Web Applications

Back-End : Processing User Input with Flask

Overview

	Web Applications
Front-End	Part 1a : Basic HTML Part 1b: Basic CSS Part 2 : HTML – Forms
Back-End	Part 3 : Form with Flask Part 4 : SQLite with Flask

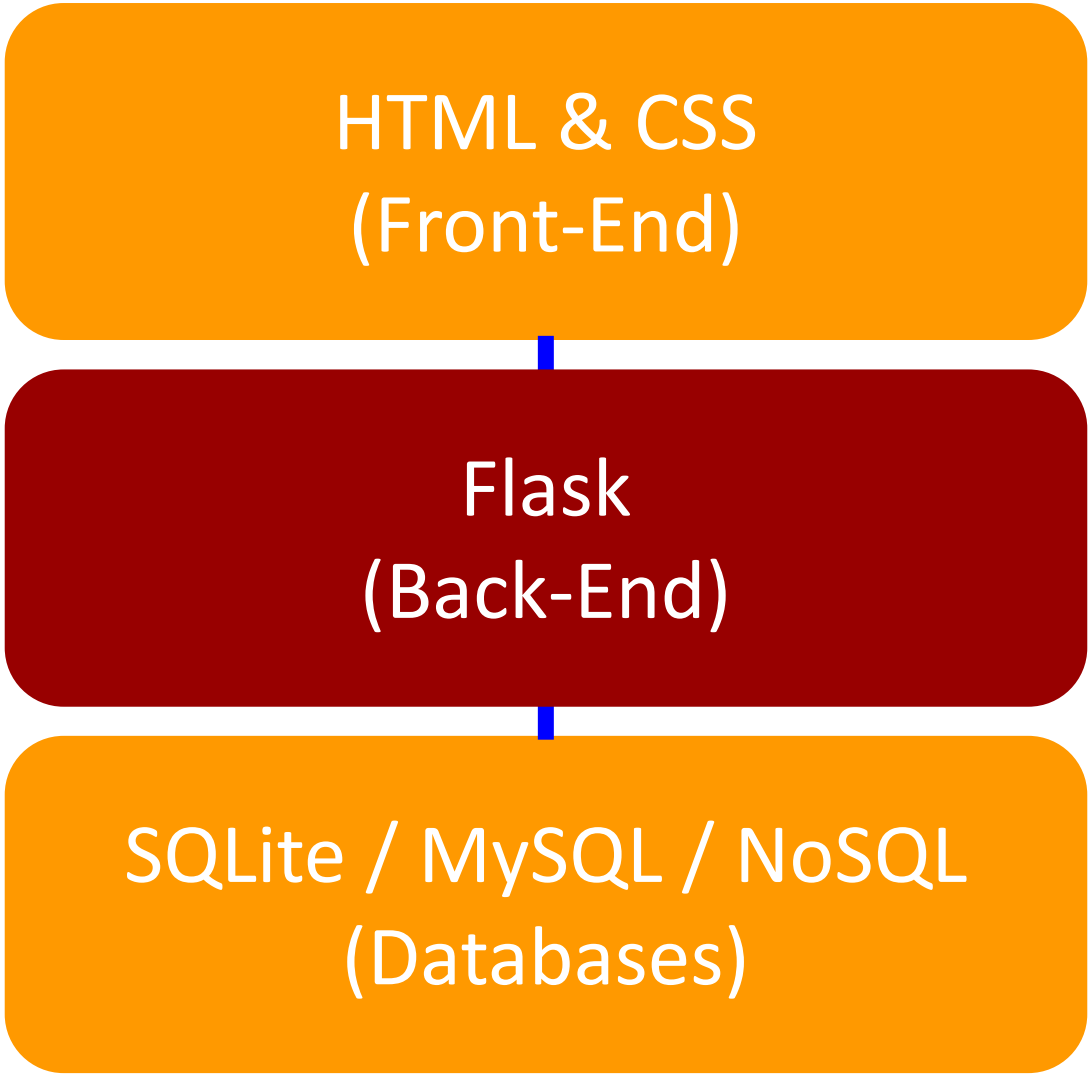
Part 3 : Flask Server

Processing user input

How Websites Work ?

www.youtube.com/watch?v=D8c4JZW73cM

Web Application Development



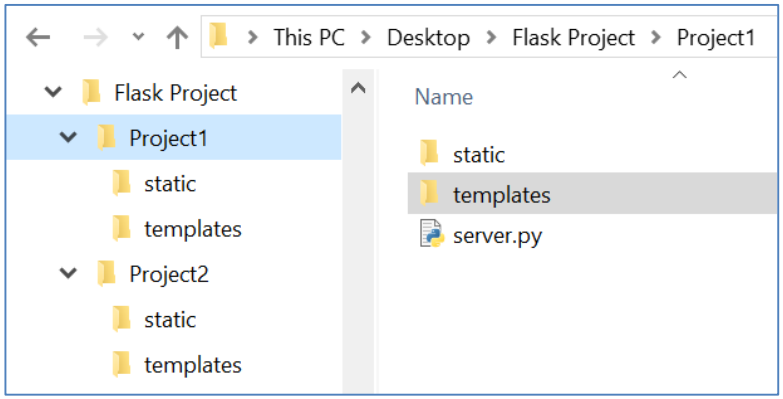
What is Flask ?

