# BACKEND

Command Prompt →

① if want to know subfolders in directory
'dir'

.android ⟶ . signifies that folder is hidden

② changing directory to go back
cd name → name of folder
cd ..

③ clear
cls

④ printing current directory → cd

---

## Python

① Number < int / float

② # comment

③ print (my_string[2:]) → grab all letters starting from index 2
my_string[:3] → prints upto 3rd index excluding 3rd index
[2:5] → all letters from 2 to 5 excluding 5
[::2] → step size 2 prints every 2nd letter
[::1] → step size

④ string is immutable
str[0] = 'x' ⟶ X

⑤ my_string.lower()

⑥ my_string.split() ⟶ ['Hello', 'World']
split('e') → ['H', 'llo World']

- Print formating →

x = "Item One": {} Item two : {} ". format("dog", "cat")

O/P Item One : Dog Item two : Cat

- List →

mylist = [1,2,3] ⟶ mylist[0] ⟶ 1
[-1] ⟶ 3

mylist = ['Hey', 1, 2, 3, 1.2, True]

print (len(mylist) ⟶ length

Add an item → mylist.append ("New") ↳ add to end of list

mylist = [1,2,3]
listtwo = [4,5,6]

mylist.extend(listtwo) ⟶ 1,2,3,4,5,6

Remove

item = mylist.pop() ⟶ 1 2 3
print (item) ⟶ 1
print (mylist) ⟶ 2 3
item = mylist.pop(1) ⟶ 3

Reverse →

mylist.sreverse () ⟶ [1,2,3]
print (mylist) ⟶ [3,2,1]

Sort

mylist.sort()

Matrix →

matrix = [[1,2,3], [4,5,6], [7,8,9]]

[1,4,7] ← first_col = [row[0] for row in matrix]

→ Dictio
my
pr
pr
→ Boolea
→ Tupl
→ Set
→ for
to tuple

→ Dictionary

my = {"key1": "value", "key2": "value2"}

print (my['key2'] → value 2

print (my)

→ Boolean

True 1 False 0

→ Tuples → lists but immutable

tup t = (1, 2, 3) t = (1, 1.2, "Hey")

print (t[0])

→ Set → unordered collection

x = set()

x.add(1)

x.add(2)

x.add(0.1)

x.add(1)

x.add(1)

print (x) → {1, 2, 0.1}

↳ any order

↳ no duplicate allowed

→ if else →

→ and or used

→ {} x → intendation

→ if 1<2:

print ('yes!')

elif (2<3):

print ('NO!')

else

print ('ya')

→ for

for item in seq

i array or anything

print (item)

to tuple → for (tup1, tup2) in mypair

print (tup1)

print (tup2)

→ while →

* comment → """ """ ''' '''

```
while i<5:
    print ("i : {}".format(i))
    i= i+1
```

for each num do num*2 and give in out

★ out=[num**2 for num in seq]

→ for item in range(10) → 0-9
    print (item)

→ Function →

syn → def my-func (param1 = 'default'):
        print (param1) → default

    my-func()

eg → def add (num1, num2)
        return num1 + num2

    result = add (1, 2)
    print ('result')

★ check type
print (type(result))

→ filter → filters needed input

```
def even (num)
    return num%2==0
evens = filter (even, mylist)
print (list(evens))
```

→ SCOPE →

Local
Enclosing fn ~~goto~~ local
Global
Built in

```
def func()
    global(x) global x → effects to global x
    x=1000
```

OOP →

```
class Dog():
    def __init__(self):          # refers to itself necessary
        print("Animal created")

    def eat(self):
        print("Eat")

class Dog(Animal):
    def __init__(self):
        Animal.__init__(self)
        print("Dog created")
```

Initialize ① `def __init__(self):`

② `def __init__(self, breed):`
`self.breed = breed`

```
mydog = Dog(breed = "Lab")
print(mydog.breed)        → Lab
```

```
class Dog():
    species = mammel
    def __init__(self, breed, name):
        self.name = name
        self.breed = name
    def area(self):          # method of class
        return self.name * self.name * Dog.species
```

```
mydog = Dog(name = "Jash", breed = "Lab")
mydog.name
mydog.breed
mydog.species
```

Calling another class →

```
mya = Animal()        → Animal created
mya.eat()             → Eat
mydog = Dog()
dog.eat()
```

