COMPUTER SCIENCE With python

Textbook for Class XII

- Programming & Computational Thinking
- Computer Networks
- Data Management (SQL, Django)
- Society, Law and Ethics

SUMITA ARORA

DHANPAT RAI & Co.

15 Creating a Django based Basic Web Application

477 – 510

15

Creating a Django based Basic Web Application

In This Chapter

15.1 INTRODUCTION

These days, Internet has invaded in all spheres of our lives whether directly or indirectly. Websites are no exception to this. To develop efficient websites, there are many web frameworks available today. **Django** is one such web framework that is used along with Python to develop dynamic websites.

In this chapter, we shall introduce Django web framework to you, cover basics of web development through Django and then we shall create a minimal simple web application based on Django web framework.

Please note that the best way to learn this chapter is to do as suggested here, on your computers, side by side.

15.2 WHAT IS WEB FRAMEWORK ?

Since Django is a web framwork, let us quickly discuss what a web framework is before starting with Django framework.

Modern day web has *web enabled applications* and *websites* that can change and update their content depending upon the users' need or a specific area. For example, you often see *Google doodle* related to history of your country or on some prominent figure of your country.

But do you know that Google doodle is country specific – it shows different doodles in different countries ? For instance, both *South Korea* and *Vietnam* celebrated *Lunar New Year* on *February 5th*, but Google showed different doodles in these countries.

This is possible because it is a **dynamic website**. A dynamic website's display contents can be changed from time to time. To build dynamic websites, web frameworks are used. There are many web frameworks popular today, *e.g.*, **Django-Python, Ruby on Rails.Ruby, Express-js-Javascript** etc.

WEBFRAMEWORK

A **webframework** is a software tool that provides a way to build and run dynamic websites and web-enabled applications.

Web Framework vs. Library

It is important to know the difference between a **web framework** and a **library**. A web framework makes available the majority of code with all the internal links and interdependencies solved where you just need to fit in the code that specifies the custom features you want to add to it. A **library**, on the other hand, provides some prewritten codes that can be used in an application being fully developed by you.

You can understand the difference between these two with this analogy. *A web framework is somewhat like as if you purchased a fully designed flat (a la web framework) wherein you fitted some windows and doors as per your liking (a la your code). A* ***library****, on the other hand is somewhat like as if you are building your own house but you are using some predesigned doors and windows (a la* ***library code****) in your house (your complete application).*

NOTE

A web framework will do majority of work and utilize your code in it and a library's function can be utilized in an application being developed by you.

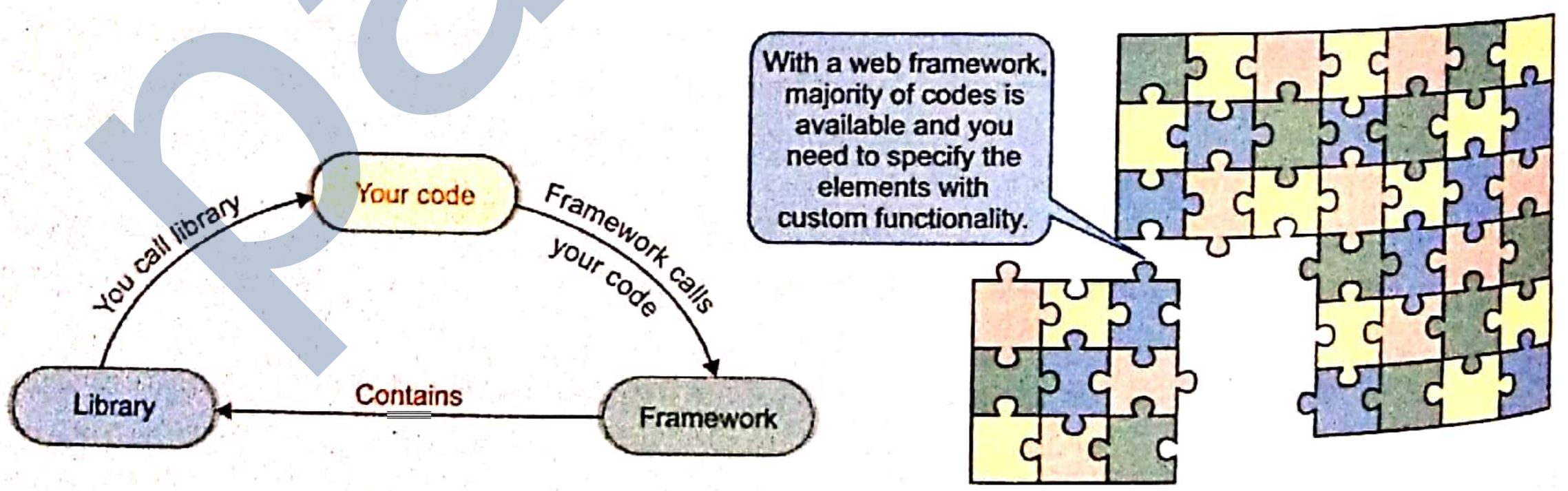


Figure 15.1 (*a*) Web framework *vs.* Library. (*b*) With web framework, you add some codes for custom functionality

15.3 HOW WEB, WEBSITES AND WEB-APPLICATIONS WORK?

Although you have an idea how web and web browsers work, let us learn about certain technical terms and methods used in doing so. This is important for you to know these technical terms as you shall be using these in Django web application development.

You already know that the Web works in the form of **client-server architecture.** Your **web browser acts as a client program** (***web client*** or the ***front-end***), and **the web server** with which it interacts **is the server (the backend).**

The clients (web browsers) work on the web mostly with HTTP protocol. The clients make an **HTTP request,** which the (web) server responds to in the form of a response called **HTTP response.** The clients mostly make these **two types of HTTP requests :**

(*i*) **HTTP GET request.** Whenever the web client has to display a webpage, it makes a GET request and sends the URL of the webpage. A GET request is always accompanied by a URL. The server (web server) receives the GET request and responds by sending the HTML of the URL, if available. If no such URL exists, it returns an error code (often the Error 404 that you see is this error that server sends when the asked URL is not found).

HTTP GET REQUEST

An **HTTP GET Request** refers to a way of retrieving information from a web server using a given URL over web.

(*ii*) **POST request.** Whenever, a web client has to send some data, *e.g.*, you filled up your details in a web page (such as while creating your account on a shopping site and so on), this data will be sent to the web server for storing in a database through a POST request. The HTTP response to a POST request is also either an error code (if not executed successfully) or success code (if completed successfully).

HTTP POST REQUEST

An **HTTP POST request** is a way to send data to the server, (*e.g.*, data filled in an online form such as student information, file upload, etc.) using HTML forms over web.

Following Fig. 15.2 shows the meaning of GET and POST requests.

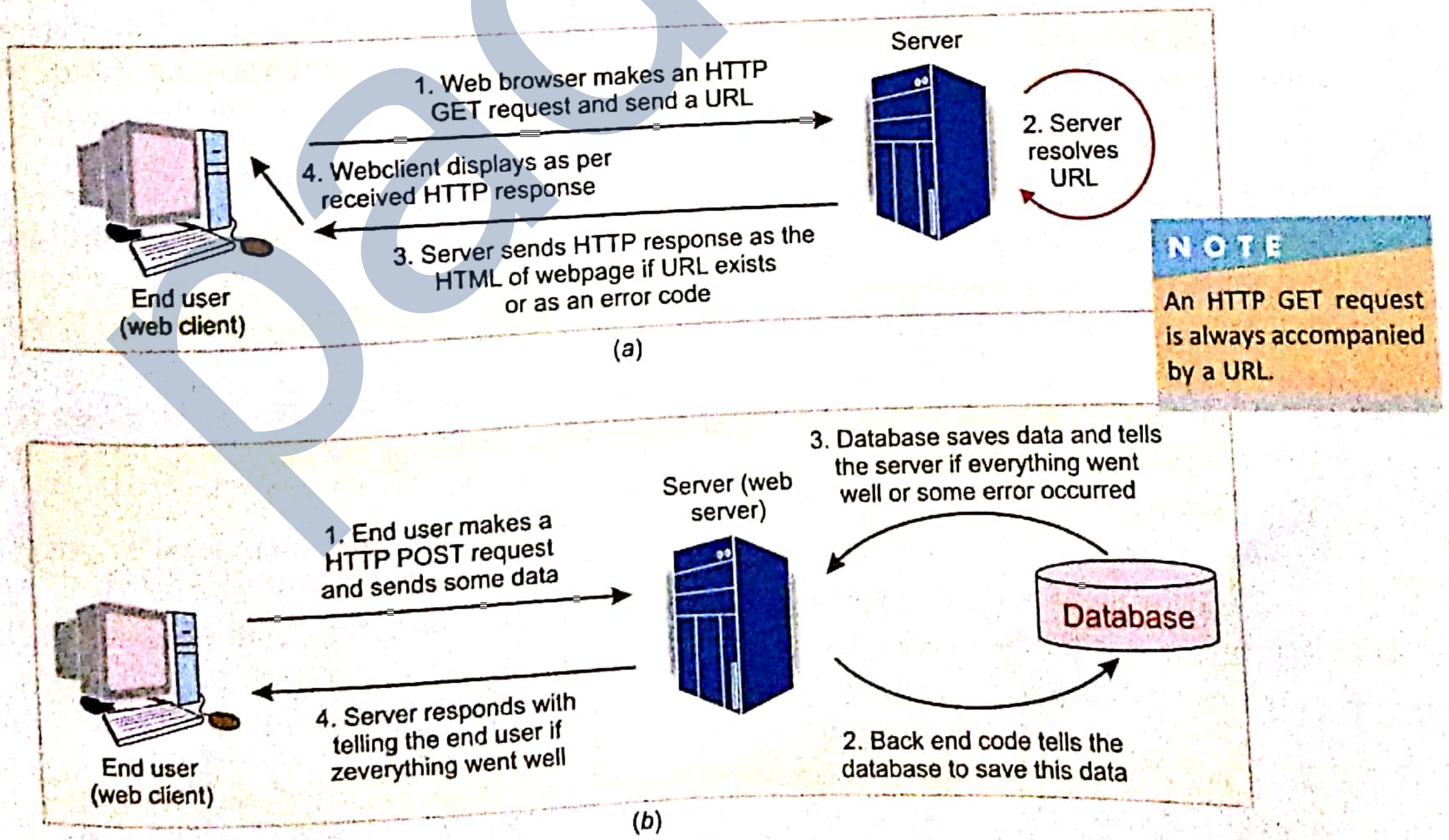


Figure 15.2 (*a*) GET request (*b*) POST request.

15.4 INTRODUCING DJANGO

Django is a Python based free and open source web application framework. You have read in the previous section that a web framework is a collection of modules used for creating dynamic websites and it takes care of much of the web development.

Why Django ?

Django is a powerful and popular server-side web framework, used by millions of developers world wide because of these advantages offered by it.

(*i*) ***Secure.*** Django offers code and enables protection against most of security vulnerabilities of websites automatically.

(*ii*) ***Versatile.*** Almost any type of website (news site, e-commerce site, a social networking site and so forth) can be built using Django.

(*iii*) ***Portable.*** Since Django is Python based, the websites developed in it can run on any platform like Linux, Windows or Mac OS.

(*iv*) ***Easy to maintain.*** Django code is easy to reuse and easily maintainable.

15.5 INSTALLING DJANGO

Before we practically begin with Django, it is important for you to install it on your computers. We shall work with the latest versions of Python and Django for this purpose. For those, who have a previous version of Python installed, make sure that the Python version should be 3.5 or later. This is because the latest version of Django (as of now) is 2.1 and this supports Python versions 3.5, 3.6 and 3.7.

For those, who have installed Python through Anaconda distribution, we recommend you to install Django through **pip**, the installer utility of Python, on standard Python 3.x distribution. This is because, it is recommended to install **Django in a virtual environment**, which we generally do by using **pip**[1].

What is *virtualenv* and Why to use it ?