- An *open-source* web *micro-framework* running on Python
- When put on a web server, it can respond to browser requests, send webpages, and handle routing

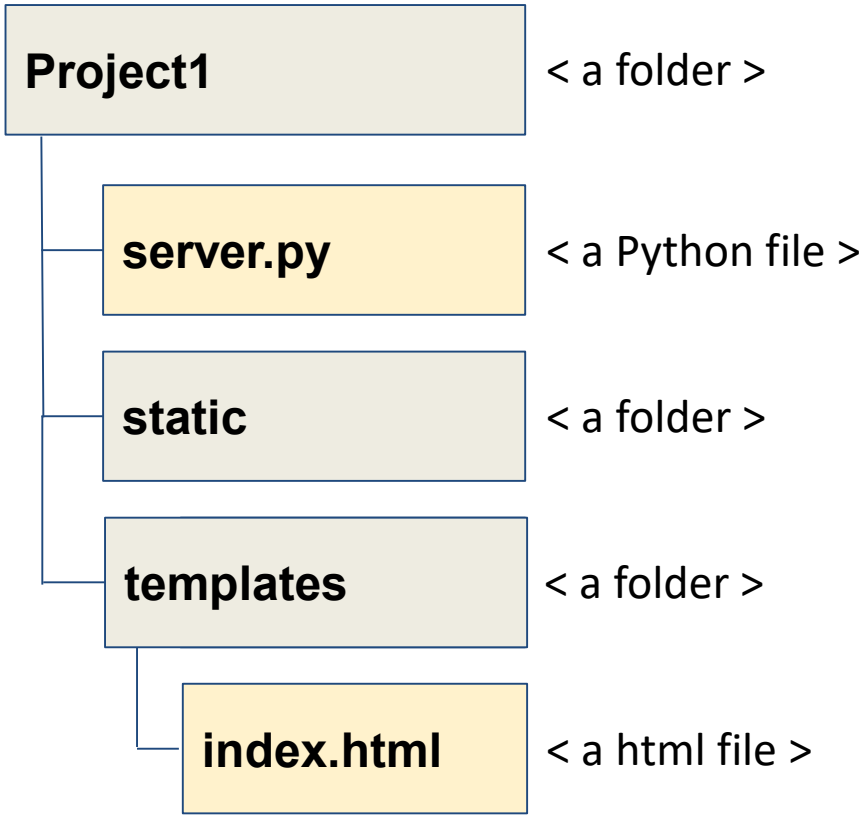
How to select a framework? (mozilla.org)

Project Folder

- Make a ‘Flask Project’ folder to contain all the different projects
- Within ‘Project1’, add 2 sub-folders templates and static; add a new file server.py
- Create the index.html in the templates sub-folder
- Note : we will keep all the picture files in the static sub-folder



Project Folder Hierarchy



Project 1

Return “Hello World!”

Project 1

server.py

```
from flask import Flask

app = Flask(__name__)

@app.route('/')

def index():
    return "Hello World!"

app.run(debug=True, port=5000)

# launch web browser with url 127.0.0.1:5000
```

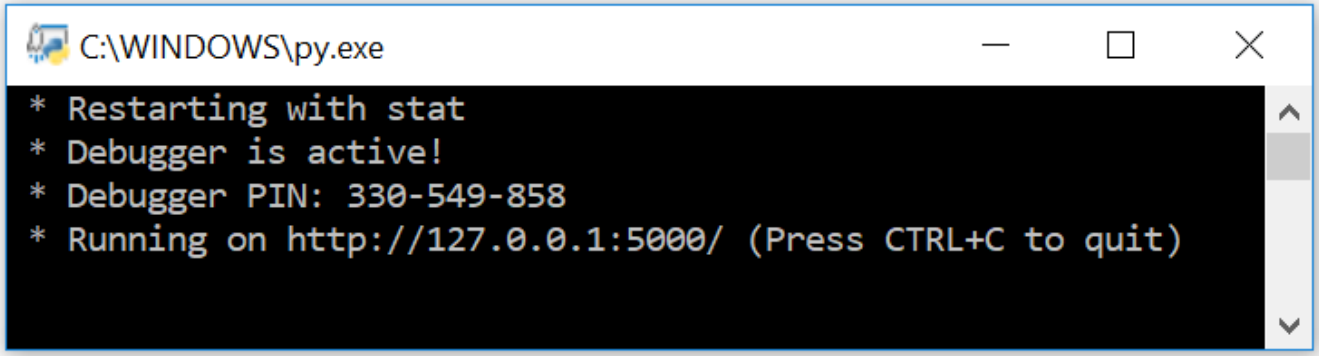
Can use value from 0 to 65535, except for 0 to 1024 which are reserved

.

