Sathya
Technologies

Django

Notes By

Naveen

Python With Naveen

Steps to Create a Project and App in Django

1. Open Command Prompt and in command prompt move to the project location.

2) To Create project : django-admin startproject project_name

3) Move into Project Directory : cd project_name

4) To create app : django-admin startapp app_name

5) Create a new Directory and name it as "templates".

6) inside the templates directory create html file

index.html

```
<html>

        <body>

                <style>

                        h1

                        {

                                background-color : red;

                                color : yellow;

                                font-size : 60px;

                                text-shadow : 5px 5px 5px green;

                        }
```

Python With Naveen

Python With Naveen

```
            </style>
            <h1> This is last Welcome to Django </h1>
        </body>
</html>
```

7) Open "settings.py" file from "project/project" directory and do the changes like

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'app8',  # adding our application name
]
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': ['templates'],   # add templates directory name
        'APP_DIRS': True,

        'OPTIONS': {

            'context_processors': [

                'django.template.context_processors.debug',

                'django.template.context_processors.request',

                'django.contrib.auth.context_processors.auth',

                'django.contrib.messages.context_processors.messages',

            ],

        },

    },

]
```

8) Open "views.py" file from "app" and define a function, this function must return the response.

views.py

```
from django.shortcuts import render

def showIndex(request):

    return render(request,"index.html")
```

9) Open "urls.py" file from "project/project" directory and call the path function.

Python With Naveen

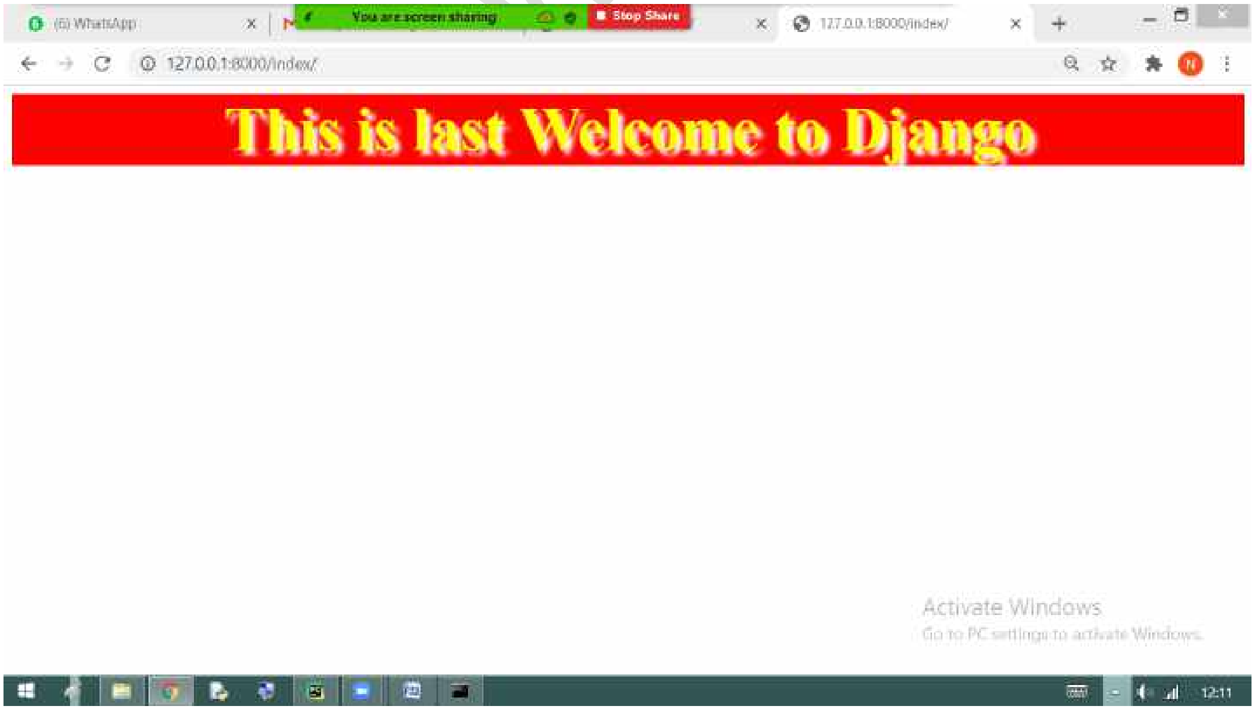
urls.py

```
from django.contrib import admin

from django.urls import path

from app8 import views

urlpatterns = [

    path('admin/', admin.site.urls),

    path('index/',views.showIndex)

]
```

10) Open command prompt and active into project location and run the server : python manage.py runderver

11) Open Browser and type the URL as : **http://127.0.0.1:8000/index**

With Naveen



This is last Welcome to Django

Template Language

A **template** is a text document, or a normal Python string, that is marked-up using the Django template language. A template can contain **block tags** or **variables**.

Block tags are surrounded by **"{%"** and **"%}"**.

Example template with block tags:

{% **if** is_logged_in %}Thanks for logging in!{% **else** %}Please log in.{% **endif** %}

Variables

Variables look like this: {{ variable }}.

When the template engine encounters a variable, it evaluates that variable and replaces it with the result.

Variable names consist of any combination of alphanumeric characters and the underscore ("_") but may not start with an underscore.

Templates

A template is simply a text file. It can generate any text-based format (HTML, XML, CSV, etc.).

A template contains **variables**, which get replaced with values when the template is evaluated, and **tags**, which control the logic of the template.

Python With Naveen

Python With Naveen

Comment

Ignores everything between {% comment %} and

{% endcomment %}.

csrf_token

This tag is used for CSRF protection, csrf stands for **Cross Site Request Forgeries.**

Boolean operators in if tag's

If tags may use **and, or** or **not** to test a number of variables or to negate a given variable:

Example 1

```
{% if athlete_list and coach_list %}
    Both athletes and coaches are available.
{% endif %}
```

Example 2

```
{% if not athlete_list %}
    There are no athletes.
{% endif %}
```

Example 3

```
{% if athlete_list or coach_list %}
```

```
    There are some athletes or some coaches.
{% endif %}
```

Example 4

```
{% if not athlete_list or coach_list %}
    There are no athletes or there are some coaches.
{% endif %}
```

Example 5

```
{% if athlete_list and not coach_list %}
    There are some athletes and absolutely no coaches.
{% endif %}
```

Use of both **and** and **or** clauses within the same tag is allowed, with and having higher precedence than or .

Example 6

```
{% if athlete_list and coach_list or cheerleader_list %}
```

will be interpreted like:

```
if (athlete_list and coach_list) or cheerleader_list
```

operators in if tag's

If tags may also use the operators like ==, !=, <, >, <=, >=, in, not in, is, and is not .

Python With Naveen

== operator Example:

```
{% if somevar == "x" %}
  This appears if variable somevar equals the string "x"
{% endif %}
```

!= operator Example:

```
{% if somevar != "x" %}
  This appears if variable somevar does not equal the string "x",
  or if somevar is not found in the context
{% endif %}
```

< operator Less than. Example:

```
{% if somevar < 100 %}
  This appears if variable somevar is less than 100.
{% endif %}
```

> operator Greater than. Example:

```
{% if somevar > 0 %}
  This appears if variable somevar is greater than 0.
{% endif %}
```

<= operator Less than or equal to. Example:

```
{% if somevar <= 100 %}
```

Python With Naveen

This appears if variable somevar is less than 100 or equal to 100.

{% endif %}

>= operator Greater than or equal to. Example:

{% if somevar >= 1 %}

This appears if variable somevar is greater than 1 or equal to 1.

{% endif %}

in operator

Contained within. This operator is supported by many Python containers to test whether the given value is in the container.

Example : {% if "bc" in "abcdef" %}

This appears since "bc" is a substring of "abcdef"

{% endif %}

Example :

{% if "hello" in greetings %}

If greetings is a list or set, one element of which is the string

"hello", this will appear.

{% endif %}

Python With Naveen

Example :

{% if user in users %}

If users is a QuerySet, this will appear if user is an

instance that belongs to the QuerySet.

{% endif %}

not in operator

Not contained within. This is the negation of the in operator.

is operator

Object identity. Tests if two values are the same object.

Example:

{% if somevar is True %}

This appears if and only if somevar is True.

{% endif %}

Example :

{% if somevar is None %}

This appears if somevar is None, or if somevar is not found in the context. {% endif %}

is not operator

Negated object identity. Tests if two values are not the same object. This is the negation of the is operator.

Python With Naveen

Example:

```
{% if somevar is not True %}
  This appears if somevar is not True, or if somevar is not found in the
  context.
{% endif %}
```

Example :

```
{% if somevar is not None %}
  This appears if and only if somevar is not None.
{% endif %}
```

load

Loads a custom template tag set.

```
{% load somelibrary package.otherlibrary %}
```

You can also selectively load individual filters or tags from a library, using the from argument.

```
{% load foo bar from somelibrary %}
```

include

Loads a template and renders it with the current context. This is a way of “including” other templates within a template.

The template name can either be a variable or a hard-coded (quoted) string, in either single or double quotes.

Python With Naveen

This example includes the contents of the template "foo/bar.html":

{% include "foo/bar.html" %}

This example includes the contents of the template whose name is contained in the variable template_name:

{% include template_name %}

now

Displays the current date and/or time, using a format according to the given string

Example:

{% now "jS F Y H:i" %}

{% **now "SHORT_DATETIME_FORMAT"** %}<**br**>
{% **now "SHORT_DATE_FORMAT"** %}<**br**>
{% **now "DATETIME_FORMAT"** %}<**br**>
{% **now "DATE_FORMAT"** %}

Filters

You can modify variables for display by using **filters**.

Filters look like this: **{{ name|lower }}**.

This displays the value of the **{{ name }}** variable after being filtered through the **lower** filter, which converts text to lowercase. Use a pipe (**|**) to apply a filter.

Python With Naveen

Some filters take arguments. A filter argument looks like this: **{{bio|truncatewords:30 }}**. This will display the first 30 words of the **bio** variable.

Filter arguments that contain spaces must be quoted; for example, to join a list with commas and spaced you'd use **{{ list|join:", " }}**.

Built-in filter reference

1) add : Adds the argument to the value.

For example: {{ value|add:"2" }}

If **value** is **4**, then the output will be **6** **and** **first** is **[1, 2, 3]** and **second** is **[4, 5, 6]**, then the output will be **[1, 2, 3, 4, 5, 6]**.

2) addslashes : Adds slashes before quotes. Useful for escaping strings in CSV, for example.

For example: {{ value|addslashes }}

If **value** is **"This is Naveen's",** the output will be **"This is Naveen\'s**.

3) capfirst : Capitalizes the first character of the value. If the first character is not a letter, this filter has no effect.

For example: {{ value|capfirst }}

If **value** is **"naveen"**, the output will be **"Naveen"**.

4) center : Center the value in a field of a given width.

For example: "{{ value|center:"15" }}"

If **value** is **"Naveen"**, the output will be **"   Naveen   "**.

Python With Naveen

5) cut : Removes all values of arg from the given string.

For example: {{ value|cut:" " }}

If **value** is **"String with spaces"**, the output will be **"Stringwithspaces"**.

6) default : If value evaluates to **False**, uses the given default. Otherwise, uses the value.

For example: {{ value|default:"nothing" }}

If **value** is **""** (the empty string), the output will be **nothing**.

7) default_if_none : If (and only if) value is **None**, uses the given default. Otherwise, uses the value.

Note that if an empty string is given, the default value will *not* be used. Use the **default** filter if you want to fallback for empty strings.

For example: {{ value|default_if_none:"nothing" }}

If **value** is **None**, the output will be the string **"nothing"**.

8) dictsort: Takes a list of dictionaries and returns that list sorted by the key given in the argument.

For example: {{ value|dictsort:"name" }}

If **value** is : [

    {'name': 'zed', 'age': 19},

    {'name': 'amy', 'age': 22},

Python With Naveen

