

Day: 18

Date: 10/06/2024

Chapter: Forms in django, Topic: django forms

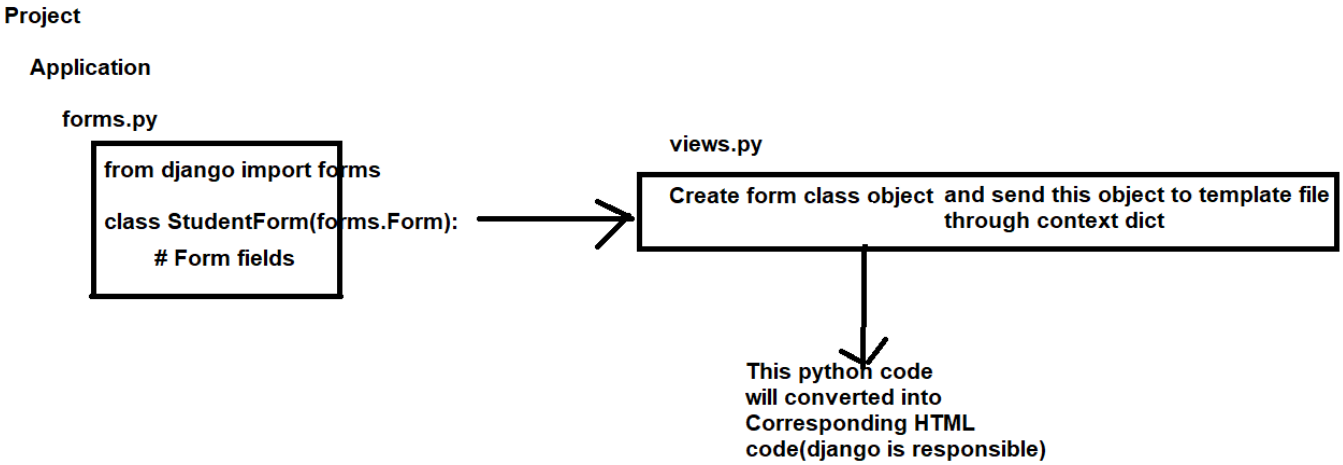
Form:

- Form is used to collect data from the end user.
- Form contains form elements like textbox, password box, checkbox, radio button etc.
- If a end-user want to insert records then the best way is to insert by using form why because end user does not know about these technical things like INSERT command, DAI (django admin interface) but by using a GUI form end user can enter the records.

Question: We can create the form by using HTML, but why do we go for django forms?

- ✓ With the help of django form we can develop form very easily.
- ✓ Here in django to develop form we will not write HTML code, instead of HTML code we will write python code. Behind the screen these python codes automatically converted into corresponding HTML code, and this is the responsibility of Django.
- ✓ We can validate forms very easily.

- If we want to develop a model the we have to define a python class and it must be the child of models.Model, similarly if we want to develop form then we have to define a python class and it must be the child class of forms.Form.
- If we want to define model, django provides models.py file but if we want to define form django does not provides forms.py file. It means as programmers we must create this forms.py file by our own in application level.



P018:

Problem Statement: Create a django based form.

Step 1: Common steps (folder creation, project, application and template creation and configuration)
Step 2: create forms.py file in application level.
Step 3: define form class
Forms.py

```
from django import forms
#Define your form class
class StudentForm(forms.Form):
    name=forms.CharField()
    age=forms.IntegerField()
    email=forms.EmailField()
    mark=forms.IntegerField()
    college=forms.CharField()
```

Step 4: Create form class object inside views.py
Views.py

```
from django.shortcuts import render
from .forms import StudentForm
# Create your views here.
def student_view(request):
    #create the object of form class
    form=StudentForm()
    d={'form':form}
    return render(request,'FormApp/forms.html',d)
```

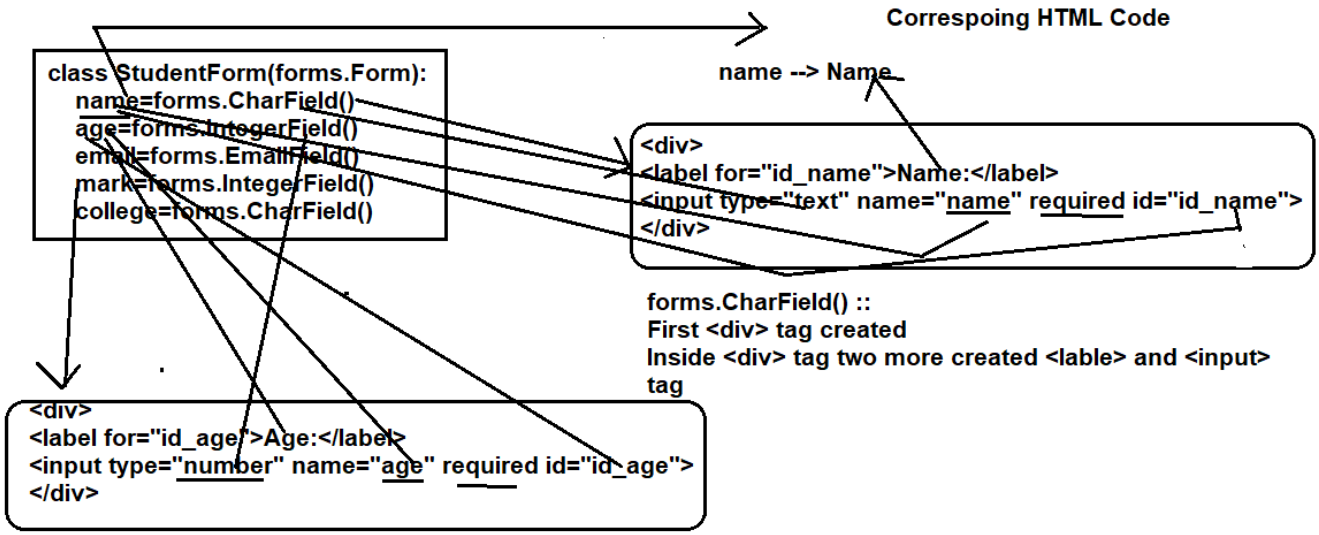
Step 5: forms.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
```