Virtualenv is a useful tool which creates **isolated Python environments**, which means that there can co-exist different Python environments without disrupting others functioning as co-dependencies and interdependencies are handled. To understand the use of virtual environments, read the example below :

Suppose you have developed a successfully running application (say *Application1*). Now on your computer, you want to build another application (say *Application2*). Here, there are specific issues you must handle :

(*i*) *Application1* uses some libraries of Django and Python, and you do not want to change the libraries used by Application1.

(*ii*) The new application (*Application2*) requires the same libraries' updated versions.

Now, what to do? If you update the libraries, the Application1 will not work correctly and if you do not, Application 2 cannot work. What is the way out then ?

1. Installing a virtual environment in conda is very much possible but this also requires the use of conda command prompt. Thus, we shall stick to the regular command prompts of our computer : ***the Terminal*** for *Linux* and *Mac OS* and **cmd** or **Windows Power Shell** for *Windows*.

The way-out or the answer to the above problem is the *virtual environment*. Using the virtual environment, you can create two isolated Python environments where one would have the older libraries, and the other would be having the updated libraries. Now two applications can run in their own virtual environments.

NOTE

Virtualenv is a useful tool which creates isolated Python environments that take care of interdependencies and let us develop different applications in isolation. Each isolated environment has different set of libraries unlike a common global set of libraries.

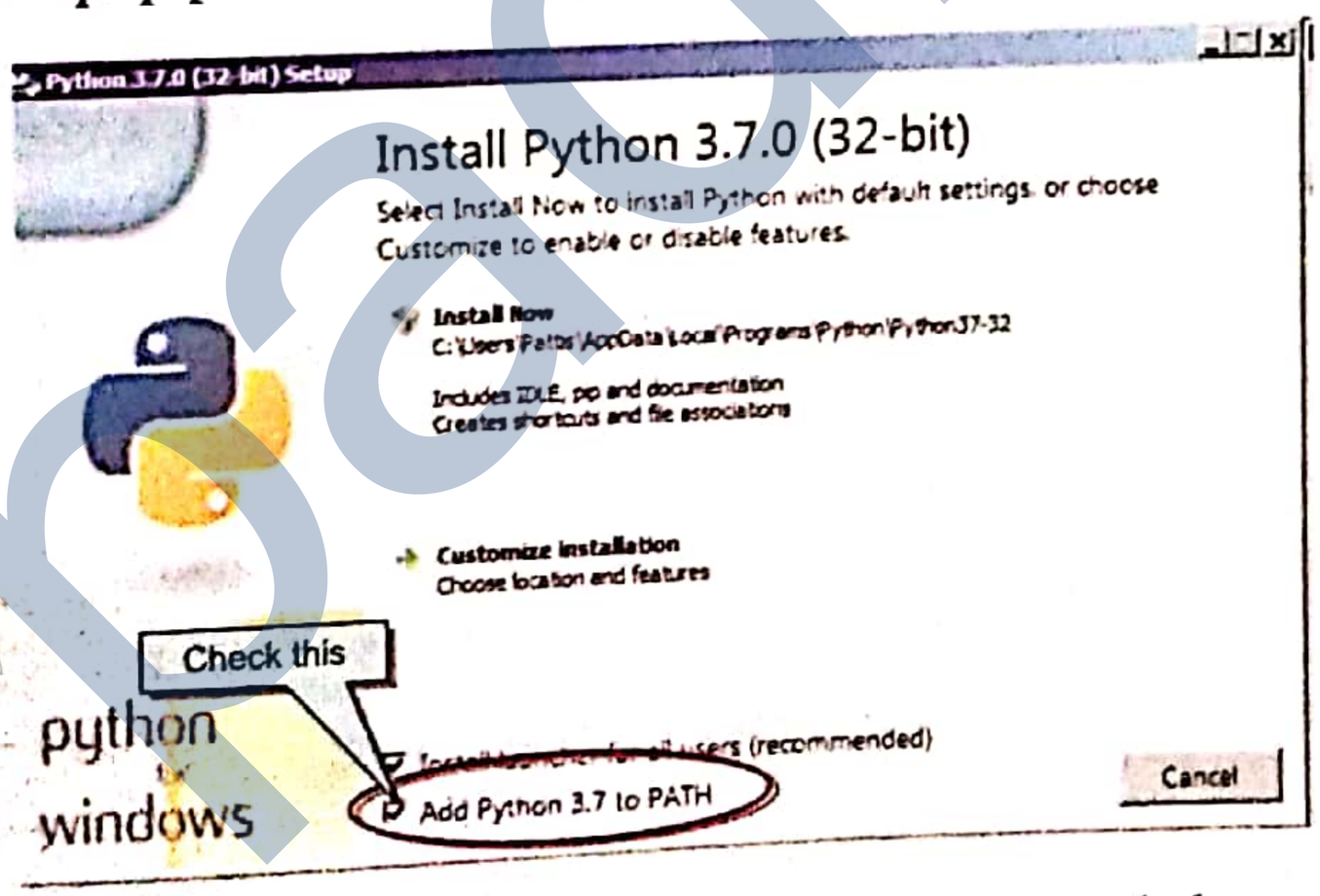
Django is named after **Django Reinhardt**, a gypsy jazz guitarist from the 1930s to early 1950s who is known as one of the best guitarists of all time.

Django web framework is managed and maintained by an independent and non-profit organization named Django Software Foundation (DSF).

15.5.1 Installing Django in Virtual Environment

Before installing Django in virtual environment, make sure that regular distribution of Python 3.5 or later is installed on your computers. You can download Python's latest 3.x distribution from the site *https://www.python.org/downloads/*. After downloading, run the file to install it on your computers. Make sure to tick *Add PATH box* to add newly installed Python to system environment variable PATH (see below).

In the following lines, we are showing the installation of Django on Windows platform. The same commands will also work for Linux platform, all you need to do is add **sudo** (super user do) **apt** before each command. *E.g.*, if command is **pip install <package>** on Windows, on Linux, it will be **sudo apt pip install <package>**. *Appendix C* shows Django installation on Ubuntu Linux.



Internet connectivity is required only for Django's Installation, not afterwards.

To install Django on Windows, follow the steps given below :

(*i*) Make sure you are connected to Internet (only for installation).

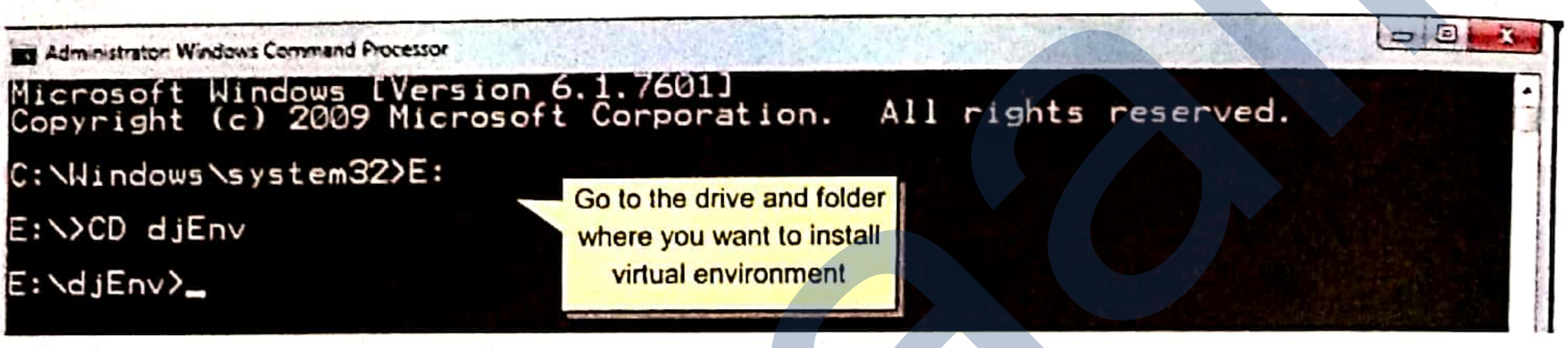
(*ii*) Start **cmd** (Windows 7) or **Windows Power Shell (Windows 10)** in administrator mode. For this, do the following :

- From the *Start button*, search for **cmd** or **Windows Power Shell**
- Right click **cmd** or **Windows Power shell** and click **Run as administrator.**
- You can also work in the normal PowerShell (Windows 10) after writing the command "**Set-Execution Policy Unrestricted**".

(*iii*) In the shell window, go to the drive and folder where you want to install virtual environment for Django. We have created folder namely **djEnv** on **E:** drive for this purpose. (You can create a folder in the same manner as you normally do but you should remember its path clearly, as you will need to do to this folder for Django installation.)

- Type <drive name> followed by a colon to go to a drive, *e.g.*, **E:** will change the working drive to E drive of your computer.
- Next go to the folder where you want to install your virtual environment. Use command **CD** to change folder, *i.e.*,

```
cd <folder's path on the current drive>
```



Command **CD djEnv** will change current folder to **djEnv** on current drive (see above)

(*iv*) Once in the desired folder, type the command :

```
pip install virtualenv
```

It will download and install virtual environment on your computer.

(*v*) Next you need to activate virtual environment by giving **activate** command. Since it lies in folder **venv\Scripts** under current directory, the command we shall type will be :

```
venv\Scripts\activate
```
(you can refer to the figure given below)

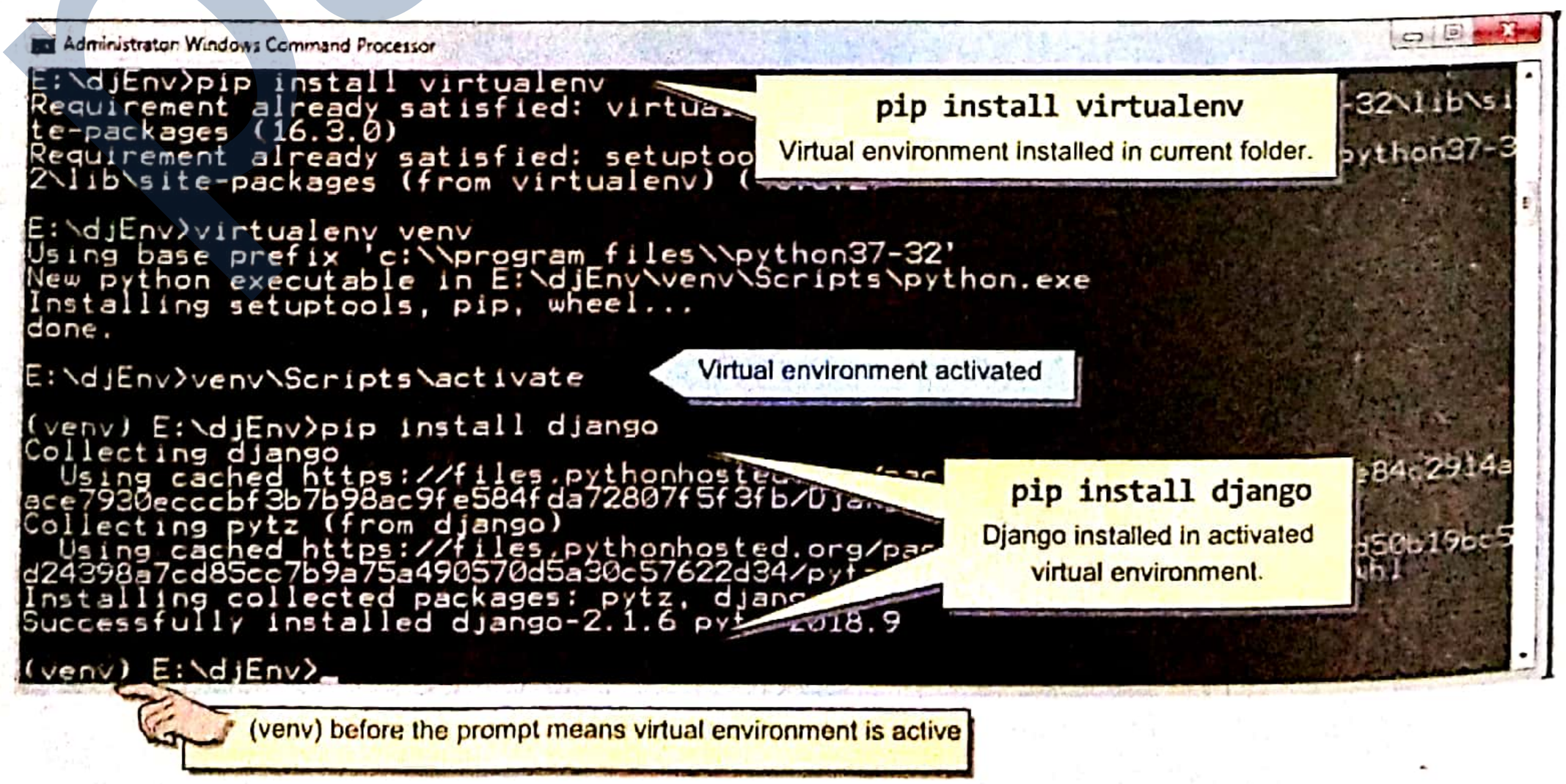
(*vi*) Once activated, you can install latest version of Django by giving command :

```
pip install django
```

Or you may install a specific version of **Django** by giving following command :

```
pip install django == <version>
```

It will download and install Django in the active virtual environment for you.