→ parent init
Animal created
→ class init
Dog created
Eat → inherited methods

Printing class →

```
class Book:
    page = 100
    def __init__(self, title, author):
        self.title = title
        self.author = author

    def __str__(self):
        return "Title: {}, Author: {}".format(self.title, self.author)

    def __len__(self):
        return self.book.page

b = Book("Python", 99)
print(b)
print(len(b))
```

## Error & Exception →

```
try:
    f = open('simple.text', 'w')
    f.write("Test write to")
except IOError:
    print("could not open file")
else:
    print("success")
    f.close()
finally:
    print("I always work")
```

open fn
open('file_name', 'r')
open('file_name', 'w')

checks for input output error
don't need to specify

else → if error

finally: → will work even if exception called

# Regular Expressions →

```
import re ——→ importing regular exp.

patterns = ['term1', 'term2']
text = 'this is a string with term1 but not other'
for pattern in patterns:
    print ("I am searching for " + pattern")
    if re.search(pattern, text):
        print("match")
    else no mat
```

match object ← re.search(pattern, text) ← attribute; key to search; where to search

---

→ read

→ write

```
text = "gt term1"
match = re.search('term1', text)
print (match.start())
```

→ 3

---

```
email = username@gmail.com
print (re.split('@', email)
```

'@' ↓ where to split; ↳ what to split

---

['match', 'match'] ← `print (re.findall ('match', 'test phrase match to match'))`

---

# Django

free & opensource web framework

used because packages get updated

virtual environment → allows to have virtual installation of python and other packages on your computer

to make virtual env with conda by
conda create --name myEnv Django
(name) (Package)

activate by
activate myEnv

deactivate by
deactivate Env

How to start →

① make a directory for your project
mkdir file_name → make a new folder
cd file_name → go to folder

② Activate your env
activate myDjangoEnv
(name)

③ create a project
django-admin startproject name

- you will get various files

① __init__.py → blank, name tells python that it can be used as package

② settings.py → to store your project settings

③ urls.py → store all URL patterns for your project

④ wsgi.py → Python script that acts as the web server gateway interface. help to deploy web app to production

⑤ manage.py → associates with many commands as we build our web apps

→ Managing & using Manage.py →

① python manage.py runserver
copy the last url on chrome — will lead to first project

② Migration → reversible, allows to move database from one design to another.

→ Terminologies →

① Django Project → collectn of applicatn & config that when combined together will make up the full web application.

② Django Application → created to perform a particular functionality for your entire web application.
Eg → registration app, polling app etc
They are pluggable. others can use apps from your project & vice versa.

→ To create app →

① creating a app folder
python manage.py startapp name

② contains file →
① __init__.py → ''
② admin.py → can register your model here which django will then use with ~~other~~ Django's admin interface
③ apps.py → place applicatn specific config
④ models.py → store applicatn's data model
⑤ test.py → store test fn to test your code
⑥ Migration folder → stores database specific info as it relates to the model

③ Need to tell Django that we created an application

→ setting.py > Installed_APPS

Search in file

→ Add a string 'file_name' to the installed_apps array

name of app file

④ Creating views →

Eg → sending Hello World →

first_app > views.py →

```
from django.http import HttpResponse

def index(request):
    return HttpResponse("Hello World")
```

1st view ← variable

HttpResponse object

⑤ To see view → linking views to urls.py

→ first_project > urls.py

→ search for urlpattern = [

Add → from APP_name import views

urlpattern=[

```
    path('', views.index, name='index'),
    path('admin/', admin.site.urls),
]
```

If run & web see Hello World

So created application, create views & map it urls file to work through url.

→ Mapping Urls → Another way [include()]

impot from django.conf.urls import include() → allows to look for a match for regular expressions and link back to our application's own urls.py file.

We have manually add in urls.py file

Method → add in urls.py of project

```
from django.conf.urls import include
url patterns = [
    path('app_name/', include('app_name.urls'))
]
```

this will be the extension name in the url

steps

① Go to urls.py of project
add ie fso

```
>{
```

② Make a new file in app folder as urls.py

add

```
from django.conf.urls import url
from first_app import views
                    ↳ name of app
urlpatterns = [
    path('', views.index, name='index')
```

## Templates

contains static parts of Html page

template tags → injects dynamic content that your Django app views will produce, effecting the final Html

template variables → inject content into the Html directly from Django

Creating template →

① under project (first_project) create a new folder named template.

