# Django -A Web Framework in Python:-

1.FOSS --> Free and Open Source Software
2.Follows MVT(Model View Template)
3.Used in Instgram, YouTube, DropBox.......
4.Developed in 2003, but it was realeased publicly in 2005.
5.Official Documentation @ djangoproject.com
6.Maintained by DSF --> Django Software Foundation

#Features/Advantages:-

1.Ridiculously fast.
2.Reassuringly secure.
3.Exceedingly scalable.
4.Fully loaded.
5.Incredibly versatile.

#IDE(Integrated Developement Env) = TextEditor+FileManager+CommandPrompt+Browser

```
      1)PyCharm, 2)VScode, 3)Eclipse+plugin---> PyCharm Comm.
      Suugested:-8GB RAM, SSD, i5(8th+)/Ryzen4/3
      Minimum:-  4GB RAM, HDD, i3
```

#Virtual Enviornment:-

A virtual environment is a tool that helps to keep dependencies required by different projects separate
by creating isolated python virtual environments for them.
This is one of the most important tools that most of the Python developers use.

Dependencies-->eg. BootStrap v3.0, Third Party Libraries-->CrispyForms,numPy,sciPy
Python 2.7, Django 1.9
After 5 yrs-->Maintainace
Python 3.8, Django 3.0

****

DJANGO ROUGH NOTES/SYNTAXES TO REMEMBER:-

```
1. Create new PC proj --> Choose location --> Create
2. Install Django --> pip install django

3. Create Django Proj   --> django-admin startproject PROJNAME
4. Change Directory     --> cd PROJNAME
5. Run                  --> py manage.py runserver
                      OR (Add conf-->'+'-->scriptPathmanage.py--
>parameter"runserver")

6. Create views.py      --> PROJNAME-->PROJNAME-->views.py
7. Define a view  -->   from django.http import HttpResponse
                        def view1(request):
                              return HttpResponse("Hello")
                        def view2(request):
                              return HttpResponse("Bye")
8. Define an urlpattern(path) --> PROJNAME-->PROJNAME-->urls.py
                              from . import views
                              urlpatterns = [
                                          path('admin/',........),
                                          path('v1/',views.view1),
                                          path('v2/',views.view2)
```

#Project Level Folder Structure:-

```
manage.py   --> command line utility(e.g. migrations, run server, createsuperuser)
__init__.py --> Blank Script,The folder in which it is present,will be treated as
python package.
settings.py --> Project related,Installed apps,Databases,Middlewares
settings/configurations
urls.py     --> url-patterns
asgi.py, wsgi.py  --> Deployment

                                          ]
(Don't use step 6,7&8 if u r going 2create Django App)
```

#DJANGO APPS:-

```
9.Create a django app   --> py manage.py startapp APPNAME
10.Register/Configure   --> PROJNAME-->PROJNAME-->settings.py
                        INSTALLED_APPS = ['','','','','APPNAME']
11.Define a view  --> PROJNAME-->APPNAME-->views.py
                        from django.http import HttpResponse
                        def view1(request):
                              return HttpResponse("Hello")
                        def view2(request):
                              return HttpResponse("<h1>Bye</h1>")

12.Create new urls.py   --> PROJNAME-->APPNAME-->urls.py (AppLevel urls)
13.Define an urlpattern(path) --> PROJNAME-->APPNAME-->urls.py
                              from . import views
                              urlpatterns = [
                                          path('v1/',views.view1),
                                          path('v2/']
14.Register appLevel url.py   --> PROJNAME-->PROJNAME-->urls.py(ProjectLevel urls)
                              from django.urls import path, include
                              urlpatterns = [
                                          path('fa/',include('APPNAME.urls')),
                                          path('sa/',include('APPNAME2.urls')),
```

#App Level Folder Structure:-

```
admin.py  --> Administrative Panel config
apps.py  --> App config
models.py  --> Models(Database tables, config)
tests.py  --> Testing Purpose(Unit test)
views.py --> Views
                                          ]
```

#TEMPLATES in Django:-

```
15.Create dir. structure      --> PROJNAME-->templates-->APPNAME-->home.html
16.Register/Config Dir.       --> PROJNAME-->PROJNAME-->settings.py
                              TEMPLATES = [{'DIRS':['templates']}]
17.Render the template in view      --> PROJNAME-->APPNAME-->views.py
                              def view3(request):
                                    template_name = "AppName/template.html"
                                    context = {}
                                    return render(request,template_name,context)
18.Create URLPattern for this view
```

