

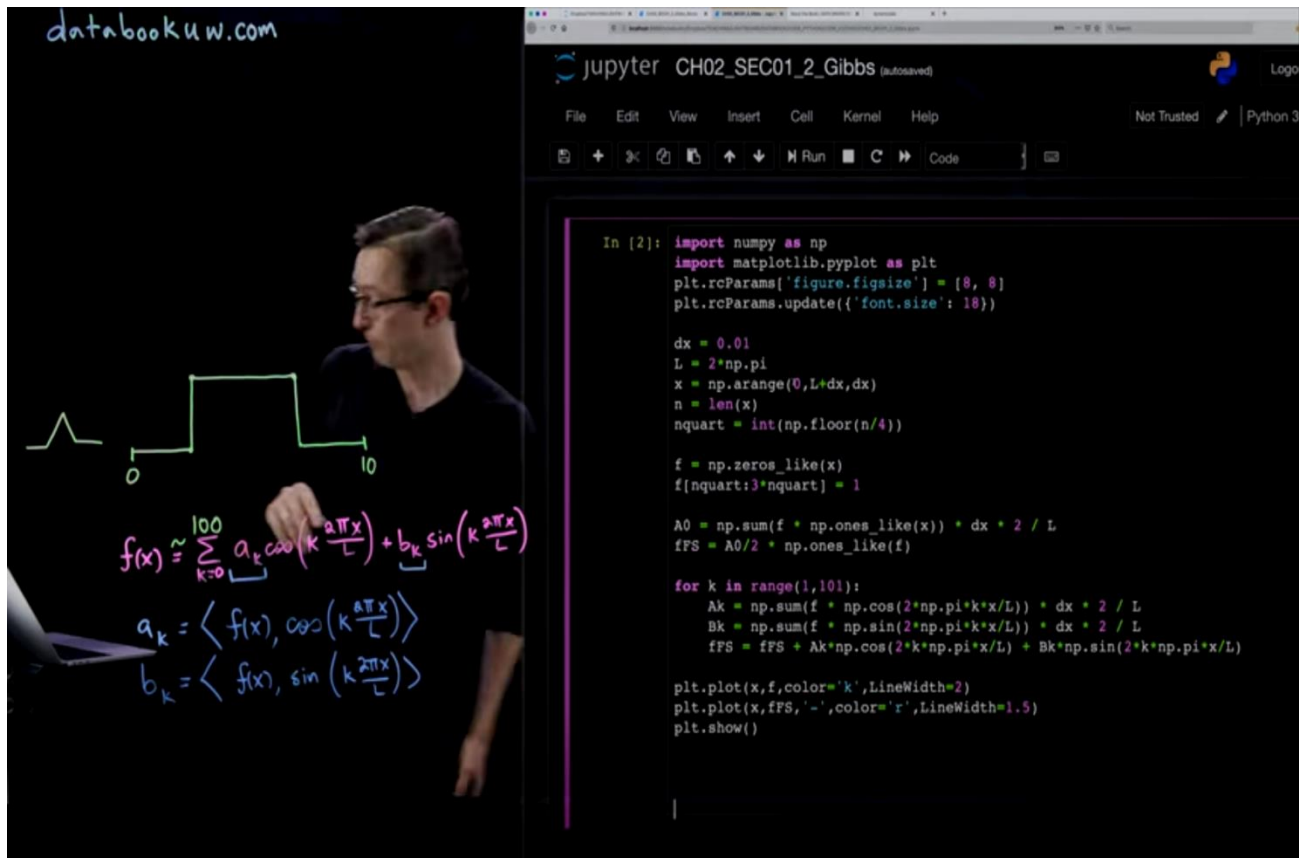
DAILY ASSESSMENT FORMAT

Date:	26/05/2020	Name:	Shilpa N
Course:	DSP	USN:	4AL16EC071
Topic:	DAY 2	Semester & Section:	8 "B"
Github Repository:	Shilpan-test		

FORENOON SESSION DETAILS

Report – Report can be typed or hand written for up to two pages.

