# Commands (of importance) 4/10/19

> Pip install django
> django-admin startproject adsite

cmc → Python manage.py runserver

making migrations
→ 'cd adsite'
→ Python manage.py make migrations
(adding a new model)
→ python manage.py migrate
→ python manage.py shell

→
python manage.py shell
>> from main.models import Tutorial
>> Tutorial.objects.all()

Creating a superuser (a skeleton project)
python manage.py createsuperuser
name: pro
pass: 'Programming'

registering the model to admin
* To modify the admin you can create a class

importing the model from 'models.py'

4/10/19

Models - definitive source of information about the data. Contains fields and behaviors of the data being stored.

- Each model map to a single database table.
- Each model is a Python class that subclasses django.db.models.Model.
- When created you edit your settings file by altering 'Installed_Apps'

Fields - required by model. Used to determine (eg. Integer, Varchar, Text)

- what data to store
- default HTML to use when rendering a form field
- validation requirements, used in Django's admin

Override special method (__str__) to make a bit more readable when being displayed in string form.

When models have been changed ⇒ make migrations

Personal Note

Complication: having difficulty utilising a Django application 'TinyMCE' editor for widgets, views integration with file browser) admin page but failing to be compatible with current Django version

mysite/ root directory is just a container for the project.

manage.py: A command line utility that lets you interact with this Django in particular ways (eg. starting a project, helps sets paths

inner 'mysite/': the actual package for your project used to import anything inside it eg (mysite.urls)

__init__.py: an empty file that tells Python that this directory should be considered a Python package.

settings.py: settings configuration for the project.

creating Models. 7/10/19

Models is the single definitive source of truth of the data, containing the essential fields and behaviours of the data being stored

Each model has a number of class variables, representing a database field in the model.

Field; is represented by an instance of a field class
eg. CharField = character fields (requires an argument + max length
DateTimeField = datetimes

Activating Models

- Create a database schema (Create Table statements)
- Create a Python database-access API for accessing Objects

Installing 'POLLS' app

Django apps are 'pluggable' use an app in multiple projects
Besides the app doesn't have to be tied to a given Django installation.

# Models (Further Notes + Useful Info.)

* each Model will be a unique class within the app models.py file.
* within the class is where the Model is being defined.

* Models will inherit from models.Models
  ↳ defining the fields with further fields (different fields are defined in different ways.

  CharField = used for something that has a limit in size
  TextField = when we dont have a limit.

__ Str __ (special method) to make it more readable when its being displayed in string form.

- Once complete make a "migration" once you make changes in your models (either new or modifying an existing one)

- Migrations will only apply to app's in which we've told Django to be installed.
  ↳ adding "main.apps.MainConfig" to installed apps

Then adding to mysite/main/apps.py

Query → create tutorial object
- title
- content
- published

Committing an object to our database (.save())

```
>> New_tutorial.save()
```

```
>> Tutorial.objects.all()
<QuerySet []>
```

# Admin & Apps (Continuation)

Creating Superuser.

- Django admin application automatically builds a site area that use to create, view, update and delete records.

Interacts with ~~the~~ models as an actual user interface.

## ~~Web Development~~ (Views & Templates)

Models, ~~add~~ templating & views

mysite/main/views.py to alter homepage function

```
def homepage(request):
    return render(request = request, template_name =
'main/home.html',
context = {"tutorial": Tutorial.objects.all})
```

- render = render an actual html file/template ~~to~~ to ~~pass~~ pass objects to that template.
- This also iterates over objects with loops, use if statements

[Setting with Vars]

* To do this we pass a [key-word] dictionary where the 'key' is the name of the variable which will reference from within the template and the 'value' is the object it represents.

Interacting with the Model needs to import

```
from .models import Tutorial
```

Django looks for templates in each app's directories by seeking and directory called "templates" - it builds a list and opens on that matches.

# Homepage

22/10/19

Creating a views in (views.py)

Render implemented is to render a template file (HTML)

- import Model previously created (Test)
- Created a templates directory which will find the template being called from the 'views.py'

# Django Template language

- Templates generate HTML dynamically
- The template contains the static parts of the HTML

Django can be configured with one or several template engines
- Django template language (DTL) (used)
- Jinja2 (Jinja similar to)
- Smarty

{{ variable }} → evaluates the variable and replaces it with the result.

(".") → access attributes of a variable

Filters

{{ name|lower }} → value of name then filtered through lower filter, converting text to lowercase

Pipe filter

Tags {% tags %} used for
- Creating text in output
- Control flow by performing loops or logic

- Loading external information into the template to be used by later variables.

{# #} = comments

## Template inheritance

block tag defines templates
(tells the template engine that template may override specific portions of the template)

{% extend %} first template tag in template otherwise inheritance will not work
{% block %} preventing duplication of content

```
{% trans "title" as title %}
{% block content %}{{title}}{% endblock %}
```

DTL generating any text format (HTML, CSV, XML etc)

12/11/19

# Registration for a user / login / logout

- creating a user model
  - import through the shell

homepage / Reg. / Login

Template for Registration

home / Reg / Logout

<form method = >

denoted that there will be a form.

allows a user to enter a text, select options, manipulate objects / controls

<input> = • where: the URL to which the data corresponds to the users input should be returned
• how: the HTTP method (Hyper Text transfer Protocol. (defines how messages are formatted and transmitted.

GET / Post methods

bundles the form data, encodes for transmission, sends to server the receives a response

yet: used for web search form, URLs that represent a GET request can easily be bookmarked, shared or resubmit.

{% csrf_token %} CSRF (Cross Site Request forgery.
forces a user to execute unwanted actions on a web application in which they're currently authenticated.
* Compromising an entire web application *

Personal note:
When checking if the form instance created was valid, in views.py
if form.is_valid():
it started receiving errors as it was not defined (*) for the time being I have turned form into a global keyword inside the function.
This will then allow me to modify a variable outside of the current scope.
→ This creates global variables from a non-global scope (ie. inside a function)

Container Class = used for HTML container elements
eg 
can be used to readjust the positioning of content.

## Login / Logout

27/11/19

Similar concept as applied
- create URL in 'urls.py'
- create the function in views
- have individual html page

validation of data received
↳ Data is cleaned / is valid? / error?
Cleaned_data ⇒ New data will populate with existing data.

form extension to collect users emails
create an extended form in django form
forms.py