#CONTEXT in Django:-

```
18. PROJNAME-->APPNAME-->views.py
            def view22(req):
                  request = req
                  template_name = "APPNAME/template.html"
                  context = {'key':val, 'key2':val2}
                  return render(request,template_name,context)

19. PROJNAME-->templates-->APPNAME-->template.html
            {{key2}}

            {% for ele in var %}
                  <tag>{{ele}}</tag>
            {% endfor %}

            {% if condition %}
                  code
            {% elif condition %}
                  code
            {% endif %}
```

#STATIC FILES IN DJANGO:-

```
20.Create Dir.Structure -> PROJNAME->"static"->"APPNAME"->css,js,images
21.Register Static Dir. -> PROJNAME->PROJNAME->settings.py->STATICFILES_DIRS =
['static']
22.Create a CSS/IMG/JS  -> PROJNAME->"static"->"APPNAME"->css->main.css
23.load static files    -> PROJNAME->"templates"->"APPNAME"->HTMLPAGE.html->{% load
static %}
24.href/src       -> "{% static '_________________' %}"
```

#TEMPLATE INHERITANCE:-

```
25.Create base temp.              ->PROJNAME->"templates"->"APPNAME"->"base.html"
      ->
                                Cut & Paste all the common part on each webpage &&&
blocks
26.Extend child template          -> {% extends 'APPNAME/base.html' %}
27.Define variable code in blocks   -> {% block BLOCKNAME %}
      (in child template)               <!-- Write ur html code here -->
                                 {% endblock %}
```

#Sample Folder Stucture Including Apps,Templates & Static Files

```
─ProjName
    │   db.sqlite3
    │   manage.py
    │
    ├───FirstApp
    │   │   admin.py
    │   │   apps.py
    │   │   models.py
    │   │   tests.py
    │   │   urls.py
    │   │   views.py
    │   │   __init__.py
    │   │
    │   ├───migrations
    │   │   │   __init__.py
    │   │   │
    │   │   └───__pycache__
    │       │           __init__.cpython-38.pyc
    │       │
    │       └───__pycache__
    │               admin.cpython-38.pyc
    │               models.cpython-38.pyc
    │               urls.cpython-38.pyc
    │               views.cpython-38.pyc
    │               __init__.cpython-38.pyc
    │
    ├───ProjName
    │   │   asgi.py
    │   │   settings.py
    │   │   urls.py
    │   │   wsgi.py
    │   │   __init__.py
    │   │
    │   └───__pycache__
    │           settings.cpython-38.pyc
    │           urls.cpython-38.pyc
    │           wsgi.cpython-38.pyc
    │           __init__.cpython-38.pyc
    │
    ├───static
    │   └───FirstApp
    │           style.css
    │
    └───templates
        └───FirstApp
                child1.html
                child2.html
                home.html
                normal.html
```

#MVT Architecture:-
MVT architecture is the software design pattern used by the Django web framework.
MVT stands for Model - View - Template.

1) Model
Just like the Model in MVC, here as well it has the same functionality of providing the interface
for the data stored in the database.

2) Template
Just like View in MVC, Django uses templates in its framework. Templates are responsible for the
entire User Interface completely. It handles all the static parts of the webpage along with the HTML, which the users visiting the webpage will perceive.

3) Views
In Django, Views act as a link between the Model data and the Templates.

Note: Just like the controller in MVC, views in Django MVT are responsible for handling all the
business logic behind the web app. It acts as a bridge between Models and Templates.

```
Model --> Table in DataBase/Class in Python -->models.py
View  --> Business Logic --> FBVs/CBVs       -->views.py
Template --> Presentation --> HTML file     -->templ.html
```

#DJANGO ADMIN PANEL:-

```
 "py manage.py makemigrations"      --> python code gets converted into SQL
 "py manage.py migrate"       --> execute the SQL code
28. "py manage.py createsuperuser" --> creates new user acc with admin. rights -->
One Time
```

#Models in Django:-

```
29. Create a Model      --> PROJNAME-->AppName-->models.py

                        from django.db import models
                        class ModelName(models.Model):
                              col1 = models.DATATYPEField()
                              col2 = models.DATATYPEField()

30. Register/config.    --> PROJNAME-->AppName-->admin.py
                  A]
                        from .models import ModelName
                        admin.site.register(ModelName)
                  B]
                        from .models import ModelName
                        class ModelNameAdmin(admin.ModelAdmin):
                              list_display = ['col1','col2','col3']
                        admin.site.register(ModelName,ModelNameAdmin)
31. "py manage.py makemigrations"  --> python code gets converted into SQL
32. "py manage.py migrate"         --> execute the SQL code
```