② In the subfolder of first_project > settings.py
(i) add below BASE_DIR → to make it run on any operating system

```
TEMPLATE_DIR = os.path.join(BASE_DIR, "template")
```

(ii) In TEMPLATES [

'DIRS' : [ ],
↓ Add
TEMPLATE_DIRS

## Adding a template →

① In template folder, add file Index.html

② Adding a template tag in body →

{{ insert_me }}
↓
tag variablename

③ Connecting variable (insert_me) to our project and application

in first_project > views.py >

```
def index(request):
    my_dict = {'insert_me': "Hello I am"}
    return render(request, 'index.html', context=my_dict)
```

key value pair with key as the tag variable name

html file to add

what to link

④ To make modular →

in templates folder add a folder named on application and put index.html in it

change * to first_app/index.html

name of app

## Static Files

{{ }} Simple text injections

% % complex injections and logics

① Creating Folder

→ in project folder add a new folder named static and create a

→ folder in it named images

→ first_project > static > images

→ add image required in the image folder

② first_projects > setting.py

(i) Below base_DIR & Template_DIR add →

STATIC_DIR = os.path.join(BASE_DIR, "static")

joins base dir to static

(ii) at the bottom of page.

below STATIC_URL = '/static/'

↳ through this url we can access

add → STATICFILES_DIRS = [

STATIC_DIR,

refers to loc on comp where static files are stored

]

★ Put static & template in app folder.

③ For loading image through html

→ <!DOCTYPE.html>

{% load static %}

→ <body>

<img src = "{% static "first_app/images/image.jpg" %}

alt = "Uh not loaded" >

④ For adding CSS

① make a folder in static name css write full name.css

② link html with CSS as

<link rel = "stylesheet" href = "{% static "css/name.css" %}

# Models

① go to model.py in project folder

② → Code

```
from django.db import models

class Topic(models.Model):                    # inherited class
    top_name = models.CharField(max_length=264, unique=True)
                                  # max length constraint
                                  # field type
                                  # unique ↓ no duplicates

    def __str__(self):
        return self.topic_name

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

    def __str__(self)
        return self.name
```

* Similarly for date → DateField etc.

Migrations

③ Apply following command →

→ python manage.py migrate
→ " " " makemigrations first_app  (↗ appname)
→ " " " migrate

Method ① → " " " shell  (1st Method to add contents)

① → from first_app.models import Topic

To save a entry → t = Topic(top_name = "Hy")
t.save()

to print → print(Topic.objects.all())

to quit → quit()

② Method to add contents **Admin** Method ②

① go to first-app > admin.py

from django.contrib import admin

from first_app.models import Topic, Webpage, AccessRecord

↳ app name

class names

admin.site.register(AccessRecord)
" " " (Topic)
" " " (Webpage)

} registered

② In cmd →

python manage.py createsuperuser

will ask for username, password, email

Jose, hellopwrld, Jose@gmail.com

python manage.py runserver

↳ site copy & run

③ /admin in site leads to → login
leads to a django admin site.

---

→ Fakers library to create script

① Install →
cmd → pip install faker in first project

② create populate_first_app.py → (code all)

③ cmd → "v" ".py

④ cmd runserver

---

## Models Template Views

① Connecting views to database

steps →

① Go to views.py

③ Add → from first_app.models import Topic, Webpage, AccessRecord

connecting views to database

```
def index(request):
    webpages_list = AccessRecord.objects.order_by('date')
    date_dict = {'access_record': webpages_list}
    return render(request, 'first_app/index.html', context
                  = date_dict)
```

method to order by date field

④ index.html → code
style as per need → css

imp → !DOCTYPE

→ {% load static %}

→ <link ---- href = "{% static "css/style.css" %}"

→ {% if access_record %}

→ {% else %}

→ {% endif %}

→ {% for acc in access_records %}

{% endfor %}

⑤ runserver

# Django forms

## Making Django Project from scratch

### Level II till module

Creating Project Folder, APP folder

(1) cmd
- → activate myDjangoEnv
- → django-admin startproject Protwo (names → Making Project)
- → Protwo > python manage.py startapp appTwo (Making app)