(*vii*) Once it displays the message ***Successfully installed django***, you can rejoice and move ahead with the *Django application development*, which we shall learn in the next section.

You might need to use these commands on shell (command prompt)

- chdir|cd [drive:][path] – to make the specified drive or path as active folder, *e.g.*,

```
C:> D:
```
Change to D drive i.e., make D drive the active drive

```
D:> cd mywork\djng
```
Make given folder the active folder i e., change to folder **djng** folder under **mywork** folder

```
D:\mywork\djng>
```
You can see the path of current folder in the prompt

- chdir|cd [..] – to make the parent directory/folder as the current folder, *e.g.*,

```
D:\mywork\djng> cd..
```
Change to folder **djng** folder s parent folder

```
D:\mywork>
```
You can see the path of current folder in the prompt

- dir – to view the files and folders inside current folder.

Test-run Django Development Webserver

You shall learn about Django project's structure and other basic functionality but here let us quickly run some commands to test if we have successfully installed Django and its built-in web server is properly working on our computer. This is just for testing purposes, so we shall not explain the commands given below in this section. These commands will be explained later after we have discussed the basic structure of Django projects.

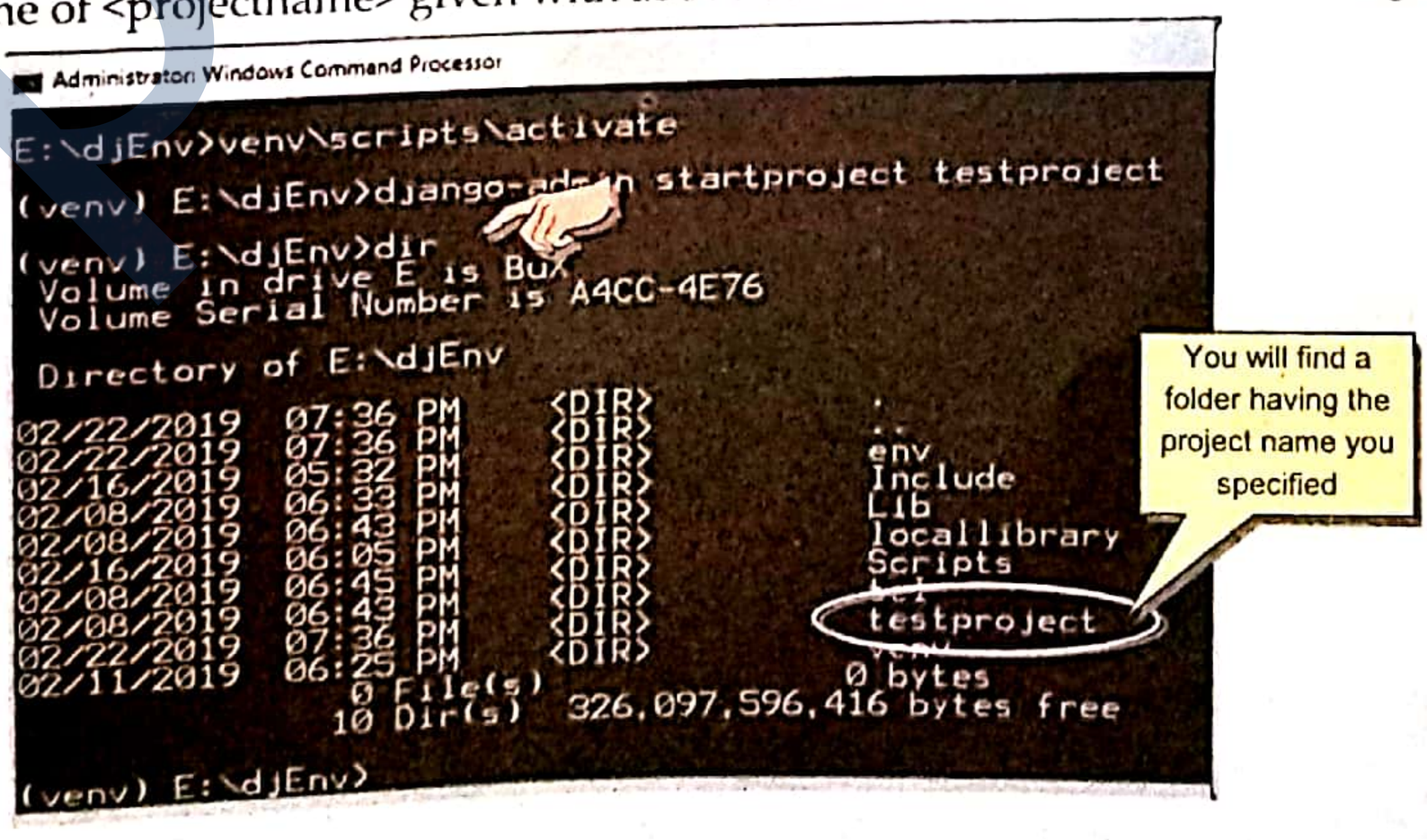
1. In the active virtual environment, type following command in front of the prompt :

   django-admin startproject <projectname>

   e.g., `django-admin startproject testproject`

```
(venv) E:\djEnv>django-admin startproject testproject
(venv) E:\djEnv>
```

2. It will create a directory/folder by the name testproject under the current folder. To confirm type command DIR in front of the command prompt and press Enter. You will find the name of <projectname> given with above command as a folder in the listing that appears.



3. Now change to this folder, your project's folder, using command CD, *i.e.*, give command

```
cd <projectname>
```

We shall give command **cd testproject** as our project's name and project's folder name is ***testproject***. (*see* figure below).

4. Once you changed the current folder to your project's folder, type following command in front of the prompt to run the built-in webserver :

```
python manage.py runserver
```

NOTE

Do not worry if you do not understand anything right now. This is just for testing purposes. All these commands will become clear to you in the coming lines.

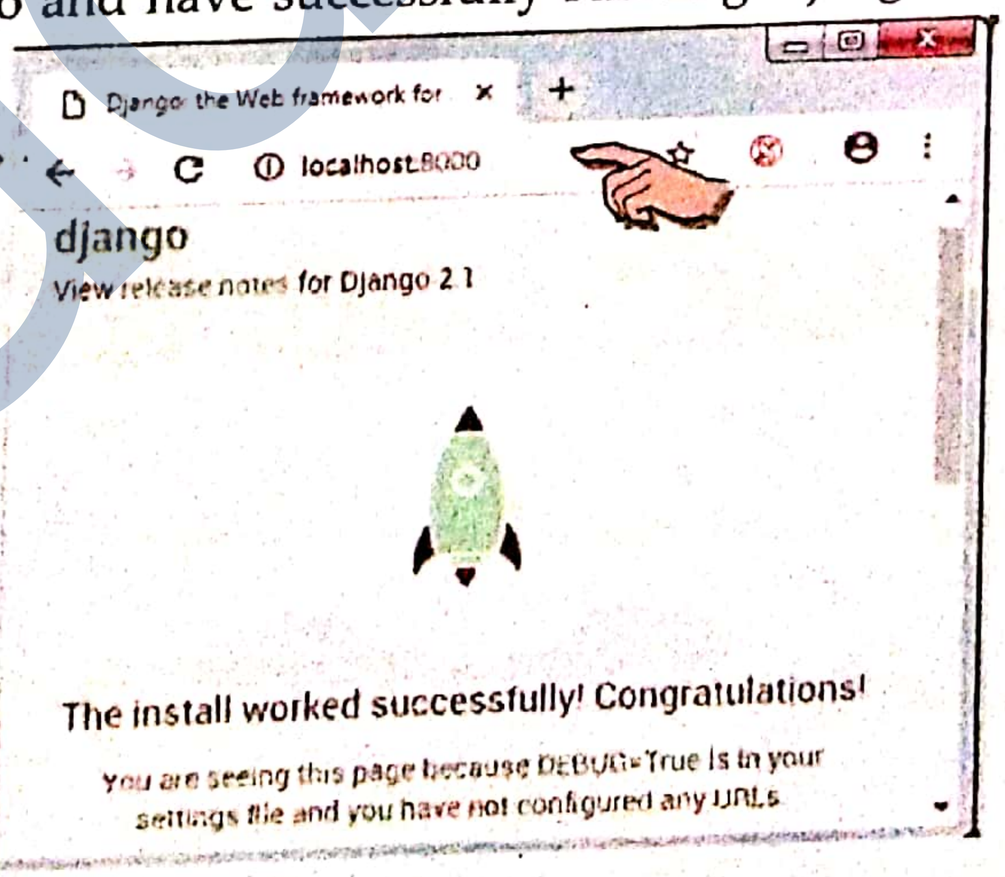
```
(venv) E:\djEnv>cd testproject

(venv) E:\djEnv\testproject>python manage.py runserver
Performing system checks...

System check identified no issues (0 silenced).

You have 17 unapplied migration(s). Your project may not work properly until you
 apply the migrations for app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
February 22, 2019 - 19:43:53
Django version 2.1.6, using settings 'testproject.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

5. Finally, open your web browser window and type either of the following commands in the address bar :