#Django-ORM Queries:-

```
We'll use the Interactive Console/Shell to learn these queries
33. Open the Interactive Console --> "py manage.py shell"
34. Import the model          --> from APPNAME.models import MODELNAME
35. Queries
      #Retrival               -->
      1. objs = MODEL.objects.all()
      2. obj = MODEL.objects.get(col=val) -->OBJ
      3. objs = MODEL.objects.filter(col=val)   -->QS
      4. objs = MODEL.objects.exclude(col=val)
      5. objs = MODEL.objects.filter(col__lt = val)
      6. objs = MODEL.objects.filter(col__gt = val)
      7. objs = MODEL.objects.filter(col__lte = val)
      8. objs = MODEL.objects.filter(col__gte = val)
      9. objs = MODEL.objects.filter(col__startswith = 'val')
      10. objs = MODEL.objects.filter(col__endswith = 'val')
      11. objs = MODEL.objects.filter(col__contains = 'val')
      12. objs = MODEL.objects.order_by('col')
      12. objs = MODEL.objects.order_by('-col')
      #Update
      13. obj = MODEL.objects.get(col=val)
          obj.COL = NEWVAL
          obj.save()
      #Delete
      14. obj = MODEL.objects.get(col=val)
          obj.delete()
      #Insert
      15. obj = MODEL(col1=val,col2=val,col3=val..)
```

```
          obj.save()
            OR
          obj = MODEL()
          obj.col1 = val
          obj.col2 = val
            obj.col3 = val
          obj.save()
      #Aggregation Funs
      from django.db.models import Max,Min,Avg,Sum,Count
      16. mx = MODEL.objects.all().aggregate(Max('col'))
      17. mi = MODEL.objects.all().aggregate(Min('col'))
      18. sm = MODEL.objects.all().aggregate(Sum('col'))
      19. av = MODEL.objects.all().aggregate(Avg('col'))
      20. ct = MODEL.objects.all().aggregate(Count('col'))
      #And
      21. objs = MODEL.objects.filter(condition1 , condition2)
      #Or
      22. objs = MODEL.objects.filter(condition1) |
MODEL.objects.filter(condition2)
      from django.db.models import Q
      23. objs = MODEL.objects.filter(Q(condition1) | Q(condition2))
```

#MySQL integration with Django:-

```
1. Install Connector    --> "pip install mysqlclient"
                                  OR
                      1."pip install pymysql"
                      2.PROJNAME->PROJNAME->__init__.py
                              import pymysql
                              pymysql.version_info = (1,4,0,"final",0)
                              pymysql.install_as_MySQLdb()

2. Config.        --> PROJNAME->PROJNAME->settings.py
                        DATABASES = {
                                     'default': {
                                                      'ENGINE':
'django.db.backends.mysql',
                                               'NAME': 'newdb',
                                               'USER':'root',
                                               'PASSWORD':'root'
                                              }
                                  }
```

#Realationships betn Models:-

```
1.One to One      --> model_nameFIELD =
models.OneToOneField(MODELNAME,on_delete=models.CASCADE)
2.One to Many     --> model_nameFIELD =
models.ForeignKey(MODELNAME,on_delete=models.CASCADE)
3.Many to Many    --> model_nameFIELD = models.ManyToManyField(MODELNAME)
```

