Form API

forms.py

```
from django.core import validators
from django import forms
class StudentRegistration(forms.Form):
 name = forms.CharField()
 email = forms.EmailField()
 password = forms.CharField(widget=forms.PasswordInput)
```

models.py

```
from django.db import models
class User(models.Model):
 name = models.CharField(max_length=70)
 email = models.EmailField(max_length=100)
 password = models.CharField(max_length=100)
```

views.py

```
def showformdata(request):
 if request.method == 'POST':
  fm = StudentRegistration(request.POST)
  if fm.is_valid():
   nm = fm.cleaned_data['name']
   em = fm.cleaned_data['email']
   pw = fm.cleaned_data['password']
   reg = User(name=nm, email=em, password=pw)
   reg.save()
```

templatefile.html

```
<form action="" method="POST" novalidate>
 {% csrf_token %}
  {{form.as_p}}
  <input type="submit" value="Submit">
 </form>
```

Model Form

forms.py

```
from django.core import validators
from django import forms
class StudentRegistration(forms.Form):
 name = forms.CharField()
 email = forms.EmailField()
 password = forms.CharField(widget=forms.PasswordInput)
```

```
from .models import User
class StudentRegistration(forms.ModelForm):
 class Meta:
  model = User
  fields = ['name', 'password', 'email']
```

models.py

```
from django.db import models
class User(models.Model):
 name = models.CharField(max_length=70)
 email = models.EmailField(max_length=100)
 password = models.CharField(max_length=100)
```

views.py

```
def showformdata(request):
 if request.method == 'POST':
  fm = StudentRegistration(request.POST)
  if fm.is_valid():
   nm = fm.cleaned_data['name']
   em = fm.cleaned_data['email']
   pw = fm.cleaned_data['password']
   reg = User(name=nm, email=em, password=pw)
   reg.save()
```

templatefile.html

```
<form action="" method="POST" novalidate>
 {% csrf_token %}
  {{form.as_p}}
  <input type="submit" value="Submit">
 </form>
```

Model Form

Django provides a helper class that lets you create a Form class from a Django model. This helper class is called as ModelForm.

ModelForm is a regular Form which can automatically generate certain fields.

The fields that are automatically generated depend on the content of the Meta class and on which fields have already been defined declaratively.

Steps:-

- Create Model Class
- Create ModelForm Class

Syntax:-

forms.py

```
class ModelFormClassName(forms.ModelForm):
	class Meta:
		model = ModelClassName
		fields = [‘fieldname1', ‘fieldname2', ‘fieldname3’]
		fields = (‘fieldname1', ‘fieldname2', ‘fieldname3’)
```

```
class Registration(forms.ModelForm):
	class Meta:
		model = User
		fields = [‘name’, ‘password’, ‘email']
```

Model Form

Model Field	Form Field
AutoField	Not Represented in the Form
BigAutoField	Not Represented in the Form
BigIntegerField	IntegerField with min_value set to -9223372036854775808 and max_value set to 9223372036854775807.
BinaryField	CharField, if editable is set to True on the model field, otherwise not represented in the form.
BooleanField	BooleanField, or NullBooleanField if null=True.
CharField	CharField with max_length set to the model field's max_length and empty_value set to None if null=True.
DateField	DateField
DateTimeField	DateTimeField
DecimalField	DecimalField
DurationField	DurationField

Model Form

Model Field	Form Field
EmailField	EmailField
FileField	FileField
FilePathField	FilePathField
FloatField	FloatField
ForeignKey	ModelChoiceField
ImageField	ImageField
IntegerField	IntegerField
IPAddressField	IPAddressField
GenericIPAddressField	GenericIPAddressField
ManyToManyField	ModelMultipleChoiceField
NullBooleanField	NullBooleanField
PositiveIntegerField	IntegerField

Model Form

Model Field	Form Field
PositiveSmallIntegerField	IntegerField
SlugField	SlugField
SmallAutoField	Not represented in the form
SmallIntegerField	IntegerField
TextField	CharField with widget=forms.Textarea
TimeField	TimeField
URLField	URLField
UUIDField	UUIDField

Model Form

- If the model field has *blank=True*, then required is set to *False* on the form field. Otherwise, *required=True*.
- The form field's *label* is set to the *verbose_name* of the model field, with the first character capitalized.
- The form field's *help_text* is set to the *help_text* of the model field.
- If the model field has *choices* set, then the form field's widget will be set to *Select*, with choices coming from the model field's choices. The choices will normally include the blank choice which is selected by default. If the field is required, this forces the user to make a selection. The blank choice will not be included if the model field has *blank=False* and an explicit *default* value (the default value will be initially selected instead).

