

UNIT 4 : FLASK FRAMEWORK

WHAT IS FLASK?

1. Flask is a web framework that allows developers to build lightweight web applications quickly and easily with Flask Libraries.
2. It is based on WSGI toolkit and jinja2 template engine.
3. Flask is considered as a micro framework.
4. Flask's framework is more explicit than Django's framework and is also easier to learn because it has less base code to implement a simple web-Application.
5. A Web-Application Framework or Web Framework is the collection of modules and libraries that helps the developer to write applications without writing the low-level codes such as protocols, thread management, etc.

What is WSGI?

It is an acronym for web server gateway interface which is a standard for python web application development. It is considered as the specification for the universal interface between the web server and web application.

What is Jinja2?

Jinja2 is a web template engine which combines a template with a certain data source to render the dynamic web pages.

Advantages of Flask:

1. Flask is a **lightweight** backend framework with minimal dependencies.
2. Flask is **easy to learn** because its simple and intuitive API makes it easy to learn and use for beginners.
3. Flask is a **flexible Framework** because it allows you to customize and extend the framework to suit your needs easily.
4. Flask can be used with **any database** like:- SQL and NoSQL and with **any Frontend Technology** such as React or Angular.
5. Flask is **great for small to medium projects** that do not require the complexity of a large framework.

Flask Environment Setup

To install flask on the system, we need to have python 2.7 or higher installed on our system. However, we suggest using python 3 for the development in the flask.

Install virtual environment (virtualenv)

virtualenv is considered as the virtual python environment builder which is used to create the multiple python virtual environment side by side. It can be installed by using the following command.

1. \$ pip install virtualenv

Once it is installed, we can create the new virtual environment into a folder as given below.

1. \$ mkdir new
2. \$ cd new
3. \$ virtualenv venv

To activate the corresponding environment, use the following command on the Linux operating system.

1. \$ venv/bin/activate

We can now install the flask by using the following command.

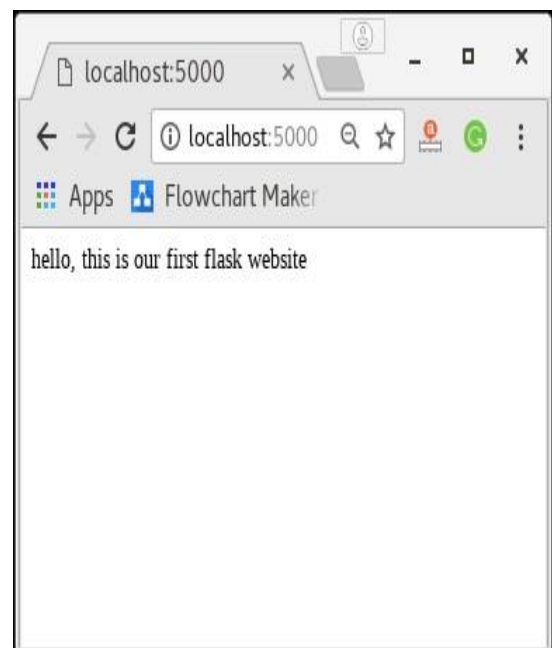
1. \$ pip install flask

NOTE : we can install the flask using the above command without creating the virtual environment.

WEB DEVELOPMENT USING FLASK

1. **from** flask **import** Flask
- 2.
3. app = Flask(__name__)
4. **#creating the Flask class object**
- 5.
6. @app.route('/') **#decorator drfines the**
7. **def** home():
8. **return** "hello, this is our first flask website";
- 9.
10. **if** __name__ == '__main__':
11. app.run(debug = True)

OUTPUT

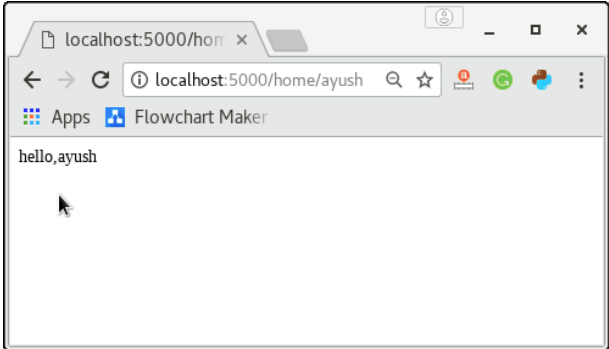
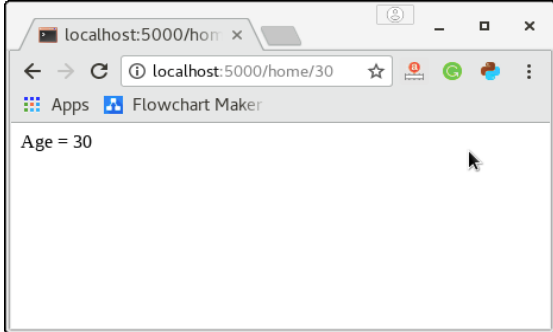


