Django

Backend web development

Python - Numbers, string, lists, Dictionaries, Tuples, sets, Booleans, control flow, functions, scope, OOP, class & exception, Decorators, Regular expression.

Python installation & setup.

Django - Background, Introduction, project, application, Mapping URLS, Templates, static files, Models, MTV, Basic forms, HTTP, GET, POST, Form Validation, Model forms, Relative URLs with template, template inheritance, Templates features filters, users model, Permissions, groups, passwords & Authentication, Logging in & out, Registration

Django Deployment - using Git Bash & Heroku CLI a project is deployed

What is backend & what its use

* The back-end uses technology to display content on the frontend.
* The backend of any website generally has three mainly components
  - The language
  - The framework
  - The Database
* So many technologies such as PhP, Node.js, Ruby / Rails, Java, Python, etc. are all possible choice for any website.

Here

- Python & PhP as the language
- Django as the framework
- SQLite & MySQL as the database
  Postgres

What is a programming language?

A programming language is a vocabulary & set of grammatical rules for instructing a computing device to perform specific tasks. The term programming language usually refers to high level languages.

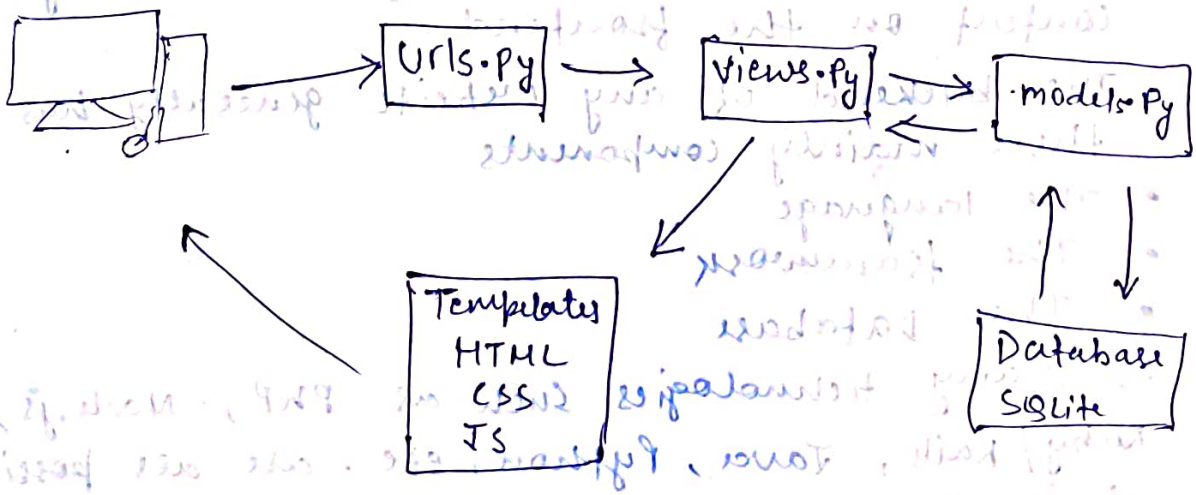
What is framework in web design?

In the world of web design, to give more straightforward defination, a framework is defined as a package made up of a structure of files & folders of standarized code (HTML, CSS, JS documents etc.) which can be used to support the development of websites, as a basis to start building a site

What is a database?

a database is a datastructure that stores organized information. Most database contain multiple tables, which may each include several different fields. for ex - a company database may include tables for products, employees, & financial records.

(very) High level overview of Django



* Backend development refers to the server side of an application & everything that communicates between the database & the browser. Backend development refers to the server side of development where you are primarily focused.

Introduction to Django Framework

* Django is free & open source web framework
* It is used by many sites, including pinterest, PBS, instagram, Bitbucket, washington times, Mozilla, & more.
* Django was created in 2003 when the web development at the lawrence journal-world newspaper started using python for their development
* Because the original developers were surrounded by writers, good written documentation is a key part of Django.
* This means you have an excellent reference to check on the official Django docs!
* Django has its own excellent basic tutorial where you are walked through creating a basic polling web app.
* When encountering Django tutorials you will often read that you should create a virtual environment or an 'venv'.
* Virtual environment
  - a virtual environment allows you to have a virtual installation of python & packages on your computer.
  - The packages change & get updated often! due to updates
  - There are changes that break backwards compatibility because of it.
  - So what do you do if you want to test our new features but not break your web app?

