Chapter 10 :

Informatics Practices

Class XII (As per CBSE Board)

New Syllabus 2019-20

Visit : python.mykvs.in for regular updates

Django

Django is an open source *web application* development framework. It was Named after famous Guitarist "Django Reinhardt".it was Developed by Adrian Holovaty and Jacob Kaplan-moss at World Online News for efficient development in python .It was Open sourced in 2005 and it's first Version released September 3, 2008.

It follows the principle of "Don't Repeat Yourself". Means keeping the code simple and non repeating. Django is also a high level, MVT architect which stands for Model View Template.

Features of Django

- **Fast: -encourages rapid development**
- **Tons of Packages: that help us to develop websites faster and easier.**
- **Secure: It helps the developers to avoid many common security mistakes, such as SQL injection, cross-site scripting, csrf and clickjacking.**
- **Versatile –can develop all sort of things – like content management systems ,social networks,scientific computing platforms etc.**

A *Web application* (Web app) is an application program that is stored on a remote server and delivered to a browser through internet.

Django

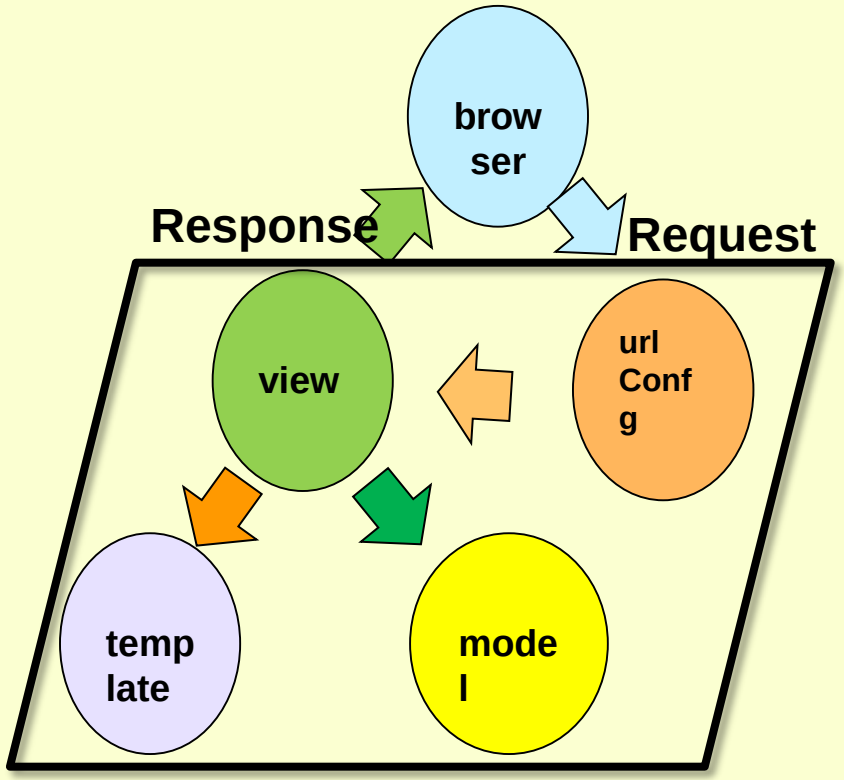
Django architecture

Django follows a MVC- MVT architecture.
MVC stands for Model View Controller.

Model – Model is used for storing and maintaining data and work as a backend to define database.

Views – In Django templates, View is all about the which user is seeing. Templates and views are designed in html.

