Contents

[Django – Design Philosophies 2](#_Toc29831919)

[Advantages of Django 2](#_Toc29831920)

[DJANGO MVC - MVT Pattern 3](#_Toc29831921)

[setup 4](#_Toc29831922)

[STEPS 6](#_Toc29831923)

[Reusability in htmls: 9](#_Toc29831924)

[Request capturing: 11](#_Toc29831925)

[Configuring welcome page 13](#_Toc29831926)

[Code in Templates: 14](#_Toc29831927)

[Firstof 14](#_Toc29831928)

[If statement 15](#_Toc29831929)

[For loop 16](#_Toc29831930)

[Conditions 17](#_Toc29831931)

[Includes: 18](#_Toc29831932)

# Django – Design Philosophies

Django is a high-level Python Web framework that encourages rapid development and clean pragmatic design.

A Web framework is a set of components that provide a standard way to develop websites fast and easily.

Django’s primary goal is to ease the creation of complex database-driven websites.

Django comes with the following design philosophies −

* **Loosely Coupled** − Django aims to make each element of its stack independent of the others.
* **Less Coding** − Less code so in turn a quick development.
* **Don't Repeat Yourself (DRY)** − Everything should be developed only in exactly one place instead of repeating it again and again.
* **Fast Development** − Django's philosophy is to do all it can to facilitate hyper-fast development.
* **Clean Design** − Django strictly maintains a clean design throughout its own code and makes it easy to follow best web-development practices.

# Advantages of Django

Here are few advantages of using Django which can be listed out here −

* **Object-Relational Mapping (ORM) Support** − Django provides a bridge between the data model and the database engine, and supports a large set of database systems including MySQL, Oracle, Postgres, etc. Django also supports NoSQL database through Django-nonrel fork. For now, the only NoSQL databases supported are MongoDB and google app engine.
* **Multilingual Support** − Django supports multilingual websites through its built-in internationalization system. So you can develop your website, which would support multiple languages.
* **Framework Support** − Django has built-in support for Ajax, RSS, Caching and various other frameworks.
* **Administration GUI** − Django provides a nice ready-to-use user interface for administrative activities.
* **Development Environment** − Django comes with a lightweight web server to facilitate end-to-end application development and testing.

# DJANGO MVC - MVT Pattern

The Model-View-Template (MVT) is slightly different from MVC. In fact the main difference between the two patterns is that Django itself takes care of the Controller part (Software Code that controls the interactions between the Model and View), leaving us with the template. The template is a HTML file mixed with Django Template Language (DTL).



The developer provides the Model, the view and the template then just maps it to a URL and Django does the magic to serve it to the user.

Django = Controller

MVT:

Template is mixed with Django Template language and HTML

# setup

Check PIP version

**pip –version**

**Upgrading pip**

C:\Users\i335484>py -m pip install -U pip

check Django version

**django-admin –version**

**Pip command location:**

C:\Users\i335484\AppData\Local\Programs\Python\Python37-32\Scripts

**Virtual env:**

Separate for every project.

**pip install virtualenvwrapper-win**

installs the virtual environment.

**Create virtual env:**

mkvirtualenv mytest

**Install python**

pip install Django

**OR**

pip install Django==2.2.3

**Result:**

Refer C:\Users\i335484\AppData\Local\Programs\Python\Python37-32\Lib\site-packages

**check Django version**

django-admin –-version

**Env changes**

Add C:\Users\i335484\AppData\Local\Programs\Python\Python37-32\Lib\site-packages\django\bin to env path variable

# STEPS

EX:

C:\Users\i335484>pip --version

C:\Users\i335484>py -m pip install -U pip

C:\Users\i335484>pip --version

pip 19.3.1 from c:\users\i335484\appdata\local\programs\python\python37-32\lib\site-packages\pip (python 3.7)

C:\Users\i335484>pip install virtualenvwrapper-win

C:\Users\i335484>mkvirtualenv mytest

(mytest) C:\Users\i335484>

(mytest) C:\Users\i335484>pip install django (mytest) C:\Users\i335484>mkdir myprojects

(mytest) C:\Users\i335484>cd myprojects

(mytest) C:\Users\i335484\myprojects>django-admin startproject myProject

Urls--🡪 urls

Wsgi -> deploymnets

(mytest) C:\Users\i335484\myprojects\myProject>python manage.py runserver

1.Import project to pycharm

2.open the environment

C:\Users\i335484\myprojects\myProject>workon mytest

3.Create app

(mytest) C:\Users\i335484\myprojects\myProject>python manage.py startapp login

4.add the app name under the settings.py under INSTALLED\_APPS section of project.

1.Create urls.py under the login project.