```
localhost:8000
127.0.0.1:8000
```

127.0.0.1 is the *IP address of your local host* (***localhost***) and 8000 is the default port on which it connects to Django webserver.

If your web browser shows you following screen – Congratulations! You have successfully installed Django and have successfully running Django webserver.



NOTE

Default IP address of built-in Django webserver is 127.0.0.1 (localhost) and default port of connection is 8000.

6. In shell window, you can quit the running server by pressing **CTRL+C** or **CTRL+Break.**

15.6 ACTIVATING VIRTUAL ENVIRONMENT

Every time you have to work in a Django project after installation, you need to activate virtual environment first. For this purpose, you need to follow the steps given below to open and work in your Django projects.

1. Start **cmd** or **Windows power shell**. (Search for its name in *Start menu* and run it).

2. Once inside the shell, go to the drive and folder where you want to create your Django project. Since we created E:\djEnv folder for this, we shall first change to *E: drive* by giving E: in front of the prompt (*see* figure below).
   And then we typed command :

   ```
   cd djEnv
   ```

   to change to folder *djEnv*, to make it current active folder. (*see* figure below)

3. Once inside the folder, type command :

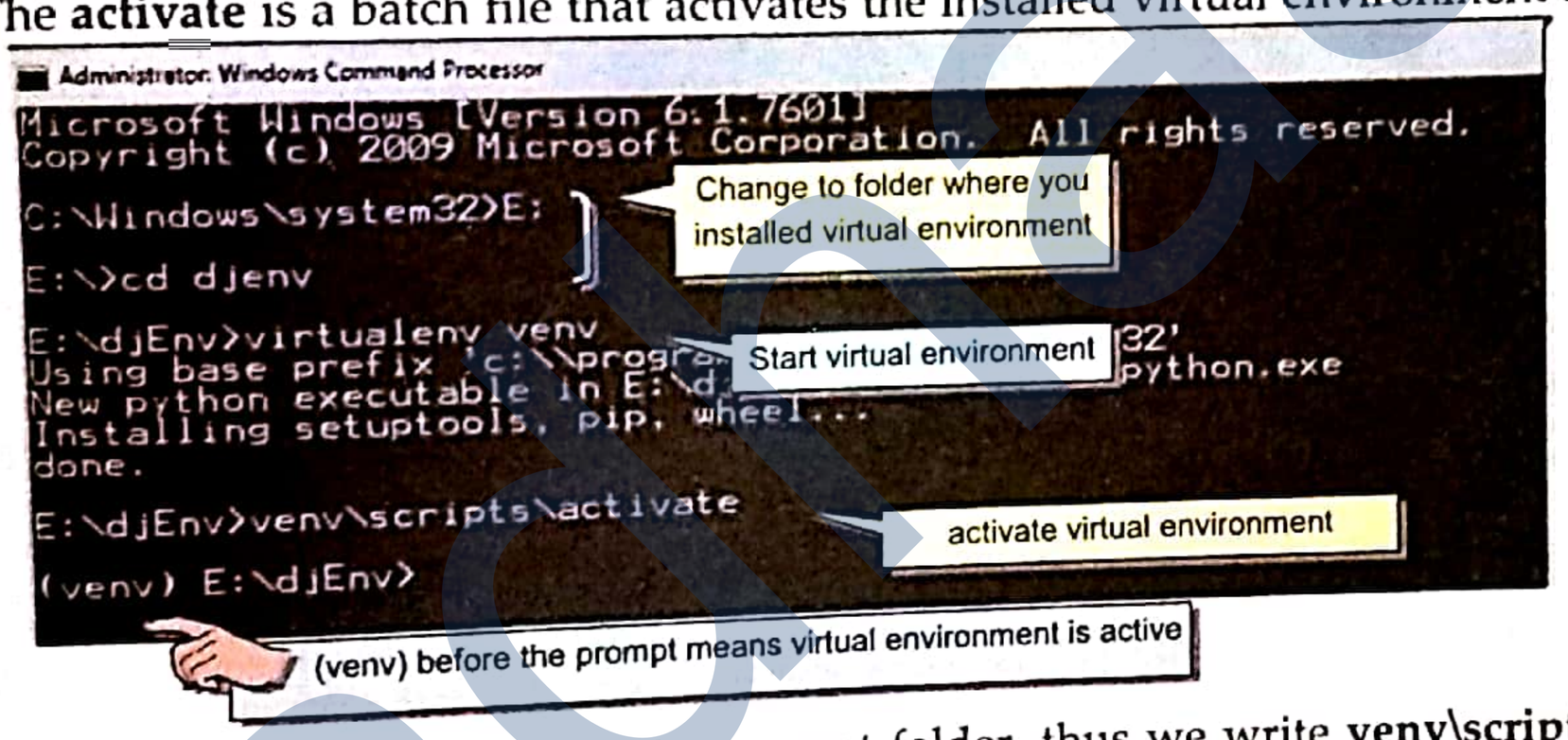
   ```
   virtualenv venv
   ```
   ← *You may also use any other identifier in place of venv*

   Here ***venv*** is an identifier; you can use any identifier other than ***venv***.
   *Please note, **virtualenv** is installed globally on your computer ; you can run it from any folder.*

4. Next, type command **venv\scripts\activate** to activate the virtual environment :

   ```
   venv\scripts\activate
   ```
   ← *If you have used any other identifier than venv, then replace **venv** with your identifier*

   The **activate** is a batch file that activates the installed virtual environment and it resides



   in the folder **venv\scripts** under current folder, thus we write **venv\scripts\activate**.

5. When it displays **(venv)** before the command prompt, it means your virtual environment is now active and now you can work on your Django projects and web applications.
6. Now you can create/open your projects, make changes in files and run them.
7. After working on your Django projects, you can deactivate the virtual environment by giving following command on the command prompt.

   ```
   venv\scripts\deactivate  or  deactivate
   ```

8. And then you can close the shell window (*cmd* or *Windows power shell*) like you close other windows.

15.7 DJANGO BASICS AND PROJECT STRUCTURE

In this section, we shall discuss the basics of Django projects, their structure, Django's working architecture and so on. In this section, we shall discuss the basic terms that make the core of Django. These are : *project, app, model, field, view, template, controller*. Please note that Django is not just limited to these; there are many more things in Django but we shall stick to the basics of Django and will cover only the ones mentioned here and how these relate to one another.

Let us first talk about terms **project** and **app** below and then we shall talk about other terms in a later section – Django Project Architecture.

15.7.1 What are Project and App in Django ?

PROJECT & APPS

A Django **project** refers to an entire application and an **app** is a submodule of the project caters to one specific functionality.

Django uses *two* important terms in development : **project** and **app**

- **A Project** refers to an entire application.
- **An app** is a submodule catering to one part of the project.

For example, you are creating an **online shopping store** on which :

- You can register as a *customer* by providing all the *profile* details such as *name, address* etc.
- There are many **products** listed on the online store giving each ***product's details*** such as *product name, category, size, weight, price* etc.
- Users can select desired products and add to their **shopping cart**.

The above **project is named as online-shop**, and it is the name of overall complete application of *online shopping store*.

This project has *three* sub applications, which will be created using apps : apps on *online-shop* are :

NOTE

Please note, for simplicity sake we have created three different Apps : *customer, products* and *shopping_cart*. But a better design would be to have them as data models in a single app.

`Customers` ← *It will manage the profiles of customers*

`Products` ← *It will manage the products' details*

`Shopping-cart` ← *It will manage the details of selected products chosen for shopping*

Now that you have an idea of what a project means and what an app is, let us quickly learn how you can create projects in Django.

15.7.1A Creating a Project in Django

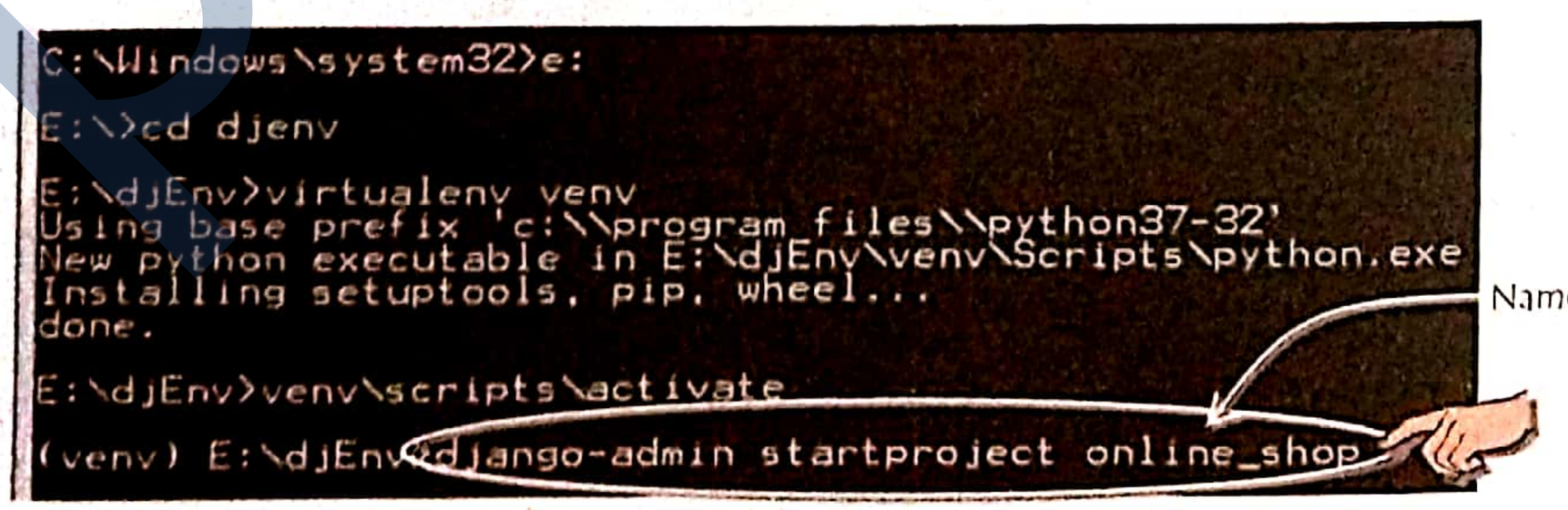
To create an application/project in Django you need to follow the steps given below :

1. Start virtual environment as explained in section 15.6. (steps 1-6)
2. On the command prompt type command :