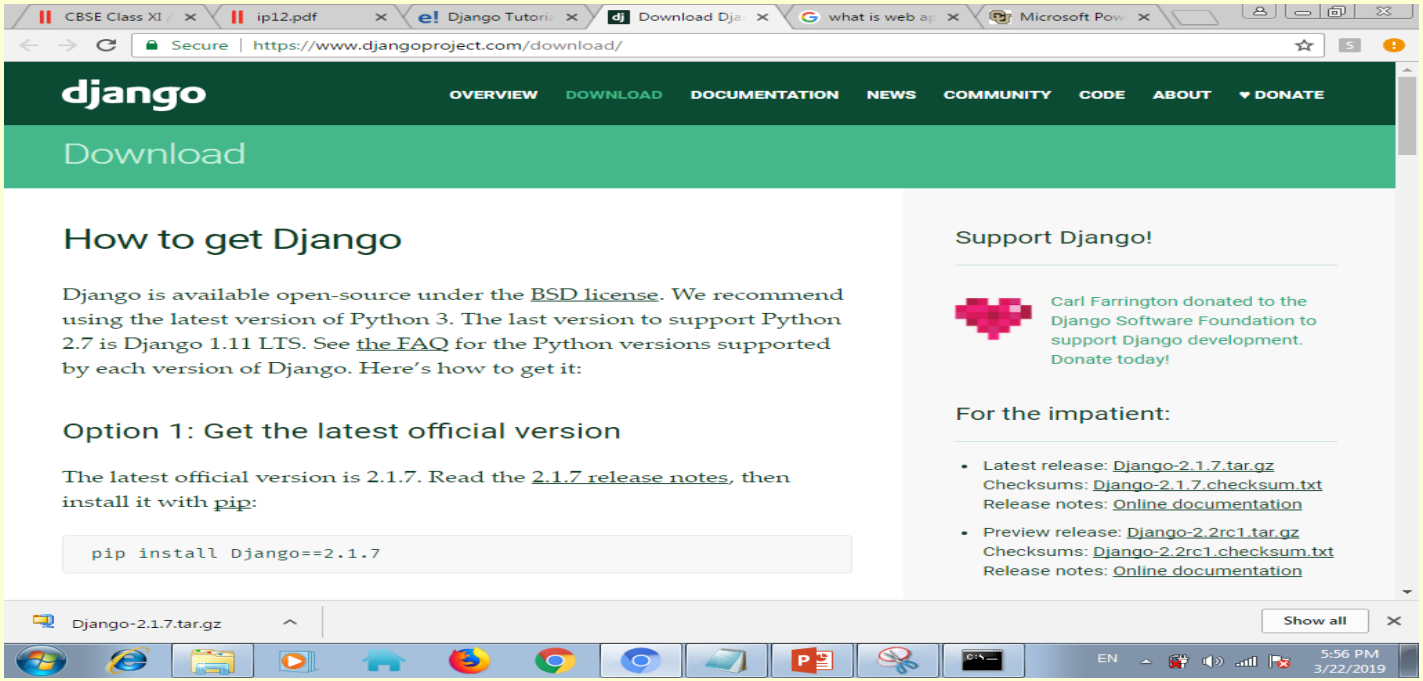
Controller – business logic which interact with the model and the view.



Django

Django Installation – Method1

Step 1: Go to : https://www.djangoproject.com/download/ , Read the release notes.



Step 2: Type the pip command in command prompt

pip install django

After its completion,installation part will be completed

Django

Django Installation – Method2

Step 1- open command prompt in windows(type cmd in windows search)

Step 2 – type command easy_install django and press enter

Wait for few minutes.

After installation a message will be shown

Finished processing dependencies for django

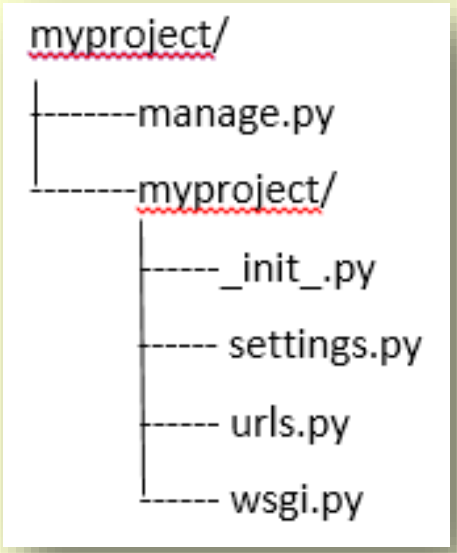
Django

Building Web Application in Django

- Create a folder on computer. E.g. create a folder named demo in c drive.
- Open command prompt through cmd command in search option of window.
- Move to folder demo in command prompt(using cd command) **cd c:\demo**
- Run the following command to create project

  c:\demo> django-admin startproject myproject

It will create list of files in demo->myproject



manage.py – used as command to interact with this Django project.
myproject/ – It is actual Python package.
init.py–tells the python to treated like a python package.
settings.py – to manage settings of project.
urls.py – to maps website.
wsgi.py – It serves as an entry point for WSGI compatible web servers.

