Cleaning and Validating Specific Field

clean_<fieldname>() - This method is called on a form subclass where <fieldname> is replaced with the name of the form field attribute.

This method does any cleaning that is specific to that particular attribute, unrelated to the type of field that it is.

This method is not passed any parameters.

You will need to look up the value of the field in self.cleaned_data and remember that it will be a Python object at this point, not the original string submitted in the form.

Cleaning and Validating Specific Field

forms.py

```
from django import forms
class StudentRegistration(forms.Form):
    name=forms.CharField()
    email=forms.EmailField()
    password=forms.CharField(widget=forms.PasswordInput)
    def clean_name(self):
        valname = self.cleaned_data['name']   //        valname = self.cleaned_data.get('name')
        if len(valname) < 4:
            raise forms.ValidationError('Enter more than or equal 4')
        return valname
```

Validation of Complete Django Form at once

clean() – The clean() method on a Field subclass is responsible for running to_python(), validate(), and run_validators() in the correct order and propagating their errors.

If, at any time, any of the methods raise ValidationError, the validation stops and that error is raised.

This method returns the clean data, which is then inserted into the cleaned_data dictionary of the form.

Implement a clean() method on your Form when you must add custom validation for fields that are interdependent.

Syntax:- Form.clean()

Validation of Complete Django Form at once

forms.py

```
from Django import forms
class StudentRegistration(forms.Form):
    name=forms.CharField()
    email=forms.EmailField()
    def clean(self):
        cleaned_data = super().clean()
        valname=self.cleaned_data['name']
        if len(valname)<4:
            raise forms.ValidationError('Name should be more than or equal 4')
        valemail=self.cleaned_data['email']
        if len(valemail)<10:
            raise forms.ValidationError('Email should be more than or equal 10')
```

Using Built-in Validators

We can use Built-in Validators, available in django.core module.

forms.py

```
form django.core import validators
from django import forms
class StudentRegistration(forms.Form):
    name=forms.CharField(validators=[validators.MaxLengthValidator(10)])
    email=forms.EmailField()
```

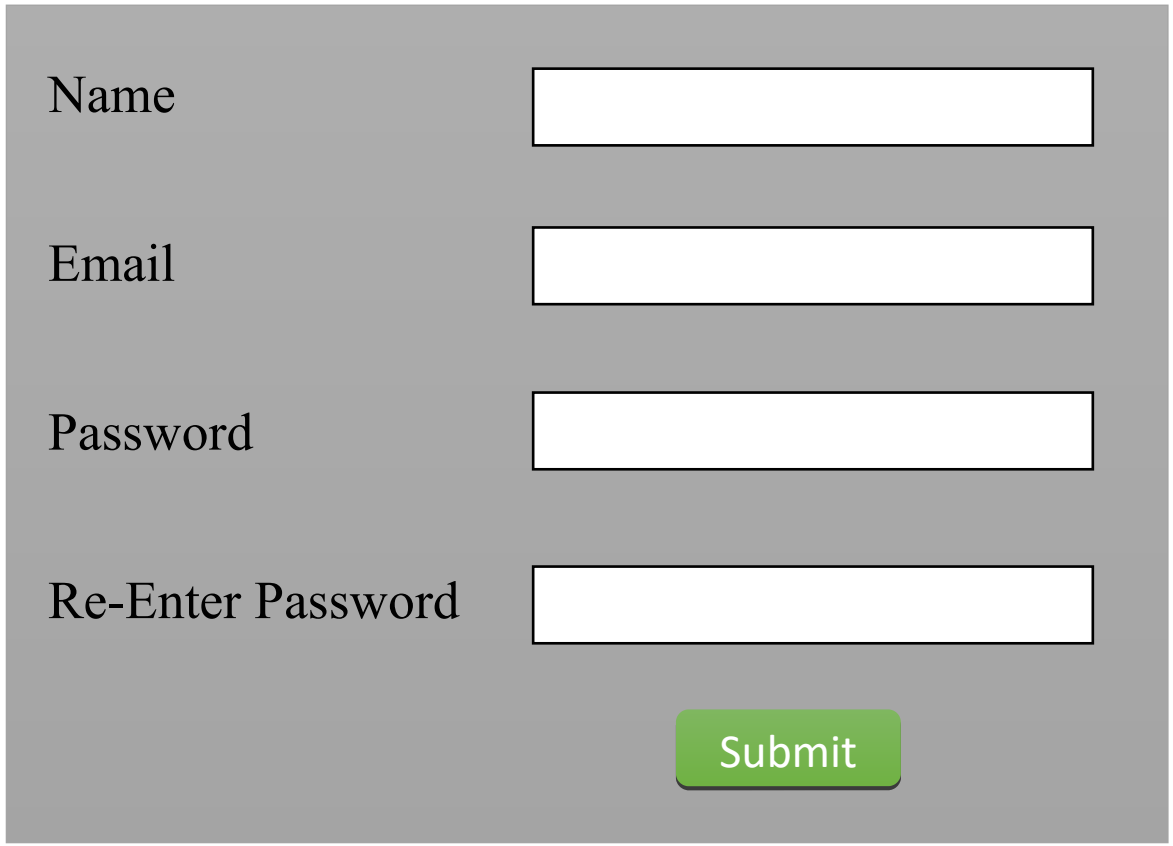
Create Custom Form Validators

forms.py

```
form django.core import validators
from django import forms
def starts_with_s(value):
    if vaule[0] != 's':
        raise forms.ValidationError('Name should start with s')

class StudentRegistration(forms.Form):
    name=forms.CharField(validators=[starts_with_s])
    email=forms.EmailField()
```

Form Validation – Match Two field value



Form Validation – Match Two field value

forms.py

```
from Django import forms
class StudentRegistration(forms.Form):
    name=forms.CharField()
    password=forms.CharField(widget=forms.PasswordInput)
    rpassword=forms.CharField(widget=forms.PasswordInput)
    def clean(self):
        cleaned_data = super().clean()
        valpwd=cleaned_data['password']          // cleaned_data.get('password')
        valrpwd=cleaned_data['rpassword']
        if valpwd != valrpwd :
            raise forms.ValidationError('Password Not Matched')
```

The call to super().clean() ensures that any validation logic in parent classes is maintained.