```
    {'name': 'joe', 'age': 31},
]
```

then the output would be:

```
[
    {'name': 'amy', 'age': 22},
    {'name': 'joe', 'age': 31},
    {'name': 'zed', 'age': 19},]
```

9) dictsortreversed : Takes a list of dictionaries and returns that list sorted in reverse order by the key given in the argument. This works exactly the same as the above filter, but the returned value will be in reverse order.

10) divisibleby : Returns **True** if the value is divisible by the argument.

For example: {{ value|divisibleby:"3" }}

If **value** is **21**, the output would be **True**.

11) filesizeformat : Formats the value like a ‘human-readable’ file size (i.e. **'13 KB'**, **'4.1MB'**, **'102 bytes'**, etc).

For example: {{ value|filesizeformat }}

If **value** is 123456789, the output would be **117.7 MB**.

12) first : Returns the first item in a list.

For example: {{ value|first }}

Python With Naveen

If **value** is the list **['a', 'b', 'c']**, the output will be **'a'**.

13) floatformat : When used without an argument, rounds a floating-point number to one decimal place – but only if there’s a decimal part to be displayed. For example:

value	Template	Output
34.23234	{{ value|floatformat }}	34.2
34.00000	{{ value|floatformat }}	34
34.26000	{{ value|floatformat }}	34.3

If used with a numeric integer argument, **floatformat** rounds a number to that many decimal places.

For example:

value	Template	Output
34.23234	{{ value|floatformat:3}}	34.232
34.00000	{{ value|floatformat:3}}	34.000
34.26000	{{ value|floatformat:3}}	34.260

Particularly useful is passing 0 (zero) as the argument which will round the float to the nearest integer.

value	Template	Output
34.23234	{{ value|floatformat:"0" }}	34

Python With Naveen

value	Template	Output
34.00000	**{{ value|floatformat:"0" }}**	**34**
39.56000	**{{ value|floatformat:"0" }}**	**40**

If the argument passed to **floatformat** is negative, it will round a number to that many decimal places – but only if there's a decimal part to be displayed. For example:

value	Template	Output
34.23234	**{{ value|floatformat:"-3" }}**	**34.232**
34.00000	**{{ value|floatformat:"-3" }}**	**34**
34.26000	**{{ value|floatformat:"-3" }}**	**34.260**

Using **floatformat** with no argument is equivalent to using **floatformat** with an argument of **-1**.

14) join : Joins a list with a string, like Python's **str.join(list)**

For example: {{ value|join:" // " }}

If **value** is the list **['a', 'b', 'c']**, the output will be the string **"a// b // c"**.

15) get_digit : Given a whole number, returns the requested digit, where 1 is the right-most digit, 2 is the second-right-most digit, etc. Returns the original value for invalid input (if input or argument is not an integer, or if argument is less than 1). Otherwise, output is always an integer.

For example: {{ value|get_digit:"2" }}

Python With Naveen

If **value** is **123456789**, the output will be **8**.

16) last: Returns the last item in a list.

For example: {{ value|last }}

If **value** is the list **['a', 'b', 'c', 'd']**, the output will be the string **"d"**.

17) length: Returns the length of the value. This works for both strings and lists.

For example: {{ value|length }}

If **value** is **['a', 'b', 'c', 'd']**, the output will be **4**.

18) length_is : Returns **True** if the value's length is the argument, or **False** otherwise.

For example: {{ value|length_is:"4" }}

If **value** is **['a', 'b', 'c', 'd']**, the output will be **True**.

19) linenumbers : Displays text with line numbers.

For example: {{ value|linenumbers }}

If **value** is :

one

two

three

the output will be:

1. one

Python With Naveen

2. two

3. three

20) lower: Converts a string into all lowercase.

For example: {{ value|lower }}

If **value** is **This is Naveen**, the output will be **this is naveen**.

21) make_list : Returns the value turned into a list. For a string, it's a list of characters. For an integer, the argument is cast into an unicode string before creating a list.

For example: {{ value|make_list }}

If **value** is the string **"Naveen"**, the output would be the list **['N', 'a', 'v', 'e', 'e', 'n']**.

If **value** is **123**, the output will be the list **['1', '2', '3']**.

22) random : Returns a random item from the given list.

For example: {{ value|random }}

If **value** is the list **['a', 'b', 'c', 'd']**, the output could be **"b"**.

23) slice : Returns a slice of the list.

Example: {{ some_list|slice:":2" }}

If **some_list** is **['a', 'b', 'c']**, the output will be **['a', 'b']**.

Python With Naveen

25) time : Formats a time according to the given format.

For example: {{ value|time:"H:i" }}

If **value** is equivalent to **datetime.datetime.now()**, the output will be the string **"01:23"**.

26) title : Converts a string into title case by making words start with an uppercase character and the remaining characters lowercase. This tag makes no effort to keep "trivial words" in lowercase.

For example: {{ value|title }}

If **value** is **"my FIRST post"**, the output will be **"My First Post"**.

28) truncatewords : Truncates a string after a certain number of words.
Argument: Number of words to truncate after

For example: {{ value|truncatewords:2 }}

If **value** is **"This is Naveen From "**, the output will be **"This is..."**.

Newlines within the string will be removed.

29) upper : Converts a string into all uppercase.

For example: {{ value|upper }}

If **value** is **"This is Naveen"**, the output will be **"THIS IS NAVEEN"**.

30) wordcount : Returns the number of words.

For example: {{ value|wordcount }}

If **value** is **"This is Naveen Kumar"**, the output will be **4**.

Python With Naveen

31) wordwrap: Wraps words at specified line length.

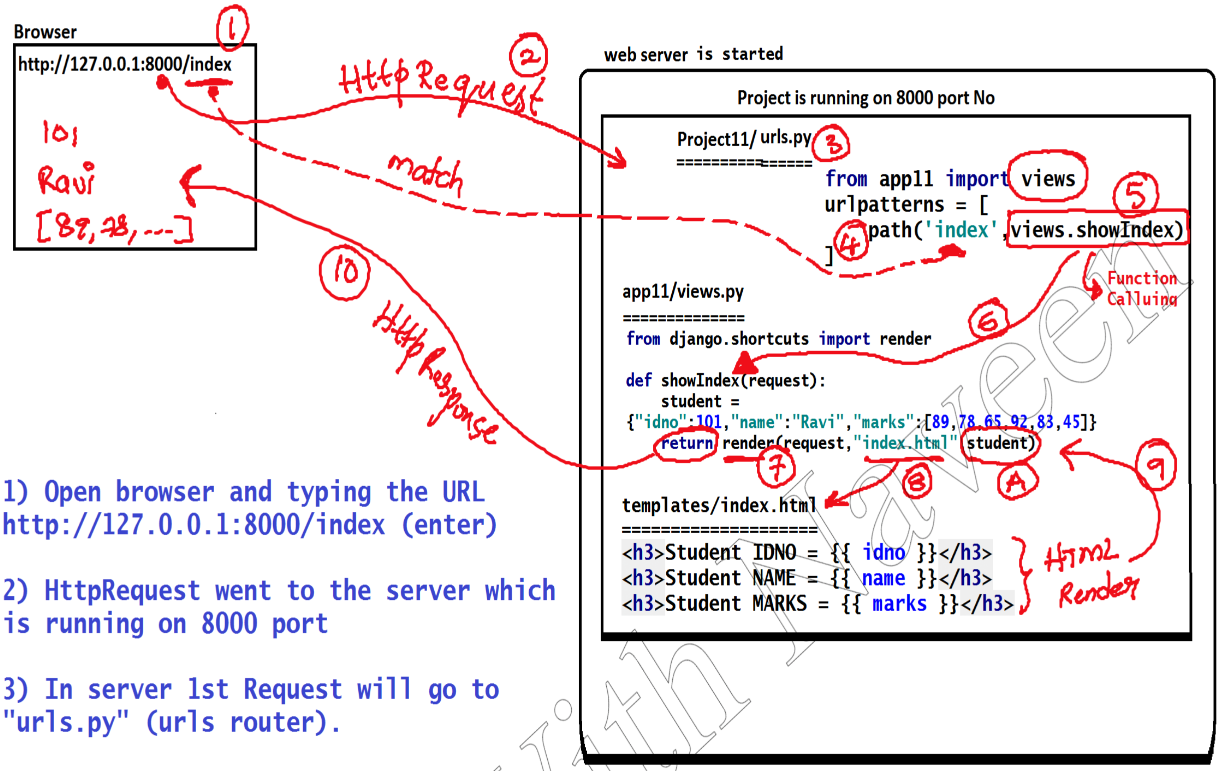
Argument: number of characters at which to wrap the text

For example: {{ value|wordwrap:5 }}

Python With Naveen

Sending Data from views.py to Template (html)

and the Work Flow



Project11/ urls.py
===============

```
from app11 import views
urlpatterns = [
  path('index',views.showIndex)
]
```

app11/views.py
==============

```
from django.shortcuts import render

def showIndex(request):
    student =
{"idno":101,"name":"Ravi","marks":[89,78,65,92,83,45]}
    return render(request,"index.html" student)
```

templates/index.html
====================