- **Move to the folder where manage.py file is stored.**

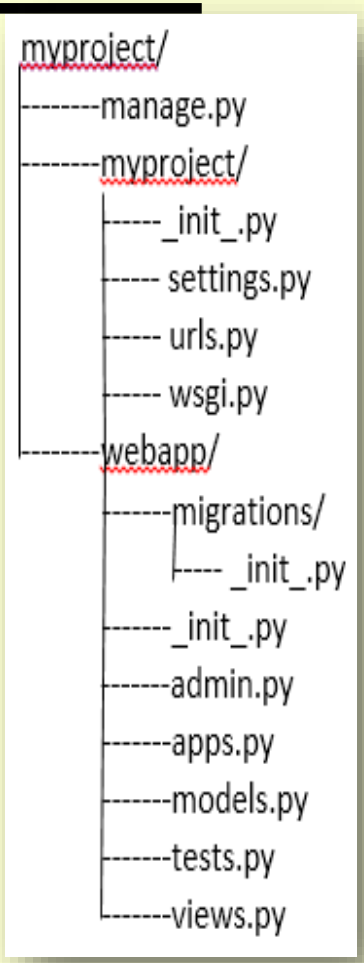
  c:\demo>cd c:\demo\myproject

Django

Building Web Application in Django

- **Type the following command to create app there**
  python manage.py startapp webapp
  it will create other files in myproject/webapp
- **Next, we need to import our application manually inside project settings.open** myproject/settings.py **in windows edit with ide and add webapp manually as below code and save it.**

```
INSTALLED_APPS = [
    'webapp',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```



- **now. Open webapp/views.py and put below code in it:**

```
from django.shortcuts import render
from django.http import HttpResponse
def index(request):
    return  HttpResponse("<H2>Hi ,it is our first django webapp </H2>")
```

Django

Building Web Application in Django

- Now we need to map this view to a URL.so create a new python file “urls.py” inside our webapp.In webapp/urls.py include the following code:

```
from django.conf.urls import url
from . import views
urlpatterns = [
 url(r'^$', views.index, name='index'),
]
```

In the above code, we have referenced a view which will return index. here url pattern is a regular expression where ^ stands beginning of the string and $ stands for the end.

- Open myproject/urls.py file and write the below code to point the root URLconf at the webapp.urls module.

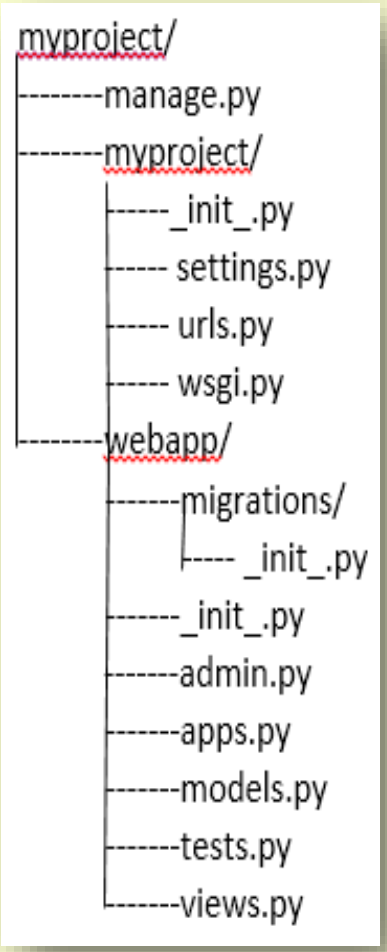
```
from django.conf.urls import include, url
from django.contrib import admin
urlpatterns = [
    url(r'^webapp/', include('webapp.urls')),
]
```

- First move to folder of manage.py file,run the server with command

python manage.py runserver

After running the server, go to **http://localhost:8000/webapp/** In any browser

Note – for above development pycharm(trial version available) like ide can be used



Django

Download the demo project from the link given below

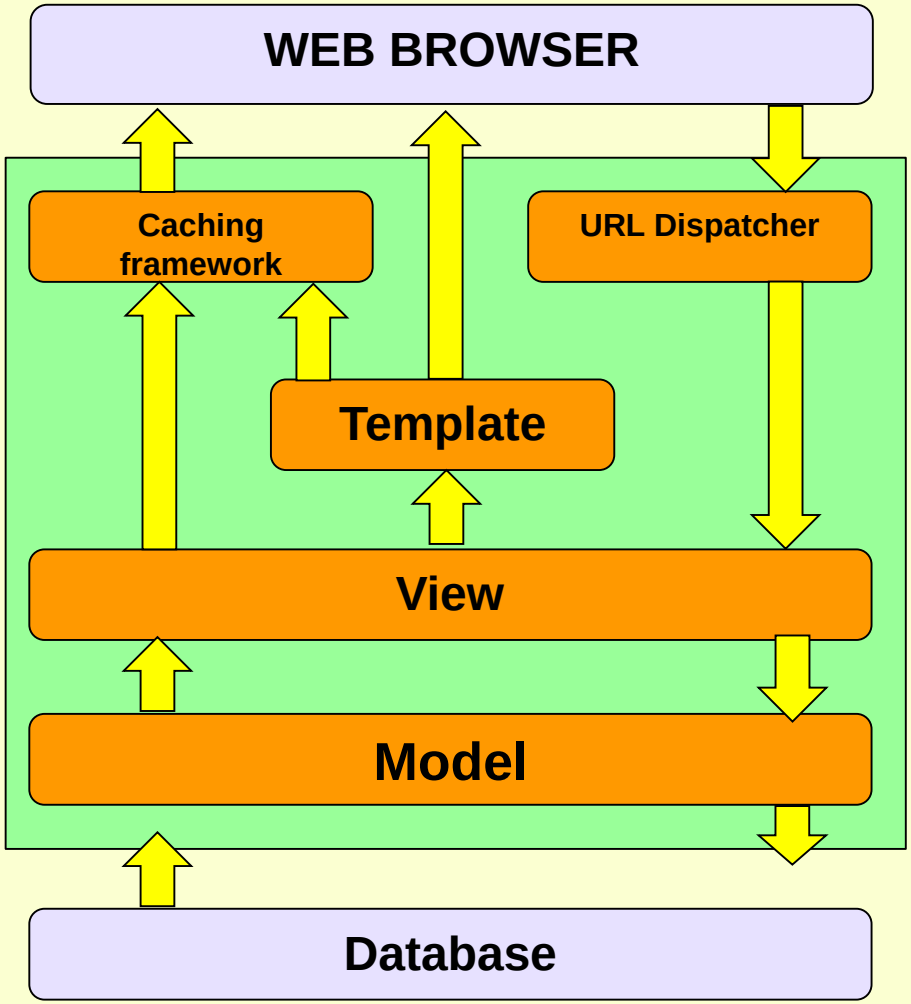
Click here

1. **Download will start**
2. **After download completes**
3. **Extract all files on c drive ->it will create demo folder**
4. **Open command prompt and move to c:\demo\myproject**
5. **Write command python manage.py runserver**
6. **Open any browser and write url http://localhost:8000/webapp/**
7. **It will display the view/page**

NOTE :- django must be installed to run the server and webapp

Django

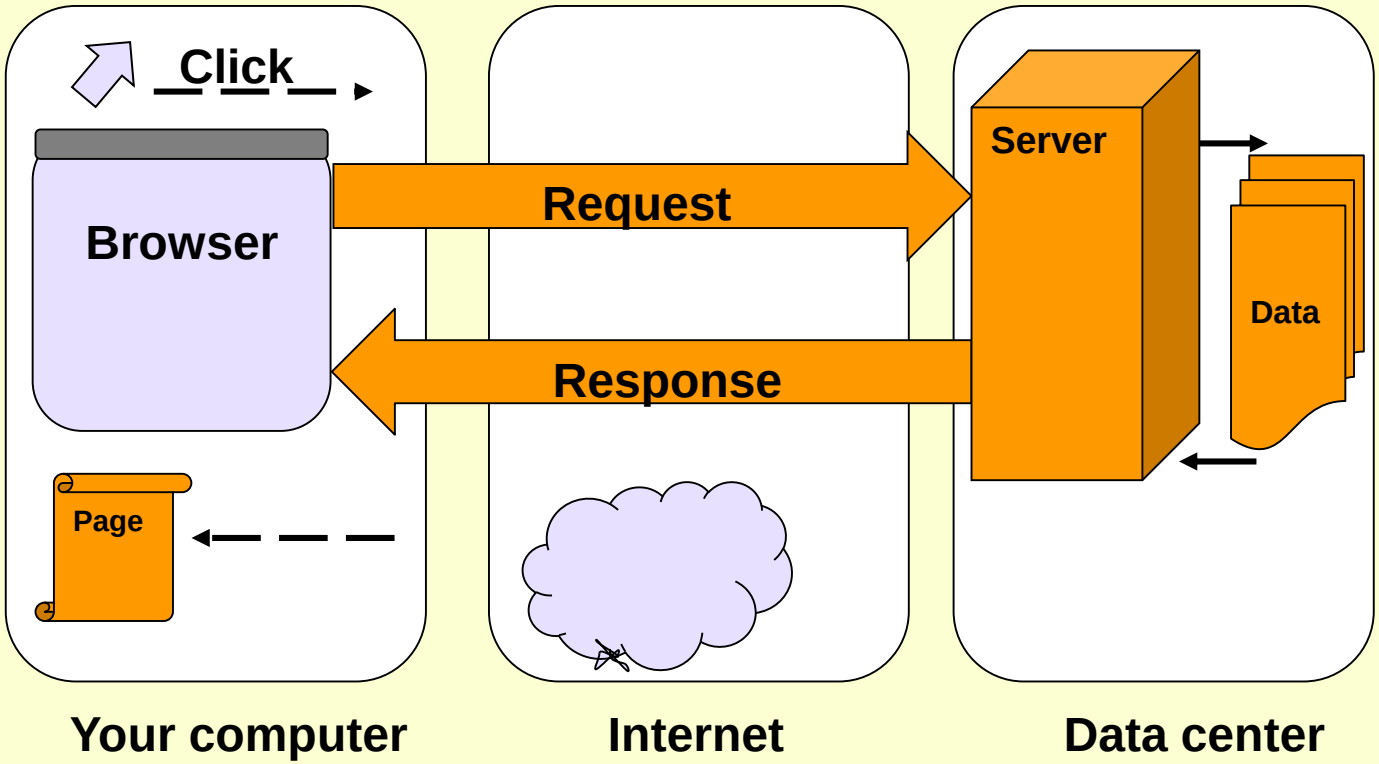
Functional architecture of django webapplication



1. URL dispatcher(urls.py)-> requests url to view function and call it.if cache version available then cache copy Will be returned
2. View function(view.py)-> perform display Part and database interaction
3. model(models.py) -> define data in Pythond and interact with data(typically Mysql,postgress,sqlite etc)
4. Templates -> return html pages with the help of django template language.
5. After any request the view returns HTTP response object to the web browser ,generally to display value

Django

Django Request and Response life cycle



Django uses request and response objects to pass state through the system. When a page is requested, Django creates an HttpRequest object which contains metadata about the request. Then Django loads the appropriate view, passing the HttpRequest as the first argument to the view function. Each view is responsible for returning an HttpResponse object.

Django

Django HttpRequest Attributes

Attribute	Description
HttpRequest.scheme	Representing the scheme of request (HTTP or HTTPs usually).