```
django-admin startproject <projectname>
```

← *Please note that there is hyphen in django-admin*

For example, if your project name is **online_shop**, you will write following command on the command prompt.



3. When you create a project with **django-admin startproject** command, it will create a folder having the same name as that of the project in the current folder. This newly created project folder will have some files and a folder preloaded in it – all because of Django framework. Django framework makes available much of the code and settings beforehand.

Let us now understand the structure of a Django project before we proceed.

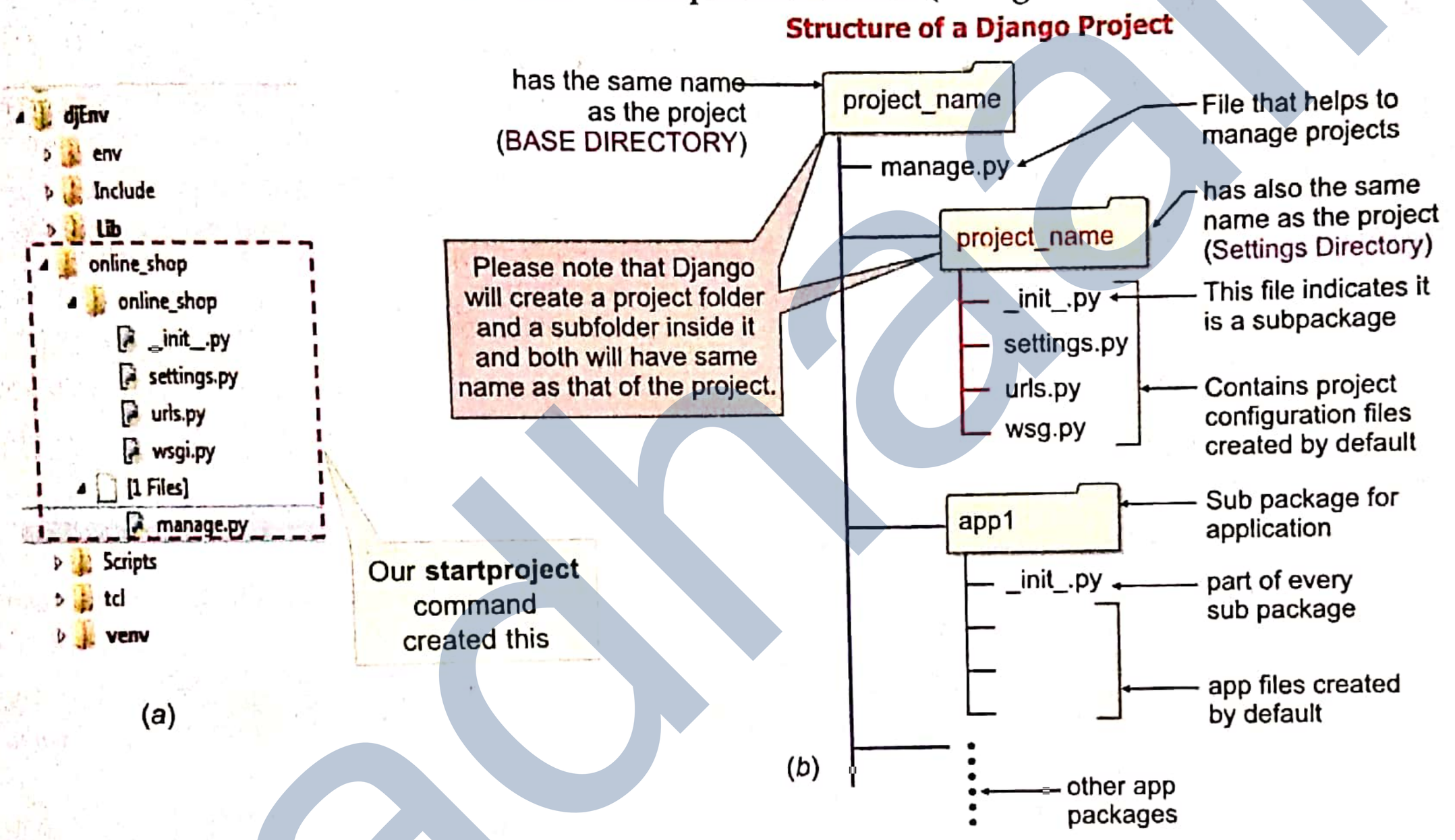
15.7.1B Structure of a Django Project

When you create a Django project with **startproject** command, Django automatically creates a folder bearing the same name as the project. This folder has some folders and files preloaded in it (see figure below). The project folder has another folder in it, also having the same name as that of the project and a file namely **manage.py**.

NOTE

A project is the complete application, while an app focuses on one logical part of the application. There can be multiple apps in a project.

The <projectname> subfolder has some files preloaded in it (*see* figure below).



So if, for example, our project is online-shop, then its structure will be as explained in Fig. 15.3(*c*).

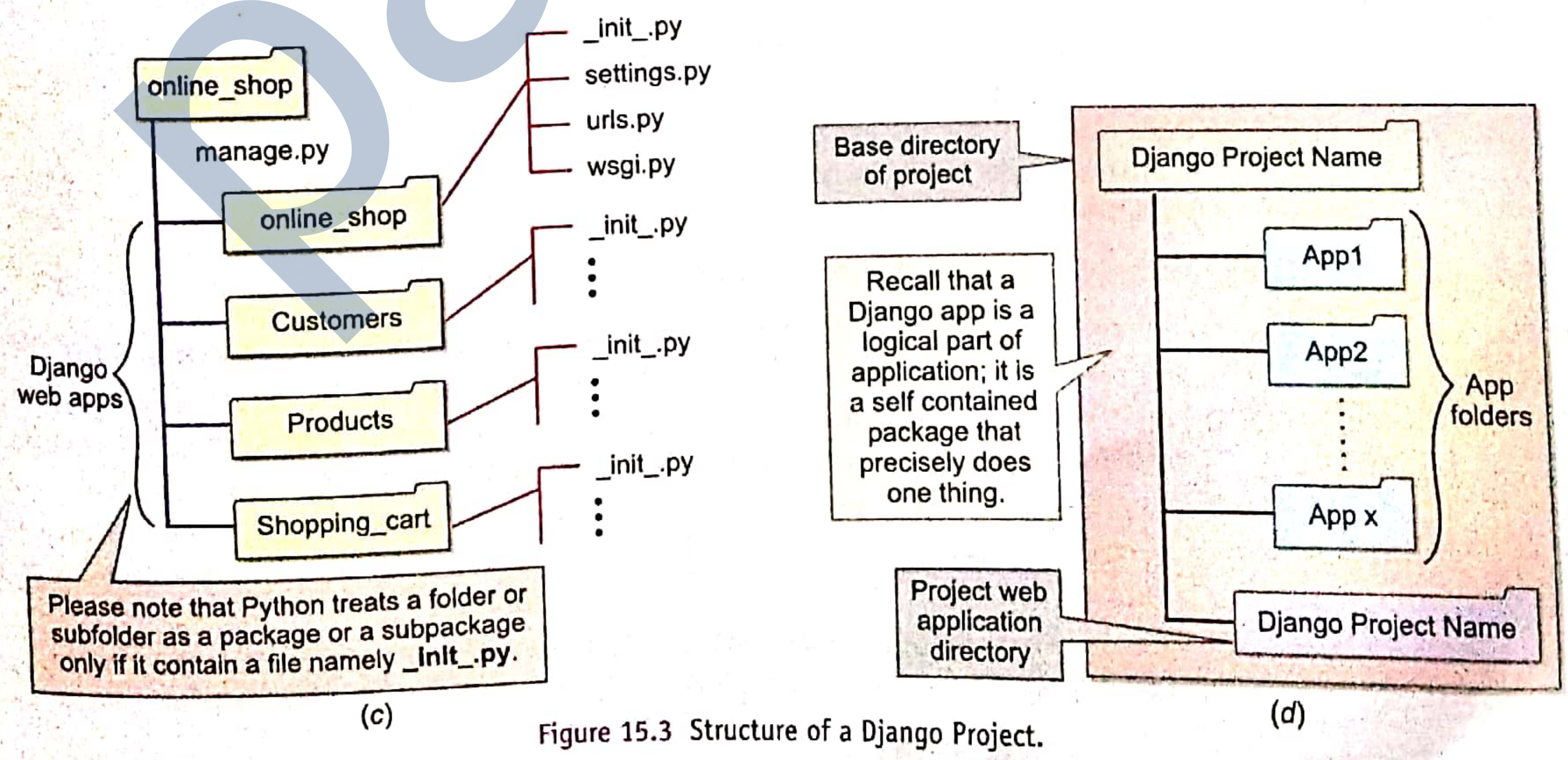
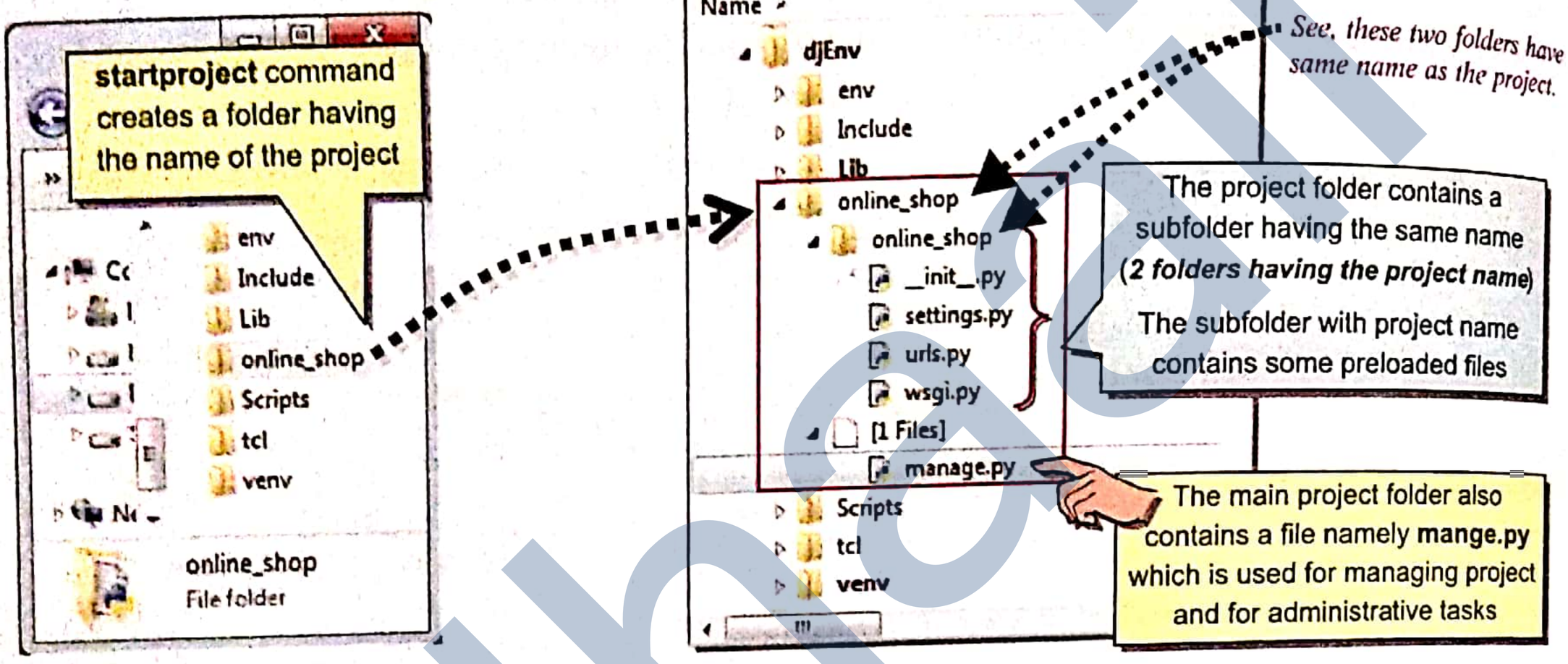


Figure 15.3 Structure of a Django Project.

As you see that as soon as you create or start a project, Django automatically creates some folders and files in it. Let us talk about these.

15.7.1C Files Created in a Django Project

Let us now talk about files created by Django by default. These files are automatically created by Django as soon as you start a project. (Do not worry if things are not very clear at this moment. These will be when you start working on your Django projects practically.)



⇨ Folders Having the Project Name

Django creates two folders having the project name, and one is a subfolder of the other. The outer folder signifies that it is a Django project.

The inner folder (subfolder) with the project name is actually a Django app that signifies that <projectname> is a web application. Recall that a Django project contains apps in it. [refer to Fig. 15.3 (*c*), (*d*)]

Do not be confused between these two folders. The outer folder signifies that it is a Django project and the inner one stores the actual settings and other functionality of web application.

The outer project folder is the BASE DIRECTORY for your Django application.

⇨ File in Outer Project name folder (the base directory)

`manage.py` : This is a file in the (outer) project folder. It is used to manage the project and perform some administrative tasks such as running built-in server and many others.

> **NOTE**
> File **manage.py** is created inside the BASE DIRECTORY, the outer project name folder.

⇨ Files in Inner Project name folder (the subfolder)

`__init__.py` This is a file inside every package/sub-package of Python. Without this file the folder is not treated as an importable package.

`settings.py` This is a file that will store most configuration settings for your Django project *e.g.*, what databases are in use, where Django should look for files, and so forth.

`urls.py` This file stores the information about locations where URLs of your Django project have been declared. A Django project identifies a URL related to its web application if this file ***urls.py*** somehow points to it.

`wsgi.py` This file is used for application production of your project. You need not go into details of this file at learning stage.

For now only this much information is sufficient. We shall talk about these files and folders when the need arises.

IMPORTANT

Roles of Two project name folders

When you create a Django project with command **django-admin startproject** command, Django creates two project folders having the project name and one is the subfolder of the other.

- The outer project folder is the **BASE DIRECTORY/FOLDER** of your project and holds all the apps and other setting modules of the project. It also holds the administrative tasks management file **manage.py** in it. This folder also holds the **templates** folder (which you create) that stores the presentation html files for your project.
- The inner project folder is a subfolder of base directory folder and is the ***project web-application folder***. It holds all the setting files of your Django project.

15.7.1D Creating Apps of a Django Project

As there can be multiple apps in a Django project, you need to create each app individually inside the project by issuing the following command while inside the project folder where **manage.py** resides :

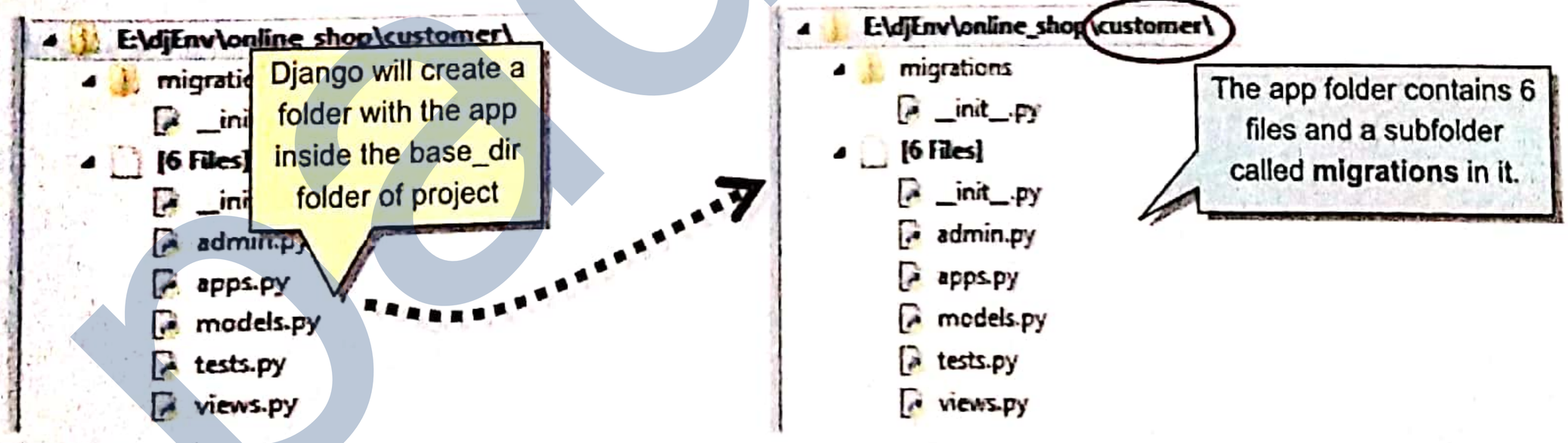
```
python manage.py startapp <appname>
```

e.g., if we want to create app *customers* inside our project **online_shop**, we shall first go inside the outer project folder *online_shop* (because **manage.py** resides in it) and then issue command:

```
cd online_shop
python manage.py startapp customers
```

← *Name of the app*

```
(venv) E:\djEnv\online_shop>python manage.py startapp customer
(venv) E:\djEnv\online_shop>
```



If you have multiple apps inside a project, you need to create each app individually by giving **startapp** task. That is, for our example shown in above figure, we need to issue two more commands to create two more apps **products** and **shopping_cart**. The only thing you need to ensure is that while issuing these commands you must be the base_dir folder (outer project folder) of your project where **manage.py** resides.

