additionally, we cover the installation and configuration of autoformatters like black, promoting code consistency and adherence to PEP8 guidelines. These tools can be seamlessly integrated into popular code editors like Visual Studio Code and Sublime Text.

Lastly, we 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 are well-equipped to initiate and maintain a Python project with best practices in mind, fostering readability, consistency, and ease of collaboration.

SET UP A VIRTUAL ENVIRONMENT

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

1. **Open your command line interface (CLI)**: this could be Command Prompt (Windows), Terminal (macOS/Linux), or any other terminal emulator you prefer.
2. **Navigate to your task directory**: use `cd` to change into the directory where your task files are located. For example:

```
cd TaskFolder
```

3. **Create a virtual environment**: use the `python -m venv` command followed by the name of your virtual environment. For example:

```
python -m venv myenv
```

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 myenv
```

4. **Activate the virtual environment**: activation sets up the terminal session to use the Python interpreter and libraries associated with the virtual environment. Use the appropriate command based on your operating system.

    - Windows:

    ```
    myenv\Scripts\activate
    ```

    - MacOS/Linux:

    ```
    source myenv/bin/activate
    ```

5. **Deactivate the virtual environment once you're done**:

    ```
    deactivate
    ```

    Since the Notebook has not been installed yet, do not deactivate the virtual environment at this point. You should deactivate it once you have finished installing the Notebook or any other packages within the virtual environment.

For further insights into Python virtual environments, refer to the official **Python venv documentation**.

ADDING A PEP8 LINTER

To add **PEP8 linting** to your Python project, you can use a tool like **Flake8**, which is a popular linting tool that checks your code against the style guide outlined in PEP8. Here are the steps to add PEP8 linting to your project:

Installing Flake8:
Open your terminal and run the following command to install Flake8 using `pip`:

```
pip install flake8
```

Running Flake8:
Navigate to your project directory using the terminal and run Flake8. It will analyse your Python code and report any PEP8 violations.

```
flake8
```

Integrating with your editor (optional):
To make PEP8 linting more convenient, you can integrate Flake8 with your code editor. Many popular editors have plugins or support for Flake8. Here is an example for setting Flake8 within Visual Studio Code (VS Code).

VS Code:

1. Navigate to the **Flake8 extension within the marketplace**.
2. Select "Install" to install the extension.
    a. If prompted, accept the notification to open VS Code, which will direct you to the Extensions panel.
    b. In the Extensions panel, click on "Install" to complete the installation of the Flake8 extension.
3. Flake8 linting should now be automatically enabled for your Python files.

For more detailed information, refer to the **VS Code documentation**.

Customise Flake8 configuration (optional):
You can create a configuration file (usually named `.flake8`) in your project to customise Flake8 settings. This allows you to ignore certain rules, exclude files or directories, and set other options. Here is a basic example of a `.flake8` file:

```
[flake8]
exclude = .git,__pycache__,venv
max-line-length = 80
```

This example excludes the `.git`, `__pycache__`, and `venv` directories and sets the maximum line length to 80 characters.

By following these steps, you can easily integrate PEP8 linting into your Python project, ensuring that your code follows the recommended style guidelines.

Editor integration:
If you have the Python extension installed within VS Code, it will also integrate some PEP8 checks directly into your code editor.

Autoformatters:
Use autoformatters to automatically format your code according to PEP8. We suggest **Black**, a popular auto-formatter for Python. Install it using:

```
pip install black
```

Run it on your codebase:

```
black your_project_directory
```

Configuration file:
Both **Flake8** and **Black** can be configured using a configuration file. This allows you to customise certain behaviours or exclude specific files or directories from linting or formatting.

Create a **.flake8** file for **Flake8** configuration:

```
[flake8]
ignore = E203, E501, W503
exclude = venv, __pycache__
max-line-length = 80
```

Create a `pyproject.toml` file for **Black** configuration:

```
[tool.black]
line-length = 80
```

Example configuration:

```
[flake8]
max-line-length = 80
extend-ignore = E203
exclude = .git, __pycache__, venv
```

Example `pyproject.toml` configuration for Black:

```
[tool.black]
line-length = 80
```

By integrating these tools and following PEP8 guidelines, you can maintain a consistent and readable codebase. Adjust the configuration according to your project's specific requirements.

GENERATING A REQUIREMENTS FILE

A `requirements.txt` file in a Python 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. To generate a `requirements.txt` file for your Python project, you can use the `pip freeze` command. This command lists all installed packages and their versions. Here's how you can create a `requirements.txt` file:

Activate your virtual environment (if you're using one):

- Windows:
  ```
  myenv\Scripts\activate
  ```
- MacOS/Linux:
  ```
  source myenv/bin/activate
  ```

Run the following command to generate the `requirements.txt` file:

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

This command creates a `requirements.txt` file containing the names and versions of all installed packages. If you haven't installed any additional packages beyond the default ones, your `requirements.txt` file may be empty initially.
Here's an example `requirements.txt` file:

```
Flask==2.0.1
SQLAlchemy==1.4.27
requests==2.26.0
```

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.

In conclusion, this comprehensive guide equips you with the fundamental knowledge to kickstart your Python project development in a structured and efficient manner. By embracing virtual environments, PEP8 linting, and the generation of a `requirements.txt` file, you establish a solid foundation for building 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.