HttpRequest.body	Returns raw HTTP request body as byte string.
HttpRequest.path	It returns the full path to the requested page but not include the scheme/domain.
HttpRequest.path_info	It shows path info portion of the path,no matter what Web server is being used
HttpRequest.method	It shows the HTTP method used in the request.like get or post
HttpRequest.encoding	It shows the current encoding used to decode form submission data.
HttpRequest.content_type	It shows the MIME type of the request, parsed from the CONTENT_TYPE header.
HttpRequest.content_params	It returns a dictionary of key/value parameters included in the CONTENT_TYPE header.
HttpRequest.GET	It returns a dictionary-like object containing all given HTTP GET parameters.
HttpRequest.POST	It is a dictionary-like object containing all given HTTP POST parameters.
HttpRequest.COOKIES	It returns all cookies available.
HttpRequest.FILES	It contains all uploaded files.
HttpRequest.META	It shows all available Http headers.
HttpRequest.resolver_match	It contains an instance of ResolverMatch representing the resolved URL.

Django

Django HttpRequest Methods

Attribute	Description
HttpRequest.get_host()	Returns the original host of the request.
HttpRequest.get_port()	Returns the originating port of the request.
HttpRequest.get_full_path()	Returns the path, plus an appended query string, if applicable.
HttpRequest.build_absolute_uri *(location)*	Returns the absolute URI form of location.
HttpRequest.get_signed_cookie *(key, default=RAISE_ERROR, salt='', max_age=None)*	Returns a cookie value for a signed cookie
HttpRequest.is_secure()	Returns True if the request is secure; that is, if it was made with HTTPS or not.
HttpRequest.is_ajax()	Returns True if the request was made via an XMLHttpRequest.

Django

Django HttpResponse Attributes

Attribute	Description
HttpResponse.content	A bytestring encoded from a string, if necessary.
HttpResponse.charset	A string denoting the charset in which the response will be encoded.
HttpResponse.status_code	It is an **HTTP status code** for the response.
HttpResponse.reason_phrase	The HTTP reason phrase for the response.
HttpResponse.streaming	It is false by default.
HttpResponse.closed	It is True if the response has been closed.

Django

Django HttpResponse Methods

Method	Description
HttpResponse.__init__*(content='', content_type=None, status=200, reason=None, charset=None)*	To instantiate an HttpResponse object with the given page content and content type.