Fourier Series & Gibbs Phenomena using Python- The Gibbs phenomenon is an overshoot (or "ringing") of Fourier series and other eigenfunction series occurring at simple discontinuities. It can be reduced with the Lanczos sigma factor. In the session explained about the Gibbs phenomenon using the example & code as shown in below image,



The image is a composite of two parts. On the left, a man is pointing to a blackboard. On the blackboard, a square wave is drawn, and the Fourier series expansion is written as:

$$f(x) \approx \sum_{k=0}^{100} a_k \cos\left(k \frac{2\pi x}{L}\right) + b_k \sin\left(k \frac{2\pi x}{L}\right)$$

Below this, the coefficients are defined as:

$$a_k = \left\langle f(x), \cos\left(k \frac{2\pi x}{L}\right) \right\rangle$$

$$b_k = \left\langle f(x), \sin\left(k \frac{2\pi x}{L}\right) \right\rangle$$

On the right, a Jupyter Notebook window titled "CH02_SEC01_2_Gibbs" is shown. It contains the following Python code:

```
In [2]: import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = [8, 8]
plt.rcParams.update({'font.size': 18})

dx = 0.01
L = 2*np.pi
x = np.arange(0,L+dx,dx)
n = len(x)
nquart = int(np.floor(n/4))

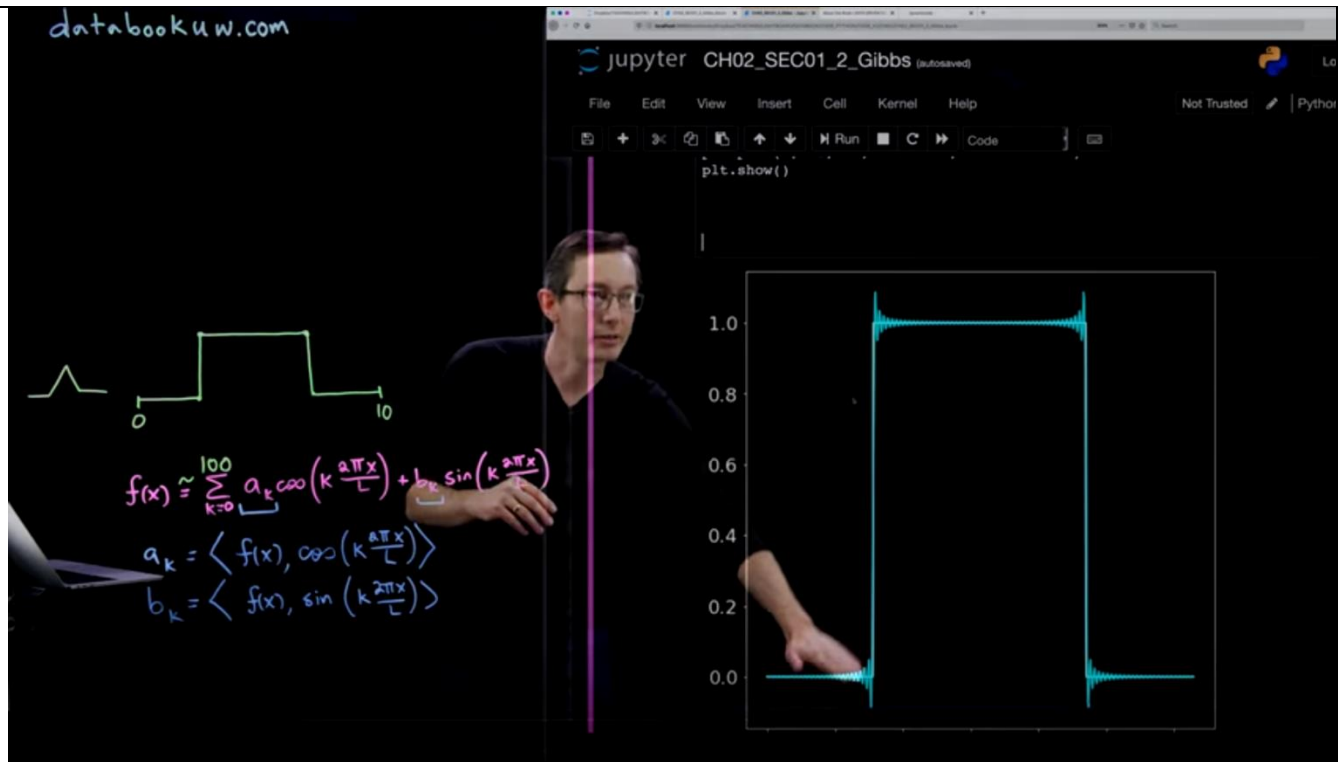
f = np.zeros_like(x)
f[nquart:3*nquart] = 1

A0 = np.sum(f * np.ones_like(x)) * dx * 2 / L
fFS = A0/2 * np.ones_like(f)

for k in range(1,101):
    Ak = np.sum(f * np.cos(2*np.pi*k*x/L)) * dx * 2 / L
    Bk = np.sum(f * np.sin(2*np.pi*k*x/L)) * dx * 2 / L
    fFS = fFS + Ak*np.cos(2*k*np.pi*x/L) + Bk*np.sin(2*k*np.pi*x/L)

plt.plot(x,f,color='k',LineWidth=2)
plt.plot(x,fFS,'-',color='r',LineWidth=1.5)
plt.show()
```