(2) Creating Template →
Right click appTwo folder > add folder Template
create → Template > appTwo > index.html, .html } needed html files

(3) Setting.Py changes →
① # See template section 7 creating template 6 page before
- → TEMPLATE_DIR (Linking & Registering template to Project)
- → 'DIRS' = [ ]

② INSTALLED APPS = [
'appTwo' ] (APPName → Registering & Linking app to project)

(4) In Models.py # Refer to Models section 7 code 3 page before
- → Create required class

(5) In views.py
code →

from Apptwo (appname) import render, HttpResponse, class_name (class name)
(Import)

- def index (request) (main webpage display index.html)
  return render (request, Apptwo/index.html)

(Methods)

# can similarly insert other methods linked to respective html files

⑥ urls.py →

① create files urls.py in Apptwo folder

② In urls.py code →

import { from django.urls import url
from apptwo import views

urlpatterns [

path('', views.users, name="users")

App name

main page method called from views.py

]

③ Go to Project > urls.py

import { from django.contrib import admin
from django.urls import path, include
from apptwo import views

urlpatterns [

mainpage ← path('', views.index, name="index")
admin page ← path('admin/', admin.site.urls)
users page ← path('users/', include('apptwo.urls'))

]

⑦ Migration → (of Models to Database)

Protwo > python manage.py migrate → (to register all the changes to the app)
python manage.py makemigrations Apptwo
python manage.py migrate

⑧ In admin.py →

# Check admin section

⑨ ★ Regularly check for succesful changes by python manage.py runserver

⑩ create index.html

⑪ Populate → create file in project
Refer laptop ← → code
→ cmd protwo > python file-name

⑫ create user table in user.html →

⑬ Run server mainpage → index.html
/admin → admin page
/users → user page

If want to create user table at user page

Scanned with CamScanner

# Django Forms

HTTP → enable communication between a client and a server
→ request / respond

for ← Request/Response
METHODS

**Get**
request data from a resource

**POST**
submits data to be processed to a resource

## # Creating basic form →

① Create project basicform, app basicapp

② Create template folder / basicapp folder / index.html / form.html

③ do changes in settings.py — Template_DIR
DIR = [ ]
INSTALLED APPS

④ create file forms.py in basicapps.

code →

```
from django import forms
~~from django.core import validators~~

class FormName(forms.Form):
    name = forms.CharField()                    # name char
    email = forms.EmailField()                  # email field
    ~~verify email = forms.EmailField(label='Enter your mail again')~~
    text = forms.CharField(widget=forms.Textarea)   # same as Html input = textarea / text area
```

CLEAN DATA → All validated checked data

⑤ Go to views.py →

```
from django.shortcuts import render
from . import forms   <-- or --> from forms import FormName
def index(request):
    return render(request, 'basicapp/index.html')

def form_name_view(request):
    form = forms.FormName()
    if request.method == 'POST':
        form = forms.FormName(request.POST)
        if form.is_valid():
            print("Name"+form.cleaned_data['name'])
    return render(request, 'basicapp/form_page.html', {'form':form})
```

forms.py

class name

import from same directory as current.py file

If hit submit with method POST

If we get POST back, check if data is valid and so grab data

Method to check if each input field has correct datatypes

dictionary to access valid data

input field as key

⑥ Go to urls.py of project

```
from basicapp import views
urlpatterns [
    path('admin/', admin.site.urls),
    path('formpage/', views.form_name_view, name='form_name')
]
```

⑦ Go to form_page.html

```
<div>
<form method = "POST">
{{ form.as_p }}
{% csrf_token %}
<input type="submit" value="Submit">
</form>
</div>
```

data to be submitted to be processed

form to be inside para tag

will accept info and secure it and check if inputs on form submitted is not going on some other website

MOST

⑧ runserver

# Form Validation

① Creating botcatcher → Hidden field remains in Html but hidden from user. This field is used to catch bots. can be seen in INSPECT.