```
python manage.py startapp products
python manage.py startapp shopping_cart
```

← *These two commands will create two more webapps namely products and shopping cart*

Like project folder, the app folders also contain some files in it. We shall use these files for various settings when we make our example web application later. But one file you can easily

identify is **__init__.py**, which means it is treated like an importable package by Python. Other files in app folder are being briefly explained below :

`admin.py`	It is a configuration file for the built-in Django Admin app
`apps.py`	It is a configuration file for the app itself
`models.py`	It is the file where you will define the database models. Django will automatically translate models defined here into database tables.
`tests.py`	This file is for our app-specific tests
`views.py`	This file is for resolving http request/response for our web app
`migrations/`	This folder keeps track of changes in **models.py** file and updates database accordingly

All the above discussed things will become clearer to you when you develop a Django application practically in coming lines. But before that you need to know Django Project Architecture.

15.7.2 Understanding Django Project Architecture

Django projects are based on a software architecture that separates :

- Data being managed (the *model*)
- Presenting/showing to user (the *template*)
- Handling Logic (the *view*)

This software architecture was called **MVC (Model View Controller)** architecture and now it is called **MVT** or **MTV (Model Template View)** architecture.

The components of MTV architecture are as follows :

Model — This component is responsible for data management. This deals with access, validation and relationships among data. Models defined in Django are automatically mapped into database tables. Each individual data item of a model is mapped to a ***field*** in the database table.

Template — This component is responsible for the presentation of data/webpage to the user/ client.

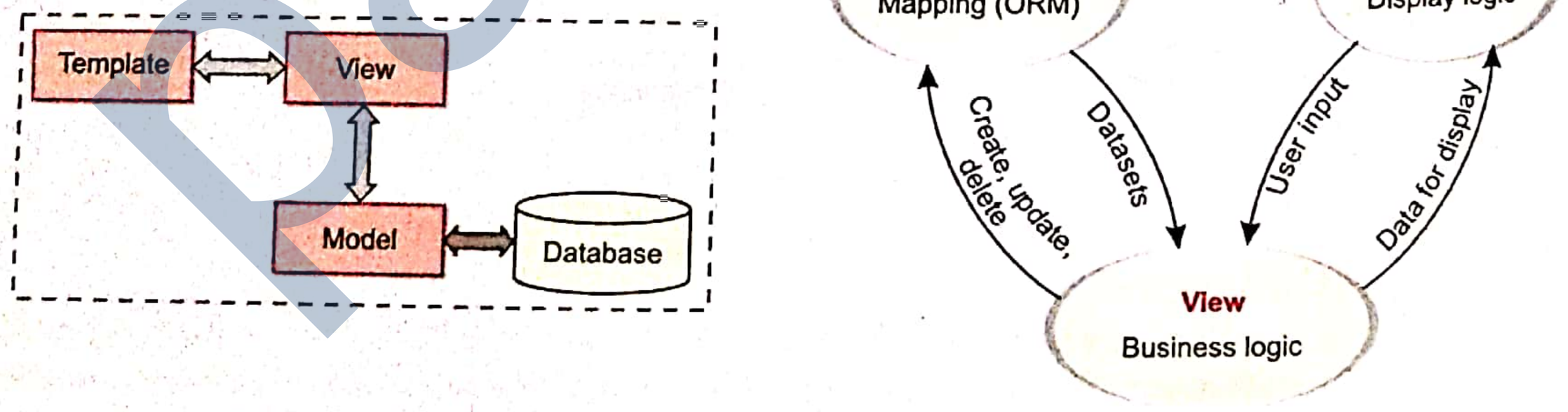


Figure 15.4 MTV architecture of Django.

View — This component defines the Business logic for your application. Django views act like a bridge between models and templates. A view accesses model data and redirects it to a template for presentation.

Controller — It is the logic tier provided by the Django framework. It links the components of Django architecture *i.e., model, view* and *template*. It passes data for display from a *model* to a *template* via a *view*.

15.8 STEPS TO CREATE A BASIC DJANGO WEB APPLICATION

Armed with basic knowledge of Django's basic architecture and terms, let us talk about the steps you need to take to create a basic Django web application. Recall that a user can interact with a web application by making any of the following *two* requests :

- **HTTP GET request.** User sends a URL via client (web browser) and web server makes available the web page's HTML at the given URL, after performing the required tasks.
- **HTTP POST request.** User sends some data (*e.g.*, through a form), which is saved in a database.

A web application can support both the above types of requests.

To create a Django web application, you need to follow these steps :

Step 1 Determine and create Project alongwith its Apps

1.1 Determine project and apps for your Django web application.

Recall that the project signifies an overall web application while an app is a component of a project that carries out a distinct function.

A project can have multiple apps in it.

1.2 Create project and apps by issuing commands for it as discussed in sections 15.7.1A and 15.7.1D.

1.3 Register apps with the project after creating them.

To register your created apps with the project, you need to add the app names in the INSTALLED_APPS setting of your project's setting file. For this open inner project-name folder's **settings.py** file in Notepad or any other editor and add your apps' names (in string form) in the **INSTALLED_APPS** list without deleting any other setting (*see* adjacent figure).

settings.py - E:\djEnv\online_shop\online_shop\settings.py (3.7.2)

File Edit Format Run Options Window Help

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

DO NOT FORGET to Register your apps by putting their names in INSTALLED-APPS

In **settings.py**, add your apps' names in the INSTALLED_APPS list

For each app of the project, you need to prepare MVT *i.e.*, *Model*, *View* and *Template* as Django architecture is MVT based, which is what we going to talk about in the next step.

Step 2 For each app of the project, create MTV (model, template, view) by determining the following :

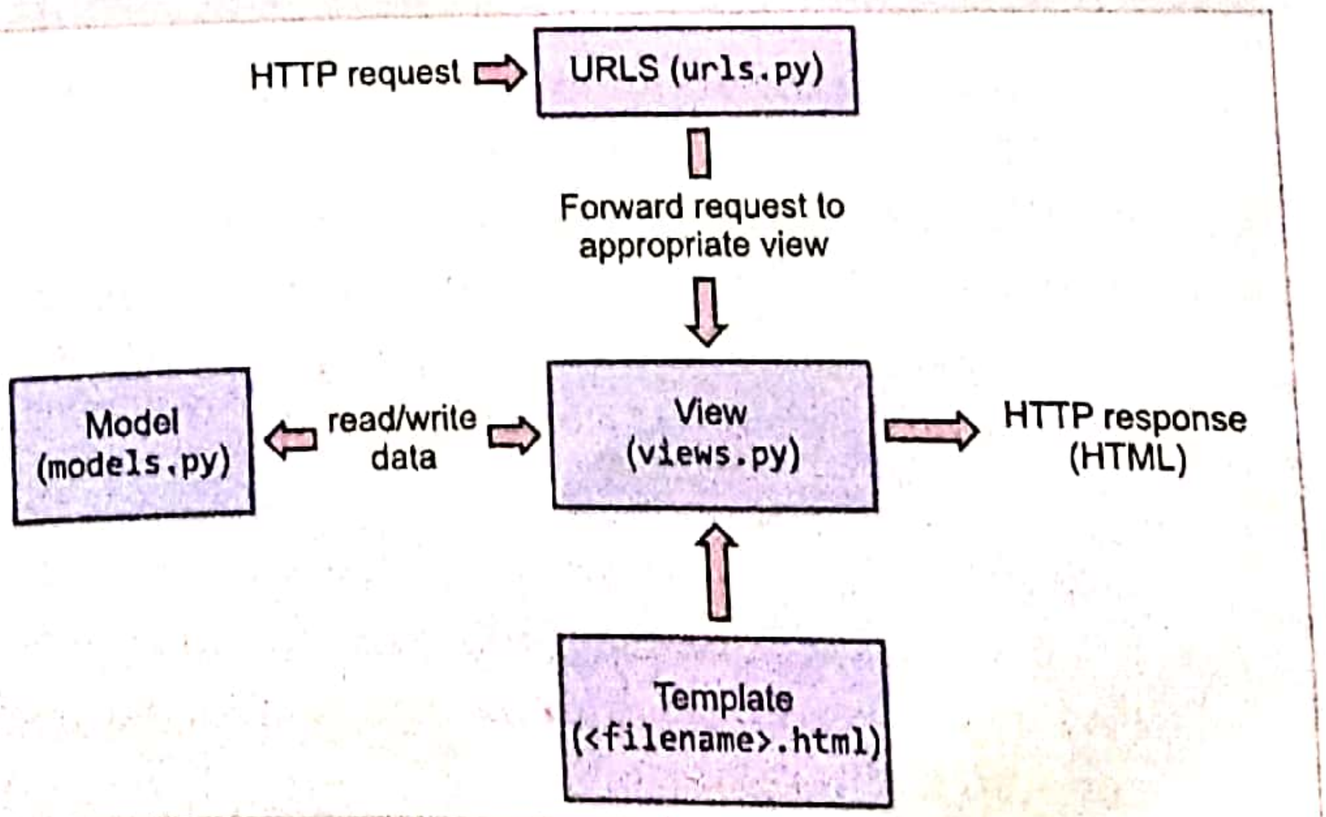
2.1 Type of HTTP request (GET/POST)

2.2 Data involved (MODEL)

2.3 HTML code generation (TEMPLATE)

2.4 URL and its mapping (VIEW)

Using these components, Django web application interacts and works as depicted here,



Step 3 For the components identified in step 2, you need to create the following :

3.1 Model — Models are the short names for data objects. Models define the structures of application's data. Data defined through models are stored and managed in databases. (*covered in section 15.9.1*)

3.2 Template — A template is responsible for creating layout of HTTP response file. A template can be used to define the structure of any type of file; can be in HTML and other formats (*covered in section 15.9.2*).

3.3 View — A View refers to a function which receives HTTP requests and returns the HTTP responses. Views access the data needed to satisfy requests via models and formatting the responses to templates. (*covered in section 15.9.3*)

3.5 URLs for HTTP requests

Every HTTP request is responded in the form of HTTP response which is a template based on some html. Every HTTP request requires a **URL** and a **URL mapper**.

A URL mapper links the URL to an appropriate view, which links to a template. Django does it through **URL Confs** (*URLS Configurations*). (*covered in section 15.9.4*)

Django apps follow the **Model, View, Template paradigm**. In nutshell, the app gets data from a **model**, the **view** does something to the data and then renders a **template** containing the processed information.

The following section 15.9 talks about creating the above components MTV for apps and URL Confs. All the above mentioned terms will make sense to you when we create an **Example Django web application** by following above steps and that is what we are going to do in a later section.

Step 4 Run built-in web server in your project's base directory and test the URLs working.

15.9 CREATING MODELS, VIEWS AND TEMPLATES

Now we know that a Django project contains multiple apps and for each app, we need to create ***models*** for data, ***templates*** for presentation and *view* for resolving *urls* and connecting them with templates. In this section, we shall learn how you can create models, templates and views for the apps of your project and then you shall learn how to link them with URLs with URL Confs.

Important. We shall discuss in the coming lines a minimal web application to give an idea of Django web application development, thus covering just the required basics. Also, just make the changes as suggested BUT do not delete anything from the default settings/code available as this will hamper your web application's working.

15.9.1 Creating Models

Models in Django refer to the way data is stored and processed. For example, if you are creating an **online_shop** application as mentioned earlier and its one functionality involves the **products** app. Now the ***products app*** will be dealing with all data and functionality related to products. Or in other words, the **products** app will be dealing with data like :