Model Form

```
class Registration(forms.ModelForm):
    class Meta:
        model = User
        fields = [‘name’, ‘password’, ‘email’]
        labels = {‘name’: ‘Enter Name’, ‘password’: ‘Enter Password’, ‘email’: ‘Enter Email’ }
        help_text = {‘name’: ‘Enter Your Full Name’ }
        error_messages = {‘name’: {‘required’: ‘Naam Likhna Jaruri Hai’},
                          ‘password’: {‘required’: ‘Password Likhna Jaruri Hai’} }
        widgets = { ‘password’: forms.PasswordInput,
                    ‘name’: forms.TextInput(attrs={‘class’: ‘myclass’, ‘placeholder’: ‘Yaha Naam likhe’}),  }
```

Model Form

```
class Registration(forms.ModelForm):
    name = forms.CharField(max_length=50, required=False)
    class Meta:
        model = User
        fields = [‘name’, ‘password’, ‘email’]
        labels = {‘name’: ‘Enter Name’, ‘password’: ‘Enter Password’, ‘email’: ‘Enter Email’ }
        widgets = {‘password’:forms.PasswordInput}
```

save ( ) Method

save (commit=False/True) Method - This method creates and saves a database object from the data bound to the form.

A subclass of ModelForm can accept an existing model instance as the keyword argument instance, if this is supplied, save() will update that instance.

If it's not supplied, save() will create a new instance of the specified model.

If the form hasn't been validated, calling save( ) will do so by checking form.errors.

Syntax:- save (commit=False/True)

If *commit=False*, then it will return an object that hasn't yet been saved to the database. This is useful if you want to do custom processing on the object before saving it, or if you want to use one of the specialized model saving options.

If your model has a many-to-many relation and you specify *commit=False* when you save a form, Django cannot immediately save the form data for the many-to-many relation. This is because it isn't possible to save many-to-many data for an instance until the instance exists in the database.

Selecting Fields

- Set the fields attribute to field names

```
class Registration(forms.ModelForm):
    class Meta:
        model = User
        fields = [‘name’, ‘password’, ‘email’]
```

- Set the fields attribute to the special value __all__ to indicate that all fields in the model should be used.

```
class Registration(forms.ModelForm):
    class Meta:
        model = User
        fields = ‘__all__’
```

Selecting Fields

- Set the exclude attribute of the ModelForm’s inner Meta class to a list of fields to be excluded from the form.

```
class Registration(forms.ModelForm):
    class Meta:
        model = User
        exclude = [‘name’]
```

ModelForm Inheritance

You can extend and reuse ModelForms by inheriting them. This is useful if you need to declare extra fields or extra methods on a parent class for use in a number of forms derived from models.

You can also subclass the parent's Meta inner class if you want to change the Meta.fields or Meta.exclude lists.

```
class StudentRegistration(forms.ModelForm):
 class Meta:
  model = User
  fields = ['student_name', 'email', 'password']


class TeacherRegistration(StudentRegistration):
 class Meta(StudentRegistration.Meta):
  fields = ['teacher_name', 'email', 'password']
```

```
class User(models.Model):
 student_name = models.CharField(max_length=100)
 teacher_name = models.CharField(max_length=100)
 email = models.EmailField(max_length=100)
 password = models.CharField(max_length=100)
```

ModelForm Inheritance

- Normal Python name resolution rules apply. If you have multiple base classes that declare a Meta inner class, only the first one will be used. This means the child's Meta, if it exists, otherwise the Meta of the first parent, etc.

- It's possible to inherit from both Form and ModelForm simultaneously, however, you must ensure that ModelForm appears first in the MRO. This is because these classes rely on different metaclasses and a class can only have one metaclass.

- It's possible to declaratively remove a Field inherited from a parent class by setting the name to be None on the subclass.