In formName

* botcatcher = forms.CharField(required = False, widget = forms.HiddenInput)

creating own Validator

① def clean_botcatcher(self):
keyword / checked to check for item / item

botcatcher = self.cleaned_data['botcatcher']
valid grab data what botcatcher returns in forms
dictionary attribute

Means a robot scraped the page and filled form and html field

if len(botcatcher) > 0:

raise forms.ValidationError("caught...")
to raise errors

* If bot is catched, will show on page [Hidden field botcatcher caught!]
check by ← i.e. if it's changes something in html

Instead ① built in method ② ...

from django.core import validators

checks that it should be 0

botcatcher = form.CharField(_____, validators = [validators.MaxLengthValidator(0)])
method ↓ built-in validator

More validators creating

• def check_z(value) → must to identify as validator if making own validator
creating a function to check for z

if value[0].lower() != 'z':
raise forms.ValidationError("Name needs to start with z")
error

add validator → name = forms.CharField(validators=[check_z])

→ verify email double check →

add a variable in class ← verify_email = forms.EmailField(label = "Enter email again")

def clean(self): ← keyword to check entire form at once

dictionary ← all_clean_data = super().clean() → grab all clean data at once

email = all_clean_data['email']

vmail = all_clean_data['verify_email']

check for match ← if (vmail != email)

raise forms.ValidationError("Make sure emails match")

---

## Connecting forms to Model

① FrontEnd

edit index and user + insert form tag in users.html

② Backend

i) forms.py in app Folder

from ____ ____

from ____ ____

class NewUserForm(forms.ModelForm) → to connect to Model

class Meta():

model = User → link to class

fields = '__all__'

③ views.py

import Render

from appTwo.forms import NewUserForm ← only 2 needed

def index(__):

def users(request):

form = NewUserForm()

if request.method == "POST":

form = NewUserForm(request.POST)

if form.is_valid():

form.save(commit=True) ← Save to form → saved in admin database

to database

return index(request) → will take to homepage

else:
print('Error')

return render(request, 'Apptwo/users.html', {'form':form})

---

## → Relative URLS with Templates →

i.e. hardcoded urls ⟶ href with URL template

steps →

① create project learning_templates, app basicapp, templates and templates/ basic_app / index.html
folder folder other.html
base.html
~~other~~ relative_url_template.html

② change settings

③ views.py

def index ——
return ——

def other ——
return ——

def relative ——
return ——

④ project > urls.py

path('' ——
path('admin/ ——
path('basicapp/', include('basicapp.urls'))

⑤ basic app > urls.py → for template tagging

app_name = 'basicapp'

urlpatterns = [
path('relative/', view ——),
path('other/' —— name='other'),
]

⑥ in relative_url_template.html

add → <a href = "{% url 'basicapp:other' %}"> Other Page </a>

same name

* no space or django will look for basicapp

Now it is relative to actual application only

For admin link → already in installed apps

<a href = "{% url 'admin:index' %}"> admin </a>

For index link →

—— "{% url 'index' %} " ——

## Template Inheritance

→ allows to create base template we can inherit for
→ saves repeatative work
→ template extending : extending base.html to other html files

Steps →

① In base.html

<links to JS, CSS, ——>
<html navbar>
<body>

{% block body_block %}

{% endblock %}

Everything outside this will be extended to all pages

In other.html file

<!DOCTYPE html> → MUST

{% extends "basicapp/base.html" %}

{% block body_block %}

{% endblock %}

everything inside will show

## Template Filters

General form of template filter →

{{ value | filter : "parameter" }}

* search django + templates for documentation

I) Built in template filter →

① Adding a tag in index →

in views.py

```
def index ——
    context_dict = {'text': 'hello', 'number': 100}
    return render(——, context_dict)
```

② adding tag in index.html & adding filter to tag

{{ text|upper }} ⟶ hello ⟶ HELLO

{{ number|add:"99" }} ⟶ 100 ⟶ 199

(filter, value, parameter)

③ For creating own filters →

① → create template tags folder > __init__.py file → to act as model

my_extsas.py file

② → in my_extsas.py →

```
from django import template
register = template.Library()
```

→ a library where tags and filters are registered

(template way)

```
@register.filter → if no arg used will take fn name as name
def cutout(value, args)
    return value.replace(arg, '')

@register.filter(name='lower')
def lower(value)
    return value.lower
```

```
def cutout( ⟶
    return ——

register.filter(
'cutout' 'cut')
```

fn name → python function

③ In index.html

```
<!DO ——
{% extend ——
{% load my_extsas %}  MUST
{% body ——
<h1> index </h1>
{{ text|cutout:'rl' }} ⟶ Hello World → Heo Wod
```

① @register.filter

② register.filter('name', python fn)