Files

settings.py

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

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

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

Commands

manage.py

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

views.py

Functional Views

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

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

request Object

> Request object is also accessible in html templates.

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

ModelName.objects

- [ ] .get
- [ ] .create(**dictionary) or .create(attribute1=value1, attribute2=value2 ...)

urls.py

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

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()
```

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

> All widgets can be found in docs.

- [ ] Modifying PreBuilt Forms
  - [ ] Add the formFields like in raw django form to overwrite them.
- [ ] 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
```