Try to use a different port no. if your browser is not updated with the changing in your index.html

Start serving!

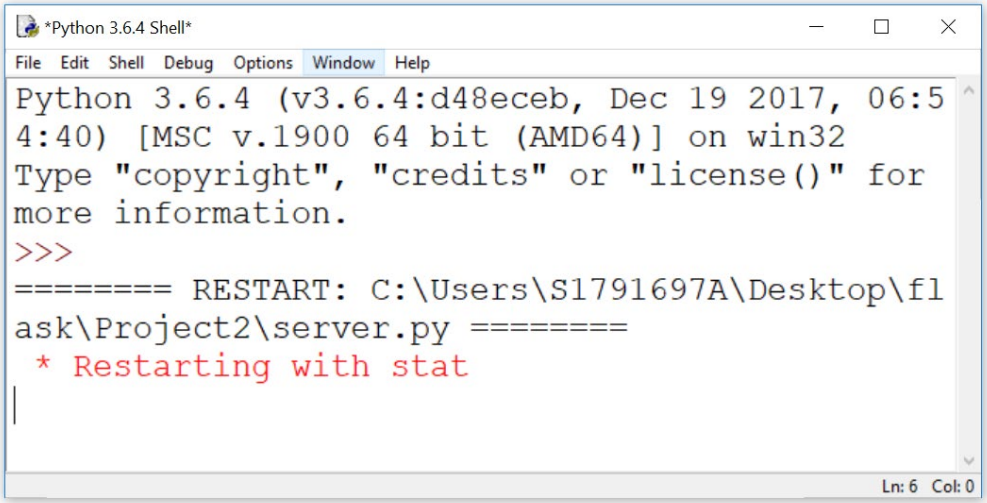
- Remember to save the edited server.py before running it.
- Double click on server.py to run the Python file.
- A dialog box will show that your server has started and is running.



- Use a web browser to access the url at http://127.0.0.1:5000.
- Press CTRL and C to shut the server.

Start serving!

- Alternatively, you may use F5 to run the Python file server.py and the Python Shell will remain active while the server is running.
- Use a web browser to access the url at http://127.0.0.1:5000.
- Closing the Python Shell will shut the server.



What just happened?

- We set up a Flask server, with a default directory structure
- In our server, we defined the **route** "/"
- When the user (your browser) reaches this route, it runs the Python function `index()` and return “Hello World!” to display on the browser

Duplicate the whole folder '**Project1**' with all the sub-folders and files in it and rename it as '**Project2**'.

For subsequent projects, you will duplicate the folders and files from the previous project.

Important: You must always shut down the server before running server.py again.

Project 2

Return “Hello World!” and “Hello David!”

Project 2

server.py

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

@app.route('/')      # url 127.0.0.1:5000/
def index():
    return "Hello World!"

@app.route('/david')# url 127.0.0.1:5000/david
def david():
    return "Hello David!"

app.run(debug=True, port=5000)
```

Project 3

Return “Hello < anybody > !”

Project 3

server.py

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

@app.route('/')      # url 127.0.0.1:5000/
def index():
    return "Hello World!"

@app.route('/<string:name>')   # url 127.0.0.1:5000/<name>
def hello(name):
    name = name.capitalize()
    return f"<h1>Hello {name}!</h1>"

app.run(debug=True, port=5000)
```

Jinja code

formatting

HTML code

Project 4a

Render index.html

Project 4a

server.py

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

@app.route('/')      # url 127.0.0.1:5000/
def index():
    return render_template("index.html")

app.run(debug=True, port=5000)
```

What just happened?

- We set up a Flask server, with a default directory structure
- In our server, we defined the **route** "/"
- When the user (your browser) reaches this route, Flask renders the template called index.html, which by default, is in the templates folder
- So the user (your browser) will receive the entire index.html file from your server.

Project 4a

index.html

```
<!DOCTYPE html>
<html>

<head> </head>

<body>

<h1> Hello World! </h1>

</body>

</html>
```

Project 4b

Render index.html with picture in the static folder

Project 4b

index.html

```
<!DOCTYPE html>
<html>
<head></head>
<body>

    <h1> The Toy Story </h1>
    <img src = " {{ url_for('static',
    filename='Toy_Story.jpg') }}">

</body>
</html>
```

HTML code

Pic file in static folder

Jinja code

Project 4c

Render index.html with css file in the static folder

Project 4c

index.html