```
<h3>Student IDNO = {{ idno }}</h3>
<h3>Student NAME = {{ name }}</h3>
<h3>Student MARKS = {{ marks }}</h3>
```

1) Open browser and typing the URL
http://127.0.0.1:8000/index (enter)

2) HttpRequest went to the server which is running on 8000 port

3) In server 1st Request will go to "urls.py" (urls router).

4) from the "path" function map the given "URL" ----"index/"

5) When the "URL" is matched call the functon from "views.py"

6) the given function will execute from "views.py" file function name is showIndex

7) the showIndex function is calling "render" function with "template name" and passing some data

8) the html file will render(execute,show,present,...) the given data and return back the showIndex function

9) The "render" function will return "HttpResponse" to the Browser

10) Present the template(HTML file) in browser

browser will understand : HTML + CSS + Javascript

Python With Naveen

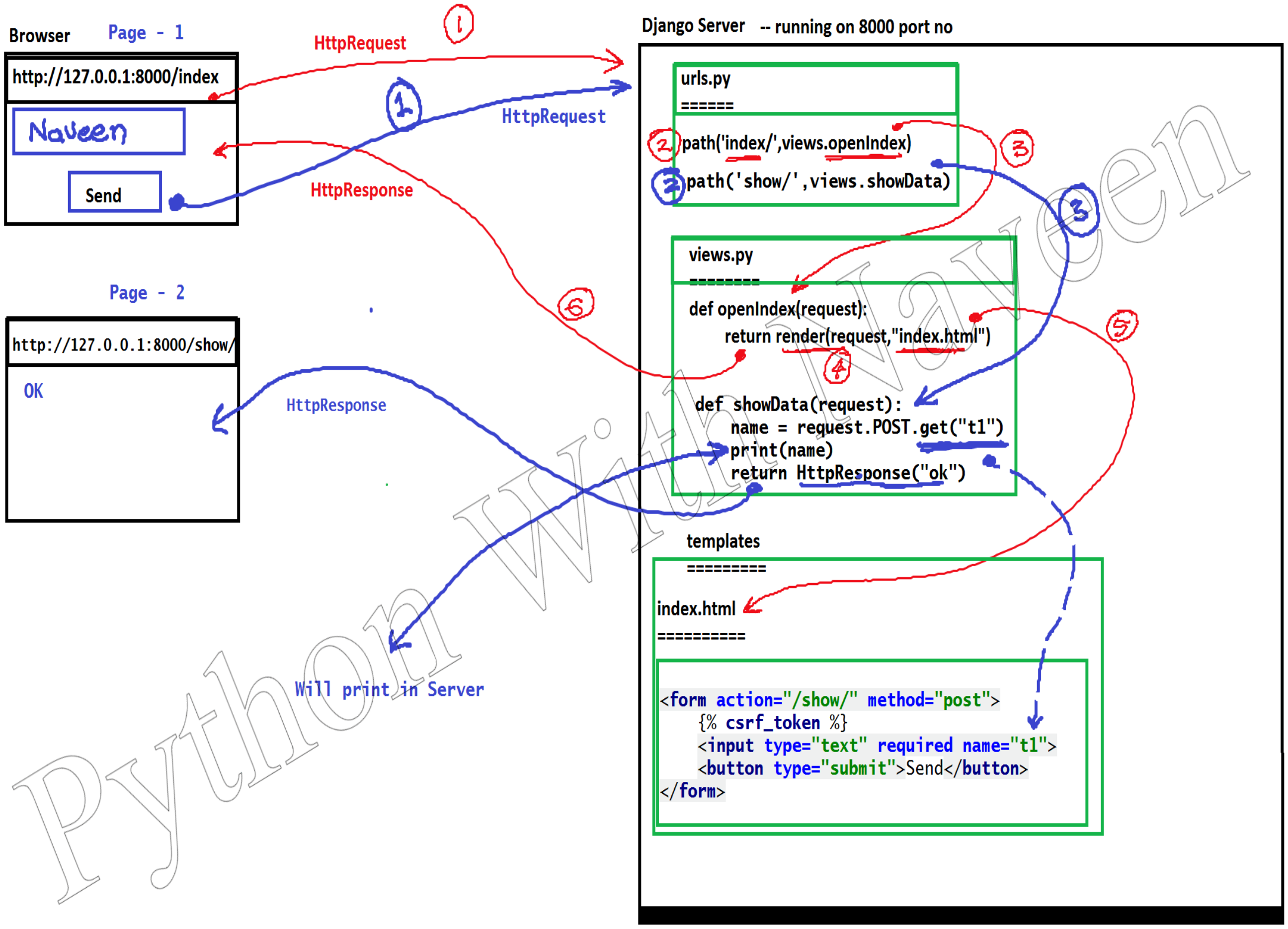
Sending Data from Template(HTML) to Views.py and the Work Flow

We can send request in 3 ways

1) Url

2) Hyperlink

3) Button



```
urls.py
======
path('index/',views.openIndex)
path('show/',views.showData)
```

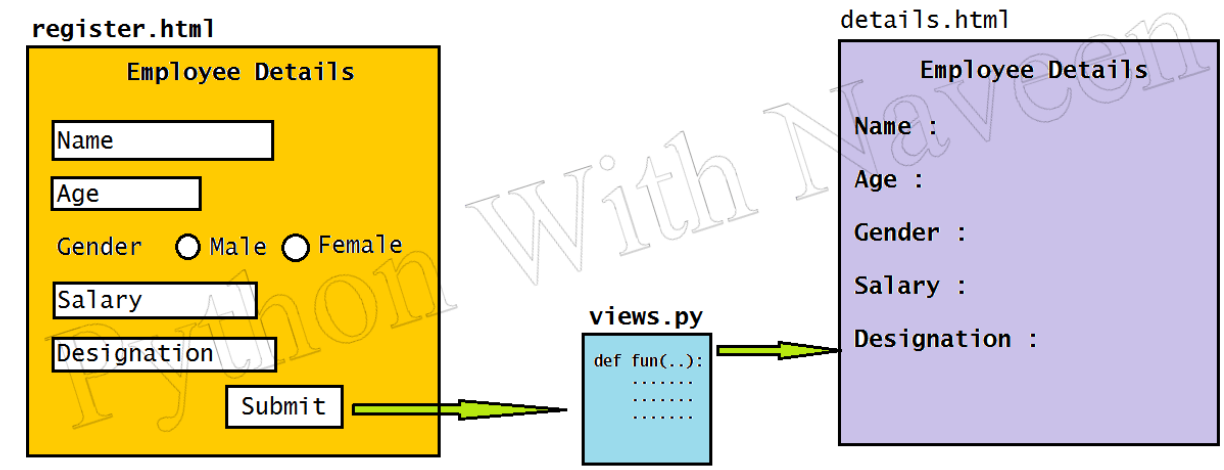
```
views.py
========
def openIndex(request):
    return render(request,"index.html")

def showData(request):
    name = request.POST.get("t1")
    print(name)
    return HttpResponse("ok")
```

```
templates
=========
index.html
==========
<form action="/show/" method="post">
    {% csrf_token %}
    <input type="text" required name="t1">
    <button type="submit">Send</button>
</form>
```

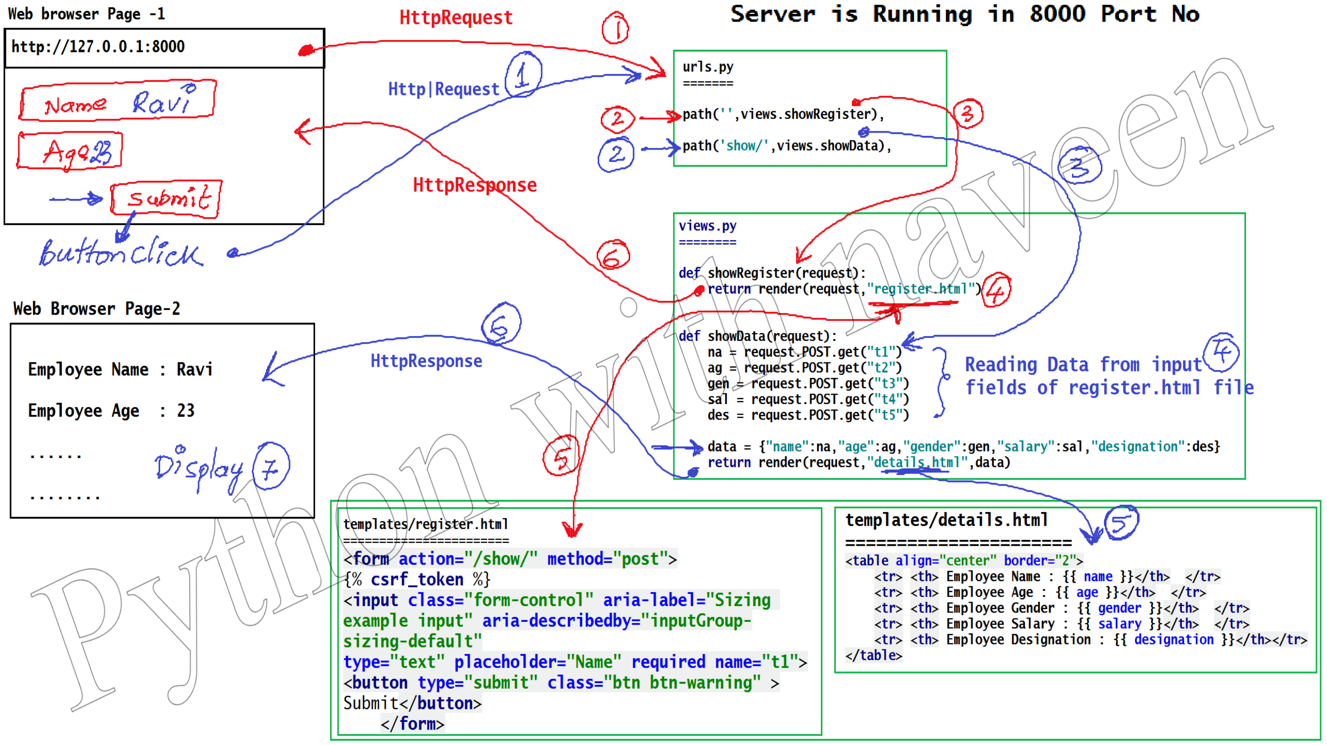
Python With Naveen

Sending Data from Template to Views and Views to Template and the Work Flow

Application Requirement
=======================



Example



Server is Running in 8000 Port No

```
urls.py
=======
path('',views.showRegister),
path('show/',views.showData),
```

```
views.py
========
def showRegister(request):
    return render(request,"register.html")

def showData(request):
    na = request.POST.get("t1")
    ag = request.POST.get("t2")
    gen = request.POST.get("t3")
    sal = request.POST.get("t4")
    des = request.POST.get("t5")

    data = {"name":na,"age":ag,"gender":gen,"salary":sal,"designation":des}
    return render(request,"details.html",data)
```

Reading Data from input fields of register.html file

