from flask import Flask

#set up our application

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

app = Flask(\_\_name\_\_) #referencing this file (#creating the Flask class object )

#We need to pass the name of the current module, i.e. \_\_name\_\_ as the argument into the Flask constructor.

#creating index route so that when we browse URL,we don't immediately get 404

@app.route('/') #App routing is used to map specific URL with associated function that is intended to perform some task.

'''The route() function of the Flask class defines the URL mapping of the associated function. The syntax is given below.

app.route(rule, options)

It accepts the following parameters.

rule: It represents the URL binding with the function.

options: It represents the list of parameters to be associated with the rule object

As we can see here, 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.'''

def index():

return "Hello world"

'''If the python interpreter is running that module (the source file) as the main program and if that module is

the entry point to your program., it sets the special \_\_name\_\_ variable to have a value “\_\_main\_\_”.

If this file is being imported from another module, \_\_name\_\_ will be set to the module’s name.'''

'''Finally, the run method of the Flask class is used to run the flask application on the local development server.

The syntax is given below.

app.run(host, port, debug, options)

SN Option Description

1 host The default hostname is 127.0.0.1, i.e. localhost.

2 port The port number to which the server is listening to. The default port number is 5000.

3 debug The default is false. It provides debug information if it is set to true.

4 options It contains the information to be forwarded to the server.

'''

if \_\_name\_\_ == "\_\_main\_\_":

app.run(debug=True)

Flask facilitates us to render the external HTML file instead of hardcoding the HTML in the view

function. Here, we can take advantage of the jinja2 template engine on which the flask is based.

Flask provides us the render\_template() function which can be used to render the external HTML

file to be returned as the response from the view function.

ex:

def message():

return render\_template('message.html')

Here, we must create the folder templates inside the application directory and save the

HTML templates referenced in the flask script in that directory.

In our case, the path of our script file script.py is E:\flask whereas the path of the

HTML template is E:\flask\templates.

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

Due to the fact that HTML is a mark-up language and purely used for the designing purpose,

sometimes, in the web applications, we may need to execute the statements for the general-purpose

computations. For this purpose, Flask facilitates us the delimiter {%...%} which can be used to

embed the simple python statements into the HTML.

Example

In the following example, we will print the table of a number specified in the URL, i.e., the URL http://localhost:5000/table/10 will print the table of 10 on the browser's window.

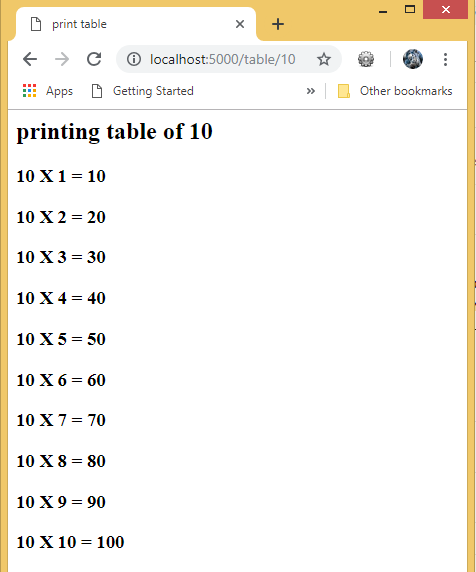
Here, we must notice that the for-loop statement is enclosed inside {%...%} delimiter, whereas, the loop variable and the number is enclosed inside {{ ... }} placeholders.

**script.py**

1. **from** flask **import** \*
2. app = Flask(\_\_name\_\_)
4. @app.route('/table/<int:num>')
5. **def** table(num):
6. **return** render\_template('print-table.html',n=num)
7. **if** \_\_name\_\_ == '\_\_main\_\_':
8. app.run(debug = True)

**print-table.py**

1. <html>
2. <head>
3. <title>**print** table</title>
4. </head>
5. <body>
6. <h2> printing table of {{n}}</h2>
7. {% **for** i  **in** range(1,11): %}
8. <h3>{{n}} X {{i}} = {{n \* i}} </h3>
9. {% endfor %}
10. </body>
11. </html>



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.

app.config('SQLALCHEMY\_DATABASE\_URI') #tells our app where database is located

Flask SQLAlchemy is an ORM tool which establishes the relationship between the objects and the tables of the relational databases.

The mapping between the both is important because the python is capable of storing the data in the form of objects whereas the database stores the data in the form of relational tables, i.e. the collection of rows and columns.

The object-relational mapping is the technique of storing python objects into the database tables without writing the raw SQL queries.

app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///test.db' #telling our app where database is located,4 slashes is absolute path

# 3 slashes is relative path,here we just want to reside in project location, we don't want exact location

db = SQLAlchemy(app) #database is initialised from our app

id = db.Column(db.Integer, primary\_key=True) #references id of each entry

    content = db.Column(db.String(200), nullable=False) #text column,holds each task,nullable=False means we don't want it to be left blank

    date\_created = db.Column(db.DateTime, default=datetime.utcnow)  #gets set automatically to the time we created the item

def \_\_repr\_\_(self):   #func to return a string everytime we create a new element

        return '<Task %r>' % self.id  #return task and id of the created task

@app.route('/', methods=['POST', 'GET']) #adding 2 methods which route can accept,instead of GET(which is default),we can also POST to this route(i.e. send data to our database)

def index():

    if request.method == 'POST':  #if request set to the route is 'POST',put it to database

        #return 'Hello'

        task\_content = request.form['content'] #input text i.e. name='content' given by user in the form

        new\_task = Todo(content=task\_content)  #object for Todo,it's content is = content of input, creating new task for that input

        #push to db

        try:

            db.session.add(new\_task)

            db.session.commit()

            return redirect('/')  #redirect back to index page

        except:

            return 'There was an issue adding your task'

    else:

        tasks = Todo.query.order\_by(Todo.date\_created.all) #look at all the database contents in the order of date they were created and return all of them.this is how we display all current tasks in the table

        return render\_template('index.html',tasks=tasks)  #just show the page