```
</head>
<body>
    {{ form }}
</body>
</html>
```

Step 7 : Run server and send HTTP request
http://127.0.0.1:8000/FormApp/forms/

Conversion of Python from class code into corresponding HTML code



P019:
Problem Statement: Use existing project(P0018) and fix the alignment problem.

What is the problem with django form?

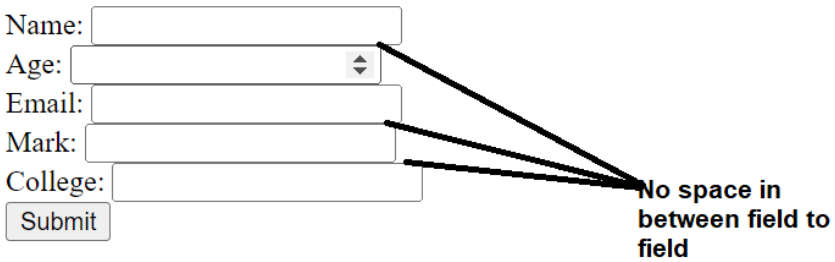
- ✓ Alignment problem
- ✓ By default, no submit button
- ✓ No form tag also

All these issues we have to fix by our own.

Add form tag and submit button
Form.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
     <form action="">
        {{ form }}
        <input type="submit">
     </form>
</body>
</html>
```

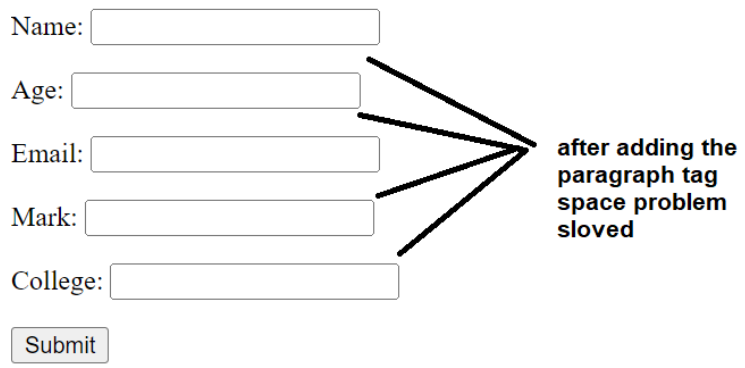
Add form tag and submit button and fix space issues using paragraph tag



Forms.html

```
<!DOCTYPE html>
<html lang="en">
<head>
```

```
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
     <form action="">
        {{ form.as_p }}
        <input type="submit">
     </form>
</body>
</html>
```



Generated corresponding HTML code

forms.py

```
from django import forms

#Define your form class

class StudentForm(forms.Form):
   name=forms.CharField()
   age=forms.IntegerField()
   email=forms.EmailField()
   mark=forms.IntegerField()
   college=forms.CharField()
```

form.html

```
<form action="">
      {{ form.as_p }}
      <input type="submit">
</form>
```

```
<form action="">
 <p>
  <label for="id_name">Name:</label>
  <input type="text" name="name" required id="id_name">
 </p>
 <p>
  <label for="id_age">Age:</label>
  <input type="number" name="age" required id="id_age">
 </p>
 <p>
  <label for="id_email">Email:</label>
  <input type="email" name="email" maxlength="320" required id="id_email">
 </p>
 <p>
  <label for="id_mark">Mark:</label>
  <input type="number" name="mark" required id="id_mark">
 </p>
 <p>
  <label for="id_college">College:</label>
  <input type="text" name="college" required id="id_college">
 </p>
 <input type="submit">
</form>
```

here <p> tag is comping because of we have printed form object as {{ form.as_p }}

Note ::
{{ form }} ---> add

{{form.as_p}} --> add <p> instread of

{{ Form.as_ul }}

```
<form action="">
      <ul>
         {{ form.as_ul }}
      </ul>
      <input type="submit">
</form>
```

```
<ul>

   <li>
   <lable>
   <input>
   </li>
   ....
   .....

</ul>
```

{{ Form.as_table }}


```
<form action="">
    <table>
    {{ form.as_table }}
    <tr>
      <td><input type="reset"></td>
      <td><input type="submit"></td>
    </tr>
    </table>
</form>
```

```
<form action="">
<table>
<tr>
 <th><label for="id_name">Name:</label></th>
 <td>
  <input type="text" name="name" required id="id_name">
 </td>
</tr>
<tr>
 <th><label for="id_age">Age:</label></th>
 <td>
  <input type="number" name="age" required id="id_age">
 </td>
</tr>
<tr>
 <th><label for="id_email">Email:</label></th>
 <td>
  <input type="email" name="email" maxlength="320" required id="id_email">
 </td>
</tr>
<tr>
 <th><label for="id_mark">Mark:</label></th>
 <td>
  <input type="number" name="mark" required id="id_mark">
 </td>
</tr>
<tr>
 <th><label for="id_college">College:</label></th>
 <td>
  <input type="text" name="college" required id="id_college">
 </td>
</tr>
<tr>
 <td><input type="reset"></td>
 <td><input type="submit"></td>
</tr>
</table>
</form>
```

Provide our own label name

```
<form action="">
  <table>
    <tr>
      <td>Enter your name : </td>
      <td>{{ form.name }}</td>
    </tr>
    <tr>
      <td>Enter your age : </td>
      <td>{{ form.age }}</td>
    </tr>
    <tr>
      <td>Enter your mark : </td>
      <td>{{ form.mark }}</td>
    </tr>
    <tr>
      <td>Enter your email : </td>
      <td>{{ form.email }}</td>
    </tr>
    <tr>
      <td>Enter your college : </td>
      <td>{{ form.college }}</td>
    </tr>
    <tr>
      <td><input type="reset" value="reset"> </td>
      <td><input type="submit" value="register"></td>
    </tr>
  </table>
</form>
```

