🐍 Python : Django Framework

🤾 Web Framework for python people

Objectives

- Django Introduction
- Installation and Project Structure
- Django Fundamentals
- Case Study: Simple App

😇 About Me

✅ Aniruddha Kudalkar
✅ 11 Years Of Experience
✅ Founder, Entrepreneur
✅ Full Stack Developer
✅ Full Stack Trainer

Programmer by Brain, Teacher by Heart

Django Introduction

Django Introductions

 Easier, Quick, Better, Less Code

- Fundamentals
- Features

📌 Fundamentals

⛹️ Python-based free and open-source web framework

History

- Invented to meet fast-moving newsroom deadlines, while satisfying the tough requirements of experienced web developers
- Named after guitarist Django Reinhard

Need Of Django

- primary goal is to ease the creation of complex, database-driven websites
- reusability and "pluggability" of components, less code, low coupling, rapid development, and the principle of don't repeat yourself.

📌 Features

🏃 From concept to launch, is a matter of hours

Python

- Completely written in python
- Python is used throughout, even for settings, files, and data models.

Features

- **Very fast.** Concept to completion as quickly as possible.
- **Fully loaded.** Handle common web development tasks
- **Reassuringly secure.** Security considered at its core.
- **Exceedingly scalable.** Meets the heaviest traffic demands.
- **Incredibly versatile.** CMS, SM, SC etc

Installation, Project Structure

Installation, Project Structures

⛹️ pip install Django==3.2.9

- System Requirement
- Installation Steps
- Running your first app
- Understanding Project Structure

📌 System Requirements

🤾 Very Lightweight, can run on Raspberry Pi also.

Hardware Requirement

- 4GB Ram
- Core I3 Processor
- 128 GB HDD/SSD

Note: Above details are given wrt learning purpose only.

Software Requirement

- Python 3.9.x
- Ubuntu 20.04
- PyCharm/VSCode
- MySql/SQLite/Postgres

Note: Above details are given wrt learning purpose only.

📌 Installation Steps

⛹️ pip or git clone

Virtual Environment

- allow you to manage separate package installations for different projects.
- allow you to create a “virtual” isolated Python installation and install packages into that virtual installation.
- python3 -m venv env (if not installed)

Django Installation

- Can be installed from Source and Pip
- Change Environment
- python -m django --version (if error)
- pip install Django==3.2.9 (only once)

Creating/Running Project

- django-admin startproject mysite
- python manage.py runserver

📌 Project Structure

⛹️ Settings, Db Config, App Settings, Django Specification

Important Files

- **manage.py:** lets you interact with this Django project
- **__init__.py:** directory should be considered a package
- **settings.py:** project configuration

Important Files

- **urls.py:** entry points of projects
- **asgi.py:** entry-point for ASGI-compatible web servers
- **wsgi.py:** entry-point for WSGI-compatible web servers

Django Fundamentals

Django Fundamentals

⛹️ These build your app

- Models
- Http
- Forms and Templates
- Testing
- Security

📌 Models

⛹️ Model maps to a single database table

Model

- model is a Python class
- attribute of the model represents a database field
- automatically-generates database-access API

Simple Model

```
from django.db import models


class Person(models.Model):
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=30)
```

Generated Table

```
CREATE TABLE myapp_person (
    "id" serial NOT NULL PRIMARY KEY,
    "first_name" varchar(30) NOT NULL,
    "last_name" varchar(30) NOT NULL
);
```

📌 Http

⛹️ Handling HTTP requests, responses and other needed stuff

URL Dispatcher

- Pure Python code; mapping between URL to python function
- Can be constructed dynamically
- Supports i8n easily

Simple URL Dispatcher

```
from django.urls import path

from . import views

urlpatterns = [

    path('articles/2003/', views.special_case_2003),

    path('articles/<int:year>/', views.year_archive),

    path('articles/<int:year>/<int:month>/', views.month_archive),

    path('articles/<int:year>/<int:month>/<slug:slug>/', views.article_detail),

  ]
```

Views

- Pure Python code;
- Takes a Web request and returns a Web response
- response can be anything

Simple View

```
from django.http import HttpResponse

import datetime

def current_datetime (request):

    now = datetime.datetime.now()

    html = "<html><body>It is now %s.</body></html>" % now

      return HttpResponse(html)
```

Middleware

- low-level “plugin” for altering Django’s input or output.
- responsible for doing some specific functions like authentication, sanitization etc
- django given built in middlewares

Simple Middleware

```
class SimpleMiddleware:
    def __init__(self, get_response):
        self.get_response = get_response

    def __call__(self, request):
         return response
```

📌 Forms and Templates

⛹️ Managing forms is complex, django makes it easy

Forms

- preparing and restructuring data to make it ready for rendering
- creating HTML forms for the data
- receiving and processing submitted forms and data from the client

Simple Form

```
<form action="/your-name/" method="post">
    <label for="your_name">Your name: </label>
    <input id="your_name" type="text" name="your_name" value="{{ current_name }}">
    <input type="submit" value="OK">
</form>
```