UNDERSTANDING THE CODE

1. To build the python web application, we need to import the Flask module. (CODE LINE 1)
 2. An object of the Flask class is considered as the WSGI application.
 3. We need to pass the name of the current module, i.e. `__name__` as the argument into the Flask constructor. (CODE LINE 3)
 4. The `route()` function of the Flask class defines the URL mapping of the associated function. (CODE LINE 6)
 5. App routing is the technique used to map the specific URL with the associated function intended to perform some task.
 6. The Latest Web frameworks use the routing technique to help users remember application URLs.
 7. It is helpful to access the desired page directly without navigating from the home page.
 8. Route function has two arguments rule and option [**`app.route(rule, options)`**]
 9. rule: It represents the URL binding with the function.
 - the / URL is bound to the main function which is responsible for returning the server response.
 - It can return a string to be printed on the browser's window or we can use the HTML template to return the HTML file as a response from the server.
 10. options: It represents the list of parameters to be associated with the rule object
 11. Then the run method of the Flask class is used to run the flask application on the local development server. [**`app.run(host, port, debug, options)`**]
- | Option | Description |
|---------|---|
| host | The default hostname is 127.0.0.1, i.e. localhost. |
| port | The port number to which the server is listening to. The default port number is 5000. |
| debug | The default is false. It provides debug information if it is set to true. |
| options | It contains the information to be forwarded to the server. |
12. The if `__name__ == '__main__':` makes sure the server only runs if the script is executed directly from the Python interpreter and not used as an imported module.

@app.route function

1. the URL ('/') is associated with the home function that returns a particular string displayed on the web page.
2. if we visit the particular URL mapped to some particular function, the output of that function is rendered on the browser's screen.

<ol style="list-style-type: none"> 1. from flask import Flask 2. app = Flask(__name__) 3. 4. @app.route('/home') 5. def home(): 6. return "hello, welcome to our website"; 7. 8. if __name__ == "__main__": 9. app.run(debug = True) 	
<ol style="list-style-type: none"> 1. from flask import Flask 2. app = Flask(__name__) 3. 4. @app.route('/home/<name>') 5. def home(name): 6. return "hello,"+name; 7. 8. if __name__ == "__main__": 9. app.run(debug = True) 	<ul style="list-style-type: none"> • Flask facilitates us to add the variable part to the URL by using the section. • We can reuse the variable by adding that as a parameter into the view function.  <p>A screenshot of a web browser window. The address bar shows 'localhost:5000/home/ayush'. The page content displays 'hello, ayush'.</p>
<ol style="list-style-type: none"> 1. from flask import Flask 2. app = Flask(__name__) 3. 4. @app.route('/home/<int:age>') 5. def home(age): 6. return "Age = %d"%age; 7. 8. if __name__ == "__main__": 9. app.run(debug = True) 	<ul style="list-style-type: none"> • The converter can also be used in the URL to map the specified variable to the particular data type. • we can provide the integers or float like age or salary respectively.  <p>A screenshot of a web browser window. The address bar shows 'localhost:5000/home/30'. The page content displays 'Age = 30'.</p>

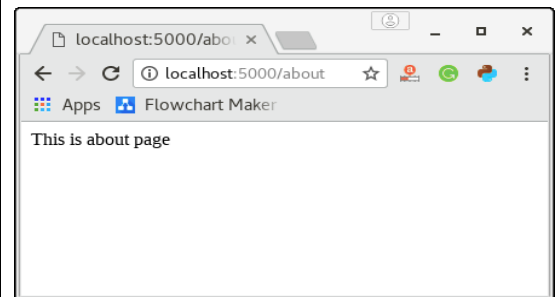
- Following are the converters that can be used to convert the default string type to the associated data type.
 - string: default
 - int: used to convert the string to the integer
 - float: used to convert the string to the float.
 - path: It can accept the slashes given in the URL.

add_url_rule() function

- This is one more approach to perform routing for the flask web application with the help of `add_url_rule()` function of the Flask class.

`add_url_rule(<url rule>, <endpoint>, <view function>)`
- This approach is used in case we are importing the view function from another module
- This is mainly used in the case if the view function is not given and we need to connect a view function to an endpoint externally by using this function.
- the app. route calls this function internally.

```
1. from flask import Flask
2. app = Flask(__name__)
3. def about():
4.     return "This is about page";
5. app.add_url_rule("/about", "about", about)
6. if __name__ == "__main__":
7.     app.run(debug = True)
```



url_for() function

- The `url_for()` function is used to build a URL to the specific function dynamically.
- The first argument is the name of the specified function, and then we can pass any number of keyword argument corresponding to the variable part of the URL.
- This function is useful in the sense that we can avoid hard-coding the URLs into the templates by dynamically building them using this function.
- is used to prepare a URL, for a function dynamically, such that, changing URLs, in the application, is avoided.
- It accepts, the name of the view function, as the first argument, and, any number of keywords, to be sent(to the view function), as the second argument.

<pre>1. from flask import * 2. 3. app = Flask(__name__) 4. 5. @app.route('/admin') 6. def admin(): 7. return 'admin' 8. 9. @app.route('/librarian') 10. def librarian(): 11. return 'librarian' 12. 13. @app.route('/student') 14. def student(): 15. return 'student'</pre>	<pre>16. @app.route('/user/<name>') 17. def user(name): 18. if name == 'admin': 19. return redirect(url_for('admin')) 20. if name == 'librarian': 21. return redirect(url_for('librarian')) 22. if name == 'student': 23. return redirect(url_for('student')) 24. if __name__ == '__main__': 25. app.run(debug = True)</pre>
--	--

- The above script simulates the library management system which can be used by the three types of users, i.e., admin, librarian, and student.
- There is a specific function named user() which recognizes the user the redirect the user to the exact function which contains the implementation for this particular function.

Benefits of the Dynamic URL Building

1. It avoids hard coding of the URLs.
2. We can change the URLs dynamically instead of remembering the manually changed hard-coded URLs.
3. URL building handles the escaping of special characters and Unicode data transparently.
4. The generated paths are always absolute, avoiding unexpected behavior of relative paths in browsers.
5. If your application is placed outside the URL root, for example, in /myapplication instead of /, url_for() properly handles that for you.