Enter your name :
Enter your age :
Enter your mark :
Enter your email :
Enter your college :
reset register

```
<form action="">
  <table>
    <tr>
      <td>Enter your name : </td>
      <td><input type="text" name="name" required id="id_name"></td>
    </tr>
    <tr>
      <td>Enter your age : </td>
      <td><input type="number" name="age" required id="id_age"></td>
    </tr>
    <tr>
      <td>Enter your mark : </td>
      <td><input type="number" name="mark" required id="id_mark"></td>
    </tr>
    <tr>
      <td>Enter your email : </td>
      <td><input type="email" name="email" maxlength="320" required id="id_email"></td>
    </tr>
    <tr>
      <td>Enter your college : </td>
      <td><input type="text" name="college" required id="id_college"></td>
    </tr>
    <tr>
      <td><input type="reset" value="reset"> </td>
      <td><input type="submit" value="register"></td>
    </tr>
  </table>
</form>
```


Day: 19

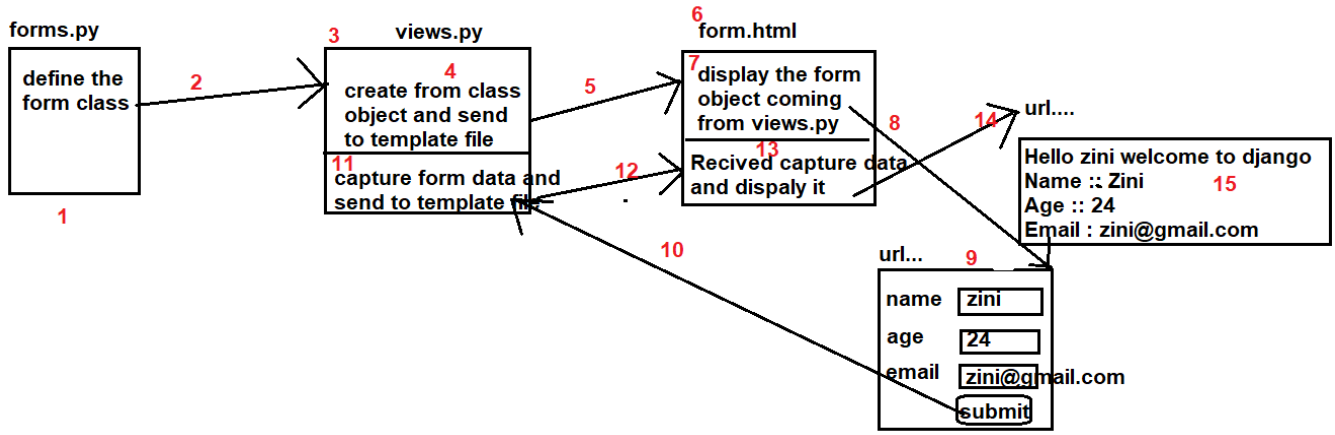
Date: 11/06/2024

Chapter: Forms in Django, Topic: Django form

Till now we have successfully created the django based form. Now we can enter data into the form then we can process this data inside views.py file.

P020 :

Problem Statement: Define form class, create the object inside views.py file send form object to templates file then display the form in browser, then end-user will fill data once the end-user click on submit button then capture this form data in views.py file. Then send this form captured data from views.py file to template file and display captured form itself.



All steps:

Step 1: Common step

Step 2: Create forms.py file in application level

Step 3: Define form class inside forms.py file

Step 4: Create form class object inside views.py file

Step 5: Send form class object from views.py to template file through context dict.

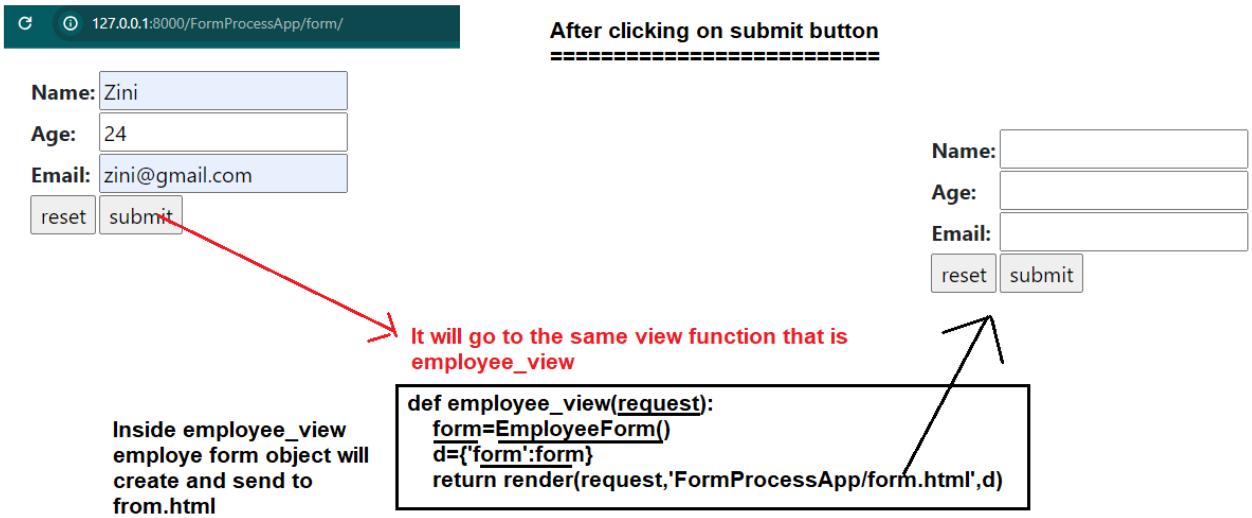
Step 6: Display form class object inside template file

Step 7: Run sever and send HTTP request

Step 8: Fill the data in the form

Step 9: Click on submit button then capture form data in views.py file.

