Flask

web development,
one drop at a time

Micro Web Framework

by

t-aravinda

Introduction

Flask is a **web development framework** developed in Python.

Flask provides a **development server** and a **debugger**.

It uses **Jinja2** templates.

It is working with **WSGI 1.0**.

It provides integrated support for **unit testing**.

Many extensions are available for Flask, which can be used to enhance its functionalities.

What does “micro” mean?

The “micro” in micro framework 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.

Why is Flask called a micro-framework?

Flask is known as a *micro-framework* because it is *lightweight* and *only* provides components that are *essential*.

It only provides the necessary components for web development, such as **routing, request handling, sessions,** and so on.

For the other functionalities such as *data handling*, the developer can write a custom module or use an *extension.*

Install Flask

```
$ pip install Flask
```

Install virtualenv

If you are using Python 2, the venv module is not available. You need to install.

On Linux, virtualenv is provided by your package manager:

```
# Ubuntu

$ sudo apt-get install python-virtualenv

# CentOS, Fedora

$ sudo yum install python-virtualenv
```

On Windows, as an administrator:

```
> \Python27\python.exe Downloads\get-pip.py

> \Python27\python.exe -m pip install virtualenv
```

Hello world Example

1) Create a folder and name it, You can give any name.

2) In created folder create a module (a python file) and name it, you can give any name. Ex : Demo1.py

3) Write the Program into the file as

```
from flask import Flask
app  =  Flask(_name_)
@app.route('/')
def wish():
   return "Hello Flask"
if__name__== "__main__":
   app.run()
```

4) save the module and run the module using command prompt.

5) Open the Command prompt and type the command as

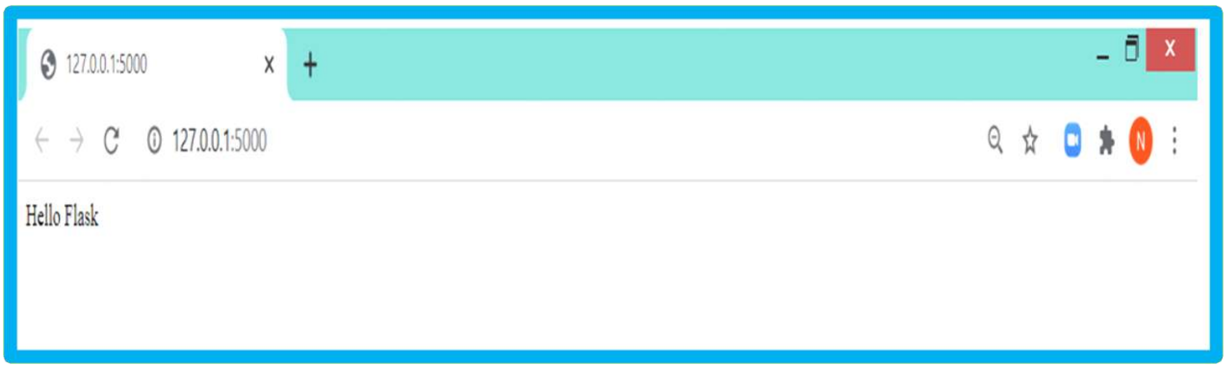
```
python   module.py
```

Example :   python Demo1.py

```
Serving Flask app "Demo1" (lazy loading)

* Environment: production

WARNING: Do not use the development server in a production environment.

 Use a production WSGI server instead.

 * Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

6) Open the browser and type the URL as **http://127.0.0.1:5000**

Output



Hello Flask

Program Explanation

# Importing flask module

from flask import Flask

# An object of Flask class is our WSGI application.

# Flask constructor takes the name of current module (__name__) as argument.

app = Flask(__name__)

# The route() function of the Flask class is a decorator which tells the application which URL should call the associated function.

@app.route('/')

# '/' URL is bound with wish function.

```
def wish():
    return 'Hello World'
```

# main driver function

```
if__name__== '__main__':
   # run() method of Flask class runs the application on the local development server.
   app.run()
```

Example to render the Html Text

```
from flask import Flask
app = Flask(__name__)
@app.route('/')
def show_html():
   html_text = '''<html>
   <head>
      <title>Python With Aravinda</title>
   </head>
   <body>
      <h1>Welcome to Flask With Aravinda</h1>
   </body>
</html>'''
   return html_text
if__name__== "__main__":
   app.run()
```

Output

Routing

Use the **route()** decorator to bind a function to a URL.