```
templates/register.html
=====================
<form action="/show/" method="post">
{% csrf_token %}
<input class="form-control" aria-label="Sizing
example input" aria-describedby="inputGroup-
sizing-default"
type="text" placeholder="Name" required name="t1">
<button type="submit" class="btn btn-warning" >
Submit</button>
     </form>
```

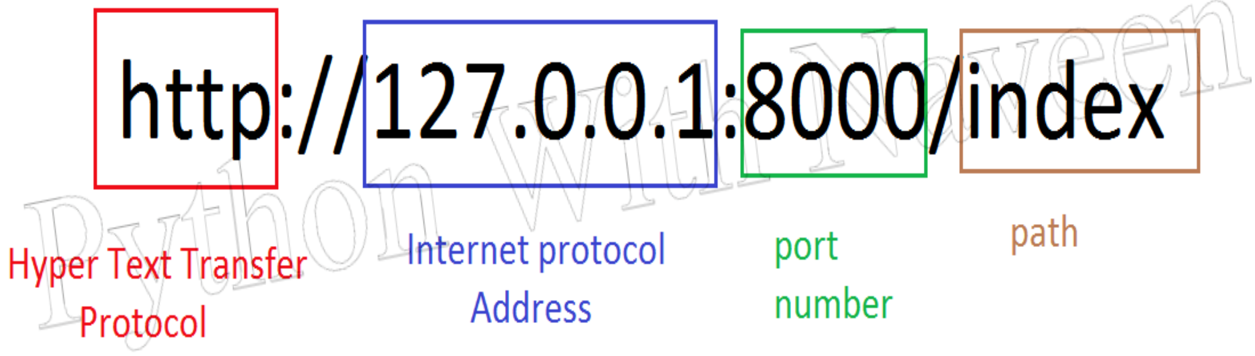
```
templates/details.html
======================
<table align="center" border="2">
    <tr> <th> Employee Name : {{ name }}</th>   </tr>
    <tr> <th> Employee Age : {{ age }}</th>   </tr>
    <tr> <th> Employee Gender : {{ gender }}</th>   </tr>
    <tr> <th> Employee Salary : {{ salary }}</th>   </tr>
    <tr> <th> Employee Designation : {{ designation }}</th></tr>
</table>
```

Python With Naveen

URL : Uniform Resource Locator.

Url Contains the address that links to a resource such as a HTML page.

Ex: http://127.0.0.1:8000/index --- it will open index.html page.


http://127.0.0.1:8000/index
Hyper Text Transfer Protocol
Internet protocol Address
port number
path

IP Address : Internet Protocol Address is used to identify a machine. Like the machine may be a Desktop, a laptop, a Mobile, a Swipe Machine,... etc.,.

What is Server ?

Server is nothing but a computer which is installed with server software.

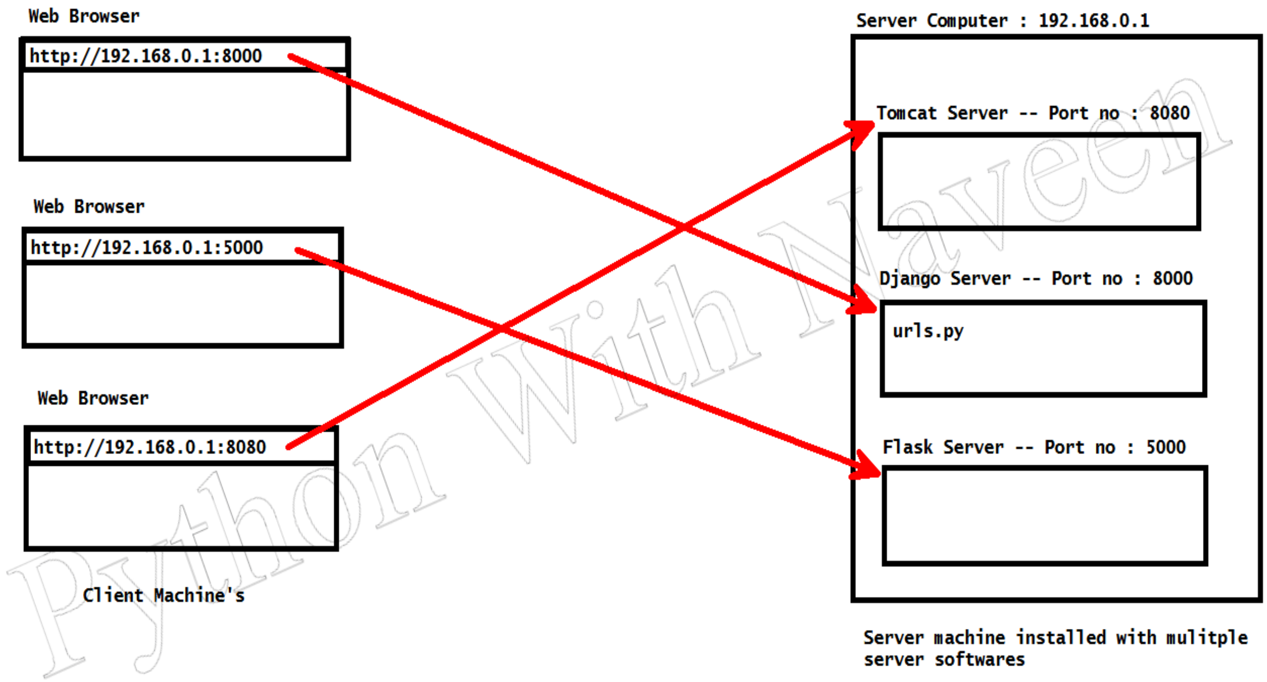
The server computer will accept a request and will give response.

Note: In one computer we can install any number of server software's.

Port No : Port No is used to identify a server software which is running in server machine.

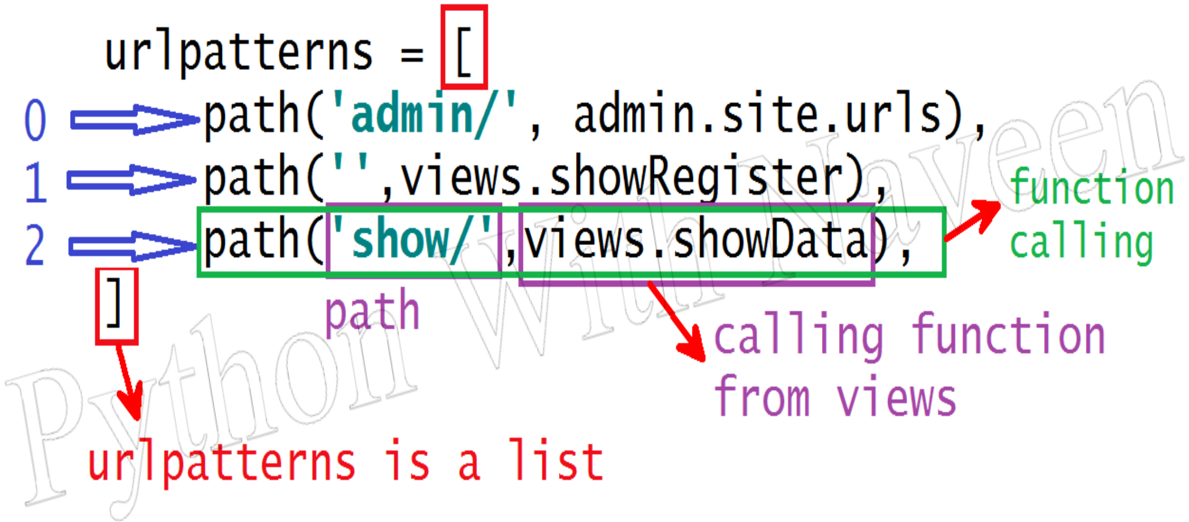
Port no must be unique.

Python With Naveen


Web Browser
http://192.168.0.1:8000
Web Browser
http://192.168.0.1:5000
Web Browser
http://192.168.0.1:8080
Client Machine's
Server Computer : 192.168.0.1
Tomcat Server -- Port no : 8080
Django Server -- Port no : 8000
urls.py
Flask Server -- Port no : 5000
Server machine installed with mulitple server softwares

Python With Naveen

Path : Path is used to identify a Template (HTML), path must be unique within one application.



```
urlpatterns = [
0 path('admin/', admin.site.urls),
1 path('',views.showRegister),
2 path('show/',views.showData),
]
```

Views

The **View** is used to execute the business logic and interact with model to carry data and renders template.

In django views are 2 types

1) Function based view

2) Class based view

Function Based views

- First, we import the class **HttpResponse** from the **django.http** module.
- Next, we define a function called **show**.
- This is the view function. Each view function takes an **HttpRequest** object as its **first parameter**, which is typically named **request**.

Note that the name of the view function doesn't matter; it doesn't have to be named in a certain way in order for Django to recognize it. We're calling it **show** here, because that name clearly indicates what it does.

- The view returns an **HttpResponse** object that contains the generated response.
- Each view function is responsible for returning an **HttpResponse** object.
- Every function must map within urls.py

Python With Naveen

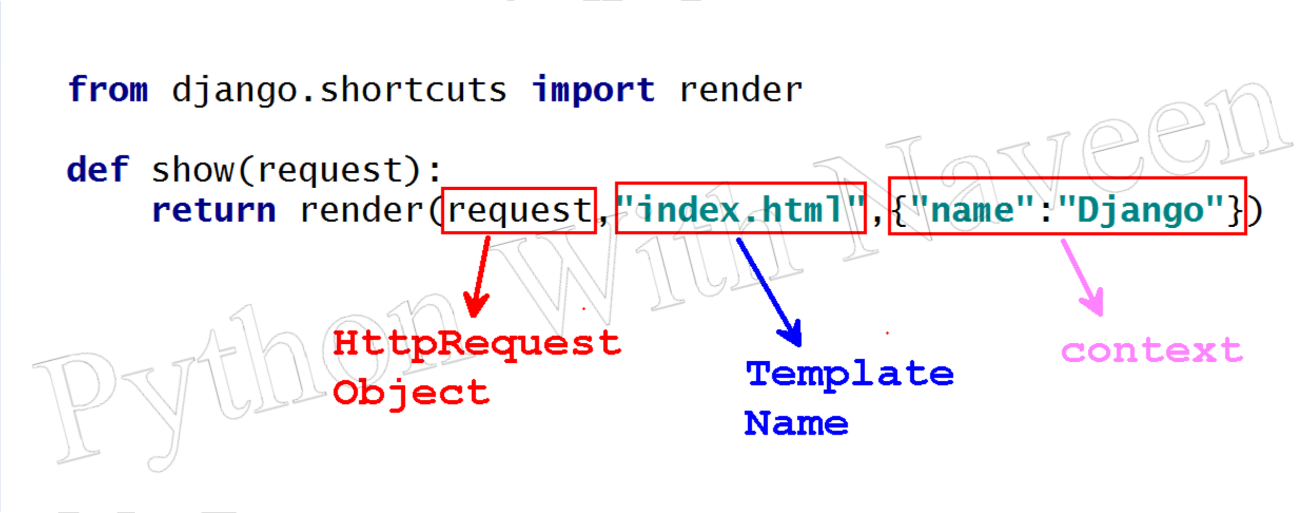
Example

```
from django.http import HttpResponse

def show(request):
    return HttpResponse("Welcome to Python
With Naveen")
```

render

import the render function from django.shortcuts package.

The render function will render(present) the given template and it will return **HttpResponse** Object.



```
from django.shortcuts import render

def show(request):
    return render(request,"index.html",{"name":"Django"})
```

Python With Naveen

Sending Data in URL

1) http://127.0.0.1:8000/index/Naveen/

2) urls.py

```
from app16 import views
urlpatterns = [
    path('index/<str:name>/',views.show)
]
```

3) views.py

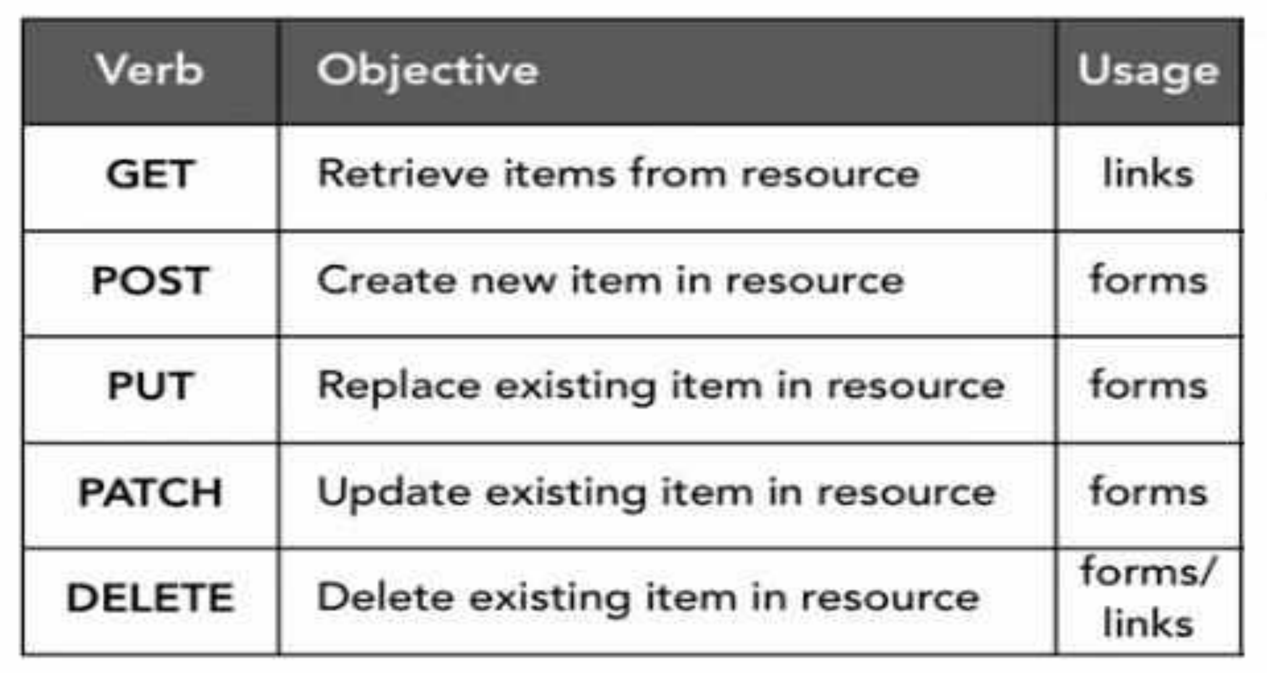
```
from django.shortcuts import render

def show(request,name):
    return render(request,"index.html",
{"name":name})
```

4) templates/index.html