Step 10: Send the captured form to template file and display captured form.



form.py

```
from django import forms
#define your form class

class EmployeeForm(forms.Form):
    name=forms.CharField()
    age=forms.IntegerField()
    email=forms.EmailField()
```

views.py

```
from django.shortcuts import render,HttpResponse
from .forms import EmployeeForm
# Create your views here.
def employee_view(request):
    if request.method=='POST':
        #capture the form data
        form=EmployeeForm(request.POST)
        d={'form':form}
        return render(request,'FormProcessApp/welcome.html',d)
    else:
        form=EmployeeForm()
        d={'form':form}
        return render(request,'FormProcessApp/form.html',d)
```

form.html

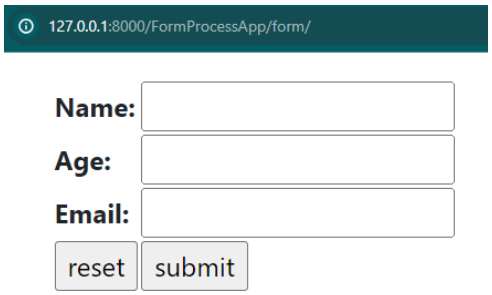
```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Bootstrap demo</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-
QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH"
crossorigin="anonymous">
  </head>
  <body>
    <div class="container mt-3">
        <form action="" method="post">
          {% csrf_token %}
            <table>
                {{ form.as_table }}
                <tr>
                    <td><input type="reset" value="reset"></td>
                    <td><input type="submit" value="submit"></td>
                </tr>
            </table>
        </form>
    </div>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"
integrity="sha384-
YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIdslK1eN7N6jIeHz"
crossorigin="anonymous"></script>
  </body>
</html>
```

welcome.html

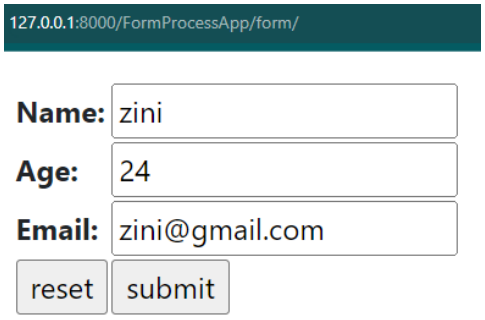
```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Bootstrap demo</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-
```

```
QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH"
crossorigin="anonymous">
  </head>
  <body>
    <div class="container mt-3">
        <form action="" method="post">
          {{ form }}
        </form>
        <a href="{% url 'form' %}">Click here Back</a>
    </div>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"
integrity="sha384-
YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIdslK1eN7N6jIeHz"
crossorigin="anonymous"></script>
  </body>
</html>
```

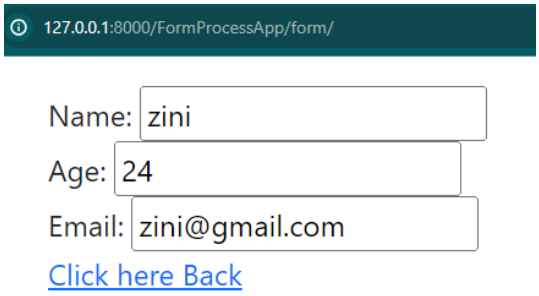
Step 11: Run server and send HTTP request
This request method is HTTP GET.



Then fill the data



Display captured form data

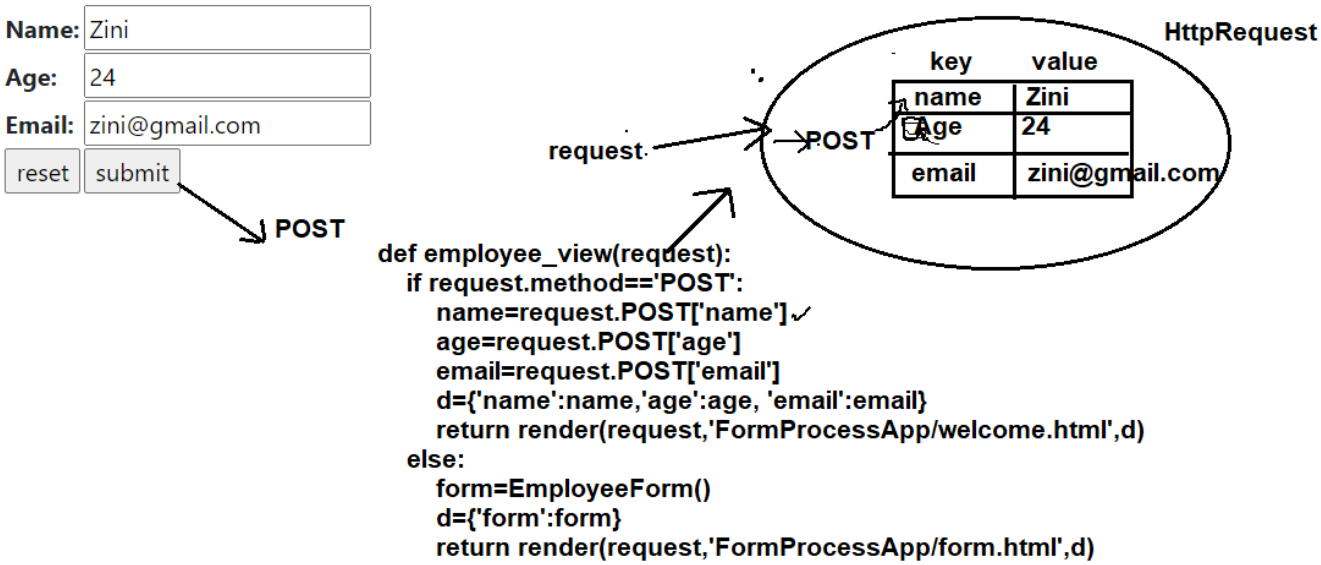


Day: 20
Chapter: Forms in django, Topic : Django Form

P021
Problem Statement : Define form class, create the object inside views.py file send form object to templates file then display the form in browser, then end-user will fill data once the end-user click on submit button then capture this form data in views.py file. Then send this form captured data from views.py file to template file and display name, age, and email.