- you create a virtual environment that contains the newest version of the package.
- Luckily, Anaconda makes this really easy for us to create and use a virtual environment.
- A virtual environment handler is included in anaconda distribution.
- How to use it?
  - To use a virtual environment with conda we use these commands.
    - conda create --name myEnv django.
  - Here we created an environment called "myEnv" with the latest version of Django.
  - you can then activate the environment
    - activate myEnv
  - Now, anything installed with pip or conda when this environment is activated, will only be installed for this environment.
    - deactivate myEnv
  - Its encouraged to use virtual environment for your projects to keep them self contained & not run into issues when packages updated!

Django (Project)

* Lets create our first project
* But first we need to install django & we can do it as shown below.
* we can install Django with anaconda prompt. by using below command
  - conda install django
* for normal python distributors, type the below code in command prompt:
  - pip install django
* when you install Django, it actualy also installed a command line tool called:
  - django-admin
* for first_Project in Vs-code terminal run
  - django-admin startproject first_Project

* After this

- Starting with __init__.py file
  - This is a blank python script that due to its special name let's python know that this directory can be treated as a package.
- Second file created is settings.py file
  - This settings.py file is a place where you will store all your project settings.
- Third file created is the urls.py file
  - This is a python script that will store all the URL pattern for your project. Basically the different pages of your web application.
- fourth file created is the wsgi.py file.
  - This is a python script that acts as the web server gateway interface.
  - It will later on help us deploy our web app to production.
- last file created is the manage.py file
  - This is a python script that we will use a lot
  - It will be associated with many commands as we build our web app.

* Use of manage.py now:

- Python manage.py runserver

- you have to type the above command in any one of the command prompts.
- Then you will see a bunch of stuff but at the bottom you will see something like
  - Django version 1.10.5, using settings 'first Project.settings'
  - Starting development server at http://127.0.0.1:8000/
- We need to copy & paste that url into any one of our browsers.
- All of you should now see your very first

web page being locally hosted on your computer

- Congratulations! if you have achieved this.
- After this you should have also noticed a warning about migrations.
- This has to do with database & how to connect them to Django.
- But you might get a doubt.

* What is Migration?

- migrations are Django's way of propagating changes you make to your models (adding a field, deleting a model, etc.) into your database schema, They are designed to be mostly auto-matic, but you'll need to know when to make migrations, when to run them, & common problems you might run into:
- A migration allows you to move database from one designed to another, this is also reversible.
- so you can "migrate" your database.
- we will see this topic later
- That's was basic of getting started with django
- we will continue now by creating very simple

"welcome to Ashish" Django application

Django (application)

* Creating our first Django application

"welcome to Ashish"

* So far we have been able to use runserver to test our installation of Django.
* Now let's move on to creating our first Django application
* We will learn about views & how to use them.

Django Project

* It is a collection of application & configurations that when combined together will make up the full web application (your complete website running with Django

Django applications

* A django application is created to perform a particular functionality for your entire web application.
* for example you could have a registration app, a polling app, comments app etc

* These Django apps can then plugged into other projects, so you can reuse them! (or use other people's apps)
* lets create a simple application with
  - Python manage.py startapp first_app
* Type the above command in any one of the command prompt (Anaconda prompt or command prompt)
* then a list of folders & files are created
  - The first file we have is the "__init__.py"
    - This is a blank python script that due to its special name let's python know that this directory can be treated as a package.
  - The second file is the "admin.py"
    - you can register your models here which Django will then use them with Django's admin interface
  - The third file is the "apps.py"
    - Here you can place application specific configuration

- The fosth file is the "models.py"
  - Here you can store the application's data models
- The fifth file is the "test.py"
  - Here you can store test functions to test your code
- The last file is the "views.py"
  - This is where you have functions that handle requests & return responses
- Before all files discussed above first will be the ~~mig~~ "migrations folder".
  - This directory stores database specific information as it relates to the models.

* Now let's learn the process of creating a view & mapping it to a URL!
- go to settings.py file, & find 'INSTALLED_APPS' & add "first-app" at last.
- go to views.py file & add 'from django.http import HttpResponse' then give a function
  - def index(request):
    - return HttpResponse("welcome to Ashish")
- Go to urls.py file & add 'from first_app import views' & 'from django.conf.urls import url' inside 'urlpatterns', add the below code
  - url(r'^$', views.index, name='index'),
- save all these files & runserver in cmd & paste url in the browser to see the output

* conclusion
1) Create projects
2) Create application
3) Create view
4) Url mapping
5) run server