```
<html>
    <body>
        <h1>Welcome Mr/Miss : {{ name }}</h1>
    </body>
</html>
```

HTTP request methods

HTTP defines a set of **request methods** to indicate the desired action to be performed for a given resource.



Verb	Objective	Usage
GET	Retrieve items from resource	links
POST	Create new item in resource	forms
PUT	Replace existing item in resource	forms
PATCH	Update existing item in resource	forms
DELETE	Delete existing item in resource	forms/ links

We can send request in **3** ways **1) URL 2) Hyperlink 3) Button**

By default **"URL"** and **"Hyperlink"** will send **"GET"** http request method.

Using **Button** we can send **"GET"** or we can send **"POST"** http request method.

Note : The HTML **<form>** element will take **"GET"** method by default, for **"POST"** method we need to write method attribute.

Python

Example on GET

urls.py

```
path('',views.showIndex),

path('show/',views.show),
```

views.py

```
from django.http import HttpResponse
from django.shortcuts import render

def showIndex(request):
    return render(request,"index.html")


def show(request):
    username = request.GET.get("t1")
    password = request.GET.get("t2")
    return HttpResponse("OK")
```

templates/index.html

```
<form action="/show/" method="get">
    <table align="center" border="2"
bgcolor="#add8e6">
<tr><th>Login with Get Request</th></tr>
<tr><th><input type="text"
placeholder="Username" name="t1"></th></tr>
<tr><th><input type="password"
```

```
placeholder="Password" name="t2"></th></tr>
<tr><th><button
type="submit">Login</button> </th></tr>
    </table>
</form>
```

Example on POST

urls.py

```
path('',views.showIndex),

path('display/',views.display),
```

views.py

```
from django.http import HttpResponse
from django.shortcuts import render

def showIndex(request):
    return render(request,"index.html")


def display(request):
    username = request.POST.get("p1")
    password = request.POST.get("p2")
    return HttpResponse("OK")
```

With Naveen

templates/index.html

```
<form action="/display/" method="post">
     {% csrf_token %}
    <table align="center" border="2"
bgcolor="#7fffd4">
        <tr><th>Login with POST
Request</th></tr>
        <tr><th><input type="text"
placeholder="Username" name="p1"></th></tr>
        <tr><th><input type="password"
placeholder="Password" name="p2"></th></tr>
        <tr><th><button
type="submit">Login</button> </th></tr>
    </table>
</form>
```

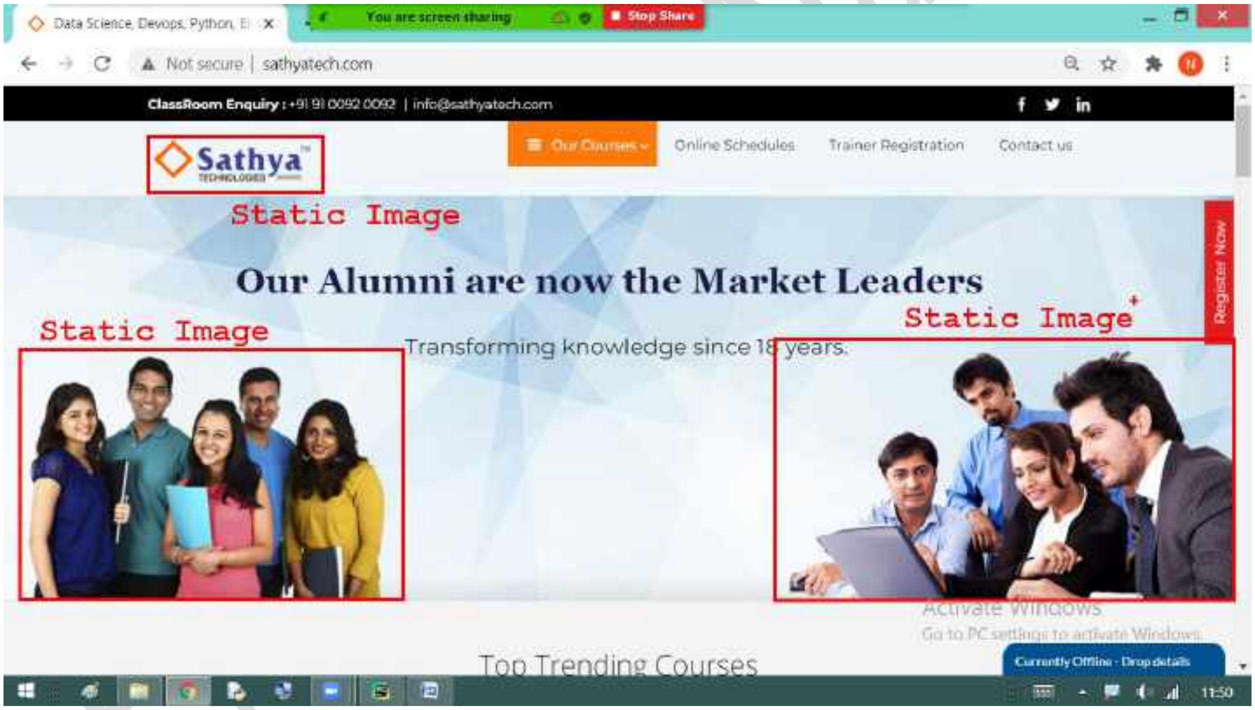
GET	POST
1) In case of Get request, only **limited amount of data** can be sent because data is sent in header.	In case of Post request, **large amount of data** can be sent because data is sent in body.
2) Get request is **not secured** because data is exposed in URL bar.	Post request is **secured** because data is not exposed in URL bar.

3) Get request **can be bookmarked.**	Post request **cannot be bookmarked.**

Static Files Handling in Django Application

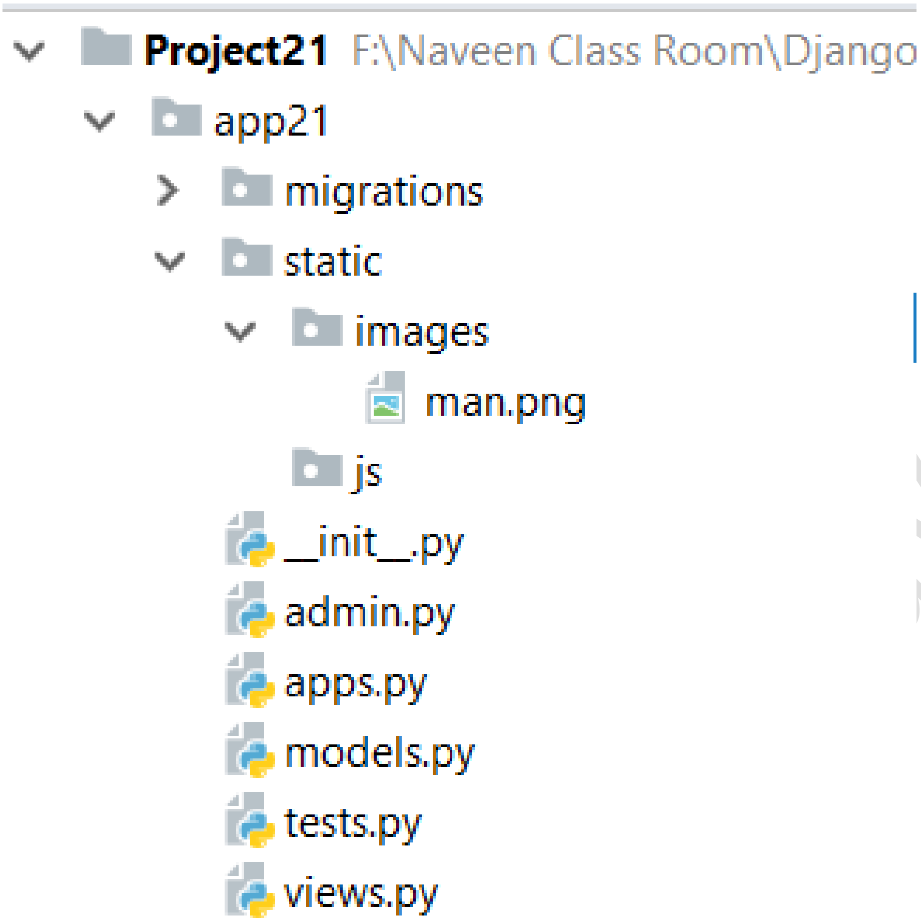
Static files means which will not change like images, CSS, JavaScript, audio, video, pdf files.

Example on Images


Data Science, Devops, Python, E
You are screen sharing
Stop Share
Not secure | sathyatech.com
ClassRoom Enquiry : +91 91 0092 0092 | info@sathyatech.com
Sathya TECHNOLOGIES
Our Courses
Online Schedules
Trainer Registration
Contact us
Static Image
Our Alumni are now the Market Leaders
Static Image
Static Image
Transforming knowledge since 18 years.
Register Now
Activate Windows
Go to PC settings to activate Windows.
Top Trending Courses
Currently Offline - Drop details
11:50

1) In your app create a **"static"** folder (Directory)

2) If you need you can create a sub folder for images and save all images into that folder.



```
Project21  F:\Naveen Class Room\Django
    app21
        migrations
        static
            images
                man.png
            js
        __init__.py
        admin.py
        apps.py
        models.py
        tests.py
        views.py
```

3) In HTML file in line no 1 load the static in block tag

Example

```
{% load static %}
```

4) In HTML use the Image

Example

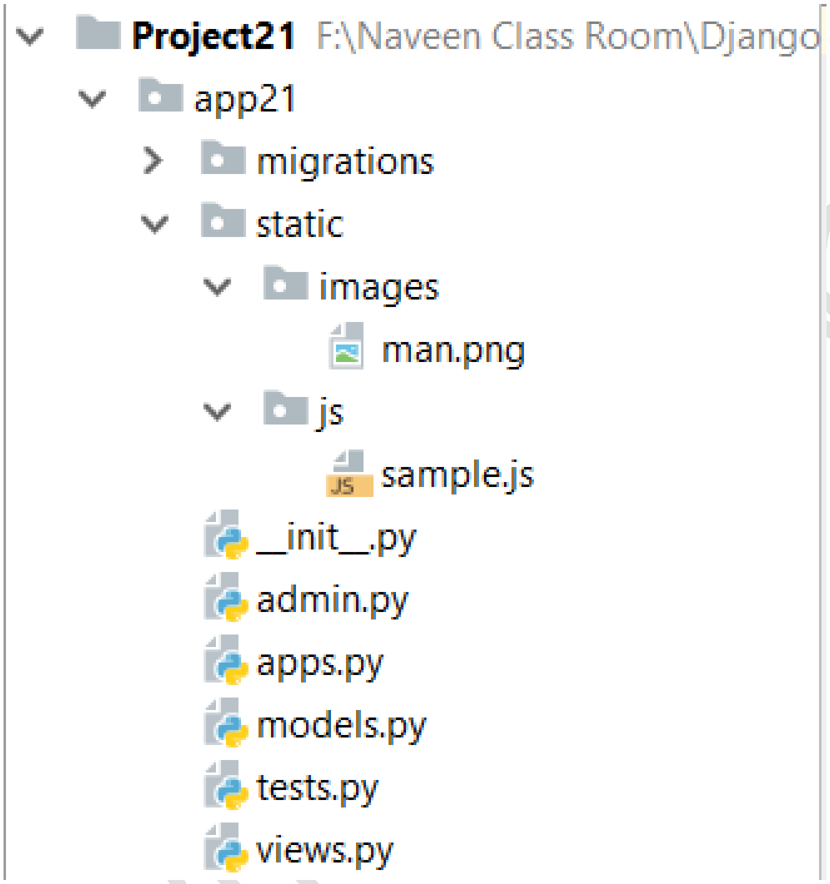
```
<img src="{% static 'images/man.png' %}">
```

Python with Naveen

Example on JavaScript

1) In your app create a **"static"** folder (Directory)

2) If you need you can create a sub folder for JavaScript files and save all JavaScript files into that folder.