the output for the code is as shown below,



Fourier Transform

The Fourier transform is simply a method of expressing a function (which is a point in some infinite dimensional vector space of functions) in terms of the sum of its projections onto a set of basis functions. Since an image is only defined on a closed and bounded domain (the image window), we can assume that the image is defined as being zero outside this window. In other words, we can assume that the image function is *integrable* over the real line. Fourier transform is given by,

$$\hat{f}(\xi) = \int_{-\infty}^{\infty} f(x) e^{-2\pi i x \xi} dx, \quad (\text{Eq.1})$$

for any real number ξ .

The inverse Fourier transform is given by,

$$f(x) = \int_{-\infty}^{\infty} \hat{f}(\xi) e^{2\pi i x \xi} d\xi, \quad (\text{Eq.2})$$

for any real number x .

Fourier Transform Derivatives

The Fourier transform of a derivative of a function, $f'(t)$:

$$\mathcal{F}\{f'(t)\} = i\omega F(\omega)$$

Proof:

$$\mathcal{F}\{f'(t)\} = \int_{-\infty}^{\infty} f'(t) \exp(-i\omega t) dt$$

Integrate by parts:

$$\begin{aligned} &= - \int_{-\infty}^{\infty} f(t) [(-i\omega) \exp(-i\omega t)] dt \\ &= i\omega \int_{-\infty}^{\infty} f(t) \exp(-i\omega t) dt \\ &= i\omega F(\omega) \end{aligned}$$

Fourier transform & convolution

$$w(t) = u(t)v(t) \Leftrightarrow W(f) = U(f) * V(f)$$

$$w(t) = u(t)v(t) \Leftrightarrow W(f) = U(f) * V(f)$$

$$w(t) = u(t) * v(t) \Leftrightarrow W(f) = U(f)V(f)$$

$$w(t) = u(t)v(t) \Leftrightarrow W(f) = U(f) * V(f)$$

$$w(t) = u(t) * v(t) \Leftrightarrow W(f) = U(f)V(f)$$

Convolution in the time domain is equivalent to multiplication in the frequency domain and vice versa.