Django - Mapping URLS

* We previously showed a very direct mapping from the views.py to the url.py
* There are several ways of mapping URLs, lets see another way of doing this now!
* Now we want to show the ability of using the include function from django.conf.urls
* The include() function allows us to look for a match with regular expression & link back to our application's own url.py file.
* we have to manually add in this urls.py file.
* we would add the below code to our project's urls.py
  - from django.conf.urls import include
  - urlpatterns = [... url(r'^first_app/', include('first_app.urls')), ...]
* This would allow us to look for any url that has the pattern:
  - http://www.domainname.com/first_app/...
* If we match that pattern, the include function basically tells django to go look at the urls.py file inside the first_app folder
* This might seem like a lot of work for a simple mapping, but later on we will want to try to keep our project's urls.py clean & modular
* so we set the reference to the app, instead of listing them all in the main urls
* lets quickly see an example of all of this to show how it works!
  - locate the urls.py file in (first_Project), there type the below code
    - from django.conf.urls import include
  - In the urlpatterns include the code below
    - url(r'^first_app', include('first_app.urls')),

- Save the urls.py file in the first-project folder after making the above changes by pressing (Cntrl+S) on your keyboard
- Create a new file 'urls.py' in first-app folder & inside this type the code which is shown below.

```
- from django.conf.url import url
- from first-app import views
- urlpatterns = [
-                 url(r'^$', views.index, name='index'),
- ]
```

- Save all these changes
- We can now test if this all works
- Open the command prompt & type
  - python manage.py runserver
- Copy & paste the url in a browser to see the output
- add an extension '/first-app' & you will notice that you get the same result as we have defined so in our code
- In the extension, if we add anything that is not defined by us, we will get an error like for example, in my case if i pass in '/verro-app', I'll get a 404 error
- The main idea here what we have done is that the application can have their own urls.py file, that we can call from the project's ~~root~~ urlpattern list (that's a small recap of what all we have done just now)

Django - Templates

* Django template is a text document or a python string marked-up using the Django ~~template~~ template language. some constructs are recognized & interpreted by the template engine. The main ones are variables & tags. A template is rendered with a context.
* Templates are a key part to understanding how django really works & interacts with your websites.
* later on we will learn about how to connect templates with models so you can display data created dynamically
* for now lets focus on the basis of template & ~~tags~~ its tags
* The template will contain the static parts of an Html page (parts that are always same)
* Then there are template tags, which have their own special syntax

what are django templates tags ?

* Django templates tags are simple python functions which accept 1 or more value, an optional argument, process those values & return a value to be displayed on the page
* the special syntax of these templates tags allow you to inject dynamic content that your django app's view will produce, effecting the final HTML
* But to first to get started with templates, you need to create a templates directory & then a subdirectory for each specific app's templates

* It goes inside of your top level directory
  • first_project / templates / first_app
* we want our django project to be easily transferrable from one computer to another but the DIR key will require a "hard-coded" path
* This means depending OS (windows, mac, linux) you are using, this hard-coded path also changes & this a problem to us.

→ so how do we resolve this - of os path

* we can use python's os module to dynamically generate the correct file path strings, regardless of computer
* we will use this os module to feed the path to the DIR key inside of the Templates dictionary
* The next step is to let Django know of the templates by editing the DIR key inside of the Templates dictionary in the setting.py file
* once we're done with all of these steps, we can now an create an HTML file called index.html inside of the templates / first_app directory
* Inside the HTML file we will insert templates tags also otherwise called as Django templates variables.
* These templates variable allow us to inject content into the HTML directly from django
* with all this, now one can understand the power of using a web framework.
* Django will be able to inject content into the HTML

* with the help of django, later on we can be use python code to inject connect from a database
* But in order to achieve this, we will use the render() function & place it into our orginal index() function inside of our views.py file
* we have discussed about the templates & templates tags in theory until now

Now

* Please Open a text editor & go to the first-project folder directory & code along with me
* Right click on the top level first-project & create a new folder 'templates'
* add this templates folder directory in our project's settings
* make sure you have import OS command in the settings.py file
* so Open this setting.py file of our project & below the BASE_DIR, add the code
  - TEMPLATE_DIR = os.path.join(BASE_DIR, "templates")
* Now in the same settings.py file, scroll down until you find TEMPLATES containing dictonaries type object & for the keys DIRS, give TEMPLATE_DIR as the value & add a comma at last
* Now all the paths will be concatenated correctly after you make above changes in the code in our project
* Now go to the templates folder & right

click on it to create a new file "index.html" and code something & {{insert_first}} before body tag