2.use include for urls.py under myProject

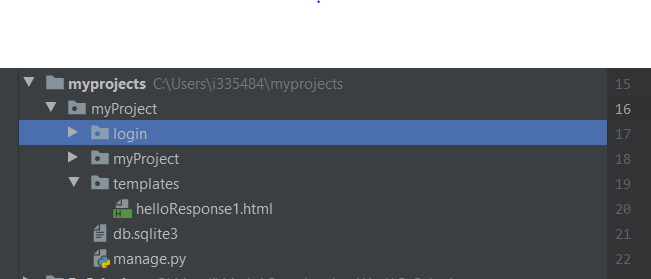
**Migrate:**

**1.python manage.py migrate**

**Create super User**

**2. python manage.py createsuperuser**

**3.Create the folder “templates” along with the app and project and make the below changes under the settings.py.**



|  |
| --- |
| 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',  ],  },  }, ] |

**Views.py**

|  |
| --- |
| def handleResponseData1(request):  return render(request, "helloResponse1.html", {"data":"hi from backend"}) |

# Reusability in htmls:

commonShow.html

|  |
| --- |
| <!DOCTYPE html> <html lang="en"> <head>  <meta charset="UTF-8">  <title>Common</title> </head> <body>  hello .. {% block content%}  {% endblock %} bye  </body> </html> |

Show1.html

|  |
| --- |
| {% extends 'commonShow.html' %}  {% block content%}  hello user {{name}}  {% endblock %} |

Show2.html

|  |
| --- |
| {% extends 'commonShow.html' %}  {% block content%} hello user {{id}}  {% endblock %} |

# Request capturing:

P1 = Request.GET[‘n1’]

P2 = Request.POST[‘n1’]

Get Form:

|  |
| --- |
| <form action="/submit1" >  <table>  <tr>  <td><label id="deptLbl">Name:</label> </td>  <td><input type="text" name="name" /></td>  </tr>  <tr>  <td><label id="deptLbl">Age</label> </td>  <td><input type="text" name="age"/></td>  </tr>    <tr>  <td colspan="2">  <input id="submit" type="submit" value="Submit"/></td>  </tr>  </table>  </form> |

Post Form:

|  |
| --- |
| <form action="/submit1/" method="post">  {% csrf\_token %}  <!-- to avoid csrf attack ; by default django provides the csrfviewmiddelware-->  <table>  <tr>  <td><label id="deptLbl">Name:</label> </td>  <td><input type="text" name="name" /></td>  </tr>  <tr>  <td><label id="deptLbl">Age</label> </td>  <td><input type="text" name="age"/></td>  </tr>    <tr>  <td colspan="2">  <input id="submit" type="submit" value="Submit"/></td>  </tr>  </table> </form> |

# Configuring welcome page

path('', views.handleIndex,name="index page"), #welcome page

def handleIndex(request):  
 return render(request, "index.html",) # one page all urls

static files [images , css , javascripts] configuration in settings.py

|  |
| --- |
| STATIC\_URL = '/static/' STATICFILES\_DIR =[  os.path.join(BASE\_DIR,"login/staticpages") ] STATIC\_ROOT= os.path.join(BASE\_DIR,"mystatic") |

# Code in Templates:

<body>  
hello user  
{{data}}  
  
<br><br><br>  
  
{{data|truncatewords:5}} − This filter will truncate the string, so you will see only the first 80 words.  
<br><br><br>

{{data|lower}} − Converts the string to lowercase.

{{data|upper}}

{{data|length}} # for list, tuple, string  
<br><br><br>

{{data|escape|linebreaks}}  
</body>  
</html>

Default

|  |
| --- |
| p2= {{p2|default:"No data"}} |

## Firstof

|  |
| --- |
| {% if var1 %}  {{ var1 }} {% elif var2 %}  {{ var2 }} {% elif var3 %}  {{ var3 }} {% endif %}  Is equal to  {% firstof var1 var2 var3 %}  Or  {% firstof var1 var2 var3 "fallback value" %}  {% if messages|length >= 100 %}  You have lots of messages today! {% endif %} |

## If statement

|  |
| --- |
| The {% if %} tag evaluates a variable, and if that variable is “true” (i.e. exists, is not empty, and is not a false boolean value) the contents of the block are output:  {% if data %}  Data exists  {% else %}  data doesnot exists  {% endif %}  {% if athlete\_list %}  Number of athletes: {{ athlete\_list|length }} {% elif athlete\_in\_locker\_room\_list %}  Athletes should be out of the locker room soon! {% else %}  No athletes. {% endif %}  Find list length = {{ athlete\_list|length }}  {% if athlete\_list and coach\_list %}  Both athletes and coaches are available. {% endif %}  {% if not athlete\_list %}  There are no athletes. {% endif %}  {% if athlete\_list or coach\_list %}  There are some athletes or some coaches. {% endif %}  {% if not athlete\_list or coach\_list %}  There are no athletes or there are some coaches. {% endif %}  {% if athlete\_list and not coach\_list %}  There are some athletes and absolutely no coaches. {% endif %} |