```
<!DOCTYPE html>
<html>
<head>
<title>My First Webpage</title>
    <link rel="stylesheet" type="text/css"
                href="{{ url_for('static',
                filename='mystyle.css') }}">
</head>
<body><h1> The Toy Story </h1>
<img src="{{url_for('static',\
filename='Toy_Story.jpg')}}"></body>
</html>
```

HTML code to link to external CSS

mystyle.css in static folder

Jinja code

Project 4c

mystyle.css

```
<style>

html { height: 100% }

h1 {
    color: blue;
    size: 30;
    font-family: Courier;
    }

</style>
```

Project 5

Render index.html with ‘headline’

Project 5

server.py

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

@app.route('/')      # url 127.0.0.1:5000/
def index():
    display = "Hello World!"
    return render_template("index.html",
                           headline = display)

app.run(debug=True, port=5000)
```

Project 5

index.html

```
<!DOCTYPE html>
<html>

<head> </head>

<body>

<h1> {{ headline }} </h1>

</body>

</html>
```

Jinja code

Project 6

‘Hello’ and ‘Goodbye’ with ‘headline’

Project 6

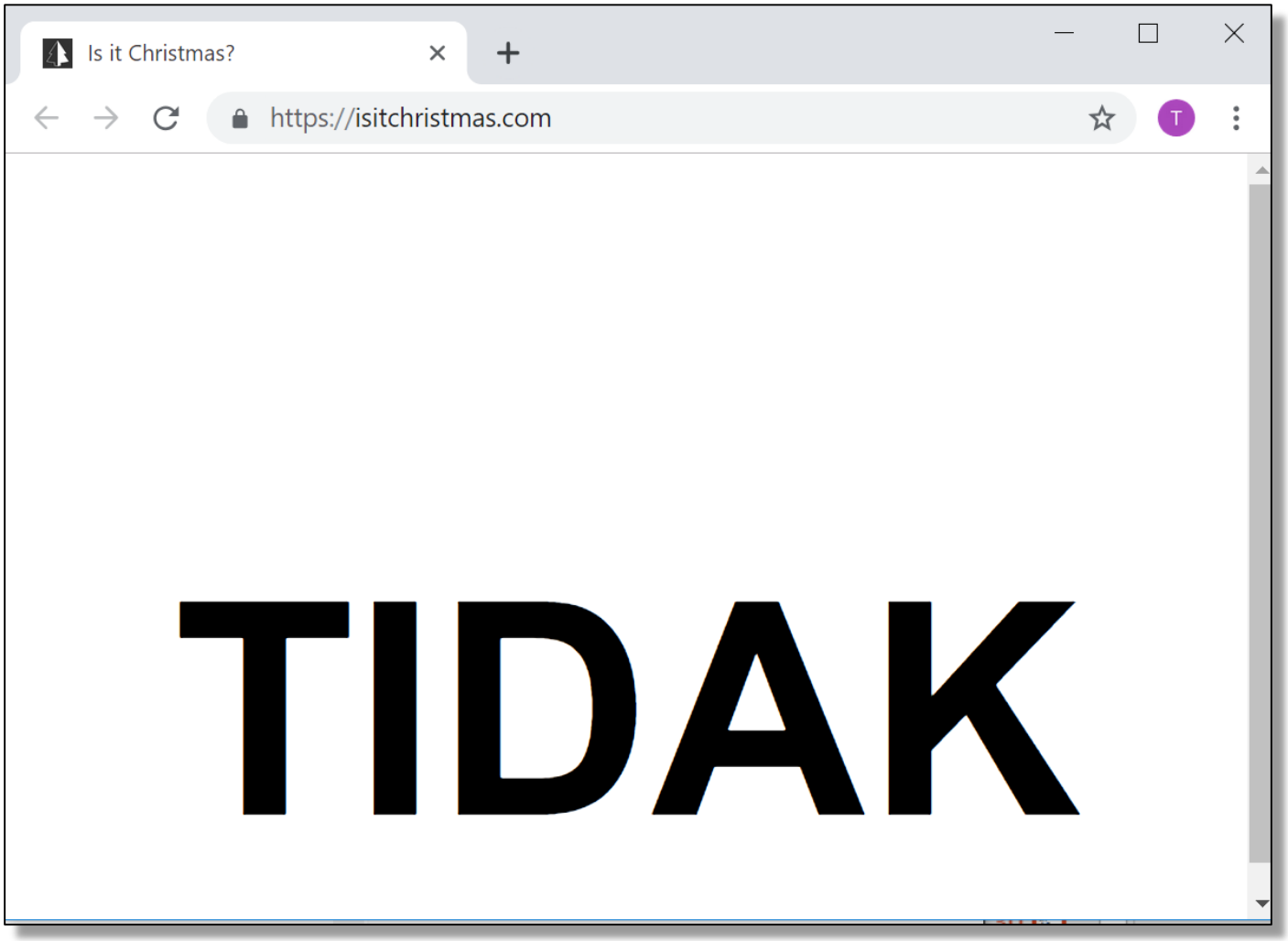
server.py

```
from flask import Flask, render_template
app = Flask(__name__)
@app.route('/')        # 127.0.0.1:5000/
def index():
    display = "Hello World!"
    return render_template("index.html",
                           headline = display)
@app.route('/goodbye')  # 127.0.0.1:5000/goodbye
def goodbye():
    display = "Goodbye!"
    return render_template("index.html",
                           headline = display)
app.run(debug=True, port=5000)
```