```
Product_no      (an integer type)
Product_price   (a real number)
Product_name (a character or string type)
Qty  (an integer type)
```

Django models also map to tables in database although they are defined in a different way. Django (data) models are created through **Object-Relational Mapping (ORM)** which maps to the underlying database. ORM is Python like programming technique used for defining data that makes working with data and relational databases much easier (*see* figure 15.5 below) as you will get to know soon.

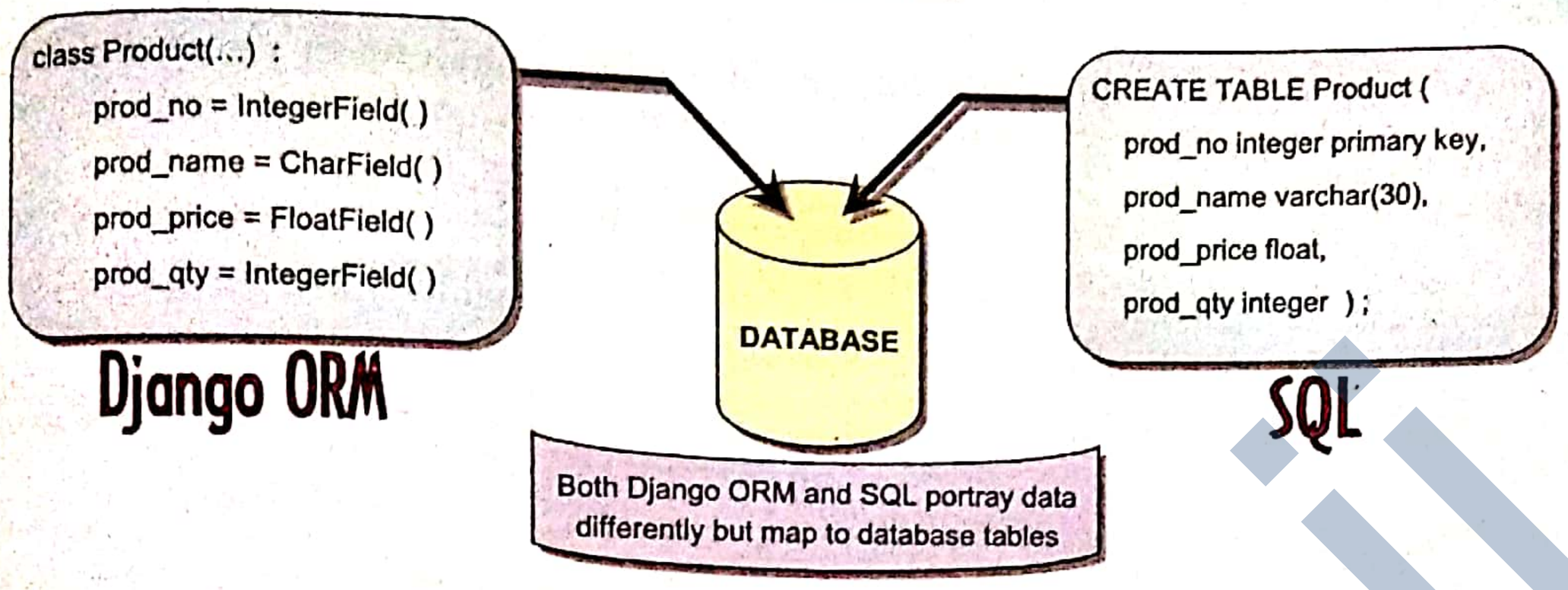


Figure 15.5 Django ORM and SQL portray data differently.

Let us talk about how you can define models using Django ORM (Object Relational Mapping). The data model for an app is defined inside **models.py** file of the app folder.

You can define data model for your app through following syntax :

```
class <modelname> ( models.Model ) :
    <fieldname> = models.<fieldtype>
    <fieldname> = models.<fieldtype>
        :
```

Example : Model for *product app* will be like :

```
class Product(models.Model) :
    prod_no = models.IntegerField( )
    prod_name = models.CharField(max_length=50)
    prod_price = models.FloatField( )
    prod_qty = models.IntegerField( )
```

```
from django.db import models

class Product(models.Model) :
    prod_no = models.IntegerField( )
    prod_name = models.CharField(max_length=50 )
    prod_price = models.FloatField( )
    prod_qty = models.IntegerField( )
```

You need to define models if you want to store apps' data in a database such as *sqlite3* or *MySQL* or *PostgreSQL* etc. For more on models, refer to Appendix D.

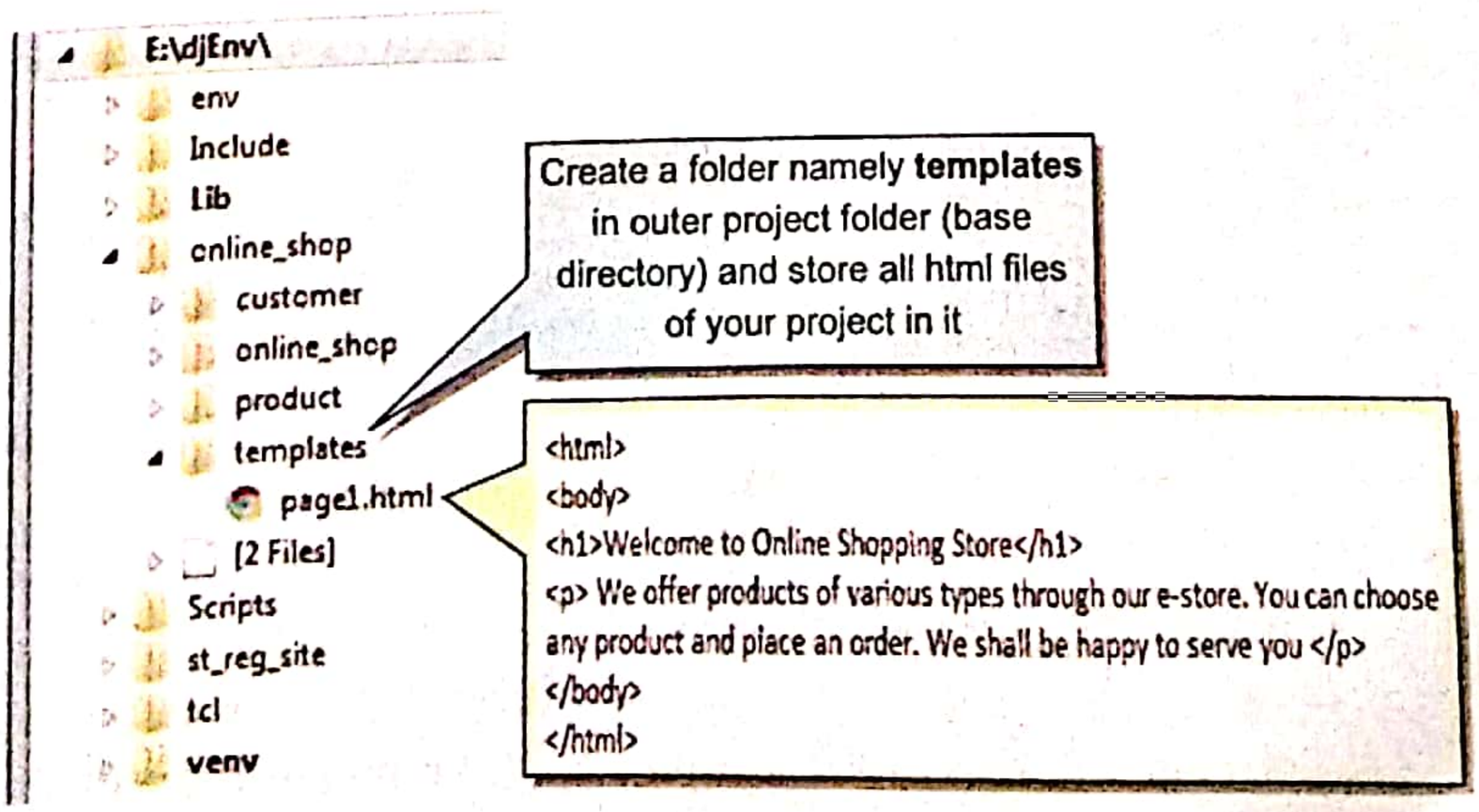
15.9.2 Creating Templates

After creating apps and their models, the next thing you need to do is to create templates for each app. Recall that Templates is the presentation mechanism of Django.

For this you need to do the following :

(*i*) In the BASE DIRECTORY (outer project name folder), create a folder by the name **templates.**

(*ii*) Next store the desired html files (which are to be displayed in response to a url) in this **templates** folder.



(*iii*) Next you need to open the **settings.py file of your project** (inner project name folder) and add the name of templates folder (in string form, *i.e.*, in quotation marks) in the DIRS setting under TEMPLATES (see below).

The DIRS key of TEMPLATES setting specifies the place where the server will look for templates while displaying webpages of the web application.

Inside inner online_shop folder
online_shop\online_shop\settings.py file

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

Initially there is no entry in DIRS key of TEMPLATES list

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [ 'templates' ],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

Type name of 'templates' folder under DIRS key of TEMPLATES list

15.9.3 Creating Views

Next thing you have to do is to create views. Views, as you know, receive the request in form of URL and provide response in form of templates which are then presented on the web browser.

To create views for an app, do the following :

Important. While making suggested settings and code changes, do not delete any other code. Only change/do as it is specified here. As Django provides most of the functionality on its own, deleting anything will result in your web application not running/working properly.

(*i*) The **views.py** file of your app (in app directory), you need to **write view functions** – *one view function* for each html page you want to render. The *view function* is defined in the following format :

The view function always takes a ***request*** *argument*

```
def <viewname> (request) :
    return render(request, <htmlfilenamestring>)
```

e.g., we want to display our file *page1.html* which is inside templates folder of base project directory through a view namely **first**, then we shall define a view function as follows inside the **views.py** file of **product** app :

Complete name of this view is ***views.first*** *as this is defined inside* ***views.py*** *file*

product/views.py file

```
def first(request):
    return render(request, 'page1.html')
```

```
from django.shortcuts import render

def first(request):
    return render(request, 'page1.html')
```

Django provides the required packages for minimal application by providing **import** statements on its own.

The view functions defined in the **views.py** have the qualified name as **views.<viewfunctionname>**, *e.g.*, for above function, the full name will be **views.first**.

You can create as many views as you need, in the views.py file.

15.9.4 Creating URL Confs

Once your model, template and views (MTV) are ready, you need to link all these with URLs through URL Confs. The process of linking URLs and displaying actual templates is also called **URL routing**.

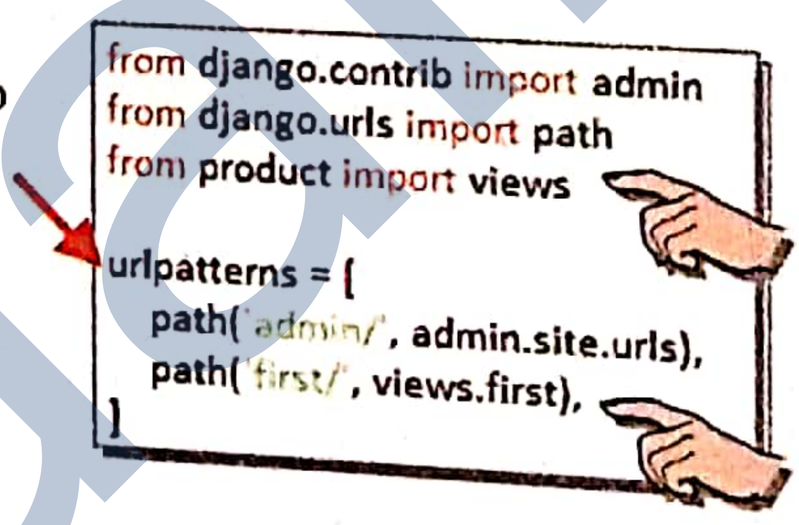
For defining URL configurations (URL Confs) for your web application project, you need to go to **urls.py** file of your inner project folder (the web application folder) and do the following :

(*i*) Import **views.py** file of your apps where you created the view functions for your apps, *e.g.*, we created a view by the name **views.first** in product app's **views.py**. So we shall write following command to import views from *product* app in addition to already existing code.