Proof of second line:

Given $u(t)$, $v(t)$ and $w(t)$ satisfying

$w(t) = u(t)v(t) \Leftrightarrow W(f) = U(f) * V(f)$ define dual waveforms $x(t)$, $y(t)$ and $z(t)$ as follows:

$$x(t) = U(t) \Leftrightarrow X(f) = u(-f) \text{ [duality]}$$

$$y(t) = V(t) \Leftrightarrow Y(f) = v(-f) \quad z(t) = W(t) \Leftrightarrow Z(f) = w(-f)$$

Now the convolution property becomes:

$$w(-f) = u(-f) v(-f) \Leftrightarrow W(t) = U(t) * V(t) \text{ [sub } t \leftrightarrow \pm f]$$

$$Z(f) = X(f)Y(f) \Leftrightarrow z(t) = x(t) * y(t) \text{ [duality]}$$

Intuition of Fourier Transform and Laplace Transform

Intuition of Fourier Transform

Fourier theory, when viewed in its historical context, is the classic example of this, since Fourier's "original" intuition" was hotly contested by contemporaries of the stature of Lagrange and Poisson. Fourier's "intuition" was at once both insightful (as history has amply shown) and audacious: he boldly claimed that his approach was capable of "developing any function whatever in an infinite series of sines or cosines of multiple areas. When Fourier published his theory in 1822 it certainly was not" intuitively" obvious that an arbitrary non-periodic function could be represented as as a convergent infinite series of sines and cosines. Yet Fourier's original motivation is still completely valid: you can construct something complex from simpler building blocks.

Intuition of Laplace Transform

The logic behind the Laplace transform takes a different route compared to the logic which underpins the Fourier transform. Laplace developed a theory of generating functions in his famous treatise on probability theory ("Analytic Theory of Probability") where he expressed a function u as follows:
 $u = a_0 + a_1 t + a_2 t^2 + \dots$

So imagine now that we have a power series representation of some function ie $A(x) = \sum_{k=0}^{\infty} a(k)x^k$ where the $a(k)$ are constants. These constants can be viewed as the values of a discrete function initially and then the leap to the continuous domain can be made. For instance, if $a(k) = 1$ for all $k = 0, 1, 2, \dots$ and $|x| < 1$ we will have a convergent power series:

$$A(x) = 1 + x + x^2 + x^3 + \dots = \frac{1}{1-x} \quad (9)$$

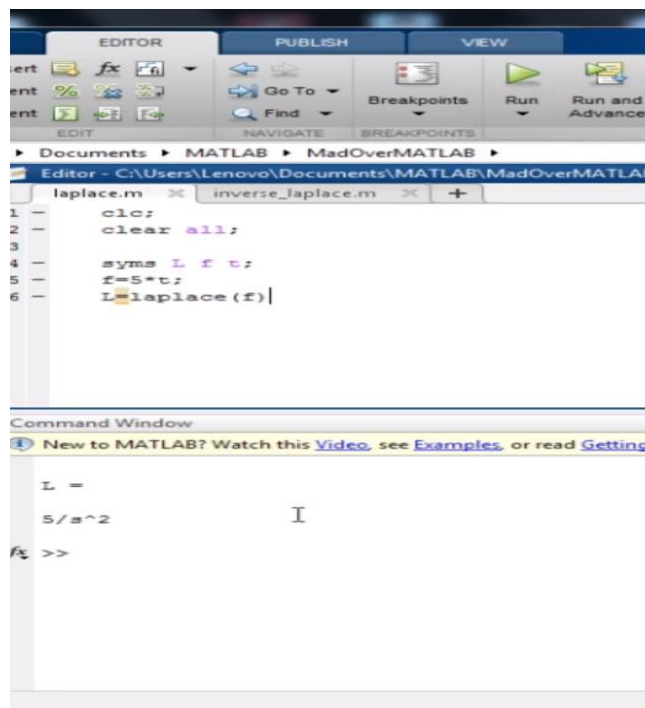
Similarly if $a(k) = \frac{1}{k!}$ we will have:

$$A(x) = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots = e^x \quad (10)$$

and this holds for all real x .