```
@app.route('/')
def index():
   return 'Index Page'
@app.route('/hello')
def hello():
   return 'Hello, World'
```

Example to render Multiple web pages

```
from flask import Flask
app = Flask(__name__)
@app.route('/one')
def page_one():
   html_text = '''<html>
      <head>
         <title>Python With Aravinda</title>
      </head>
      <body>
         <h1>This is Page 1</h1>
      </body>
   </html>'''
```

```
    return html_text


@app.route('/two')
def page_two():
    html_text = '''<html>
        <head>
            <title>Python --- Aravinda</title>
        </head>
        <body>
            <h1>This is Page 2</h1>
        </body>
    </html>'''
    return html_text

if__name__== "__main__":
    app.run()
```

Output

Variable Rules

You can add variable sections to a URL by marking sections with <variable_name>.

Your function then receives the <variable_name> as a keyword argument.

Optionally, you can use a converter to specify the type of the argument like <converter:variable_name>.

```
from markupsafe import escape


@app.route('/user/<username>')

def show_user_profile(username):

    # show the user profile for that user

    return 'User %s' % escape(username)


@app.route('/post/<int:post_id>')

def show_post(post_id):

    # show the post with the given id, the id is an integer

    return 'Post %d' % post_id
```

```
@app.route('/path/<path:subpath>')
def show_subpath(subpath):
    # show the subpath after /path/
    return 'Subpath %s' % escape(subpath)
```

Converter types:

string	(default) accepts any text without a slash
int	accepts positive integers
float	accepts positive floating point values
path	like string but also accepts slashes
uuid	accepts UUID strings **(Universally unique identifier)**

Example on variable in URL

```
from flask import Flask
app = Flask(__name__)
@app.route('/<name>')
def wish(name):
    html_text = '''<html>
            <head>
                <title>Python With Aravinda</title>
            </head>
            <body>
                <h1>Welcome Mr/Miss : '''+name+'''</h1>
            </body>
```

```
        </html>'''
    return html_text
if __name__ == "__main__":
    app.run()
```

Output

Example on variable with type conversion in URL

```
from flask import Flask
app = Flask(__name__)
@app.route('/<name>/<int:age>')
def display_age(name,age):
    if age>=23:
        message = "Welcome Mr/Miss : "+name+" Your age is valid"
    else:
        message = "Welcome Mr/Miss : " + name + " Your age is
Invalid"
    html_text = '''<html>
            <head>
              <title>Python With Aravinda</title>
            </head>
            <body>
               <h1>''' + message + '''</h1>
            </body>
        </html>'''
    return html_text
```

```
if__name__== "__main__":
    app.run(debug=True)
```

Output

Unique URLs / Redirection Behaviour

The following two rules differ in their use of a trailing slash.

```
@app.route('/projects/')
def projects():
   return 'The project page'
@app.route('/about')
def about():
   return 'The about page'
```

The canonical URL for the projects endpoint has a trailing slash. It’s similar to a folder in a file system. If you access the URL without a trailing slash, Flask redirects you to the canonical URL with the trailing slash.

The canonical URL for the about endpoint does not have a trailing slash. It’s similar to the pathname of a file. Accessing the URL with a trailing slash produces a 404 “Not Found” error. This helps keep URLs unique for these resources, which helps search engines avoid indexing the same page twice.

Rendering Templates

The Flask configures the **Jinja2** template engine automatically.

Jinja2 : https://jinja.palletsprojects.com/en/2.11.x/

To render a template you can use the **render_template()** method.

All you have to do is provide the name of the template and the variables you want to pass to the template engine as keyword arguments.

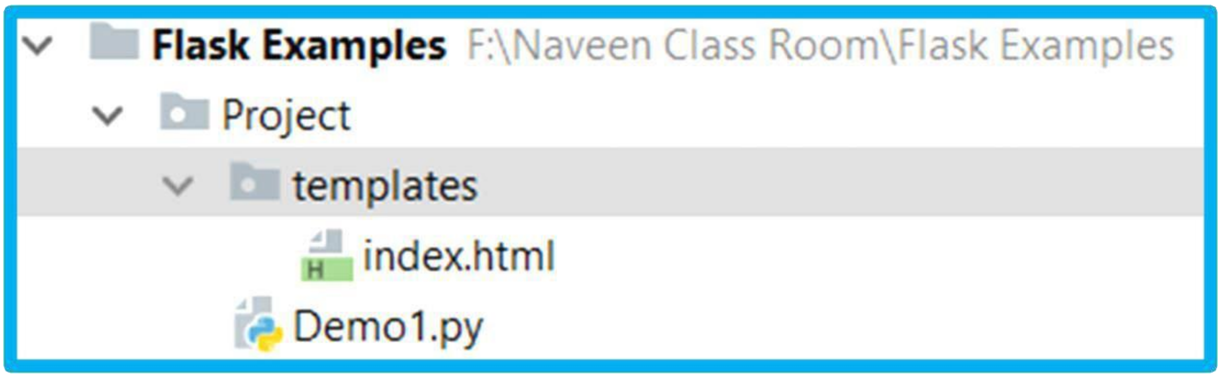
Here’s a simple example of how to render a template:

```
from flask import render_template

@app.route('/hello/')

@app.route('/hello/<name>')

def hello(name=None):
    return render_template('hello.html', name=name)
```

Flask will look for templates in the templates folder.

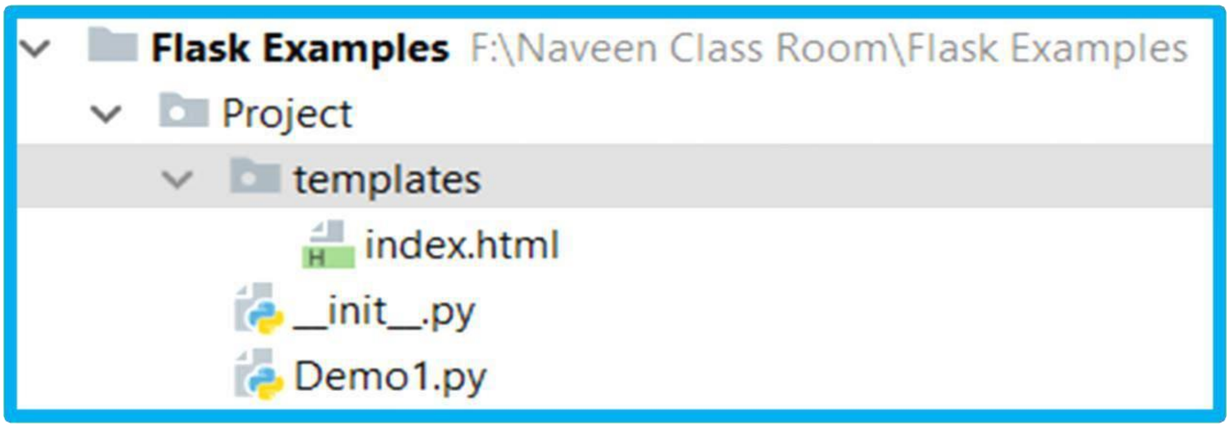
So if your application is a module, this folder is next to that module, if it's a package it's actually inside your package:

Case 1: a module:



```
Flask Examples F:\Naveen Class Room\Flask Examples
  Project
    templates
      index.html
    Demo1.py
```

Case 2: a package:



```
Flask Examples F:\Naveen Class Room\Flask Examples
  Project
    templates
      index.html
    __init__.py
    Demo1.py
```

Template Designer

Jinga 2 template engine provides some delimiters which can be used in the HTML to make it capable of dynamic data representation.

The template system provides some HTML syntax which are placeholders for variables and expressions that are replaced by their actual values when the template is rendered.

The jinga2 template engine provides the following delimiters to escape from the HTML.

- {% ... %} for statements
- {{ ... }} for expressions to print to the template output
- {# ... #} for the comments that are not included in the template output
- # ... ## for line statements

Example

```
from flask import Flask
from flask import render_template
```

```
app = Flask(__name__)

@app.route('/<name>')
def showIndex(name):
   return render_template("index.html",name=name)

if __name__== "__main__":
   app.run()
```

templates

```
<html>
   <head>
      <title>Python With aravinda</title>
   </head>
   <body>
   <style>
       h1{
            font-family: "Agency FB";
            font-size: 40px;
            background-color: red;
            color: yellow;
            text-align: center;
         }
   </style>
      <h1>Welcome Mr/Miss : {{ name }}</h1>
   </body>
</html>
```

Output

Static Files

Just create a folder called **static** in your package or next to your module and it will be available at **/static** on the application.

To generate URLs for static files, use the special 'static' endpoint name:

url_for**(**'static'**,** filename='style.css'**)**

The file has to be stored on the filesystem as static/style.css.

Example

```
from flask import render_template
from flask import Flask

app = Flask(__name__)
```

```
@app.route('/open/<name>')
def showtemplate(name):
    return render_template('index.html',name=name)

if__name___== "__main__":
    app.run(debug=True)
```

template

```
<head>
    <link rel="stylesheet" href="{{
url_for('static',filename='example.css')}}">
</head>
<body>
    <h1>Welcome to Templates {{ name }}</h1>
    <a href="http://www.facebook.com">Open Face Book</a>
</body>
</html>
```

example.css

```
h1{
    text-align: center;
    font-size: 60px;
    background-color: blue;
    color: yellow;
    font-family: "Agency FB";
}
```

Pending Notes will be Updated in short

Thank you

Python With Aravinda