HttpResponse.__setitem__*(header, value)*	It is used to set the given header name to the given value.
HttpResponse.__delitem__*(header)*	It deletes the header with the given name.
HttpResponse.__getitem__*(header)*	It returns the value for the given header name.
HttpResponse.has_header*(header)*	It returns either True or False based on a case-insensitive check for a header with the provided name.
HttpResponse.setdefault*(header, value)*	It is used to set default header.
HttpResponse.write*(content)*	It is used to create response object of file-like object.
HttpResponse.flush()	It is used to flush the response object.
HttpResponse.tell()	This method makes an HttpResponse instance a file-like object.
HttpResponse.getvalue()	It is used to get the value of HttpResponse.content.
HttpResponse.readable()	This method is used to create stream-like object of HttpResponse class.
HttpResponse.seekable()	It is used to make response object seekable.

Django

Minimal Django based web application that parses a GET

For any interactive web application ,we have to develop web application in two parts

1. Front end – Generally any html form where user enters data
2. Back end – where data are processed and stored

User enter data on html form and then submit these data,which moves to the server with the help of **HttpRequest** then these data are processed at the server as per need and then returns the result to requesting browser with the help of **HttpResponse** object in the form of html

Django

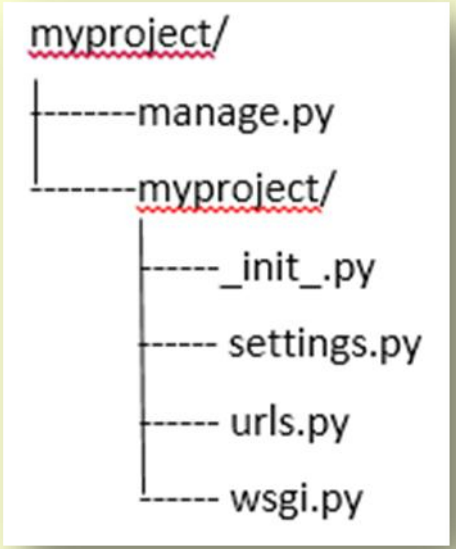
Minimal Django based web application that parses a GET

Here we are developing interactive web app step wise

- **Create a folder on computer. E.g. create a folder named demo1 in c drive.**
- **Open command prompt through cmd command in search option of window.**
- **Move to folder demo1 in command prompt(using cd command) cd c:\demo1**
- **Run the following command to create project**

  c:\demo1> django-admin startproject myproject

  It will create list of files in demo1->myproject



- **Move to the folder where manage.py file is stored.**

  c:\demo1>cd c:\demo1\myproject

Django

Minimal Django based web application that parses a GET

- **Type the following command to create app there**
  python manage.py startapp webapp
  it will create other files in myproject/webapp
- **Next, we need to import our application manually inside project settings.open** myproject/settings.py **in windows edit with ide and add webapp manually as below code and save it.**

