Files

settings.py

- [x] **BASE_DIR**
  - [x] Directory where manage.py exists.
  - [x] This allows to work relative to directory.
  - [x] You can try to print BASE_DIR and runserver.
- [x] **SECRET_KEY**
  - [x] Should be unique to each project.
  - [x] Modify few characters if using someone else project.
- [x] **DEBUG**
  - [x] Shows details for debugging
  - [x] Should be changed to False when in production.
- [x] **ALLOWED_HOSTS**
  - [x] Allowed domain names and ips.
  - [x] Used as security measure in production.
- [x] **INSTALLED_APPS**
  - [x] Components used in the whole project.
  - [x] Remember to add all apps you create and also third-party apps you install in this list.
- [x] **MIDDLEWARE**
  - [x] Manages how requests are handled and securities are handled.
- [x] **ROOT_URLCONF**
  - [x] Tells django how to manage routes.
- [x] **TEMPLATES**
  - [x] How are html templates rendered, where are they stored.
  - [x] In DIRS list, add os.path.join(BASE_DIR, "templates").
- [x] **WSGI_APPLICATION**
  - [x] Tells django how to use servers.
  - [x] Sometimes we may need to change it.
- [x] **DATABASES**
  - [x] Which database engine used and where is database stored.
  - [x] By default uses sqlite3 database.
    - [x] Change database name to create new database. Eg: change name to db2.sqlite3.
- [x] **AUTH_PASSWORD_VALIDATORS**
  - [x] Which password validators are applied.
- [x] **STATIC_URL**
  - [x] Talk about later.

models.py

> In docs, arguments given in fields are required arguments. When adding new field, either do null=True or provide some default value(Eg. default="default value").

- [x] **CharField**
  - [x] Must have max_length=120 argument.
- [x] **TextField**

    - [x] blank=False : Makes field as required while taking input.
    - [x] null=True : Makes field nullable in database.
- [x] **DecimalField**
    - [x] decimal_places=2 is required.
    - [x] max_digits=1000 is required.
- [x] **BooleanField**
- [ ] **FileField**
    - [ ] upload_to="images/"

Commands

manage.py

- [x] **runserver**
    - [x] Starts a development server.
    - [x] You can allow the server to keep running and do all changes in another terminal, including migrations.
- [x] **makemigrations** and **migrate**
    - [x] Updates database.
    - [x] Both commands are run together in sequence.
    - [x] Run these upon any change in models.py.
    - [x] To reset database,
        1. Delete all files in migrations folder (except __init__.py)
        2. Delete __pycache__ folder in migrations directory.
        3. Delete db.sqlite3 file.
- [x] **createsuperuser**
    - [x] Allows to create a superuser to login into admin page (urls/admin).
- [x] **startapp appname**
    - [x] Creates new app (component in project).
    - [x] An app does one thing very good.
    - [x] You need to add it in INSTALLED_APPS list.
- [x] **shell**
    - [x] Allows you to import models and manipulate data to database using the model.
    - [x] Eg. >>> from products.models import Product >>> Product.objects.all() >>> Product.objects.create(name="Watch", price=22)

views.py

Functional Views

- [x] Need to add views in urls.py.
- [x] Takes a request object as argument.
- [x] Conventionally, functions end with _view.
- [x] Add *args, **kwargs also as arguments in function definitions.
- [x] Returns either HttpResponse or render(request, template_name, context_dictionary)
- [x] Convention is to pass model objects as 'object' in context, and then access the attributes from it.
- [x] To use forms, Eg:

```
from .forms import ProductForm
def product_detail_view(request):
  form = ProductForm(request.POST or None)
  if form.is_valid():
    form.save()
  context['form'] = form
```

- [x] form.cleaned_data can be used to clean data.
- [x] form.errors can be used to view errors.

request Object

> Request object is also accessible in html templates.

- [x] .user
  - [x] Gives username of user logged in.
  - [x] If no one is logged in, it gives AnonymousUser.
  - [x] .is_authenticated (in template)
- [x] .method
  - [x] can have value 'GET', 'POST' or few other methods.
- [x] .GET dictionary that contains data sent through get request.
- [x] .POST dictionary contains data sent through post request.

