Django

1. What is Django
   - Django is a high-level Python full-stack web framework that encourages rapid development and clean, pragmatic design.
   - Django follows the model–template–views (MVT) architectural pattern.
     - **Models**: Represent data structures and interact with the database.
     - **Views**: Handle user requests and determine how to respond.
     - **Templates**: Define the presentation layer (HTML) of the application.
2. Django Project vs Application
   - Project
     - A Django project acts as the top-level directory that houses your entire web application. It includes configurations, and settings.
     - A project can contain multiple apps.
   - Application
     - A Django app is a self-contained module designed to accomplish a specific task within the overall project.
     - You can use an app in multiple projects, and you can distribute apps, because they don't have to be tied to a given Django project (installation).
3. Installing django
   - To create a Django project, first, you need to install Django on your operating system.
   - **pip install Django**
   - After installation check django on your system. **django-admin --version**
4. Create django project
   - Once you have installed Django on your operating system, now you can create a project.
   - Commmad:- **django-admin startproject <project_name>**
   - Navigate to the project directory and start the development server for test - '**python manage.py runserver**'.

5. Create an Application
    - Django, it’s a common practice to create a new app for each distinct feature or functionality you want to add to your project.
    - Creating separate apps for each feature helps to keep your code organized and modular.
    - It allows you to reuse apps across different projects, and it makes it easier to maintain and scale your code as your project grows.
    - Command - **python manage.py startapp <app_name>**
    - Register the app with our Django project by updating **INSTALLED_APPS** tuple in the **settings.py**.
    -

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'myapp'
]
```

6. Creating Views
    - Django views are Python functions that takes http requests and returns http response.
    -
```python
from django.http import HttpResponse


def index(request):
    return HttpResponse("Welcome to OB")
```
7. URL Mapping in the application
    - To call the view, we need to map it to a URL - and for this we need a URL configuration.
    - To create a URL configuration in the app directory, create a file called **urls.py**.
    -
```python
from django.urls import path

from . import views

urlpatterns = [
    path("", views.index, name="index"),
]
```

- The next include your app **urls** in the **root** URL configuration.
-

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('myapp', include('myapp.urls'))
]
```

8. Template System

- In Django, a template is a text document or a Python string marked-up using the Django template language (DTL).
- It's used to generate dynamic HTML content.
- Django's template engine offers a mini-language to define the user-facing layer of the application.
- The syntax of the Django template language involves four constructs.
  - Variables
    - A variable outputs a value from the context.
    - Variables are surrounded by **{{ var_name }}** like this.
    - E.g. **User role is {{ user_role }}**
  - Tags
    - Tags provide arbitrary logic in the rendering process.
    - Tags are surrounded by **{%** and **%}** like this:
    - Tags lets you perform the following operations: **if** condition, **for** loop, template inheritance and more.
    - Just like in Python you can use **if, else** and **elif** in your template

- Filters
  - Filters help you modify variables at display time.
  - Filters structure looks like the following: **{{ var|filters }}**
  - E.g. **{{ customer_name| lower}}** − Converts the customer name to lowercase.
- Comments
  - Comments look like this: **{# this is commented #}**
  - OR **{% comment %} this is commented {% endcomment %}**

- **Create django template.**
  - **NB**. Before configuring templates, make sure you have added the app to **INSTALLED_APPS** in settings.py.
  - Django looks for a **templates** folder within each app to load templates.
  - Create html file in **templates** folder.
  - The Render Function
    - It plays a crucial role in presenting dynamic web pages by bringing together various elements.
    - It takes three parameters.
      - **Request** − The initial request.
      - **The path to the template** − This is the path relative to template file.
      - **Dictionary of parameters** − A dictionary that contains all variables needed in the template.
    - How to pass context to the template
      - The render function accepts the context parameter to pass variables to the templates.

9. Database

- Django officially supports the following databases (refer official documentation https://docs.djangoproject.com/en/5.0/ref/databases/)
  - PostgreSQL
  - MariaDB
  - MySQL
  - Oracle
  - SQLite
- In this tutorial, we will use MySQL.
  - For use MySQL you need to install the MySQL driver
    - **MySQL Connector/Python** is a pure Python driver from Oracle that does not require the MySQL client library or any Python modules outside the standard library.
      - ✓ **pip install mysql-connector-python**
      - ✓ Egine**: mysql.connector.django**
    - **Mysqlclient** - mysqlclient library for Python does use the MySQL C client library.
      - ✓ **pip install mysqlclient**
      - ✓ Engine: **django.db.backends.mysql**

- Change database connection in **settings.py**

```
DATABASES = {
    'default': {
        'ENGINE': 'mysql.connector.django',
        'NAME': 'ob_training',
        'USER': 'ban',
        'PASSWORD': '1234',
        'HOST': 'localhost',
        'PORT': '3306',
    }
}
```

10. Django - Models

- A Django model is a Python class that represents a database table or collection.
- It is used to define the structure and behavior of the data that will be stored in the database.
- Each attribute of the model class represents a field in the corresponding database table.
- Each model maps to a single database table.
- **NB**. Before configuring model, make sure you have added the app to **INSTALLED_APPS** in settings.py.
- Defining a Model
  - A model is defined as a subclass of **django.db.models.Model** class

- Create model class in **models.py.**
  -
```python
from django.db import models


class Blogs(models.Model):
    title = models.CharField(max_length=100)
    page = models.IntegerField()
```
  - Prepare migrations.
    - **python manage.py makemigrations** - the command that creates new migrations based on the changes detected in your models.
  - Apply migrations changes.
    - **python manage.py migrate** - command in Django used to apply migrations that have been generated by the **makemigrations** command.
  - Save model to DB.
  -
```python
# Intialize the model object
blog = Blogs(title="New", page=33)
# Save the object into the database.
blog.save()
```

- Query from Database
-
```python
# Fetch all
blogs = Blogs.objects.all()
print(blogs)

# Get one
blog = Blogs.objects.get(pk=1)
print(blog)
```

11. F
12. Sd
13. d