```
INSTALLED_APPS = [
    'webapp',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

- **now. Open webapp/views.py and put below code in it:**

```
from django.shortcuts import render
from django.http import HttpResponse
def search_form(request):
        return render(request, 'webapp/search_form.html')
```

Django

Minimal Django based web application that parses a GET

- Next step is to create the template
- Create a new \templates folder inside webapp folder. Then go ahead and create another folder webapp inside the templates folder.Our final folder structure will be **webapp\templates\webapp\ .**

This inner webapp folder is important for namespacing templates. Because Django will search all apps for a matching template, creating a namespace for the app templates ensures that Django uses the correct template if two apps used the same template name.

- Create the following search_form.html file and save it to new folder webapp:

```
<html>
<head>
<title>Search</title>
</head>
<body>
<form action="/search/" method="get">
<input type="text" name="q">
<input type="submit" value="Search">
</form>
</body>
</html>
```

Django

Minimal Django based web application that parses a GET

- Now we need to create a URLconf so Django can find our new view.for this create urls.py file in base webapp folder

```
# mysite\books\urls.py
from django.conf.urls import url
from webapp import views
urlpatterns = [
url(r'^search-form/$', views.search_form),
]
```

- when Django searches for URL patterns, it will only search the base myproject\urls.py file, unless we explicitly include the URL patterns from other apps. So let's go ahead and modify our site urlpatterns :

```
# myproject\urls.py
from django.contrib import admin
from django.urls import path
from django.conf.urls import include, url
urlpatterns = [
url(r'^', include('webapp.urls')),
]
```

- First move to folder of manage.py file,run the server with command

python manage.py runserver

and then visit **http://127.0.0.1:8000/search-form/** , we'll see the search interface.

Django

Minimal Django based web application that parses a GET

- If we press search button then **Django 404 error** is displayed because no seach part is defined.
- To fix it ,make changes **webapp/urls.py** file like below code

```
from django.conf.urls import url
from webapp import views
urlpatterns = [
url(r'^search-form/$', views.search_form),
url(r'^search/$', views.search),
]
```

- And make change **webapp/views.py** file like below code

```
from django.shortcuts import render
from django.http import HttpResponse
def search_form(request):
        return render(request, 'webapp/search_form.html')
def search(request):
    if 'q' in request.GET:
        message = 'You searched for: %r' % request.GET['q']
    else:
        message = 'You submitted an empty form.'
    return HttpResponse(message)
```

- Now first move to folder of manage.py file,run the server with command
  python manage.py runserver
  and then visit **http://127.0.0.1:8000/search-form/** , press search button after text entry.

Django

Minimal Django based web application that parses a GET

- In above development The HTML <form> defines a variable q . When it's submitted, the value of q is sent via GET (method="get") to the URL /search/ .
- The Django view that handles the URL /search/ (search()) has access to the q value in request.GET .An important thing to point out here is that we explicitly check that 'q' exists in request.GET

Query String Parameters

Because GET data is passed in the query string (e.g., /search/?q=django), we can use request.GET to access query string variables.

Use GET when the act of submitting the form is just a request to "get" data. Use POST whenever the act of submitting the form will have some side effect – changing data, or sending an e-mail,or something else that's beyond simple display of data means encoded data move ,which are not traceable through URL.

Q.1 Develop a web application which prompt two numbers and display the sum of these two numbers after pressing sum button.

Q.2 Develop a web application which prompt principle amount,rate and time and display simple interest.

Django

Download the demo1 project from the link given below

Click here

1. **Download will start**
2. **After download completes**
3. **Extract all files on c drive ->it will create demo1 folder**
4. **Open command prompt and move to c:\demo1\myproject**
5. **Write command python manage.py runserver**
6. **Open any browser and write url http://127.0.0.1:8000/search-form/**
7. **It will display the view/page**

NOTE :- django must be installed to run the server and webapp

Django

Differences between the GET and POST methods in form submitting

GET Method	POST Method
• GET requests can be cached • GET requests remain in the browser history • GET requests can be bookmarked • GET requests should never be used when dealing with sensitive data like password • GET requests have length restrictions • GET requests is only used to request data (not modify)	• POST requests are never cached • POST requests do not remain in the browser history • POST requests cannot be bookmarked • POST requests have no restrictions on data length • URL query string is encoded so cant be used for malicious purpose

Django

Minimal Django based web application that parses a POST

Now we are acquainted with web application development in Django and using GET Method.So using POST method is very easy.

To learn POST method make below changes in the existing Demo1 project.

1. **Replace the existing code with following code in webapp/views.py file**

```
from django.shortcuts import render
from django.http import HttpResponse
def search_form(request):
        return render(request, 'webapp/search_form.html')
def sum_number(request):
    if request.method=='POST':
        a=request.POST.get('n1')
        b=request.POST.get('n2')
        c=int(a)+int(b)
        return HttpResponse("sum='"+str(c)+"'")
```

Django

Minimal Django based web application that parses a POST

2. Replace the existing code with following code in webapp/templates/webapp/search_form.html file

```
<html>
<head>
<title>sum two numbers</title>
</head>
<body>
<form action="/getdata/" method="post">
{% csrf_token %}
Enter first number<input type="text" name="n1"><br>
Enter second number<input type="text" name="n2"><br>
<input type="submit" value="Sum">
</form>
</body>
</html>
```

Django

Minimal Django based web application that parses a POST

3. Replace the existing code with following code in webapp/urls.py file

```
from django.conf.urls import url
from webapp import views
urlpatterns = [
url(r'^search-form/$', views.search_form),
url(r'^getdata/$', views.sum_number),
]
```