```
Project21  F:\Naveen Class Room\Django
    app21
        migrations
        static
            images
                man.png
            js
                sample.js
        __init__.py
        admin.py
        apps.py
        models.py
        tests.py
        views.py
```

3) Define a JavaScript function in "sample.js" file

Example

```
function display()
{
    alert("JavaScript Example in Django")
}
```

4) Call the JavaScript Function in HTML (Template)

5) In HTML file in line no 1 load the static in block tag

Example

```
{% load static %}
```

6) In HTML use the javascript function.

Example

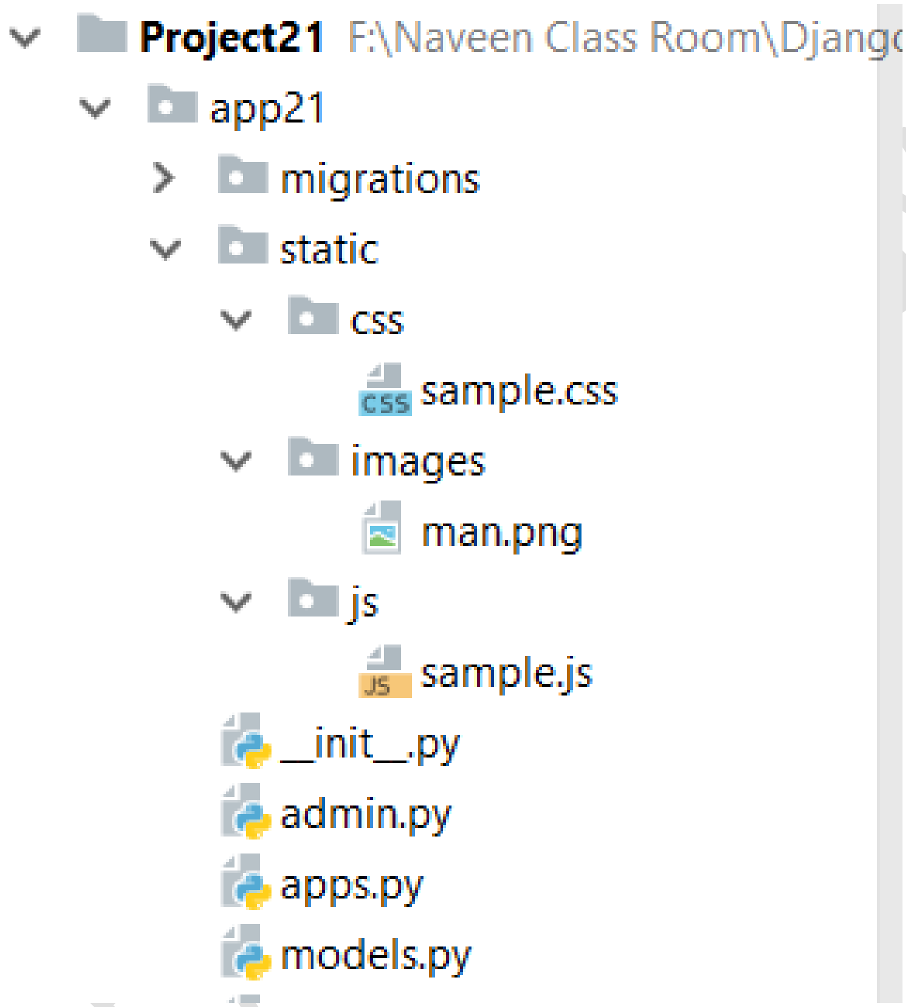
```
<head>
<script src="{% static 'js/sample.js' %}">
</script>
</head>
<body>
<button type="submit" onclick="display()">Click
Me </button>
</body>
```

Python with Naveen

Example on CSS

1) In your app create a **"static"** folder (Directory)

2) If you need you can create a sub folder for CSS files and save all CSS files into that folder.



```
Project21  F:\Naveen Class Room\Djang
  app21
    migrations
    static
      css
        sample.css
      images
        man.png
      js
        sample.js
    __init__.py
    admin.py
    apps.py
    models.py
```

3) Write a CSS Code in "sample.css" file

Example

```
h1{
    background-color: orangered;
    color: white;
    text-shadow: 5px 5px 5px blue;
    font-family: "Agency FB";
    font-size: 60px;
    text-align: center;
}
```

4) In HTML file in line no 1 load the static in block tag

Example

```
{% load static %}
```

5) In HTML use the CSS Code.

Example

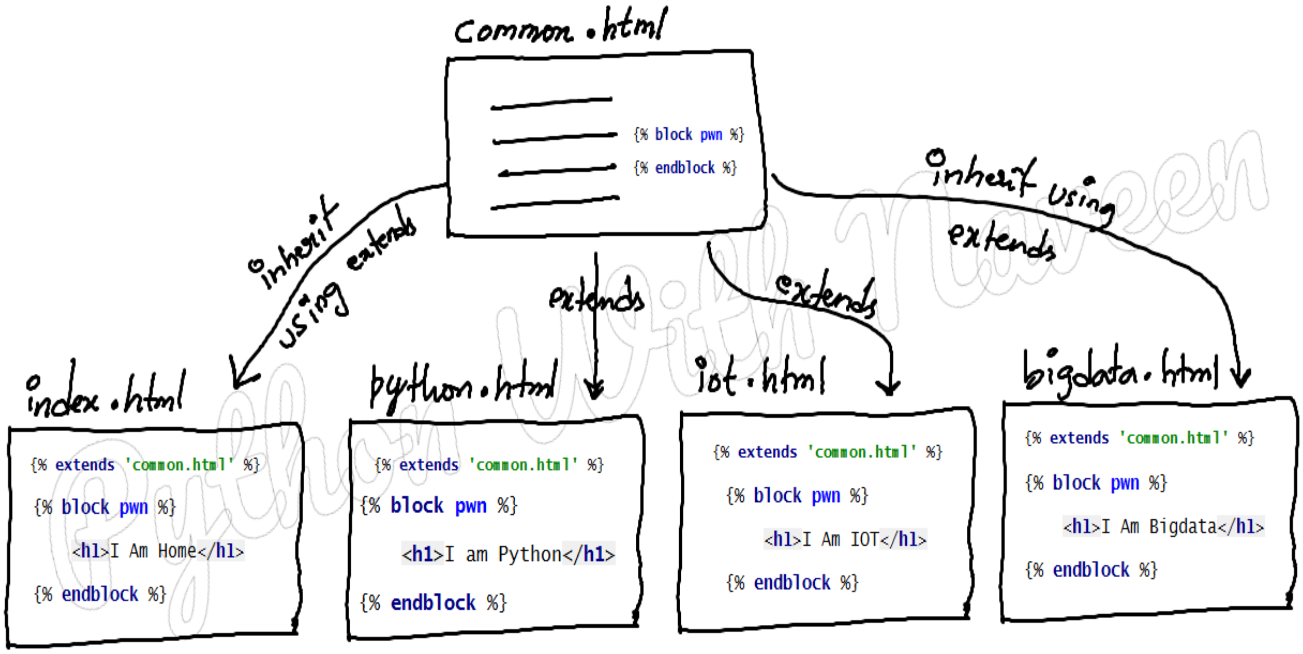
```
<head>
<link rel="stylesheet" href="{% static
'css/sample.css' %}">
</head>

<body>
     <h1> Welcome to CSS </h1>
</body>
```

Example on Bootstrap

Python with Naveen

Template Inheritance

Template inheritance allows you to build a base "skeleton" template that contains all the common elements of your site and defines **blocks** that child templates can override.



```
{% block pwn %}
{% endblock %}
```

```
{% extends 'common.html' %}
{% block pwn %}
    <h1>I Am Home</h1>
{% endblock %}
```

```
{% extends 'common.html' %}
{% block pwn %}
    <h1>I am Python</h1>
{% endblock %}
```

```
{% extends 'common.html' %}
 {% block pwn %}
    <h1>I Am IOT</h1>
 {% endblock %}
```