Implementation of Laplace Transform using Matlab

ex 1: find Laplace transform of $5(t)$



The screenshot shows the MATLAB Editor with a script named 'laplace.m'. The script contains the following code:

```
1 - clc;
2 - clear all;
3
4 - syms L f t;
5 - f=5*t;
6 - L=laplace(f)
```

The Command Window shows the output of the script:

```
L =
    5/s^2
```

The Command Window also displays a message: "New to MATLAB? Watch this Video, see Examples, or read Getting Started".

ex 2: Laplace transform of exponential

```
clc;
clear all;

syms L f t a;
f=exp(-5*t);
L=laplace(f)
```

```
laplace.m  X  inverse_laplace.m  X  +
1 -      clc;
2 -      clear all;
3
4 -      syms L f t;
5 -      f=(exp(-3*t)*sin(2*t))/t
6 -      L=laplace(f)
```

Command Window

 New to MATLAB? Watch this [Video](#), see [Examples](#), or

```
f =

(sin(2*t)*exp(-3*t))/t

L =

atan(2/(s + 3))
```

For inverse Laplace

```
1 - clc;
2 - clear all;
3
4 - syms F s;
5 - F = (s+29)/(s^3+4*s^2+9*s+36);
6 - ilaplace(F)
```

Command Window

i New to MATLAB? Watch this [Video](#), see [Examples](#), or read [Getting Started](#).

F =

$$(s + 29) / (s^3 + 4s^2 + 9s + 36)$$

ans =

$$\exp(-4t) - \cos(3t) + (5\sin(3t))/3$$

Applications of Z-Transform

- A closed-loop (or feedback) control system is shown in Figure.

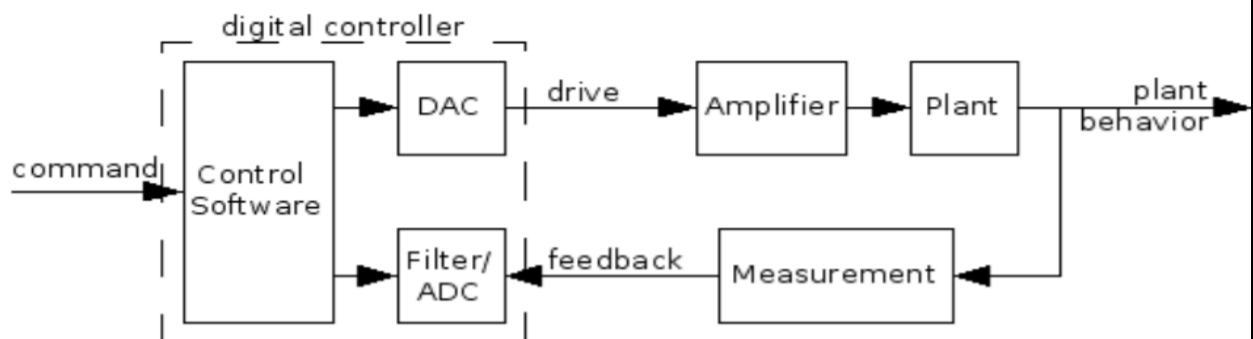


Figure 1: A Generic Computerized Control System.

- If you can describe your plant and your controller using linear difference equations, and if the coefficients of the equations don't change from sample to sample, then your controller and plant are linear and shift-invariant, and you can use the z transform.

- Suppose x_n =output of the plant at sample time n
 u_n =command to the DAC at sample time n
 a and b =constants set by the design of the plant
- You can solve the behaviour equation of the plant over time.

$$x_n = (a_1 x_{n-1} + a_2 x_{n-2} + \dots) + (b_1 u_{n-1} + b_2 u_{n-2} + \dots)$$
- Furthermore you can also investigate what happens when you add feedback to the system.
- The z transform allows you to do both of these things.