All steps:
Step 1: Common step
Step 2: Create forms.py file in application level

Step 3: Define form class inside forms.py file
Step 4: Create form class object inside views.py file
Step 5: Create empty form object
Step 6: Display form class object inside template file
Step 7: Run sever and send HTTP request
Step 8: Fill the data in the form
Step 9: Click on submit button then capture form fields data in views.py file.
Step 10: Send the captured form fields to template file and display captured form.



Views.py

```
from django.shortcuts import render,HttpResponse
from .forms import EmployeeForm
# Create your views here.
def employee_view(request):
    if request.method=='POST':
        name=request.POST['name']
        age=request.POST['age']
        email=request.POST['email']
        d={'name':name,'age':age, 'email':email}
        return render(request,'FormProcessApp/welcome.html',d)
    else:
        form=EmployeeForm()
        d={'form':form}
```

```
    return render(request,'FormProcessApp/form.html',d)
```

welcome.html

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Bootstrap demo</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-
QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH"
crossorigin="anonymous">
  </head>
  <body>
    <div class="container mt-3">
        <h3>Name :  {{ name }} </h3>
        <h5>Age :  {{ age }} </h5>
        <h5>Email :  {{ email }} </h5>
        <a href="{% url 'form' %}">Click here Back</a>
    </div>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"
integrity="sha384-
YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIdslK1eN7N6jIeHz"
crossorigin="anonymous"></script>
  </body>
</html>
```

Capture form filed data in another way using dict get() method

```
from django.shortcuts import render,HttpResponse
from .forms import EmployeeForm
```

```
# Create your views here.
def employee_view(request):
    if request.method=='POST':
        name=request.POST.get('name')
        age=request.POST.get('age')
        email=request.POST.get('email')
        d={'name':name,'age':age, 'email':email}
        return render(request,'FormProcessApp/welcome.html',d)
    else:
        form=EmployeeForm()
        d={'form':form}
        return render(request,'FormProcessApp/form.html',d)
```

We can also capture HTTP GET request data

Views.py

```
from django.shortcuts import render,HttpResponse
from .forms import EmployeeForm
# Create your views here.
def employee_view(request):
    form=EmployeeForm()
    d={'form':form}
    return render(request,'FormProcessApp/form.html',d)

def employee_process_view(request):
    name=request.GET['name']
    age=request.GET['age']
    email=request.GET['email']
    d={'name':name,'age':age, 'email':email}
    return render(request,'FormProcessApp/welcome.html',d)
```

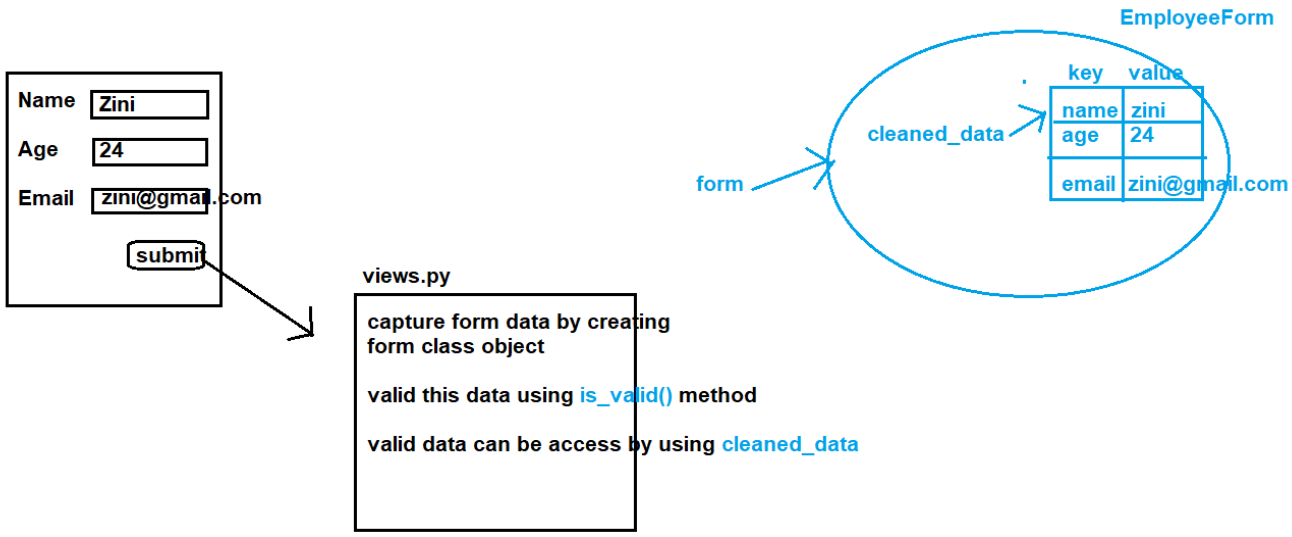
application level

```
from django.urls import path
from . import views
urlpatterns = [
    path('form/', views.employee_view, name='form'),
    path('emp_process/',views.employee_process_view,name='emp_process')
]
```

Capture form filed data in another way using dict get() method

```
from django.shortcuts import render,HttpResponse
from .forms import EmployeeForm
# Create your views here.
def employee_view(request):
    form=EmployeeForm()
    d={'form':form}
    return render(request,'FormProcessApp/form.html',d)

def employee_process_view(request):
    name=request.GET.get('name')
    age=request.GET.get('age')
    email=request.GET.get('email')
    d={'name':name,'age':age, 'email':email}
    return render(request,'FormProcessApp/welcome.html',d)
```

P022:
Problem Statement: Define form class, create the object inside views.py file send form object to templates file then display the form in browser, then end-user will fill data once the end-user click on submit button then capture this form data in views.py file. Then send this form captured data from views.py file to template file and display name, age, and email using cleaned_data .



Views.py

