Foreword

Read this before you get started with Flask. This hopefully answers some questions about the purpose and goals of the project, and when you should or should not be using it.

What does “micro” mean?

“Micro” does not mean that your whole web application has to fit into a single Python file (although it certainly can), nor does it mean that Flask is lacking in functionality. The “micro” in microframework means Flask aims to keep the core simple but extensible. Flask won’t make many decisions for you, such as what database to use. Those decisions that it does make, such as what templating engine to use, are easy to change. Everything else is up to you, so that Flask can be everything you need and nothing you don’t.

By default, Flask does not include a database abstraction layer, form validation or anything else where different libraries already exist that can handle that. Instead, Flask supports extensions to add such functionality to your application as if it was implemented in Flask itself. Numerous extensions provide database integration, form validation, upload handling, various open authentication technologies, and more. Flask may be “micro”, but it’s ready for production use on a variety of needs.

Configuration and Conventions

Flask has many configuration values, with sensible defaults, and a few conventions when getting started. By convention, templates and static files are stored in subdirectories within the application’s Python source tree, with the names `templates` and `static` respectively. While this can be changed, you usually don’t have to, especially when getting started.

Growing with Flask

Once you have Flask up and running, you’ll find a variety of extensions available in the community to integrate your project for production. The Flask core team reviews extensions and ensures approved extensions do not break with future releases.

As your codebase grows, you are free to make the design decisions appropriate for your project. Flask will continue to provide a very simple glue layer to the best that Python has to offer. You can implement advanced patterns in SQLAlchemy or another database tool, introduce non-relational data persistence as appropriate, and take advantage of f
work-agnostic tools built for WSGI, the Python web interface.

Flask includes many hooks to customize its behavior. Should you need more customization, the Flask class is built for subclassing. If you are interested in that, check out the Becoming Big chapter. If you are curious about the Flask design principles, head over to the section about Design Decisions in Flask.

Continue to Installation, the Quickstart, or the Foreword for Experienced Programmers.