**Chapter 01**

We are going to extend the most basic Flask proect we have created earlier in this chapter. Thouhg, it does not yet interact with any kind of DB and it is more or less similar to serve some static HTML files through any web server but, that said, going through this will help you understand the basics of interacting between Static Files and Dynamic Python code.

The pre-requisites:

* Python 2.x (this particular tutorial code is developed using 2.7.3)
* pip and virtualenv installed. (<http://www.pip-installer.org/en/latest/installing.html>, <http://docs.python-guide.org/en/latest/dev/virtualenvs/>)
* Having access to a note-pad or similar text editor to edit/create files.(recommended Sublime Text2)
* Git versioning system. (<http://git-scm.com/book/en/Getting-Started-Installing-Git>)

Setup the environment:

Actually, if you have followed the steps in chapter 00, then nothing much left to be told here. Just instead of changing your directory with “cd” to chapter\_00\_hello\_world, you have to change to chapter\_01\_basic

Start Exploring the project:

You will see a few new directories has been created in this chapter. In the folowing section we give an explaination of what they stand for.

The Directory Structure:

* static This directory contains the subdirectories and files for static web assets (in a production system you may want to server this from a different server or a cdn)
* templates This directory contains the html template files (Jinja 2 templates). These files serve as the view for each action we take on the app (loading a page, navigating to a url etc.)

Explaining app.py:

Open up the file in a text editor and let us travel from the begning of the file.

The first few lines of the file looks similar to what we have seen in chapter 00, except for a few new imports. All those new imports are there so that we can take the help of the “out-of-the-box” support of Flask for several utilities. For an example reading, compiling and rendering templates from the templates directory.

If you have a look into the same old def index() method now you will see instead of returning a simple string there we are actually calling a template file using render\_template method we have imported earlier. This is Flask's default way to attach a template (a “sort-of” HTML file) with a particular route. So that when that route is called this file will be rendered.

There is another interesting method called page\_not\_found. This is one method which has @app.errorhandler(404) decorator instead of the normal @app.route decorator. This particular decorator here tells Flask what to do when it hits a 404 (Page Not Found, which in our terms means “We have not defined this route yet”) error. (For an in depth understanding of what the codes are and what they mean please visit - <http://www.w3.org/Protocols/HTTP/HTRESP.html>)

Explaining base.html:

This is a base template for the whole application, which means all the common parts (header, footer, including javascripts and CSS) all go here. This is Jinja2 Template (Jinja2 is the default template engine for Flask). If you want to know more about Jinja2 please have a look at - <http://jinja.pocoo.org/docs/> . We have created this Jinja2 template in a bootstrap way. Bootstrap is the awesome CSS framework from twitter. If you want to learn more please have a look at - <http://getbootstrap.com/>

Explaining index.html:

This is the template associated with our root/index route (that is when you hit, [http://localhost:5000](http://localhost:5000/)) if you see this file then you will see that it starts with {% extends "base.html" %} This is Jinja's way of inheriting templates. This line here means that our index.html is a “child” of base.html and in base.html there is a space where Jinja will insert the contents of index.html (see the lines in base.html - {% block content %} {% endblock %} and also please notice that all the contents of our index.html is actually written between the same “block” definitions as well.). To know more about Jinja please visit the link already provided.

Conclusion:

When you are done with the code study and you can run the application (python app.py) you should visit [http://localhost:5000](http://localhost:5000/) to see the result. Try the links there. Also try visiting some not implemented page (e.g. <http://localhost:5000/myname>)