```
from django.shortcuts import render,HttpResponse
from .forms import EmployeeForm
# Create your views here.
def employee_view(request):
    if request.method=='POST':
        form=EmployeeForm(request.POST)
        if form.is_valid():
            name=form.cleaned_data['name']
            age=form.cleaned_data['age']
            email=form.cleaned_data['email']
            d={'name':name,'age':age,'email':email}
            return render(request,'FormProcessApp/welcome.html',d)
        else:
            return HttpResponse('Invalid data')
    else:
        form=EmployeeForm()
        d={'form':form}
    return render(request,'FormProcessApp/form.html',d)
```

Capture form data using dict get method

```
from django.shortcuts import render,HttpResponse
from .forms import EmployeeForm
# Create your views here.
def employee_view(request):
    if request.method=='POST':
        form=EmployeeForm(request.POST)
        if form.is_valid():
            name=form.cleaned_data.get('d')
            age=form.cleaned_data.get('age')
            email=form.cleaned_data.get('email')
            d={'name':name,'age':age,'email':email}
            return render(request,'FormProcessApp/welcome.html',d)
        else:
            return HttpResponse('Invalid data')
    else:
        form=EmployeeForm()
        d={'form':form}
    return render(request,'FormProcessApp/form.html',d)
```

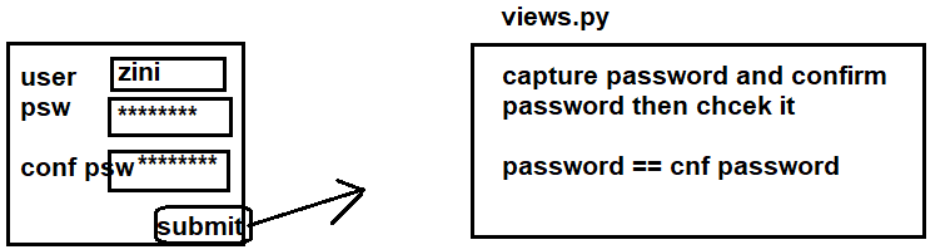
Note

```
print(type(request.POST)) -><class 'django.http.request.QueryDict'>
print(type(form))  -><class 'FormProcessApp.forms.EmployeeForm'>
print(type(form.cleaned_data)) -> <class 'dict'>
```

Passing argument in form fields:
Syntax : name=forms.CharFiled(argument)

```
from django import forms

#define your form class
class StudentForm(forms.Form):
    name=forms.CharField(required=False)
    age=forms.IntegerField(label_suffix="-")
    mark=forms.IntegerField(label="hello")
    email=forms.EmailField(required=False,label='Mail-ID')
    country=forms.CharField(initial='India',disabled=True)
    bio=forms.CharField(widget=forms.Textarea,help_text='Maximun 120
char',max_length=120)
    upload_cv=forms.FileField()
    upload_photo=forms.ImageField()
    iaggree=forms.CharField(widget=forms.CheckboxInput)
```

P024:
Problem Statement: Enter password and confirm password then check both are same or not.



form.py

```
from django import forms

#define your form class
class StudentForm(forms.Form):
    name=forms.CharField()
```

```
    password=forms.CharField(widget=forms.PasswordInput)
    confirm_password=forms.CharField(widget=forms.PasswordInput)
```

Views.py

```
from django.shortcuts import render,HttpResponse
from .forms import StudentForm
# Create your views here.
def student_form_view(request):
    if request.method=='POST':
        form=StudentForm(request.POST)
        if form.is_valid():
            password=form.cleaned_data['password']
            confirm_password=form.cleaned_data['confirm_password']
            if password==confirm_password:
                d={'matched':True}
                return render(request,'FormFieldArgumentApp/welcome.html',d)
            else:
                d={'matched':False}
                return render(request,'FormFieldArgumentApp/welcome.html',d)
    form=StudentForm()
    d={'form':form}
    return render(request,'FormFieldArgumentApp/home.html',d)
```

home.html

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
```

```
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Bootstrap demo</title>
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.cs
s" rel="stylesheet" integrity="sha384-
QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH"
crossorigin="anonymous">
  </head>
  <body>
    <div class="container mt-3">
        <form action="" method='POST'>
          {% csrf_token %}
            <table>
                {{ form.as_table }}
                <tr>
                    <td><input type="submit" value="submit"></td>
                </tr>
            </table>
        </form>
    </div>
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.m
in.js" integrity="sha384-
YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIdslK1eN7N6jIeHz"
crossorigin="anonymous"></script>
  </body>
</html>
```