ModelName.objects

- [x] .get(id=[number])
  - [ ] This must return exactly one object.
- [x] .create(**dictionary) or .create(attribute1=value1, attribute2=value2 ...)
- [ ] .filter(attr1=value1, attr2=value2)
  - [ ] returns a list of objects.
- [ ] model_object.save() can be used to save the model_objects.

urls.py

- [x] Best practice is to create a urls.py for each app and include it in the main project urls.py.
- [x] Copy paste main project urls.py to create apps urls.py.
- [x] Adding urls is given in the starter page.
- [ ] To add dynamic urls,:

```
  # In urls.py
    path('tweets/<int:tweet_id>', tweet_detail_view)
  # In views.py
    def tweet_detail_view(request, tweet_id, *args, **kwargs):
      return HttpResponse(f"tweet_id={tweet_id}")
```

templates

- ☑ Django first looks at the DIRS list for templates, then in installed apps templates directory (in sequence).
- ☑ Create a base.html with common headers and other things. Add {% block body %}{% endblock body %} In all other html pages, {% extends 'base.html' %} {% block body %} Then content here will be placed between body block in base.html {% endblock body %}
- ☑ To create components separately, create html documents separately and add {% include 'component.html' %}
- ☑ Context variables can be used inside template with {{ variable }} format.
- ☑ To render a list, use for loop:

```
{% for item in list_of_items %}
  <li>item</li>
{% endfor %}
```

- ☑ To check for conditions, use

```
{% if variable == "some_value" %}
  <h4> variable is 'some value'<h4>
{% elif variable == "some_other_value" %}
  <h4>variable is some other value<h4>
{% endif %}
```

> Refer builtin template tags in docs to know about more tags.

- ☑ comment

```
{% comment "Comment title" %}
<tag>Commented text</tag>
{% endcomment %}
```

- ☑ cycle:

```
{% for item in items %}
<tr class="{% cycle 'row1' 'row2' %}"></tr>
```

- ☑ To render forms, use

```
<form action="[url]" method='POST'>
{% csrf_token %}
{{ form.as_p }}
<input type="submit" >
```

> forms.as_ul is also a valid method. Default action sends request to current url. You can put action='.' to get same effect as default. To perform google search from your website,

```
<form action='http://www.google.com/search' method='GET'>
  <input type='text' name='q' placeholder='Google Search'/>
  <input type='Submit' value='Search'/>
</form>
```

Filters

- ☑ Filters are used in {{ }} this type of syntax.
- ☑ Filters can be used one on top of other. {{ variable|capfirst|upper }}
- ☑ See docs for builtin filters.
- ☑ Custom filters can be created.
- ☑ Common ones are:
  - ☑ safe : To render text as html (this can be done in view using *mark_safe*).
  - ☑ title : Capitalizes first letter of each word.
  - ☑ striptags : Removes all html tags.
  - ☑ slugify : Replaces spaces with '-'.
  - ☑ add:[number] : Adds a number.

forms.py

- ☑ Create this file in the app.
- ☑ Inbuilt forms Eg.

```
from django import forms
from .models import Product
class ProductForm(forms.ModelForm):
  class Meta:
    model = Product
    fields = [
      'title',
      'description',
      'price'
    ]
```

- ☑ Raw django forms. Eg:

```
from django import forms
class RawProductForm(forms.Form):
  title = forms.CharField()
  description = forms.CharField()
  price = forms.DecimalField()
```

- [x] Raw django forms
  - [x] By default, all fields are required, to change required=False.
  - [x] Search for django form fields for more info.
    - [x] Core field arguments in docs tell about defaults.
  - [x] Arguments in a FormField
    - [x] required=False
    - [x] label='New Label'
    - [x] initial=199.99 (in DecimalField)
    - [x] widget=forms.Textarea(attrs={"class":"class1 class2", "id":"some-id", "rows":20, "cols":120})
    - [x] widget=forms.TextInput(attrs={"placeholder":"A placeholder"})

> All widgets can be found in docs.

- [x] Modifying PreBuilt Forms
  - [x] Add the formFields like in raw django form to overwrite them.
- [x] To validate data, create functions with name clean_[field_name]:

```
def clean_title(self, *args, **kwargs):
  title = self.cleaned_data.get('title')
  if 'CFE' not in title:
    raise forms.ValidationError("Title must contain CFE")
  if 'NEWS' not in title:
    raise forms.ValidationError("Title must contain 'NEWS'")
  return title
```

admin.py

- [ ] Register models to be viewed from admin page.
  - [ ] admin.site.register(ModelName)