Step1:-

Create forms.py file in app folder

```
from django import forms
from django.core.validators import MinValueValidator, MaxValueValidator


class MyForm(forms.Form):
    name = forms.CharField(max_length=100, required=True)
    email = forms.EmailField()
    age = forms.IntegerField(
        validators=[
            MinValueValidator(0, message="Age must be a positive number."),
            MaxValueValidator(120, message="Age must be 120 or below.")
        ]
    )
    website = forms.URLField(
        required=False,
        label="(optional)",
        help_text="Enter a valid URL (e.g., https://www.example.com)")
```

now work on views.py: -

```
from .forms import MyForm
```

```
def myform(request):
    if request.method == 'POST':
        form = MyForm(request.POST)
        if form.is_valid():

            name = form.cleaned_data['name']
            email = form.cleaned_data['email']
            age = form.cleaned_data['age']
            return render(request, 'done.html', {'name': name})
    else:
        form = MyForm()
    return render(request, 'form.html', {'form': form})
```

now create a html file (done.html) and form.html:-

form.html

```
<form method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <button type="submit">Submit</button>
</form>
```

done.html

```
<h1>Success!</h1>
<p>Thank you, {{ name }}, for submitting the form.</p>
```

Now for accessing the file through the url set this url:-

```
from web.views import *
```

```
path('form/', myform, name='myform'),
```

now run the server and check url then submit the code ……..

There are several ways to validate a form in Django, each serving a specific purpose or scenario. Here are some common approaches to form validation in Django:

Field Validation:

This is the most basic form of validation and involves validating individual fields in a form. You can use field-specific attributes like required, min_length, max_length, min_value, max_value, and more to enforce validation rules directly on the form fields.

Custom Field Validation Methods:

You can define custom validation methods for individual fields by creating methods named clean_<fieldname> in your form class. These methods allow you to perform field-specific validation and raise validation errors if necessary.

Form-Level Validation:

Form-level validation involves checking conditions that depend on multiple fields. To perform form-level validation, override the clean() method in your form class. This method should call super().clean() to ensure that field-specific validation is also performed.

Using Validators:

Django provides a variety of built-in validators that you can use to validate fields. These can be added to a field using the validators attribute. For example, you can use EmailValidator, MaxLengthValidator, and more.

Model Form Validation:

If you're working with model forms (forms based on Django models), you can leverage the built-in model validation and constraints defined in your database schema. Model forms will automatically validate fields based on the model's field types and constraints.

Form Field Widgets:

While not a direct validation method, using appropriate form field widgets can help prevent certain types of incorrect data entry. For example, using a DateInput widget for a date field limits user input to valid date formats.

Third-Party Libraries:

You can use third-party validation libraries or packages to enhance your form validation process. For example, the Django django-crispy-forms package can help you create stylish forms with error messages and field validation.

AJAX Validation:

You can use AJAX techniques to perform asynchronous validation on the client-side and provide instant feedback to users as they fill out the form. However, server-side validation should always be performed to ensure data integrity.

Regular Expressions:

In cases where you need complex pattern matching, you can use regular expressions to validate form fields. This can be particularly useful for validating email addresses, phone numbers, or other custom formats.

Cleaned Data Transformation:

You can transform and manipulate cleaned data after validation, for example, converting a user's input to a specific format before storing it.

It's important to select the appropriate validation methods based on the specific requirements and complexity of your forms. Using a combination of these approaches can help you create a robust and user-friendly form validation process in your Django applications.