Project 7a

Is it New Year Day? Use Boolean without conditional

Is it Christmas ?



Project 7a

server.py

```
from flask import Flask, render_template
from datetime import datetime
app = Flask(__name__)
@app.route('/')      # 127.0.0.1:5000/
def index():
    # Is it New Year Day?
    now = datetime.now()
    display = now.month == 1 and now.day == 1
    return render_template("index.html",
                           headline = display)

app.run(debug=True, port=5000)
```

Project 7a

index.html

```
<!DOCTYPE html>
<html>

<head> </head>

<body>

<h1> Is today the New Year Day?
Ans: {{ headline }} </h1>

</body>

</html>
```

Jinja code

Project 7b

Is it New Year Day? Use conditional in Python

Project 7b

server.py

```
from flask import Flask, render_template
from datetime import datetime
app = Flask(__name__)
@app.route('/')        # 127.0.0.1:5000/
def index():           # Is it New Year Day?
    now = datetime.now()
    true_false = now.month == 1 and now.day == 1
    if display:
        return render_template("index.html",
                               headline = “Yes!”)
    else:
        return render_template("index.html",
                               headline = “No!”)
app.run(debug=True, port=5000)
```

Project 7c

Is it New Year Day? Use conditional in index.html

Project 7c (same as 7a)

server.py

```
from flask import Flask, render_template
from datetime import datetime
app = Flask(__name__)
@app.route('/')        # 127.0.0.1:5000/
def index():           # Is it New Year Day?
    now = datetime.now()
    display = now.month == 1 and now.day == 1
    return render_template("index.html",
                           headline = display)

app.run(debug=True, port=5000)
```

Note : `headline` **is a Boolean !**

Project 7c

index.html

```
<!DOCTYPE html>
<html>
<head> </head>

<body>
{% if headline %}
    <h1> Yes! It is New Year Day! </h1>
{% else %}
    <h1> Nope! </h1>
{% endif %}

</body>
</html>
```

Jinja codes

Project 8

Use iteration in index.html

Project 8

server.py

```
from flask import Flask, render_template
app = Flask(__name__)
@app.route('/')      # 127.0.0.1:5000/
def index():
    names = ["Alice", "Bob", "Charlie"]
    return render_template("index.html",
                           namelist = names)

app.run(debug=True, port=5000)
```

Note : `namelist` **is a list !**

Project 8

index.html

```
<!DOCTYPE html>
<html>
<head> </head>

<body> <h1> Hello Everyone </h1>
<ul>
{% for name in namelist %}
    <li> {{ name }} </li>
{% endfor %}
</ul>
</body>
</html>
```

HTML code for unordered list

Jinja codes

Project 9

index.html link to hello.html

Project 9

server.py

```
from flask import Flask, render_template
app = Flask(__name__)
@app.route('/')      # 127.0.0.1:5000/
def index():
    return render_template("index.html")

@app.route('/hello')      # 127.0.0.1:5000/hello
def hello():
    return render_template("hello.html")

app.run(debug=True, port=5000)
```

Project 9

index.html

```
<!DOCTYPE html>
<html>
<head> </head>

<body>
<h1> This is the Index Page </h1>

<a href=" {{ url_for('hello') }} ">
        Click here to go to hello page </a>

</body>
</html>
```

Jinja codes

HTML code
hyperlink reference

Project 9

hello.html

```
<!DOCTYPE html>
<html>
<head> </head>

<body>
<h1> This is the Hello Page </h1>

<a href=" {{ url_for('index') }} ">
        Click here to go to index page </a>

</body>
</html>
```

Jinja codes

HTML code
hyperlink reference

Project 10

Do you see the similarities between the `index.html` and `hello.html` ?