Find the Z-Transform of sequence using Matlab

the code discussed in the session is as below,

1.to find the sequence $a=n+1$

```
syms n;

a = n+1;

b = ztrans(a);
disp(b)
(z - 1) + z/(z - 1)^2
```

2. by using command pretty we get

```
>> pretty(b)

      z          z
----- + -----
z - 1          2
              (z - 1)
```

3.sequence= 2^n


```
>> a = 2^n;  
>> b = ztrans(a);  
>> b  
  
b =  
  

$$z/(z - 2)$$

```

4. sine and cosine functions

```
>> syms n w;  
>>  
>> a = sin(w*n);  
>>  
>> b = ztrans(a);  
>>  
>> disp(b)  

$$(z \sin(w)) / (z^2 - 2 \cos(w) z + 1)$$

```

Date:	26/05/2020	Name:	Shilpa N
Course:	PYTHON	USN:	4AL16EC071
Topic:	DAY 7	Semester & Section:	8 "B"

AFTERNOON SESSION DETAILS

What is Web Framework?

Web Application Framework or simply Web Framework represents a collection of libraries and modules that enables a web application developer to write applications without having to bother about low-level details such as protocols, thread management etc.

What is Flask?

Flask is a web application framework written in Python. It is developed by **Armin Ronacher**, who leads an international group of Python enthusiasts named Pocco. Flask is based on the Werkzeug WSGI toolkit and Jinja2 template engine. Both are Pocco projects.

WSGI

Web Server Gateway Interface (WSGI) has been adopted as a standard for Python web application development. WSGI is a specification for a universal interface between the web server and the web applications.

Werkzeug

It is a WSGI toolkit, which implements requests, response objects, and other utility functions. This enables building a web framework on top of it. The Flask framework uses Werkzeug as one of its bases.

Jinja2

Jinja2 is a popular templating engine for Python. A web templating system combines a template with a certain data source to render dynamic web pages.

Flask is often referred to as a micro framework. It aims to keep the core of an application simple yet extensible. Flask does not have built-in abstraction layer for database handling, nor does it have form a validation support. Instead, Flask supports the can customize above code as per our requirement like duration, websites, custom messages etc.

In order to test **Flask** installation, type the following code in the editor as **Hello.py**

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
```

```
return 'Hello World'

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

Importing flask module in the project is mandatory. An object of Flask class is our **WSGI** application.

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

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

```
app.route(rule, options)
```

1. The **rule** parameter represents URL binding with the function.
2. The **options** is a list of parameters to be forwarded to the underlying Rule object.

In the above example, '/' URL is bound with **hello_world()** function. Hence, when the home page of web server is opened in browser, the output of this function will be rendered.

Finally the **run()** method of Flask class runs the application on the local development server.

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

It is possible to build a URL dynamically, by adding variable parts to the rule parameter. This variable part is marked as **<variable-name>**. It is passed as a keyword argument to the function with which the rule is associated.

In the following example, the rule parameter of **route()** decorator contains **<name>** variable part attached to URL **'/hello'**. Hence, if the **http://localhost:5000/hello/TutorialsPoint** is entered as a **URL** in the browser, **'TutorialPoint'** will be supplied to **hello()** function as argument.

```
from flask import Flask
app = Flask(__name__)

@app.route('/hello/<name>')
def hello_name(name):
    return 'Hello %s!' % name

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

Save the above script as **hello.py** and run it from Python shell. Next, open the browser and enter URL **http://localhost:5000/hello/TutorialsPoint**.

The following output will be displayed in the browser.

The **url_for()** function is very useful for dynamically building a URL for a specific function. The function accepts the name of a function as first argument, and one or more keyword arguments, each corresponding to the variable part of URL.

The following script demonstrates use of **url_for()** function.

```
from flask import Flask, redirect, url_for
app = Flask(__name__)

@app.route('/admin')
def hello_admin():
    return 'Hello Admin'

@app.route('/guest/<guest>')
def hello_guest(guest):
    return 'Hello %s as Guest' % guest

@app.route('/user/<name>')
def hello_user(name):
    if name == 'admin':
        return redirect(url_for('hello_admin'))
    else:
        return redirect(url_for('hello_guest', guest = name))

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

The above script has a function **user(name)** which accepts a value to its argument from the URL.

The **User()** function checks if an argument received matches 'admin' or not. If it matches, the application is redirected to the **hello_admin()** function using **url_for()**, otherwise to the **hello_guest()** function passing the received argument as guest parameter to it.