## Conditions

|  |
| --- |
| {% if somevar == "x" %}  This appears if variable somevar equals the string "x" {% endif %}   {% if somevar != "x" %}  This appears if variable somevar does not equal the string "x",  or if somevar is not found in the context {% endif %}     {% if somevar < 100 %}  This appears if variable somevar is less than 100. {% endif %}   {% if somevar > 0 %}  This appears if variable somevar is greater than 0. {% endif %}    {% if somevar <= 100 %}  This appears if variable somevar is less than 100 or equal to 100. {% endif %}    {% if somevar >= 1 %}  This appears if variable somevar is greater than 1 or equal to 1. {% endif %}    {% if "bc" in "abcdef" %}  This appears since "bc" is a substring of "abcdef" {% endif %}  {% if "hello" in greetings %}  If greetings is a list or set, one element of which is the string  "hello", this will appear. {% endif %}  {% if user in users %}  If users is a QuerySet, this will appear if user is an  instance that belongs to the QuerySet. {% endif %}   {% if somevar is True %}  This appears if and only if somevar is True. {% endif %}  {% if somevar is None %}  This appears if somevar is None, or if somevar is not found in the context. {% endif %}     It is {% now "jS F Y H:i" %} It is the {% now "jS \o\f F" %} |

## For loop

|  |
| --- |
| <ul> {% for athlete in athlete\_list %}  <li>{{ athlete.name }}</li> {% empty %}  <li>Sorry, no athletes in this list.</li> {% endfor %} </ul>  forloop.counter The current iteration of the loop (1-indexed) forloop.counter0 The current iteration of the loop (0-indexed) forloop.revcounter The number of iterations from the end of the loop (1-indexed) forloop.revcounter0 The number of iterations from the end of the loop (0-indexed) forloop.first True if this is the first time through the loop forloop.last True if this is the last time through the loop forloop.parentloop For nested loops, this is the loop surrounding the current one |

## Includes:

Commons.html

|  |
| --- |
| <!DOCTYPE html> <html lang="en"> <head>  <meta charset="UTF-8">  <title>Title</title> </head> <body> welcome my dear <br/>  p1 = {{p1}} <br/> p2= {{p2}}  </body> </html> |

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>Title</title>  
</head>  
<body>  
  
{% include "common.html" %}  
  
<br/>  
hello  
  
{% include "common.html" with p1="testP1" p2="testP2" %}  
  
</body>  
</html>

# Model class

|  |
| --- |
| ImageField is a [FileField](https://www.geeksforgeeks.org/filefield-django-models/" \t "_blank) with uploads restricted to image formats only. Before uploading files, one needs to specify a lot of settings so that file is securely saved and can be retrieved in a convenient manner. The default form widget for this field is a [ClearableFileInput](https://docs.djangoproject.com/en/2.2/ref/forms/widgets/" \l "django.forms.ClearableFileInput" \t "_blank). In addition to the special attributes that are available for FileField, an ImageField also has height and width attributes. ImageField requires the Pillow library. To install the same run,  Pip install pillow  class GeeksModel(Model):      geeks\_field = models.ImageField() |

|  |
| --- |
| **Changes in Models.py**  from django.db import models   # Create your models here.   class Person(models.Model):  firstName = models.CharField(max\_length=30)  lastName = models.CharField(max\_length=30)  age = models.IntegerField  **changes in admin.py:**  from django.contrib import admin  # Register your models here. from django.contrib import admin from .models import Person  admin.site.register(Person) |

**Steps:**

|  |
| --- |
| **(test) C:\Users\i335484\myprojects\myProject>py manage.py makemigrations**  Migrations for 'login':  login\migrations\0001\_initial.py  - Create model Person  **(test) C:\Users\i335484\myprojects\myProject>py manage.py sqlmigrate login 0001**  BEGIN;  --  -- Create model Person  --  CREATE TABLE "login\_person" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "firstName" varchar(30) NOT NULL, "lastName" varchar(30) NOT NULL);  COMMIT;  **(test) C:\Users\i335484\myprojects\myProject>py manage.py sqlmigrate login 0002**  BEGIN;  --  -- Add field age to person  --  CREATE TABLE "new\_\_login\_person" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "age" integer NOT NULL, "firstName" varchar(30) NOT NULL, "lastName" varchar(30) NOT NULL);  INSERT INTO "new\_\_login\_person" ("id", "firstName", "lastName", "age") SELECT "id", "firstName", "lastName", -1 FROM "login\_person";  DROP TABLE "login\_person";  ALTER TABLE "new\_\_login\_person" RENAME TO "login\_person";  COMMIT;  **(test) C:\Users\i335484\myprojects\myProject>py manage.py migrate**  Operations to perform:  Apply all migrations: admin, auth, contenttypes, login, sessions  Running migrations:  No migrations to apply. |