#Django Form:-

```
A] Raw HTML form  --> 1.template(HTML)-->2.render in view-->3.urlpattern for view

                  --> dataFromFE = request.POST.get('name')

B] Django Form Class    --> Python Class(Fetch data from fields 1by1 & save it by
creating obj)
                  --> dataFromFE = form.cleaned_data['name']

c] Django ModelForm     --> Existing Model(form.save())

                  --> dataFromFE = form.cleaned_data['name']


B] Django Form Class
38. Create forms.py     --> PROJNAME-->APPNAME-->forms.py
39. Define form   -->
                        from django import forms
                        class NAMEForm(forms.Form):
                              field1 = forms.DATATYPE()
                              field2 = forms.DATATYPE()
                              fieldN = forms.DATATYPE()

40.Render form          --> PROJNAME-->APPNAME-->views.py
                        from .forms import NAMEForm
                        def viewName(request):
                              if request.method == "GET":
                                    form = NAMEForm() #Blank Form
                              else:
                                    form = NAMEForm(request.POST)#Form with entered
data
                                    if form.is_valid():
                                          val1 = form.cleaned_data['field1']
                                          val2 = form.cleaned_data['field2']
                                          # code
                                          return HttpResponse("Success msg")
                              template_name = 'APPNAME/template.html'
                              context = {'form':form}
                              return render(request,template_name,context)

41. Create URLpattern   --> PROJNAME-->APPNAME-->urls.py
                        from . import views
                        urlpatterns = [ path('pattern/',views.ViewName)]
42. template.html -->
                        <form method="post">
                              {% csrf_token %}
                              {{form.as_p}}
                              <input type="submit'>
                        </form>

c] Django ModelForm
43. Create forms.py     --> PROJNAME-->APPNAME-->forms.py
44. Define Model form   -->   from .models import Model
                        class ModelnameForm(forms.ModelForm):
                              class Meta:
                                    model = Model
                                    fields = '__all__'
                                          OR
                                    fields = ('field1','field3','field7')
                                          OR
                                    exclude = ['field1']

45.Render form in View  --> PROJNAME-->APPNAME-->views.py
                        from .forms import ModelnameForm
                        def ViewName(request):
                              obj = ModelnameForm()
                              template_name = 'APPNAME/template.html'
                              context = {'form':obj}
                              return render(request,template_name,context)

46. Create URLpattern   --> PROJNAME-->APPNAME-->urls.py
                        from . import views
                        urlpatterns = [ path('pattern/',views.ViewName)]
42. template.html -->
                        <form method="post">
                              {% csrf_token %}
                              {{form.as_p}}
                              <input type="submit'>
                        </form>
```

#FORM VALIDATION:-

```
A] Built-in Validators  --> Custom Validator
B] clean methods

A] Using built-in validators  --> PROJNAME-->APPNAME-->forms.py
                        from django import forms
                        from django.core import validators
                        class NAMEForm(forms.Form):
                              field1 = forms.DATATYPE(validators =
[validators.ANYValidator,

validators.ANYValidator2])
                              field2 = forms.DATATYPE()
                              fieldN = forms.DATATYPE()

B] Using clean methods        --> PROJNAME-->APPNAME-->forms.py
                        from django import forms
                        class NAMEForm(forms.Form):
                              field1 = forms.DATATYPE()
                              field2 = forms.DATATYPE()
                              fieldN = forms.DATATYPE()

                              def clean_field1(self):
                                    entered_data= self.cleaned_data['field1']
                                    #validation logic with entered_data MANDATORILY
                                    #                            RETURN VALUE
                                    return value

                                    OR
                              def clean(self):
                                    all_data = super().clean()
                                    entered_data1 = all_data['field1']
                                    entered_data2 = all_data['field2']

                                                #validation logic NO RETURN
                                          STATEMENT SHOULD BE USED
```

#STYILING A FORM:-

```
-->PROJNAME-->APPNAME-->forms.py
                              from django import forms
                              class NAMEForm(forms.Form):
                                    field1 = forms.DATATYPE(required=False)
                                    field2 = forms.DATATYPE(label='Your custom
label:')
                                    field3 =
forms.DATATYPE(widget=forms.PasswordInput())
                                    field4 = forms.DATATYPE(widget=forms.TextInput(
                                                attrs={'attr1':'val1',
'attr2':'val2'}))
                                    fieldN = forms.DATATYPE()
```

#BootStrap CDN:-

```
<!-- CSS only -->
<link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css"
 integrity="sha384-
9aIt2nRpC12Uk9gS9baDl411NQApFmC26EwAOH8WgZl5MYYxFfc+NcPb1dKGj7Sk"
 crossorigin="anonymous">

<!-- JS, Popper.js, and jQuery -->
<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"
integrity="sha384-DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXaRkfj"
crossorigin="anonymous"></script>

<script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js"
integrity="sha384-Q6E9RHvbIyZFJoft+2mJbHaEWldlvI9IOYy5n3zV9zzTtmI3UksdQRVvoxMfooAo"
crossorigin="anonymous"></script>

<script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/js/bootstrap.min.js"
integrity="sha384-OgVRvuATP1z7JjHLkuOU7Xw704+h835Lr+6QL9UvYjZE3Ipu6Tp75j7Bh/kR0JKI"
crossorigin="anonymous"></script>
```