Save the above code and run from Python shell.

Open the browser and enter URL as – **http://localhost:5000/user/admin**

The application response in browser is –

Hello Admin

Enter the following URL in the browser – **http://localhost:5000/user/mvl**

The application response now changes to –

Hello mvl as Guest

By default, the Flask route responds to the **GET** requests. However, this preference can be altered by providing methods argument to **route()** decorator.

In order to demonstrate the use of **POST** method in URL routing, first let us create an HTML form and use the **POST** method to send form data to a URL.

Save the following script as login.html

```
<html>
<body>
```

```
<form action = "http://localhost:5000/login" method = "post">
  <p>Enter Name:</p>
  <p><input type = "text" name = "nm" /></p>
  <p><input type = "submit" value = "submit" /></p>
</form>
</body>
</html>
```

Now enter the following script in Python shell.

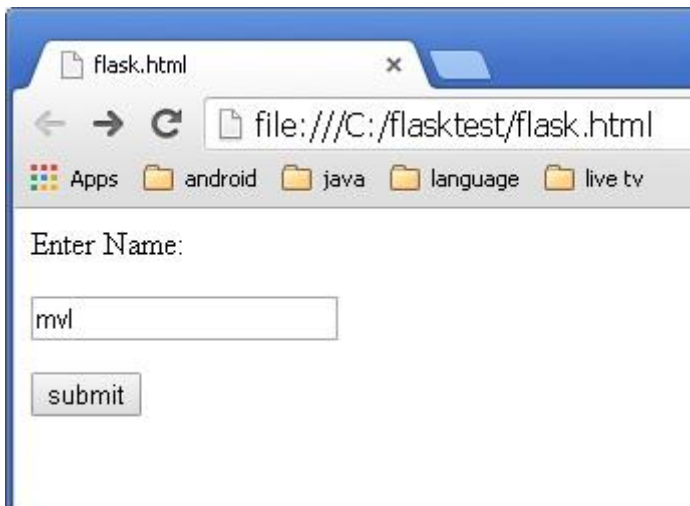
```
from flask import Flask, redirect, url_for, request
app = Flask(__name__)

@app.route('/success/<name>')
def success(name):
    return 'welcome %s' % name

@app.route('/login', methods = ['POST', 'GET'])
def login():
    if request.method == 'POST':
        user = request.form['nm']
        return redirect(url_for('success', name = user))
    else:
        user = request.args.get('nm')
        return redirect(url_for('success', name = user))

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

After the development server starts running, open **login.html** in the browser, enter name in the text field and click **Submit**.

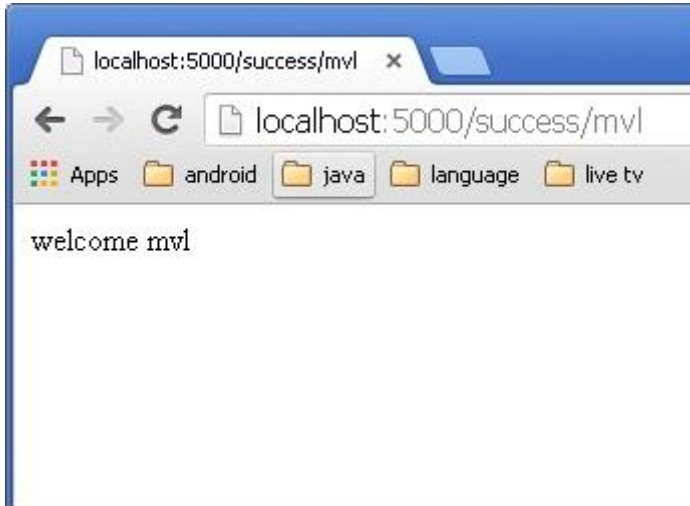


Form data is POSTed to the URL in action clause of form tag.

http://localhost/login is mapped to the **login()** function. Since the server has received data by **POST** method, value of 'nm' parameter obtained from the form data is obtained by –