Similar “heading”

index.html

```
<!DOCTYPE html>
<html>
<head> </head>

<body>
<h1> This is the Index Page </h1>

<a href=" {{ url_for('hello') }} ">
        Click here to go to hello page </a>

</body>
</html>
```

hello.html

```
<!DOCTYPE html>
<html>
<head> </head>

<body>
<h1> This is the Hello Page </h1>

<a href=" {{ url_for('index') }} ">
        Click here to go to index page </a>

</body>
</html>
```

Similar “body”

Project 10

Template Inheritance – layout.html

Project 10

layout.html

```
<!DOCTYPE html>
<html>
<head> </head>

<body>
<h1> {% block heading %}{% endblock %} </h1>

{% block body %} {% endblock %}

</body>
</html>
```

“heading” block

“body” block

Project 10

index.html

```
{% extends "layout.html" %}

{% block heading %}
        This is the Index Page
{% endblock %}

{% block body %}
        <a href="{{ url_for('hello') }}">
        Click here to go to hello page</a>
{% endblock %}
```

Project 10

hello.html

```
{% extends "layout.html" %}

{% block heading %}
        This is the Hello Page
{% endblock %}

{% block body %}
        <a href="{{ url_for('index') }}">
        Click here to go to index page</a>
{% endblock %}
```

Project 11a

Interacting with HTML Form with “POST” method

Project 11a (modify from Project 4a)

server.py

```
from flask import Flask, render_template, request
app = Flask(__name__)
@app.route('/')      # 127.0.0.1:5000/
def index():
    return render_template("index.html")

@app.route('/form', methods=["POST"])
def form():
        username = request.form.get("name")
        return str(username)

app.run(debug=True, port=5000)
```

Project 11a

index.html

```
{% extends "layout.html" %}
{% block heading %}
        This is the Index Page
{% endblock %}

{% block body %}
<form action="{{url_for('form')}}" method="POST">
    Enter the Username :
    <input type="text" name="name" >
    <button>Submit</button>
</form>

{% endblock %}
```

Project 11b

Interacting with HTML Form with "POST" and "GET" methods

Project 11b – part 1

server.py

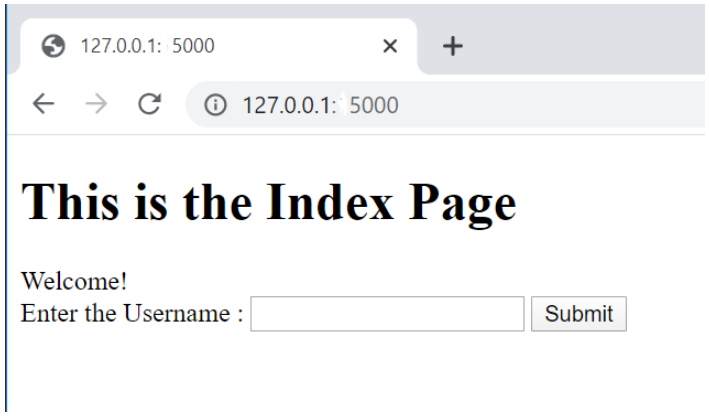
```
from flask import Flask, render_template, request
app = Flask(__name__)
@app.route('/')      # 127.0.0.1:5000/
def index():
    return render_template("index.html")

@app.route('/form', methods=["POST"])
def form():
        username = request.form.get("name")
        return render_template("hello.html",
                                name=username)
app.run(debug=True, port=5000)
```

Project 11b

hello.html

```
{% extends "layout.html" %}

{% block heading %}
        This is the Hello Page
{% endblock %}

{% block body %}
        Hello, {{ display }} !
{% endblock %}
```

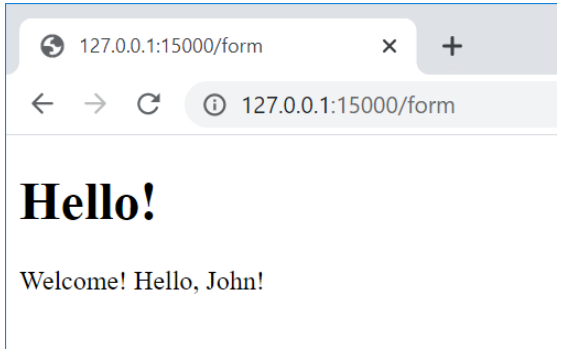
Test the form at the url : `127.0.0.1:5000/form`



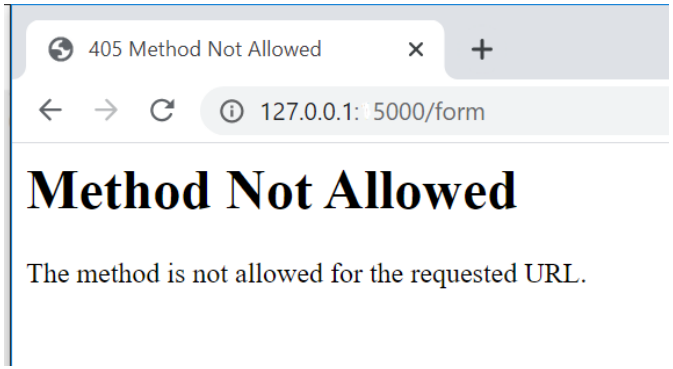
Using the “POST” method

When we click on the Submit button after typing the Username, the 'name/value' info is transmitted to the server using the “POST” method.