```
{% extends 'common.html' %}
{% block pwn %}
    <h1>I Am Bigdata</h1>
{% endblock %}
```

Python With Naveen

1) To inherit a template we use "extends" in block tag.

Example : `{% extends 'common.html' %}`

2) While inheriting a template **"extends"** must be the first line in a Template.

3) One Template file can inherit one template only.

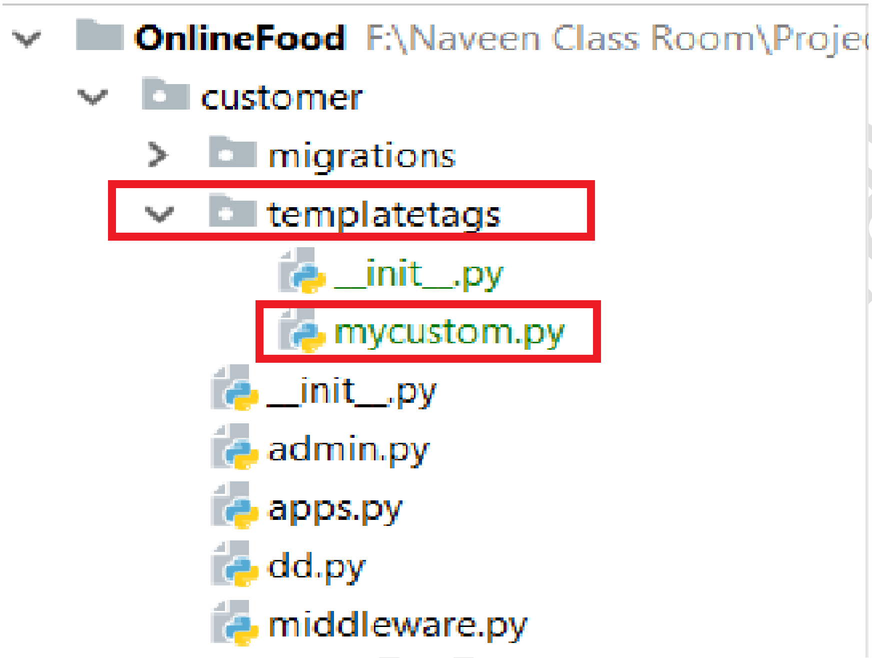
NOTE: In template language **Multiple inheritance is not possible** where Multilevel inheritance is possible.

Custom Template Tags (User defined template Tags)

1) In your app create a python package and name it as "templatetags".

2) Create a new python file in "templatetags" package and name it.

Example :



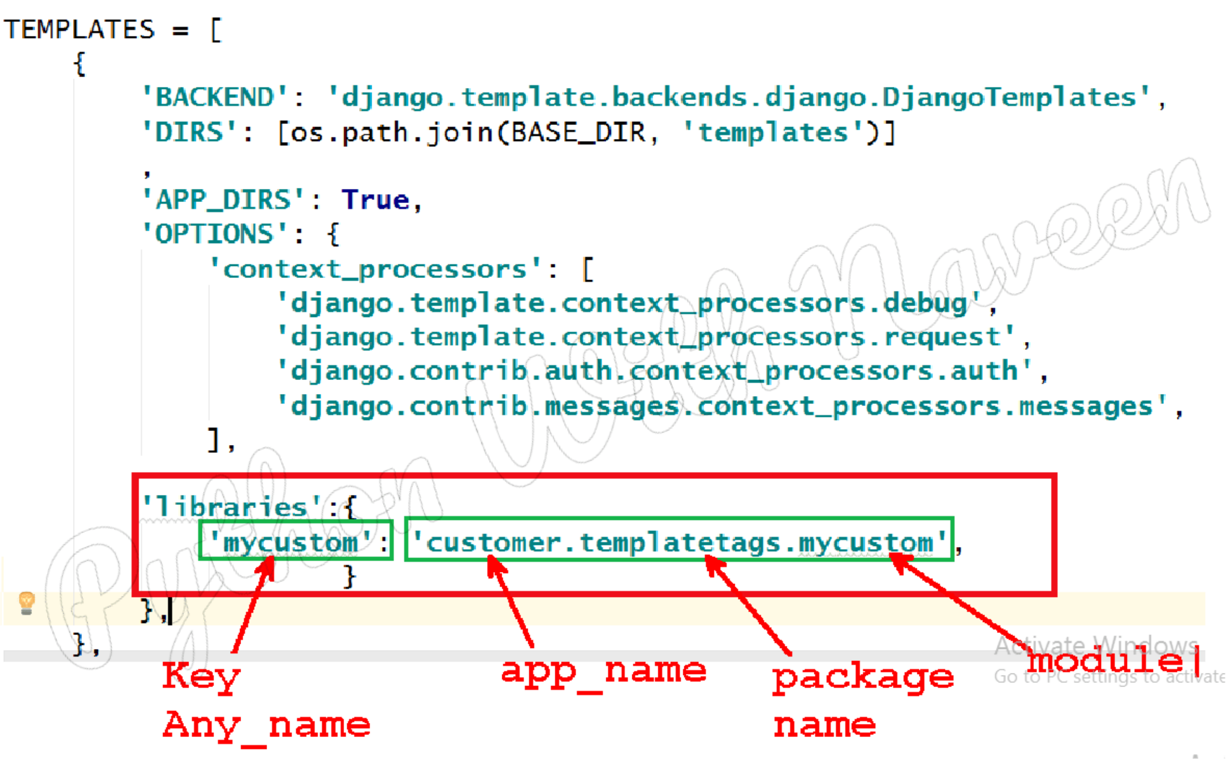
```
OnlineFood  F:\Naveen Class Room\Projec
    customer
        migrations
        templatetags
            __init__.py
            mycustom.py
        __init__.py
        admin.py
        apps.py
        dd.py
        middleware.py
```

3) Write the Program into **"mycustom.py"** file.

```
from django import template

register = template.Library()

@register.simple_tag()
def totalAmount(quantity,price):
    return price*quantity
```

Note: variable name must be **"register"** only.

4) In **"settings.py"** file go to **"TEMPLATES"** list and add a library



```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')]
        ,
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],

        'libraries':{
            'mycustom': 'customer.templatetags.mycustom',
                }
        },
    },
```

Python With Naveen

5) In template "load" custom tag.

Example :

```
{% load mycustom %} ---- Key
```

6) Use the function name as a tag name

Example

```
{% totalAmount x.quantity x.food.price %}
```

7) after doing the changes rebuild the server.

Sending Email From Django Application

Step 1 : Create a project in django.

Step 2 : In Project **settings.py** add the below code.

```
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_USE_TLS = True
EMAIL_PORT = 587
EMAIL_HOST_USER = '..............................@gmail.com'
EMAIL_HOST_PASSWORD = 'nave...................'
```

Step 3 : Open **views.py**

```
from django.core.mail import send_mail
from Example import settings as se

def login(request):
    send_mail("Check","This is a check
mail",se.EMAIL_HOST_USER,[".........@gmail.com",".........@gmail.com",
".........@gmail.com",".........@gmail.com"])
    return HttpResponse("Valid")
```

Step 4: after login to gmail account open the below link
https://myaccount.google.com/lesssecureapps

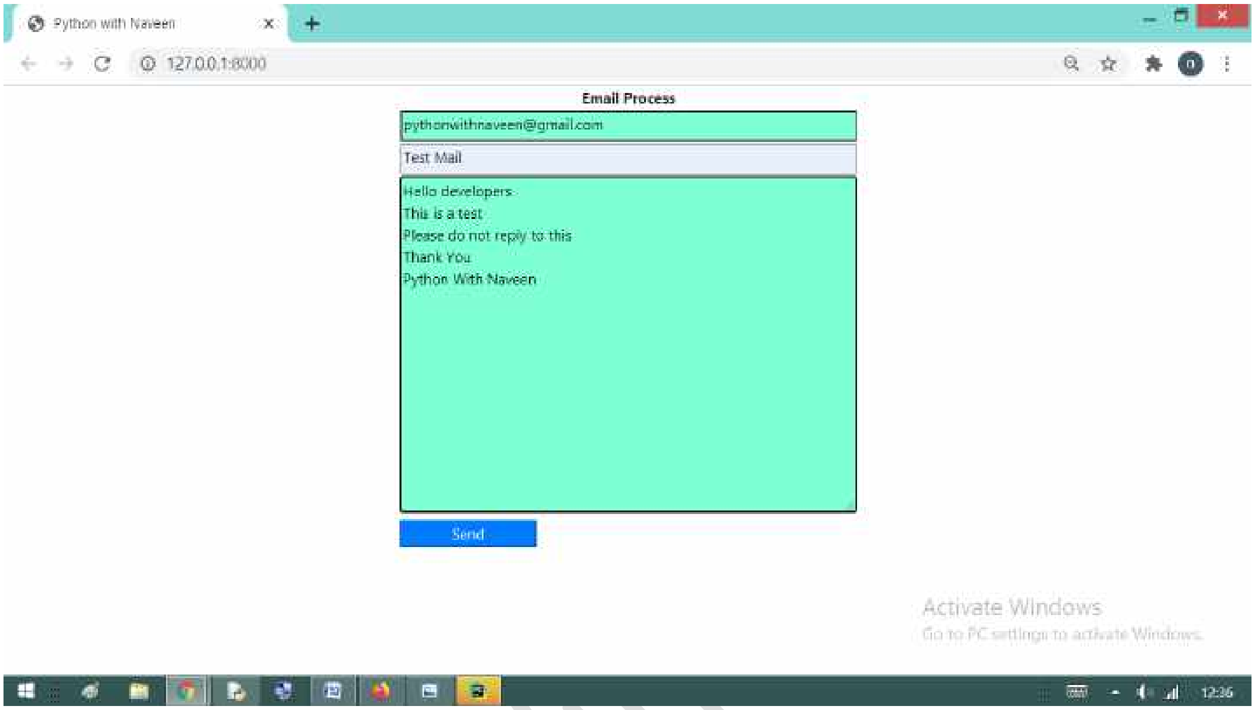
Step 5: Turn on the Allow less secure apps: ON

Syntax of send_mail function

def send_mail(subject, message, from_email, recipient_list)

Python With Naveen

Output



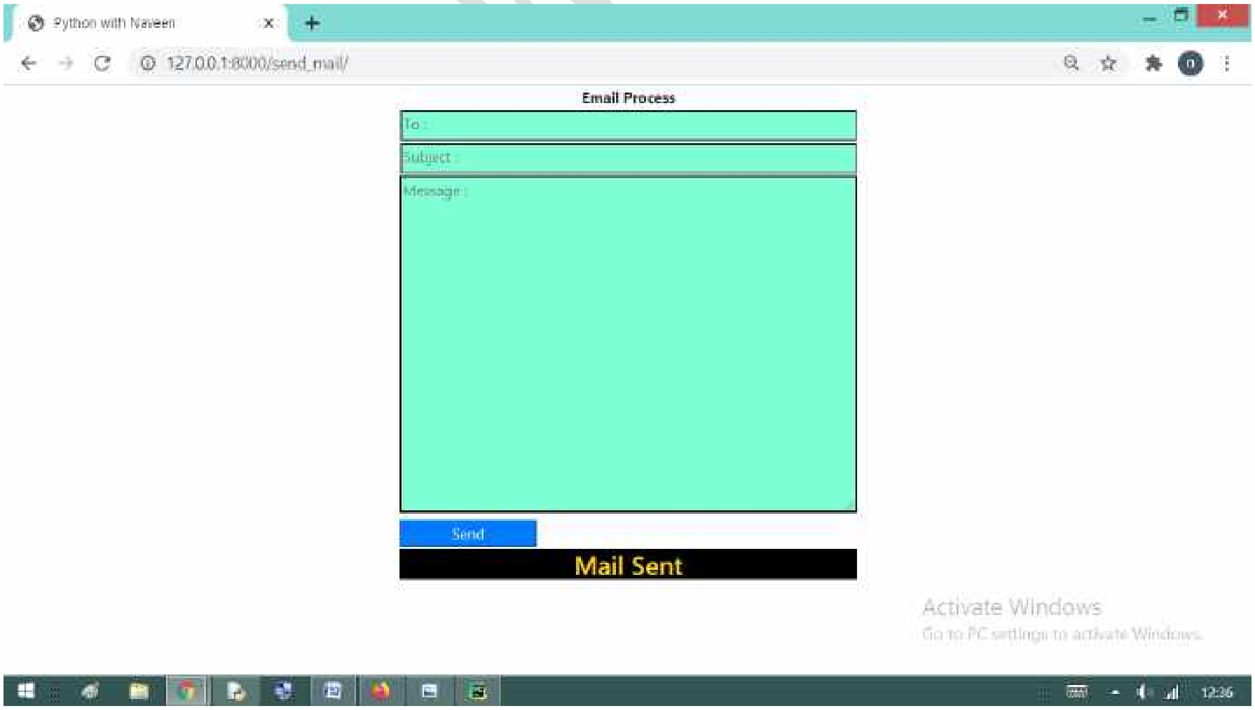
Email Process

pythonwithnaveen@gmail.com

Test Mail

Hello developers
This is a test
Please do not reply to this
Thank You
Python With Naveen

After Mail Sent



Python with Naveen
127.0.0.1:8000/send_mail/

Email Process

To :

Subject :

Message :

Mail Sent

Activate Windows
Go to PC settings to activate Windows.
12:36

Humanization

To activate these filters, add **'django.contrib.humanize'** to your **INSTALLED_APPS** setting.

Once you've done that, use **{% load humanize %}** in a template, and you'll have access to the following filters.

apnumber

For numbers 1-9, returns the number spelled out. Otherwise, returns the number.

Examples:

- **1** becomes **one**.
- **2** becomes **two**.

intcomma

Converts an integer or float (or a string representation of either) to a string containing commas every three digits.

Examples:

- **4500** becomes **4,500**.
- **4500.2** becomes **4,500.2**.

intword

Converts a large integer (or a string representation of an integer) to a friendly text representation. Works best for numbers over 1 million.

Python With Naveen

Examples:

- **1000000** becomes **1.0 million**.
- **1200000** becomes **1.2 million**.

naturalday

For dates that are the current day or within one day, return “today”, “tomorrow” or “yesterday”, as appropriate. Otherwise, format the date using the passed in format string.

Argument: Date formatting string as described in the **date** tag.

Examples (when ‘today’ is 17 Feb 2007):

- **16 Feb 2007** becomes **yesterday**.
- **17 Feb 2007** becomes **today**.
- **18 Feb 2007** becomes **tomorrow**.

ordinal

Converts an integer to its ordinal as a string.

Examples:

- **1** becomes **1st**.
- **2** becomes **2nd**.
- **3** becomes **3rd**.

Python With Naveen

Django Cookie

A cookie is a small piece of information which is stored in the client browser. It is used to store user's data in a file permanently (or for the specified time).

Cookie has its expiry date and time and removes automatically when gets expire. Django provides built-in methods to set and get cookie.

The **set_cookie()** method is used to set a cookie and **get()** method is used to get the cookie.

The **request.COOKIES['key']** array can also be used to get cookie values.

Pass max_age to set expiry

In **views.py,** two functions setcookie() and getcookie() are used to set and get cookie respectively