```
user = request.form['nm']
```

It is passed to **‘/success’** URL as variable part. The browser displays a **welcome** message in the window.



Change the method parameter to **‘GET’** in **login.html** and open it again in the browser. The data received on server is by the **GET** method. The value of 'nm' parameter is now obtained by –

```
User = request.args.get('nm')
```

Here, **args** is dictionary object containing a list of pairs of form parameter and its corresponding value. The value corresponding to 'nm' parameter is passed on to **‘/success’** URL as before.

It is possible to return the output of a function bound to a certain URL in the form of HTML. For instance, in the following script, **hello()** function will render **‘Hello World’** with **<h1>** tag attached to it.

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def index():
    return '<html><body><h1>Hello World</h1></body></html>'

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

However, generating HTML content from Python code is cumbersome, especially when variable data and Python language elements like conditionals or loops need to be put. This would require frequent escaping from HTML.

This is where one can take advantage of **Jinja2** template engine, on which Flask is based. Instead of returning hardcoded HTML from the function, a HTML file can be rendered by the **render_template()** function.

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def index():
    return render_template('hello.html')

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

Flask will try to find the HTML file in the templates folder, in the same folder in which this script is present.

- Application folder
 - Hello.py
 - templates
 - hello.html

The term '**web templating system**' refers to designing an HTML script in which the variable data can be inserted dynamically. A web template system comprises of a template engine, some kind of data source and a template processor.

Flask uses **jinja2** template engine. A web template contains HTML syntax interspersed placeholders for variables and expressions (in these case Python expressions) which are replaced values when the template is rendered.

The following code is saved as **hello.html** in the templates folder.

```
<!doctype html>
<html>
  <body>

    <h1>Hello {{ name }}!</h1>

  </body>
</html>
```

Next, run the following script from Python shell.

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

@app.route('/hello/<user>')
def hello_name(user):
    return render_template('hello.html', name = user)
```

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

As the development server starts running, open the browser and enter URL as – **http://localhost:5000/hello/mvl**

The **variable** part of URL is inserted at **{{ name }}** place holder.



The **jinja2** template engine uses the following delimiters for escaping from HTML.

- {% ... %} for Statements
- {{ ... }} for Expressions to print to the template output
- {# ... #} for Comments not included in the template output
- # ... ## for Line Statements

In the following example, use of conditional statement in the template is demonstrated. The URL rule to the **hello()** function accepts the integer parameter. It is passed to the **hello.html** template. Inside it, the value of number received (marks) is compared (greater or less than 50) and accordingly HTML is conditionally rendered.

The Python Script is as follows –

```
from flask import Flask, render_template  
app = Flask(__name__)  
  
@app.route('/hello/<int:score>')  
def hello_name(score):  
    return render_template('hello.html', marks = score)  
  
if __name__ == '__main__':  
    app.run(debug = True)
```

HTML template script of **hello.html** is as follows –


```

<!doctype html>
<html>
  <body>
    {% if marks>50 %}
      <h1> Your result is pass!</h1>
    {% else %}
      <h1>Your result is fail</h1>
    {% endif %}
  </body>
</html>

```

Note that the conditional statements **if-else** and **endif** are enclosed in delimiter **{%..%}**.

Run the Python script and visit URL **http://localhost/hello/60** and then **http://localhost/hello/30** to see the output of HTML changing conditionally.

The Python loop constructs can also be employed inside the template. In the following script, the **result()** function sends a dictionary object to template **results.html** when URL **http://localhost:5000/result** is opened in the browser.

The Template part of **result.html** employs a **for loop** to render key and value pairs of dictionary object **result** as cells of an HTML table.

Run the following code from Python shell.

```

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.

```

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

```

</html>

Here, again the Python statements corresponding to the **For** loop are enclosed in `{%..%}` whereas, the expressions **key and value** are put inside `{{ }}`.

After the development starts running, open **<http://localhost:5000/result>** in the browser to get the following output.

