Python Project Starter Kit

Get started with Python projects the right way

Michael Borck

Welcome to the Python Project Starter Kit!

As a Python beginner, starting a new project can be overwhelming. With so many possibilities and options, it's easy to get lost in the sea of code and best practices. That's why we've created this Python Project Starter Kit - to help you get started with your Python project the right way.

This cheat sheet is designed to provide you with a comprehensive guide to creating a Python project, from setting up your project structure to writing clean and efficient code. Whether you're a beginner or an experienced programmer, this kit will help you create a solid foundation for your project and ensure that you're following best practices.

Why is this cheat sheet useful?

- **Get started quickly**: With this cheat sheet, you'll be able to set up your project and start coding in no time.
- **Follow best practices**: By following the guidelines outlined in this cheat sheet, you'll ensure that your project is well-structured and easy to maintain.
- **Improve your coding skills**: By following best practices, you'll improve your coding skills and write more efficient and readable code.
- **Collaborate with others**: With a well-structured project, you'll be able to collaborate with others more easily and effectively.

What's included in this cheat sheet?

- **Project structure**: Learn how to set up your project directory and organize your files and folders.
- **Code organization**: Discover how to write clean and efficient code, and how to structure your code for readability and maintainability.

- **Best practices**: Get tips and tricks for writing Python code, from variable naming to commenting your code.
- **Testing and debugging**: Learn how to test and debug your code, and how to use tools like pdb and unittest.

By following this cheat sheet, you'll be well on your way to creating a successful Python project. Happy coding!

Project Structure

1. **Create a new directory** for your project and navigate into it.
2. **Create a `src` or `scripts` directory** for your Python scripts.
3. **Create a `notebooks` directory** for your Jupyter Notebooks.
4. **Create a `requirements.txt` file** to manage dependencies.
5. **Create a `README.md` file** to describe your project.

Notebook Best Practices

1. **Use meaningful notebook names**.
2. **Organize notebooks into sections** (e.g., introduction, methods, results).
3. **Use clear and concise headings**.
4. **Use Markdown** for formatting.
5. **Keep notebooks concise** and focused on a single idea.

Script Best Practices

1. **Use meaningful script names**.
2. **Keep scripts short and focused** (e.g., < 50 lines).
3. **Use clear and concise variable names**.
4. **Use comments to explain code**.
5. **Test scripts regularly** using print statements or a testing framework.

Code Style

1. **Use a consistent coding style** (e.g., PEP 8).
2. **Use meaningful variable names**.
3. **Use type hints** for function arguments and return types.
4. **Use consistent indentation** (4 spaces).
5. **Avoid magic numbers**; use named constants instead.

Testing

1. **Test scripts regularly** using print statements or a testing framework.
2. **Use a testing framework** (e.g., unittest) for more complex scripts.
3. **Test for edge cases** and **error handling**.

Best Practices

1. **Use version control** (e.g., Git) to track changes.
2. **Use a code editor** with syntax highlighting.
3. **Use a linter** (e.g., flake8) to catch errors.
4. **Keep your code organized** and **maintainable**.
5. **Collaborate with others** and **ask for help** when needed.