The server then render a webpage to display on the client browser.



What happen if we try to access the “form” route by using the url : `127.0.0.1:5000/form` ?



“Method Not Allowed”

This is because we are using the “POST” method.

We shall modify the `server.py` to use the “GET” method. (By not specifying the method, it will by default be using the “GET” method.)

Project 11b – part 2

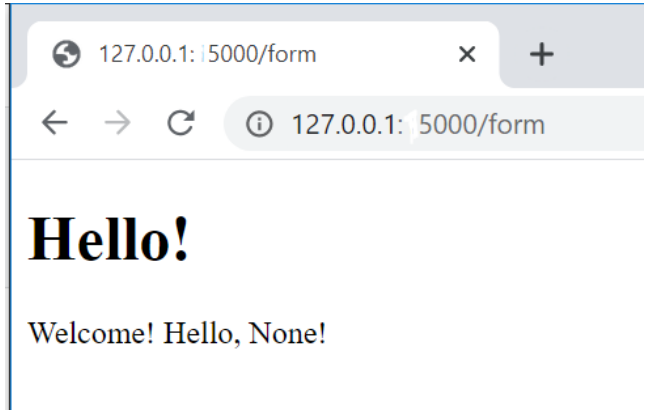
server.py

```
from flask import Flask, render_template, request
app = Flask(__name__)
@app.route('/')      # 127.0.0.1:5000/
def index():
    return render_template("index.html")

@app.route('/form')  # GET method by default
def form():
        username = request.form.get("name")
        return render_template("hello.html",
                                name=username)
app.run(debug=True, port=5000)
```

What happen if we try to access the “form” route now? (using the url : `127.0.0.1:5000/form`)

The `hello.html` will show

“Hello None !”



This is because there is no “username” provided.

We shall now modify the `server.py` again to accept both “GET” and “POST” methods.

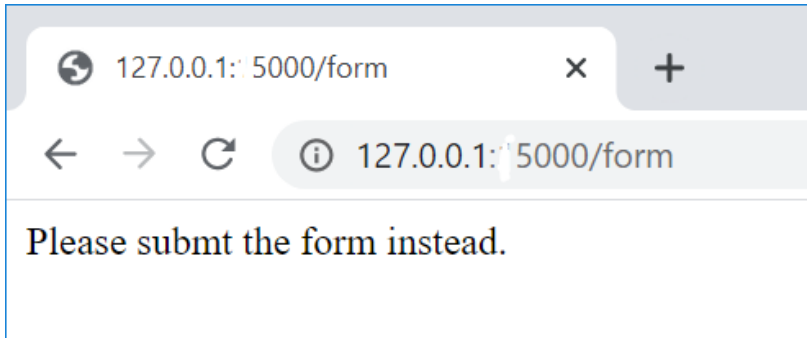
Project 11b – part 3

server.py

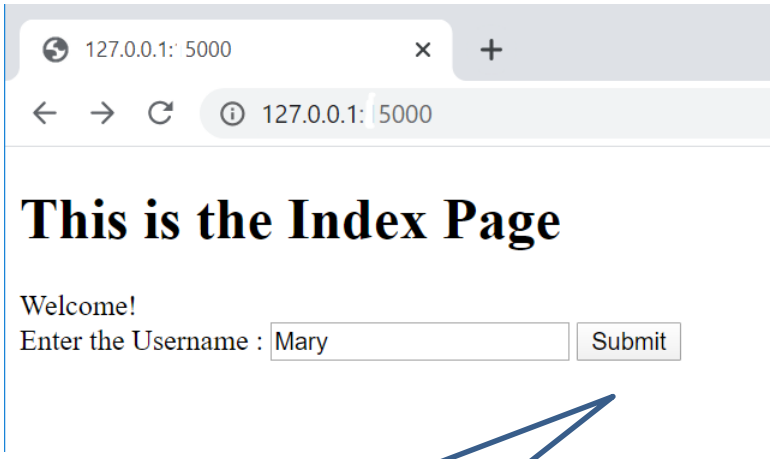
```
...

@app.route('/form', methods=["GET", "POST"])

def form():
    if request.method == "GET":
        return "Please submit the form instead."
    else:
        username = request.form.get("name")
        return render_template("hello.html",
                               name=username)

app.run(debug=True, port=5000)
```