```
# views.py

from django.shortcuts import render
from django.http import HttpResponse

def setcookie(request):
    response = HttpResponse("Cookie Set")
    response.set_cookie('sathya', 'sathyatech.com')
    return response
```

Python With Naveen

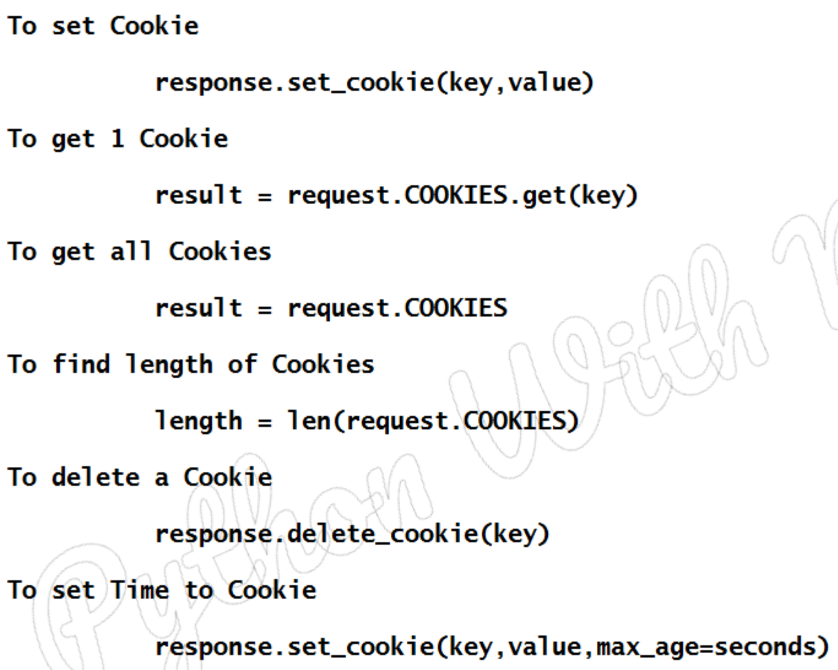
```
def getcookie(request):
    name = request.COOKIES['sathya']
    return HttpResponse("Naveen@: " + name);
```

And URLs specified to access these functions.

```
# urls.py

from django.contrib import admin
from django.urls import path
from myapp import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('index/', views.index),
    path('scookie',views.setcookie),
    path('gcookie',views.getcookie)
]
```

Python With Naveen

Cookie Class Methods



```
To set Cookie

          response.set_cookie(key,value)

To get 1 Cookie

          result = request.COOKIES.get(key)

To get all Cookies

          result = request.COOKIES

To find length of Cookies

          length = len(request.COOKIES)

To delete a Cookie

          response.delete_cookie(key)

To set Time to Cookie

          response.set_cookie(key,value,max_age=seconds)
```

Sending Data in Hyperlink

```
<th class="cart">
    <a href="{% url 'cookie_data'
%}?na=Dell&price=45000">
        Add To Cart
    </a>
</th>
```

Database Operations Using ORM

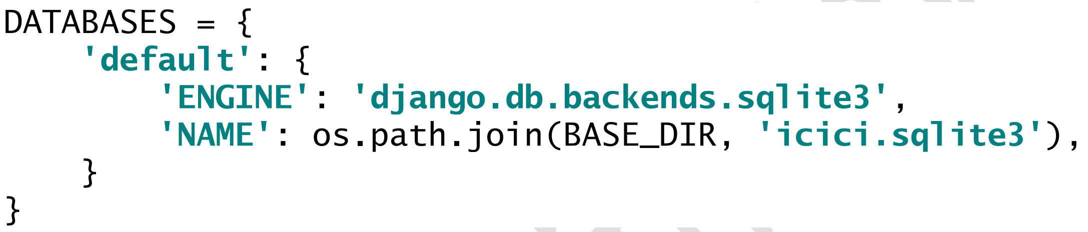
1) Create a Project and app

2) Open **"settings.py"** file

3) In **"settings.py"** do the below changes.

By default the database will be SQLite3

By default the database name will be **"db".**



```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'icici.sqlite3'),
    }
}
```

In above example i have changed the database name to **"icici"**.

4) In **"app"** open **"models.py"** file.

In models.py we define the classes.

Each class will behave like a table and each variable in a class will behave like a column.

5) Your defined class in **"models.py"** must inherit **"models.Model"** class.

Python With Naveen

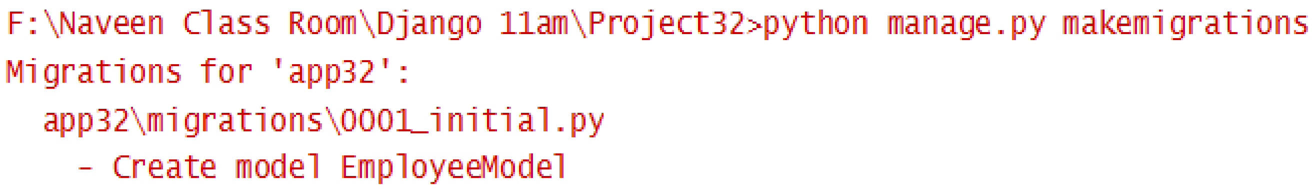
Example

```
from django.db import models

class EmployeeModel(models.Model):
    # EmployeeModel is inheriting Model class
    # So EmployeeModel is also behaving like a Model

    # each variable is a column in table
    idno = models.IntegerField()
    name = models.CharField(max_length=100)
    salary = models.FloatField()
```

6) Open **"Terminal"** in Pycharm or Open **Command Prompt** from your System and type the commands as

--> python manage.py makemigrations

makemigrations means telling to my Django to Convert the above written class into SQL Query.

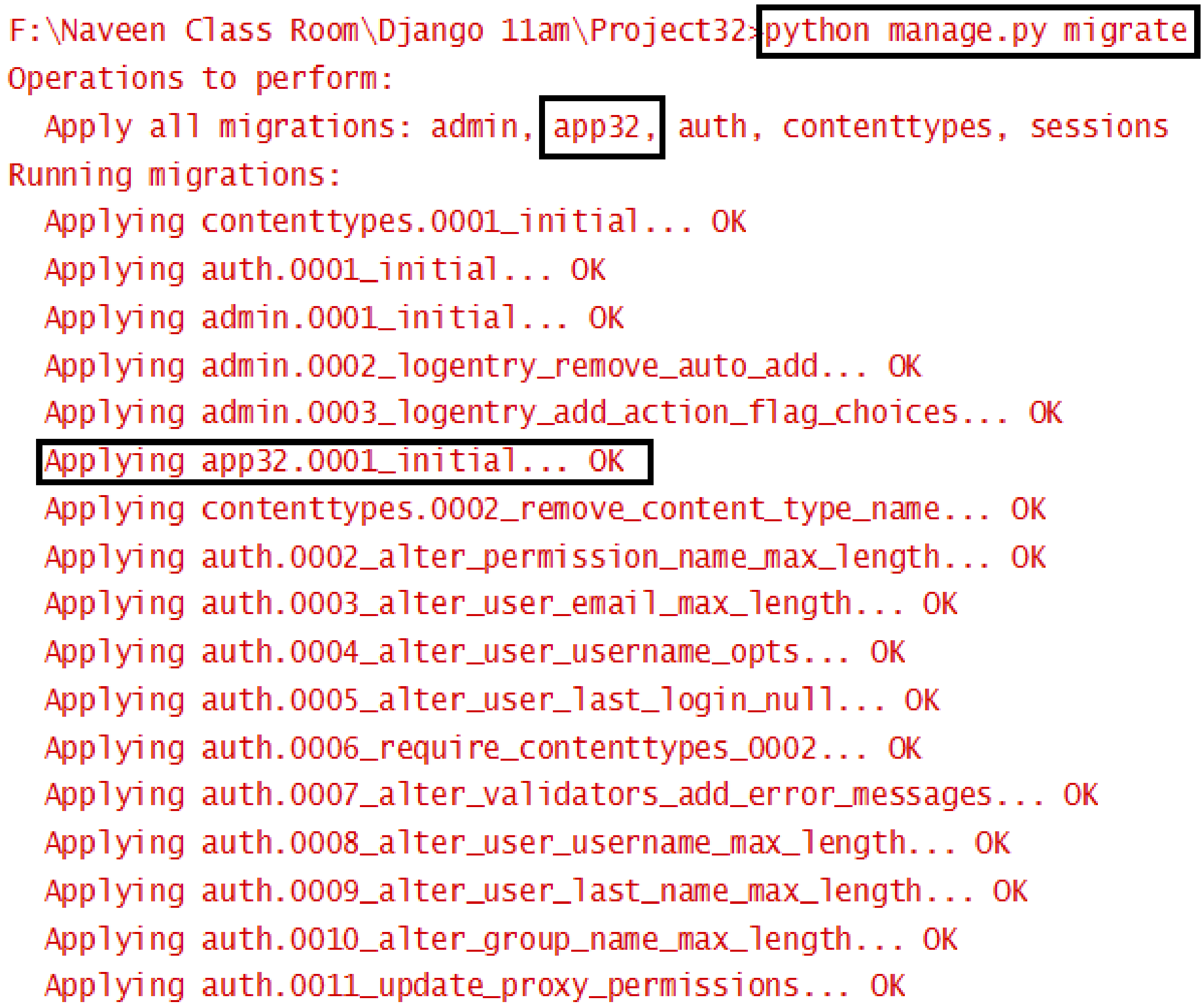


```
F:\Naveen Class Room\Django 11am\Project32>python manage.py makemigrations
Migrations for 'app32':
  app32\migrations\0001_initial.py
    - Create model EmployeeModel
```

--> python manage.py migrate

migrate means telling to my Django to execute the SQL Query so it will create a table into the Database.

Python With Naveen



```
F:\Naveen Class Room\Django 11am\Project32>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, app32, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying app32.0001_initial... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
```

Python

7) After migrate the Django will create 10 Automated tables plus your tables.

8) The django will name your table with app name.

app_name_ + class name = table_name

Example :

app_name is "app32" and class name is "EmployeeModel"

---> app32_employeemodel

Python With Naveen