#USING CRISPY FORMS IN DJANGO:-

```
48] install crispy forms      --> pip install django-crispy-forms
49] register                  --> PROJNAME--PROJNAME-->settings.py-->
                              INSTALLED_APPS-->'crispy_forms'
50] config              --> PROJNAME--PROJNAME-->settings.py-->
                              CRISPY_TEMPLATE_PACK = 'bootstrap4'
51] load in templates         --> {% load crispy_forms_tags %}
52] apply               --> {{form|crispy}}


a] Blank Form :-              form = FormName()
b] Form with data filled by User:-  form = FormName(request.POST)
c] Form with data present in DB:-   form = FormName(instance=OBJECT)
d] Updating a form present in DB with Values filled(updated) by user:-
                              form = FormName(request.POST, instace=OBJECT)

Dynamic URL patterns in Django
in template --> {% url 'staticpart' dynamicpart %}
in urls.py  --> path('staticpart/<data_type:dynamicpart>/',views.AnyView,
name='name_to_urlPattern')
```

#Authentication In Django:-

A]Registration
B]Login
C]Logout

A]Registration-

Built In SignUp form(UserCreationForm)

53] Create a registerView --> PROJNAME-->APPNAME-->views.py

```
from django.contrib.auth.forms import UserCreationForm
def registerView(request):
     if request.method == 'GET':
          form = UserCreationForm()
          #code to render this form
     elif request.method == 'POST':
          form = UserCreationForm(request.POST)
          #code to save the data from form
```

54] Create a urlPattern --> PROJNAME-->APPNAME-->urls.py

```
#imports
urlpatterns = [ path('urlpattern/',views.registerView,
                        name='patternName')
            ]
```

B]Logging In:-

55] Create a login template   --> PROJNAME-->templates-->APPNAME-->login.html

56] Create a view to render this template -->PROJNAME-->APPNAME-->views.py

```
##
#Structure without actual code:
def loginView(request):
      if request.method == 'GET':
            #code to render login.html
      elif request.method == 'POST':
            #code to login

##
#Example with actual code:
from django.contrib.auth import authenticate, login
      def loginView(request):
            if request.method == 'GET':
                  template_name = 'APPNAME/login.html'
                        context = {}
                  return render(request, template_name,
context)
            elif request.method == 'POST':
                  u = request.POST['uname']
                  p = request.POST['pw']
                  print(u,p)
                  user =
authenticate(username=u,password=p)
                        #user object OR None
                  if user is not None:
                              login(request,user)
                            return redirect('allstudents')
                        #send it to any urlpattern
                  else:
                            return HttpResponse("Invalid
Credentials!")
```

57] Create a urlPattern   --> PROJNAME-->APPNAME-->urls.py

```
#imports
```

```
                              urlpatterns = [
                                          path('urlpattern/',views.loginView,
                                                name='patternName')
                                          ]
```

#Mandatory Login for accesing particular view(Redirects to Login Page if not logged in)

```
1.from django.contrib.auth.decorators import login_required
      @login_required(login_url='login')
            OR
      @login_required()
      def secureViewToBe(request):
            #code
2.PROJNAME->PROJNAME->settings.py-> LOGIN_URL = 'login'
      (Need this step,if login_url attribute isn't specified in @login_required
decorator)

3.Working:
      a.If not logged in        :  secureView--->loginView---->secureView
      b.If already logged in    :  secureView
```

C]Logout:-

```
58] Create a logout view      -->PROJNAME-->APPNAME-->views.py
                                          def logoutView(request):
                                                logout(request)
                                                #code to redirect

59] Create a urlPattern       --> PROJNAME-->APPNAME-->urls.py
                              #imports
                              urlpatterns = [
                                          path('urlpattern/',views.logoutView,
                                                name='patternName')
                                          ]
```

#CLASS BASED VIEWS (CBVs):-

```
PROJNAME->PROJNAME->APPNAME->views.py
67. class ClassName(View):
      def get(self,request):
            #code for GET req
      def post(self,request):
            #code for POST req

68] Create a urlPattern       --> PROJNAME-->APPNAME-->urls.py
                              #imports
                              urlpatterns = [
                                          path('urlpattern/',
                                                views.YourClassBasedView.as_view(),
                                                name='patternName')
```