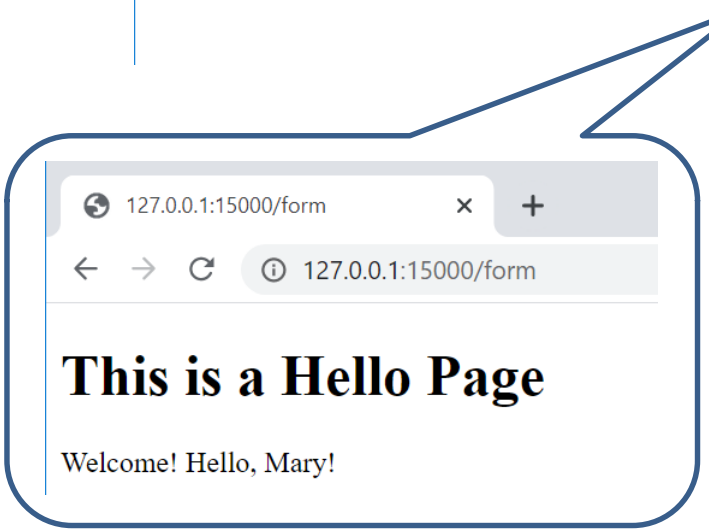
Now the server.py is able to accept both the "GET" and "POST" methods.

"GET" method



"POST" method





Project 12

Note-taking Application

Project 12 (Same as 4a)

server.py

```
from flask import Flask, render_template
app = Flask(__name__)
@app.route('/')      # 127.0.0.1:5000/
         # by default, this is a "GET" method
def index():
    return render_template("index.html")



app.run(debug=True, port=5000)
```

Project 12

server.py

```
from flask import Flask, render_template
app = Flask(__name__)
@app.route('/', methods=["GET"])    # 127.0.0.1:5000/
def index():
    return render_template("index.html")



app.run(debug=True, port=5000)
```

Project 12

index.html

```
{% extends "layout.html" %}
{% block heading %}
        Note-Taking Application
{% endblock %}

{% block body %}
<form action="{{url_for('index')}}" method="POST">
    Enter your note here :
    <input type="text" name="note" >
    <button>Add Note</button>
</form>
{% endblock %}
```

Project 12 (modify from Project 4a)

server.py

```
from flask import Flask, render_template
app = Flask(__name__)
@app.route('/', methods=["GET","POST"])
notes = []     # using a list to keep the notes
def index():
    if request.method == "POST":
        note = request.form.get("note")
        notes.append(note)
    return render_template("index.html")



app.run(debug=True, port=5000)
```

We managed to keep all the notes entered by the user into the list, but how can we display them?

Project 12

server.py

```
from flask import Flask, render_template
app = Flask(__name__)
@app.route('/', methods=["GET","POST"])
notes = []     # using a list to keep the notes
def index():
    if request.method == "POST":
        note = request.form.get("note")
        notes.append(note)
    return str(notes)

app.run(debug=True, port=5000)
```

Note : `notes` **is a list !**

Project 12

server.py

```
from flask import Flask, render_template
app = Flask(__name__)
@app.route('/', methods =["GET","POST"])
notes = []     # using a list to keep the notes
def index():
    if request.method == "POST":
          note = request.form.get("note")
          notes.append(note)
    return render_template("index.html",
                                    display=notes)
app.run(debug=True, port=5000)
```

Note : `display` **is a list !**

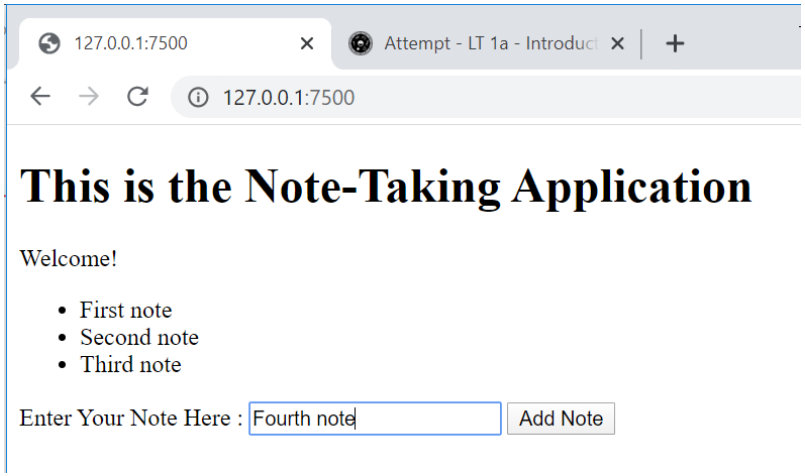
Project 12

index.html

```
...
{% block body %}
<ul>
{% for item in display %}
    <li>{{ item }} </li>
{% endfor %}
</ul>

<form action="{{url_for('index')}}" method="POST">
    Enter your note here :
    <input type="text" name="note" >
    <button>Add Note</button>
</form>
{% endblock %}
```

What happen if we close the browser after recording some notes?

Just restart the browser and access the server, the list of notes is still there and appear.



What happen if you shut the server?

All the notes will be gone!

Hence, we need a database to store all the notes.

The End

Back-End Web Applications (Part 4)