Welcome.html

```
<!doctype html>
<html lang="en">
```

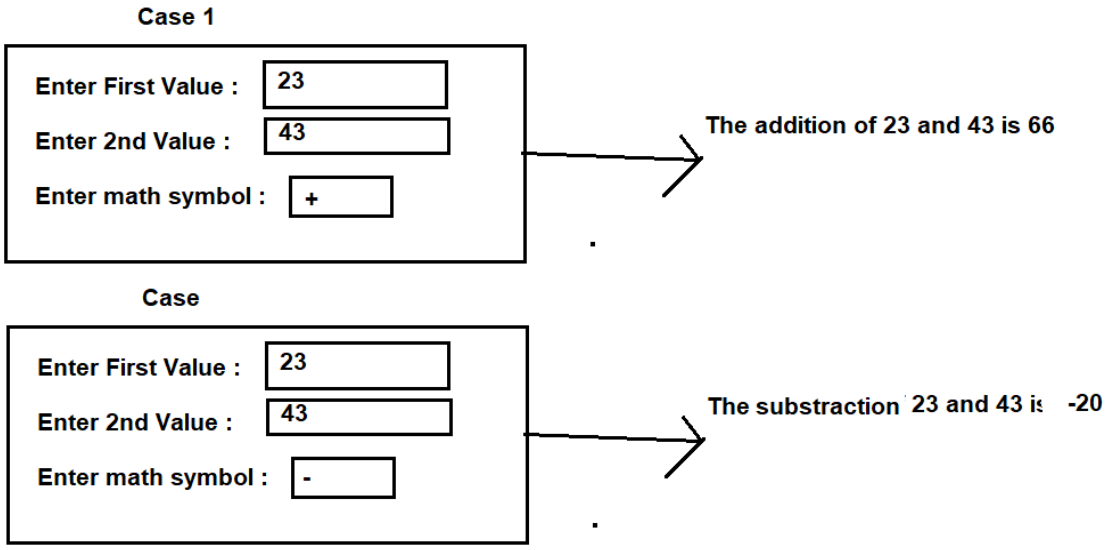
```
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Bootstrap demo</title>
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.cs
s" rel="stylesheet" integrity="sha384-
QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH"
crossorigin="anonymous">
  </head>
  <body>
    <div class="container mt-3">
       {% if matched %}
       <h1>Both password matched</h1>
       {% else %}
       <h1>Both password does not matched</h1>
       <a href="{% url 'form' %}">Go to Registraction Page</a>
       {% endif %}
    </div>
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.m
in.js" integrity="sha384-
YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIdslK1eN7N6jIeHz"
crossorigin="anonymous"></script>
  </body>
</html>
```

Application level url

```
from django.urls import path
```

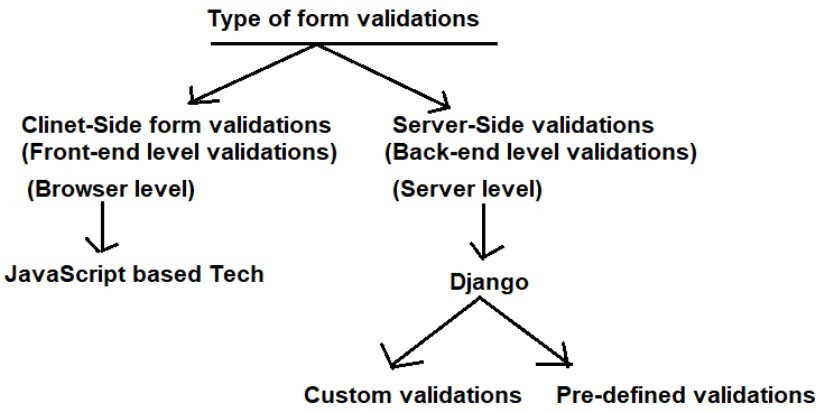
```
from . import views
urlpatterns = [
    path('form/',views.student_form_view, name='form'),
]
```

Assignment:
Problem Statement: Enter first and second value then enter math symbol then display result



At the time of data collection from the form there is huge change to get invalid data from the user just like first name 123 last name abc45, mobile number is 111, address is @25 etc.

Advantages of validations:
There is huge chance to get valid data from the user.

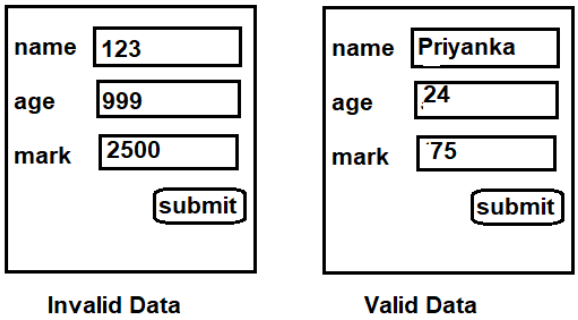


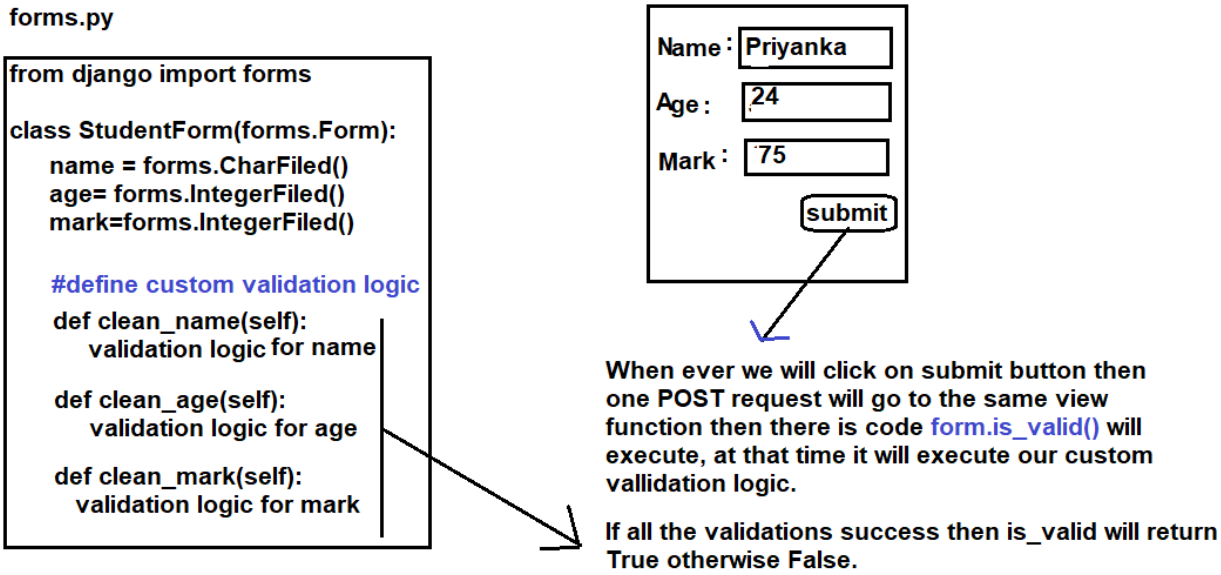
Note: server-side validations are of two types first one custom validation and second one pre-defined validations.

