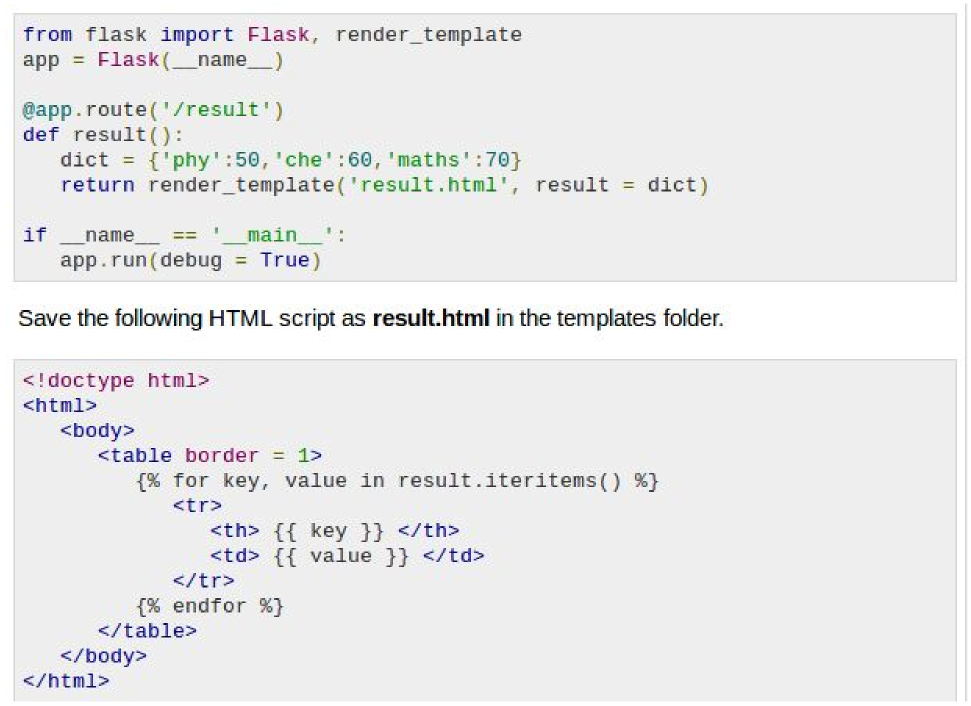
- Flask is a micro framework that uses WSGI (Web server gateway interface) that implements the requests and builds responses and allows to create a web application on top of it
- Werkzeug is used as one of the bases and Jinga is used as web templating engines
- virtualenv is a virtual Python environment builder. It helps a user to create multiple Python environments side-by-side. Thereby, it can avoid compatibility issues between the different versions of the libraries.
- mkdir newproj
  cd newproj
  virtualenv venv - Creating a new virtual environment
- source venv/bin/activate - This command is used to activate the environment
- app.route(rule, options) - route function is used to specify the URL. options are extra parameters that are specified
- app.run(host, port, debug, options) - debug is set to False by default and if set to true then it will return debug information. Options is the information that is sent to the Werkzeug server
- When debug is true then when any changes are made then no need to refresh the server again.
- Parameters to the url can also be specified via <> which contains the argument and this argument must then be passed to the function that requires it. Here we can also specify integers, floats and also path (which accepts slash as the directory separator). Ex : <int: ID>
- We can redirect the url to some other link based on the argument passed using redirect and url_for packages
- In case of POST, parameters are collected from the form data and in case of GET, parameters are collected from 'args' which is a collection of all the parameters
- User = request.args.get('nm') - This is done in case of GET methods
- In web templating system, the HTML script is written in such a way that the text is inserted dynamically in the HTML page by python script that is written. Jinga2 template engine is used in this case
- When html pages are rendered through render_template() function then put all those html files inside templates folder in the same level as that of python file
- Jinga2 templates have following syntaxes for few statements:
    1. {% … %} for statements
    2. {{ ... }} for template outputs
    3. {# … #} for comments that is not included template output
    4. #...## for line statements
- Always end the if statement with endif. This has to be followed for loops too



```python
from flask import Flask, render_template
app = Flask(__name__)

@app.route('/result')
def result():
   dict = {'phy':50,'che':60,'maths':70}
   return render_template('result.html', result = dict)

if __name__ == '__main__':
   app.run(debug = True)
```

Save the following HTML script as **result.html** in the templates folder.

```html
<!doctype html>
<html>
   <body>
      <table border = 1>
         {% for key, value in result.iteritems() %}
            <tr>
               <th> {{ key }} </th>
               <td> {{ value }} </td>
            </tr>
         {% endfor %}
      </table>
   </body>
</html>
```

- This is an example for displaying dictionary in the form of table
- All the javascript files and CSS files have to be put inside static folder
- ******static files not comming!!*******
- Request object :
    1. Form - dictionary object that contains form parameters in the form of key, value pairs
    2. Cookies - dictionary object containing cookie names and values
    3. Args - dictionary object containing query parameters after ?
    4. Method - current request method
    5. Files - data pertaining to uploaded file
- request.form gives all the form data that was passed by the client. It contains key value pairs where key attribute contains the content of “name” attribute and value contains the content that is entered by the user
- Request object contains the cookie attribute. It contains the name of the cookie and also it’s expiry date, path and also the domain name of the website
- make_response() is used to obtain the response object that is returned from the template that is rendered.
- set_cookie() of the response object is the used to create a new cookie
- response.cookies.get() is used to get the list of all the cookies that are set
- Session is the time interval in which the client logs in and logs out. So all the data that is required in a particular session, we need to store that data in the server

- This is an example for uploading file in flask:

```python
from flask import Flask, render_template, request
from werkzeug import secure_filename
app = Flask(__name__)

@app.route('/upload')
def upload_file():
   return render_template('upload.html')

@app.route('/uploader', methods = ['GET', 'POST'])
def upload_file():
   if request.method == 'POST':
      f = request.files['file']
      f.save(secure_filename(f.filename))
      return 'file uploaded successfully'

if __name__ == '__main__':
   app.run(debug = True)
```

- Secret key has to be first created before a session is created. Every session has a specific session id. To create session data we need to do : session[“name”].
- Here session data is created with the key “name”. In order to remove session data from the session created we need to do use pop() function.