* Now what we need to do is to connect this insert_first to our project
* The way we can do this is by editing this views.py file as below

```
from django.shortcuts import render
from django.http import HttpResponse

# create your views here
def index(request):
    our_dict = {'insert_first': "first, coming from views.py file!"}
    return render(request, 'index.html',
                  context = our_dict)
```

* Now to see if all of these changes are applied to our project, open a command prompt & type python manage.py runserver
* copy paste the url in the browser to see the output
* Now we need to seperate our templelate for our application

'first_app/index.html',
context = our_dict
(or any name like my_dict)

{{}}
This is known as jinja template

Django - Static files

* Websites generally need to serve additional files such as images, javascript, or CSS. In django we refer to these files as "static files".
  Django provides django.contrib.staticfiles to help you manage them.
* Now learn how to insert static media files into our project.

- So far in our course we have used templates to insert simple text.
- But we don't always just want text all time right what about other types of media files, like for example, returning a user photo?
- So now lets discuss static media files & how to use them in our projects!

- But to do this we need to create a new directory inside of the project called static (just like we did for templates earlier in our course)
- Then we will add this directory path to the project's settings.py file in the settings.py file
- We will also add a STATIC_URL variable
- Once we're done doing all of this, now we need to place to store our static images files
- So, we create a directory inside of static called images & place a favourite.jpg file inside the image directory (or just download one if you don't have)

* Now lets set up a template tag for this
  - To do this inside an html file, we add in a few specific tags, at the top:
    - {% load static %}
  - Then we want to insert image with an HTML <img src=> style tag using:
    -

- Notice how this templates tag is a little different in that it uses
  - {% %}
- instead of
  - {{ }}
- we will discuss & show these difference more clearly in future classes of our course, but for now consider {{ }} as being used for simple text injection, & we can use {% %} for more complex injection & logic
- Now let's code through an example of serving up a static image.
- please open a text editor & open the project folder directory
- on the top level project directory, right click & create a new folder directory 'images'
- we need django to know about this static folder directory so we'll open the settings.py file & we'll edit it.
- Below the TEMPLATE_DIR, add the below code
  - STATIC_DIR = os.path.join(BASE_DIR, "static")
- In the same settings.py file, scroll down until you find STATIC_URL & below it.
  - STATICFILES_DIRS = [
    STATIC_DIR,
  -]

Django - Models

* An essential part of any websites is the ability to accept information ~~about~~ from a user & input it into a database & retrieve information from a database & use it to generate content for the user

* We use models to incorporate a database into a Django project.
* Django comes equipped with SQLite.
* SQLite will work for our simple ~~exples~~ examples but Django can connect to a variety of SQL engine backends!
* In settings.py file you can edit the ENGINE parameters used for databases
* To create an actual model, we use a class structure inside of the relevant application models.py file
* This class object will be subclass of Django's built-in class:
  - django.db.models.Model
* Then each attribute of class represents a field, which is just like a column name ~~which~~ with constraints in SQL
* This will all feel very natural if you have some SQL experience, but in case you don't let's quickly review what a SQL database like!
* SQL operates like a giant table, with each column representing a field, & each row representing an entry.

WebsiteId	WebsiteName	URL
1	Ashish Classes	https://ashishclass.com
2	SSS	https://sss.com/

* Each column has a type of field, such as CharField, IntegerField, DateField etc.
* Each field can also have constraints
* for example, a charfield should have a max_length constraint, indicating the maximum number of characters allowed.
* The last thing to note is table(or models) relationship
* often models will reference each other

* Imagine we have two models
* one to store information, & another to store date information

Websiteid	Websitename	URL
1	ashish classes	https://ashishclas.com)
2	SSS	https://sss.com)

Websiteid	Date accessed
1	2020-11-23
2	2019-10-22
3	

* We could say that the websiteid column is a primary key in the first table & foreign key in the last table
* A primary key is a unique identifies for each row in a table
* A foreign key just denotes that the column coincidis with a primary key of another table
* Lates on we will move on to discuss One-to-one or Many-to-many relationships
* Now lets show an example of the models class that would go into the model.py file of your application

```
class Topic (models.Model):
    top_name = models.CharField(max_length=264,
                                unique=True)
class Webpage (models.Model):
    category = models.ForeignKey(Topic)
    name = models.CharField(max_length=264)
    url = models.URLField()
class Webpage (models.Model):
    topic = models.ForeignKey(Topic)
```

- name = models.CharField(max_length=264)
  url = models.URLField()

- def __str__(self):
      return self.name