- Now Open command prompt and move to **c:\demo1\myproject**
- Write command **python manage.py runserver**
- Open any browser and write url **http://127.0.0.1:8000/search-form/**
- It will display the view/page(search_html) with two text boxes,user have to enter two numbers and press sum button.values will be sent to server using POST method without displaying in URL and will be processed with the help of views.py file and result will be displayed

Django

Download the demo1 project for POST Method from the link given below

Click here

1. Download will start
2. After download completes
3. Extract all files on c drive ->it will create demo2 folder
4. Open command prompt and move to **c:\demo2\myproject**
5. Write command **python manage.py runserver**
6. Open any browser and write url **http://127.0.0.1:8000/search-form/**
7. It will display the view/page

NOTE:-django must be installed to run the server and webapp

Django

Writes the fields to a file – flat file

The File object- Internally, Django uses a django.core.files.File instance any time it needs to represent a file.Now Open demo1 project and overwrite below code in webapp/views.py file

```
from django.shortcuts import render
from django.http import HttpResponse
from django.core.files import File          #import django file handling library
def search_form(request):
        return render(request, 'webapp/search_form.html')
def sum_number(request):
    if request.method=='POST':
        a=request.POST.get('n1')
        b=request.POST.get('n2')
        c=int(a)+int(b)

        #file handling code
        f = open('a.txt', 'w')
        myfile = File(f)
        myfile.write(str(c))
        myfile.close()
        return HttpResponse("sum='"+str(c)+"'")
```

Flat file writing code

Here file a.txt is opened in f further referenced by myfile to write value of c in string form ,then file is closed and at last result is also shown over browser

Django

Writes the fields to a file – flat file

1. Open command prompt and move to **c:\demo1\myproject**
2. Write command **python manage.py runserver**
3. Open any browser and write url **http://127.0.0.1:8000/search-form/**

It will display two text boxes ,after entering value and press sum button. Result will be stored in a file a.txt (@ C:\demo1\myproject) as well as displayed over browser.
We can open a.txt file and find the result in it also.

Python file handling concepts can be used for other file handling operations

Django

Read and Writes the fields on a file – flat file

```
#Make changes in views.py as below code for write/read operation
from django.shortcuts import render
from django.http import HttpResponse
from django.core.files import File
def search_form(request):
        return render(request, 'webapp/search_form.html')
def sum_number(request):
    if request.method=='POST':
        a=request.POST.get('n1')
        b=request.POST.get('n2')
        c=int(a)+int(b)

        f = open('a.txt', 'w')
        myfile = File(f)
        myfile.write(str(a)+" ")
        myfile.write(str(b)+" ")
        myfile.write(str(c)+" ")
        myfile.close()

        f = open('a.txt', 'r')
        t=''
        for text in f.readlines():
            for word in text.split( ):
                t=t+word+'<br>'
        myfile.close()
        return HttpResponse(t)
```

Writes the value of a,b and c on a.txt file separated by space through file object's write method.

Read the line by line through first loop and each word is separated in inner loop with the help of split method.
Each word is appended as html text in t along with
 tag

Django

Writes the fields to a file – CSV file

Python comes with a CSV library, csv. The key to using it with Django is that the csv module's CSV-creation capability acts on file-like objects, and Django's HttpResponse objects are also file-like objects.Make changes in views.py file for csv file creation.

```
from django.shortcuts import render
from django.http import HttpResponse
import csv
def search_form(request):
        return render(request, 'webapp/search_form.html')
def sum_number(request):
    if request.method=='POST':
        a=request.POST.get('n1')
        b=request.POST.get('n2')
        c=int(a)+int(b)

        #write csv file
        response = HttpResponse(content_type='text/csv')
        response['Content-Disposition'] = 'attachment; filename="mycsvfile.csv"'
        writer = csv.writer(response)
        writer.writerow([a, b, c])
        return response
```

Django

Writes the fields to a file – CSV file

In the program following points to be noted

- **The response gets a special MIME type, *text/csv*. This tells browsers that the document is a CSV file, rather than an HTML file.**
- **The response gets an additional Content-Disposition header, which contains the name of the CSV file.**
- **Hooking into the CSV-generation API is easy: Just pass response as the first argument to csv.writer. The csv.writer function expects a file-like object, and HttpResponse objects**
- **For each row in your CSV file, call writer.writerow, passing it an iterable object such as a list or tuple.**