**Django tutorial**

It lets web development easy and time efficient. Security, scalability,

It comes with built in **admin interface.(0 time consumed in managing admin)**

**Front end:** html, css, javascript

**Back end**: php, python: webframework: django, asp

Webpages: static (same for all)vs dynamic (Django makes it possible)

Django-admin gives the list of sub commands in the command prompt

Wsgi.py: communicates the python code with the server

**Model View template: Django follows it**

**Set up:**

**Virtual environment:**

**Installing virtual env :**pip install virtualenvwrapper-win

Creating virtual env directory: mkvirtualenv test (test is the name of the directory)

Installing Django in the created virtual environment: pip install Django

Making directory and starting project: mkdir projects

Django-admin startproject project1

Running server: python manage.py runserver,,, it creates a host at port <http://127.0.0.1:8000/> which can surely be changed,, port number: 8000

**Django py files.**

Settings.py .. debug .. make it false before deploying…important file….

**First application:**

now going to your IDE

visual studio… cmd.exe on PowerShell to call command prompt

remember to workon virtual environment

working on virtual environment: workon test

starting app: python manange.py startapp polls

urls.py created for the created appp… this allows to map the url of the appp

views.py allows to interact with the view in the created app

urls.py from the main project should be mapped with the urls.py from calc

add the path to the settings.py ,,, in installed appp 'blog.apps.BlogConfig'

**create template:**

why: it saves us time and complex coding…

* create a folder within the app,,template…..create another folder within the template named same as the app(blog.)…create html file within the folder…
* add the path to the settings.py ,,, in installed appp 'blog.apps.BlogConfig',
* in the views.py add the template using render. return render(request,"blog/home.html")

**lets make the template more dynamic**

* by adding context
* return render(request,"blog/home.html",context)
* by creating base.html and {%block content%} we do not have to type same html content in different html pages (extending function in the html)

**admin:**

database migrations allows to make changes in the databases

* python manage.py makemigrations=prepares python for the migrations
* python manage.py migrate==makes the migration
* python manage.py createsuperuser ==creates the admin;

**databases and migrations**

* **ORM**: Django has its own orm object relational mapper. It allows to access databes and easy to use database. can use different database program without changing the database code
* we work with models. migrate using make migrations…make it easy to make changes an apply the changes by just using makemigration…. migrate
* once the models file is created, we need to make migrations, 000-initial.py
  + to view the sql code created : python manage.py sqlmigrate blog 001
* use of interactive shell allows to interactively communicate with the models and perform database queries:
  + python mange.py shell
  + exit () to exit the shell
* Django has built in models, which has built in methods like charfield
* To see the posts in the admin .. we have to set the path into blog.admin

**database queries:**

**from django.contrib.auth.models import User**

**from blog.models import Post**

* User.objects.all() :: return all the object in the models
* User.objects.first()
* User.objects.filter(username='aarav')
* User.objects.filter(username='aarav').first()=== returns the first user with name aarav
* user=User.objects.filter(username='aarav').first() … setting the variable
* >>> user.id or user.pk
  + 1
* User.objects.get(id=1)===returns the user with id ==1
* Creating the new post through shell
  + post\_1=Post(title='blog 1',content='first blog throuh shell',author=user) or
    - post\_2=Post(title='blog\_2',content="second post",author\_id=user.id)
    - >>> post\_2.save()
    - post\_1.save()===need to save after creatinf the object
  + Post.objects.all() === checks all the post posted===
* **Operation on the post created (running queries)**
  + >>> post=Post.objects.first()
  + >>> post.title
  + 'blog 1'
  + >>> post.date\_posted
  + datetime.datetime(2019, 11, 5, 21, 2, 50, 282219, tzinfo=<UTC>)
  + >>> post.author.email
  + 'aaravranaonline@gmail.com'
  + >>> post.author.password
  + 'pbkdf2\_sha256$150000$BxnWVRsF938H$VU6PWOhOeBcMvPT7nqZQYHriHciaAmo+8KfGvAK1Q48
* Operations using .modelname.set
  + For our post
  + user.post\_set.all()
  + <QuerySet [<Post: blog 1>, <Post: blog\_2>]>
* **To view more descriptive object result**… add dunder method(--str--) in the model file
  + To view the changes… exit the existing shell and open new one…

**To add the data base in to views file**

* Import the model
  + from .models import Post
  + context={'posts':Post.objects.all(),'title':'blogpost'}

**To see the posts in the admin ..**

* we have to set the path into blog.admin
* **admin.site.register(Post)**

**user registration:**

**main purpose: lets the user create an account, and do amazing stuff inside the website.**

* **Python manage.py startapp users**
* **Add to installed app list,,, in the settings.py : appname.apps.Appnameconfig,**

**Eg: users.apps.UsersConfig…present in the apps.py**

* **Views.py file**
  + **Django has a buit in register form .. we can import it using**