ADDITIONAL READING

Modular Testing and Implementation: Walkthrough Guide

Visit our website

WELCOME TO THE MODULAR TESTING AND IMPLEMENTATION WALKTHROUGH GUIDE!

In this guide, we walk 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 then look into creating a new Django project, explaining the essential steps from installing Django to setting up a new application within your project.

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, managing dependencies independently. Widely used in software development, it ensures consistent and reproducible application behaviour across different computing environments.

Using `venv` (built-in in Python 3.3 and newer) or `python -m venv`:

Open a Terminal or Command Prompt:

- Open a terminal or command prompt on your computer or IDE (VS CODE).
- Navigate to Your Project Directory (File):
- Use the `cd` command to navigate to the directory where you want to create your virtual environment.

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

Create a Virtual Environment:
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
```

If you are using Python 3.3 or newer, you can use:

```
python3 -m venv venv
```

Activate the Virtual Environment:
Activate the virtual environment. On Windows, you might use:

```
Cd venv\Scripts\activate.bat
```

On Unix or MacOS, you might use:

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

After activation, your terminal prompt should change to indicate that you are now in the virtual environment. You should see the following:

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

Install Dependencies:
Now, you can use `pip` to install Python packages such as Django within your virtual environment.

```
pip install django
```

Deactivate the Virtual Environment:

When you're done working in your virtual environment, you can deactivate it.

On Windows, you might use:

```
venv\Scripts\deactivate.bat
```

On Unix or MacOS, you might use:

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

Using `virtualenv` (an external package):
If you prefer using **`virtualenv`**, you can install it globally and then use it to create virtual environments.

Install **`virtualenv`** using **`pip`**. This is a once-off step.

```
pip install virtualenv
```

Navigate to your project directory and create a virtual environment.

```
virtualenv venv
```

Activate the virtual environment and install dependencies as described above.

```
source venv/bin/activate  # On Unix/Mac
venv\Scripts\activate.bat     # On Windows
```

When you're done, deactivate the virtual environment.

```
source venv/bin/deactivate  # On Unix/Mac
venv\Scripts\deactivate.bat     # On Windows
```

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 don't interfere with other projects or the system-wide Python installation.

CREATING A NEW DJANGO PROJECT

Ensure your VENV is active, there will be an extension to your file path in your terminal such as "`(.venv)`" or "`(virtual_environment_name)`". If this is not visible you can select the environment by using ctrl+shift+p, in the dropdown type Python: Select Interpreter and choose your virtual environment from the list.

Next, we will need to install the latest version of Django using "`pip`". Pip is the package installer for Python. It's a tool that helps you install and manage Python packages, which are collections of code and resources that provide specific functionality.

```
pip install django
```

Creating a Django project:

```
 django-admin startproject project_name
```

You will notice a new directory with your `project_name` with a matching folder inside. This folder is your main site of your project that links all your site's functionality together.

```
project_name/ # This is your project directory
    manage.py
    project_name/ - # This is your main site
        __init__.py
        settings.py
        urls.py
        asgi.py
        wsgi.py
```

Please take note of the python file called **manage.py**; this program allows us to interact with the project. We can view our interaction options by running the following command. (**Ensure your are inside the upper project_name folder**)

```
python manage.py
```

With this command, we are specifying the program **(python)** and the program we want to use to run **(manage.py)**.

To create an application for our project we can use the **manage.py**. First we need to cd into our project folder to access the manage.py program.

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

Then by running the following command:

```
python manage.py startapp app_name
```

This will add the following folder and structure to your project.

```
app_name
    __init__.py
    migrations/
        __init__.py
        admin.py
        apps.py
        models.py
        tests.py
        views.py
```

Once your app is built, you need to modify some aspects to create a functioning site.

Navigate to the **setting.py** program **(project_name\project_name\settings.py)**, inside this program, we will scroll to the `INSTALLED_APPS = []` section. We append the list with the name of our application:

```
INSTALLED_APPS = [
    'polls.apps.PollsConfig',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'app_name',
]
```

Run the initial database migrations to set up the database tables:

```
python manage.py migrate
```

Create a Superuser (Optional):

If you want to access the Django Admin interface, you can create a superuser account with the following command:

(Note when creating a super user the password field will remain blank, while you are typing the password. Django is in fact accepting your input.

```
python manage.py createsuperuser
```

Start the Django development server to see your project in action:

```
python manage.py runserver
```

By default, it will run on **http://localhost:8000/**.

Access the Admin Interface (Optional):
If you created a superuser, you can access the admin interface at **http://localhost:8000/admin/** and log in with the superuser credentials.

Now that your project is set up, you can start building your app by defining models, views, and templates within your app directory.

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 are examples for two common editors, Visual Studio Code (VS Code) and Sublime Text.

VS Code:

- Install the "Python" extension by Microsoft.
- Open your project in VS Code.
- Press Ctrl + , to open the settings.
- Search for "Python: Linting: Flake8 Enabled" and make sure it's set to true.

Now, Flake8 will automatically lint your Python code as you work in VS Code.

Sublime Text:

- Install the "SublimeLinter" package.
- Install the "SublimeLinter-flake8" package.
- Open your project in Sublime Text.
- Ensure that the virtual environment with flake8 installed is activated.
- As you edit your Python files, flake8 will automatically lint your code, and any violations will be displayed in the Sublime Text editor.

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:
Integrate PEP8 checking directly into your code editor. Many popular editors have plugins that highlight violations as you write code. For example:

Visual Studio Code: Install the "Python" extension by Microsoft, which includes PEP8 linting.

Sublime Text: Install the "SublimeLinter" and "SublimeLinter-flake8" packages.

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 Django 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):

```
source venv/bin/activate  # On Unix/Mac
venv\Scripts\activate     # On Windows
```

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.

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.

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 (-e option) 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.

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, Django project setup, PEP8 linting, and the generation of a `requirements.txt` file, you 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 really get stuck, you can contact your mentor for help.