(i) <u>Custom validation:</u>

a. As programmers we must write validation logic by our own.
b. All the validation logic we must write inside form class in forms.py file.
c. For custom validation we will use clean() method or clean_field_name() method.

<u>Custom validations using clean_field_name():</u>





P025:

Problem Statement: Define a StudentForm class having 3 fields named as name, age and mark.

Apply validation to each filed.

Validation Rule :

Rule 1: Name length must be in between 2 to 16.

Rule 2: Age must be in between 18 to 23.

Rule 3: Mark must be in between 60 to 100.

Task:

Validation rule: mobile number must be 10 digits.

Name: only alphabet

Password: mix (min 1 upper + 1 lower case + 1 digit and 1 special char)

Etc.

Take 6 fields and apply validation to each field.

Day: 23

Date: 21/06/2024

Chapter: Forms in django, Topic: django forms validations

(ii) Pre-define/inbuilt validators

a. Pre-define or inbuilt validations means the validation logic already written by django.
b. As a programmer we must use it without writing the validation logic.
c. All these pre-define validators are available inside django.core module.
d. In custom validations we have to define clean_field() method for validations, but in inbuilt validators we have to use validators along with field name.

Custom validations	Inbuilt validators
class StudentForm(forms.Form): #define form fields #define claen_filed() method #for validation logic	class StudentForm(forms.Form): #define form fields #we have to write use validators #along with form filed name ~~#define claen_filed() method~~ ~~#for validation logic~~

P026

Problem Statement: Form validations using inbuilt validators. Validate user name field

Rule :: Name length must be greater than or equal 2 and less than or equal16.

form.py

```
from django import forms
from django.core import validators
#define your form class
class StudentForm(forms.Form):

name=forms.CharField(validators=[validators.MinLengthValidator(2),validators
.MaxLengthValidator(16)])
```

```
    age=forms.IntegerField()
    mark=forms.IntegerField()
```

Example 2: Custom validations with inbuilt validators
Problem Statement: Form validations using inbuilt validators. Validate username field
Rule 1: Name length must be greater than or equal to 2 and less than or equal16.
Rule 2: Second letter must be ‘u’

```
class StudentForm(forms.Form):

name=forms.CharField(validators=[validators.MinLengthValidator(2),validators
.MaxLengthValidator(16)])
    age=forms.IntegerField()
    mark=forms.IntegerField()

    def clean_name(self):
        name=self.cleaned_data['name']
        if name[1]=='u' or name[1]=='U':
            return name
        else:
            raise forms.ValidationError('Name second char must be u')
```

Example 3: Custom validations with inbuilt validators
Problem Statement: Form validations using inbuilt validators. Validate username field
Rule 1: Name length must be greater than or equal to 2 and less than or equal16.
Rule 2: Second letter must be ‘u’
Rule 3: Mark must be greater than 50 and less than or equal to 100.

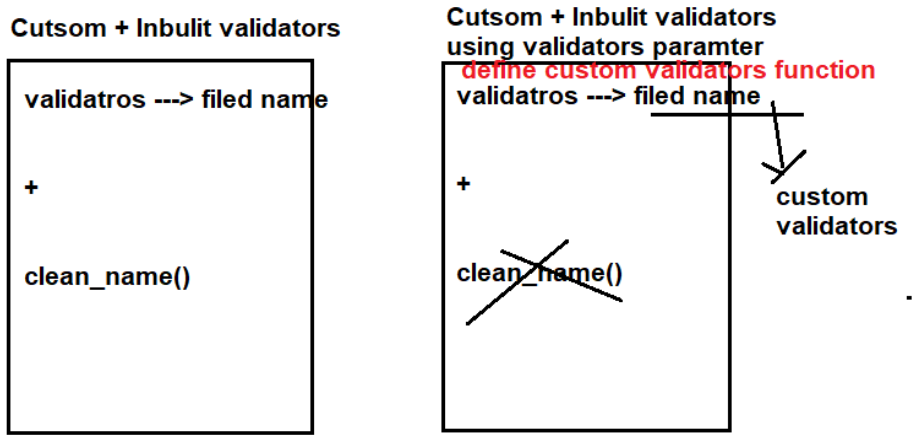
```
from django import forms
from django.core import validators
#define your form class

class StudentForm(forms.Form):
```

```
name=forms.CharField(validators=[validators.MinLengthValidator(2),validators
.MaxLengthValidator(16)])
    age=forms.IntegerField()

mark=forms.IntegerField(validators=[validators.MinValueValidator(50),validat
ors.MaxValueValidator(100)])

    def clean_name(self):
        name=self.cleaned_data['name']
        if name[1]=='u' or name[1]=='U':
            return name
        else:
            raise forms.ValidationError('Name second char must be u')
```

Example 4: inbuilt validators
Problem Statement: Form validations using inbuilt validators. Validate email field

```
from django import forms
from django.core import validators
#define your form class

class StudentForm(forms.Form):
    name=forms.CharField()
    age=forms.IntegerField()
    mark=forms.IntegerField()
    email=forms.EmailField(validators=[validators.EmailValidator()])
```

Note :: We can also define custom validation logic using validators parameter.



Example 5: inbuilt validators + custom validators using validators parameter
Problem Statement: Validate name fields.
Rule 1: Name length must be greater than or equal to 2 and less than or equal16.
Rule 2: Second letter must be ‘u’.

```
from django import forms
from django.core import validators

def second_letter_u(name):
    if name[1]!='u':
       raise forms.ValidationError('Name second letter must be u')

#define your form class

class StudentForm(forms.Form):

name=forms.CharField(validators=[validators.MinLengthValidator(2),validators
.MaxLengthValidator(16),second_letter_u])
    age=forms.IntegerField()
    mark=forms.IntegerField()
```