Django Form

```
from django import forms


class NameForm(forms.Form):
    your_name = forms.CharField(label='Your name', max_length=100)
```

Templates

- generates HTML dynamically
- can be configured with one or several template engines
- built-in engine called the Django template language (DTL)
- standard API for loading and rendering templates

Simple Template

```
My first name is {{ first_name }}. My last name is {{ last_name }}.
```

```
{% if user.is_authenticated %}Hello, {{ user.username }}.{% endif %}
```

📌 Testing

🤾 Layered architecture makes testing difficult

Unit Testing

- Http request simulation
- Insert test data
- Output inspection
- Uses python unittest module

Simple Test

```
from django.test import TestCase
from myapp.models import Animal
class AnimalTestCase(TestCase):
    def setUp(self):
        Animal.objects.create(name="lion", sound="roar")
    def test_animals_can_speak(self):
        """Animals that can speak are correctly identified"""
        lion = Animal.objects.get(name="lion")
        self.assertEqual(lion.speak(), 'The lion says "roar"')
```

📌 Security

🤾 Security is baked in an architecture

Built In Support

- XSS
- CSRF
- SQL Injection
- Clickjacking
- SSL
- Host Header Validation
- Referrer Policy
- Session Security
- User Uploaded Content

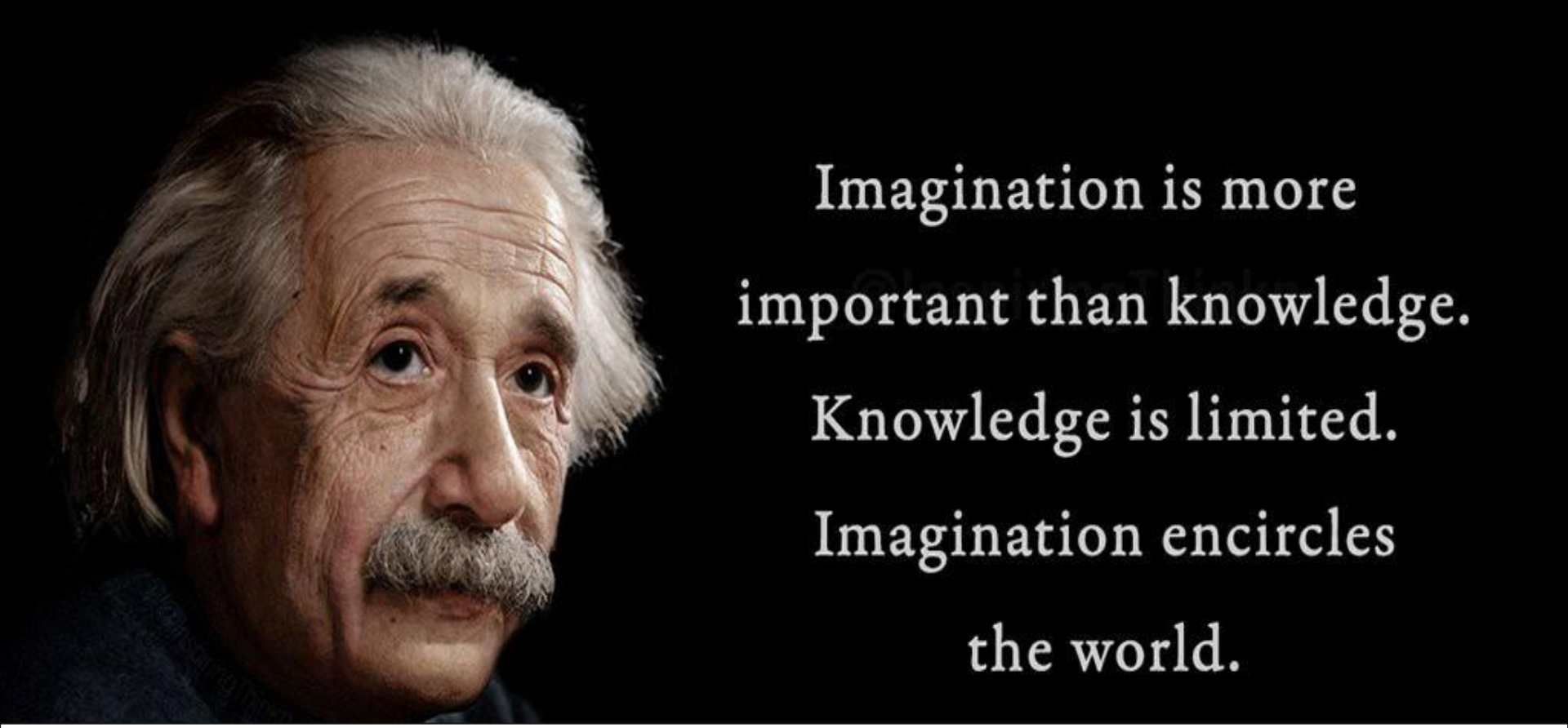
Case Study : Simple App

Credits

https://en.wikipedia.org/

https://www.djangoproject.com/

https://github.com/django/django



🎉 Thanks For Your Time 🎉