```
from product import views
```

(*ii*) To define **URl Conf** for your defined views, you need to add following line to **urlpatterns** list in the **urls.py** file.

```
path('first/', views.first),
```



To understand above path( ), read the following explanation.

Working of path( ) under urlpatterns of urls.py

Let us understand how **path( )** added to **urlpatterns** works.

```
path('first/', views.first),
```

name of the view to be displayed for the given path

add this string to default path of your web application i.e., if by default your web application starts with url ***localhost:8000/*** *or* ***127.0.0.1:8000/****, then above path( ) specifies that path*

```
localhost:8000/first/
```

see it got added to default path

will render **views.first** view from the views file.

In the **views.py file**, the view with name **first** is defined as function :

views.first view-function

```
def first (request) :
    return render(request, 'page1.html')
```

so for path **localhost:8000/first/**, it will look for **views.first** view function and from there, it will take **render( )'s** html argument as the web page, which page1.html here. It will now look for this page in **templates** folder and display it there.

That is,

```
path('first/' views.first)
```

is resolved as follows (Fig. 15.6) :

1. add **first/** to default path ⇒ **localhost:8000/first/** it is the URL to which **views.first** is linked.
2. Look for **views.first** function in **views.py**.
3. From there, look for html file's name in **return render( )** statement's value which is **page1.html**.
4. Look for this html page in attached **templates** folder of the BASE DIRECTORY.
5. If found, render this html file in response to URL **localhost:8000/first/**, otherwise display error.

Note

In older versions of Django, the URLs were given with regular expressions (RegEx), but with latest version of Django, things are very much simplified for URLs in path( ).

URL CONFS.

The mapping between URL path to views of Django Project are called URL Confs (URL configurations). The views of Django project are python functions.

Let us now run to check if all our settings (MVT + URL Conf) is working. For this

(*i*) Start/activate virtual environment as explained in section 15.6.

(*ii*) Go in the BASE DIRECTORY folder of your project.

```
cd online_shop
```

(*iii*) Now run the Django built-in server for your web application :

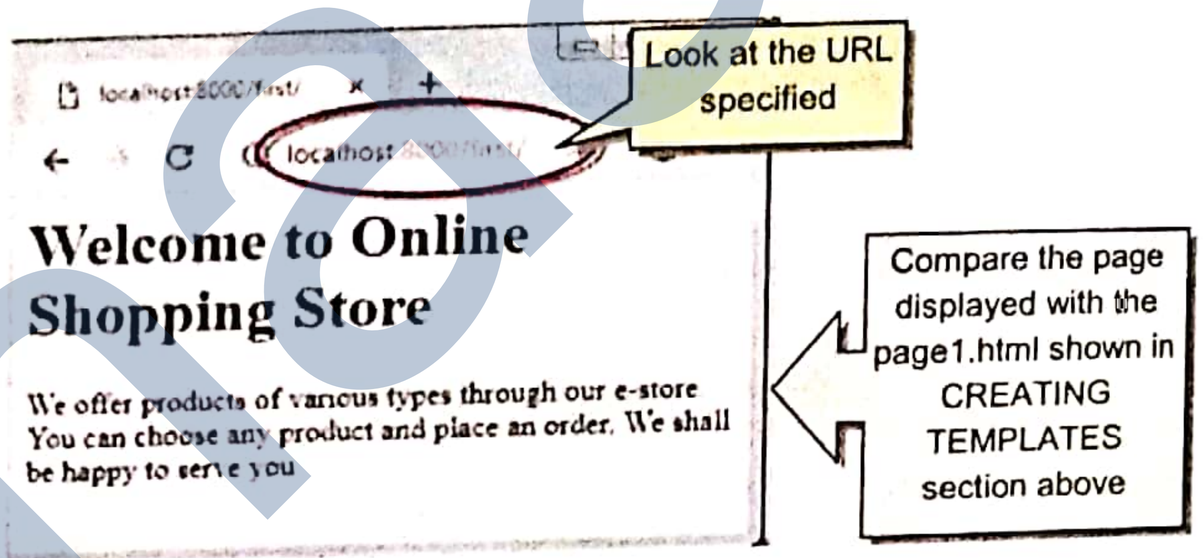
```
python manage.py runserver
```

(*iv*) Open web browser (*e.g.*, start Chrome or Safari on your computer). Open first page of your web application by giving URL as **localhost:8000** in web browser's address bar.

(*v*) Now type the following URL in the address bar of your web browser :

```
localhost:8000/first/
```

And see what happens! Hurray!!



Let us quickly do **URL Conf** for another page.

(*i*) Create a file namely **page2.html** under **templates** folder of base directory folder

```
<html>
<body>
<h2> This is second page</h2>
This is defined for our second view.
</body>
</html>
```

Page2.html under **templates** folder

(*ii*) In the **views.py** file of your product app, add following *view function* namely *two* (**views.two**)

```
from django.shortcuts import render

def first(request):
    return render(request, 'page1.html')

def two(request) :
    return render(request, 'page2.html')
```

Views.py file of product app

The **render( )** method is the Django shortcut method to more easily render an html response.

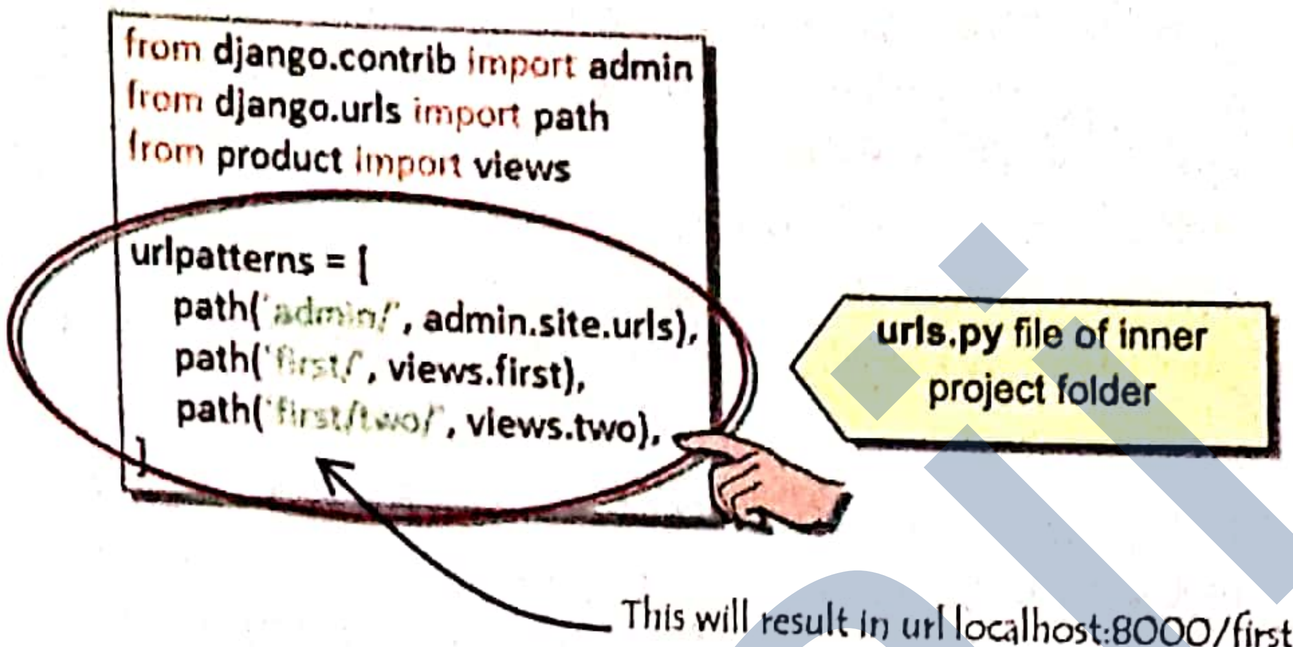
Check Point 15.1

1. What are the advantages of using Django for web development ?
   (*a*) It facilitates you to divide code modules into logical groups to make it flexible to change
   (*b*) It provides auto-generated web admin to make website administration easy
   (*c*) It provides pre-packaged API for common user tasks
   (*d*) All of the above
2. The architecture of Django consists of ?
   (*a*) Models (*b*) Views
   (*c*) Templates (*d*) All of these
3. Which option do you use with django-admin to create project in Django?
   (*a*) Newproject (*b*) myproject
   (*c*) createproject (*d*) startproject
4. Write Django command to create a Django project namely **easysell.**
5. What is the Django command used with **Python** command to start a new app named 'list' in an existing project ?
   (*a*) manage.py --newapp list
   (*b*) manage.py newapp list
   (*c*) manage.py -startapp list
   (*d*) manage.py startapp list

(*iii*) In the project's web-application folder (inner project name folder), add following path to the **urlpatterns** setting under **urls.py**

Here if you need to import views.py from different app folders of your project, you should specify alias name for views being imported as both apps have their views stored in views and it may lead to ambiguity, *e.g.*, if you write :

```
from product import views
from register import views
```



To avoid ambiguity, you can import the *views* modules of apps as :

```
from product import views as prviews
from register import views as regviews
```

Then you can qualify *product* app's view-functions as :

```
prviews.<viewfunction-name>
```

and *register* app's view-functions as :

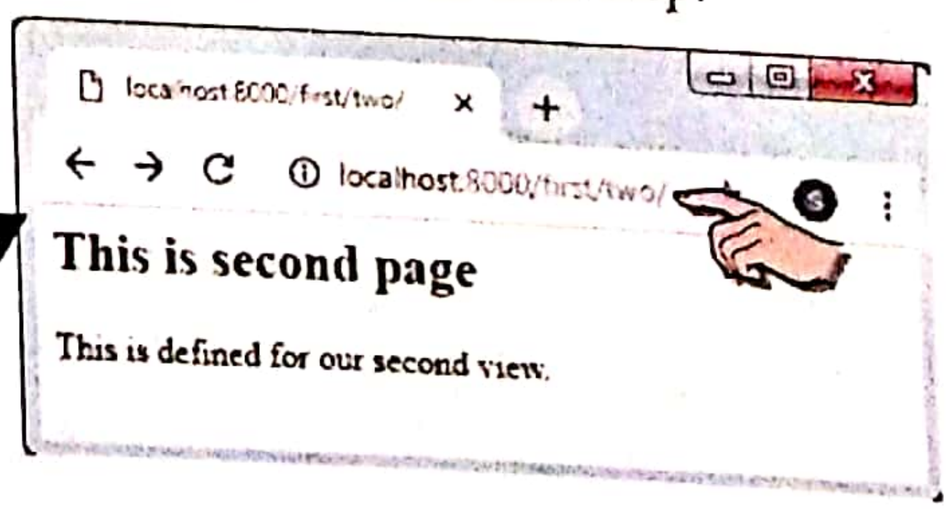
```
regviews.<viewfunction-name>
```

> **NOTE**
>
> URL Routing is a process of resolving and mapping URLs through URL Confs in **urls.py** to actual templates which can be displayed in the web browser as a web page.

(*iv*) Now run server by giving the following command (if not already running) in the base directory of your project. If server is already running move to next step.

```
python manage.py runserver
```

(*v*) Issue url **localhost:800/first/two/** in the web browser's address bar :



Following figure illustrates URL routing in Django.

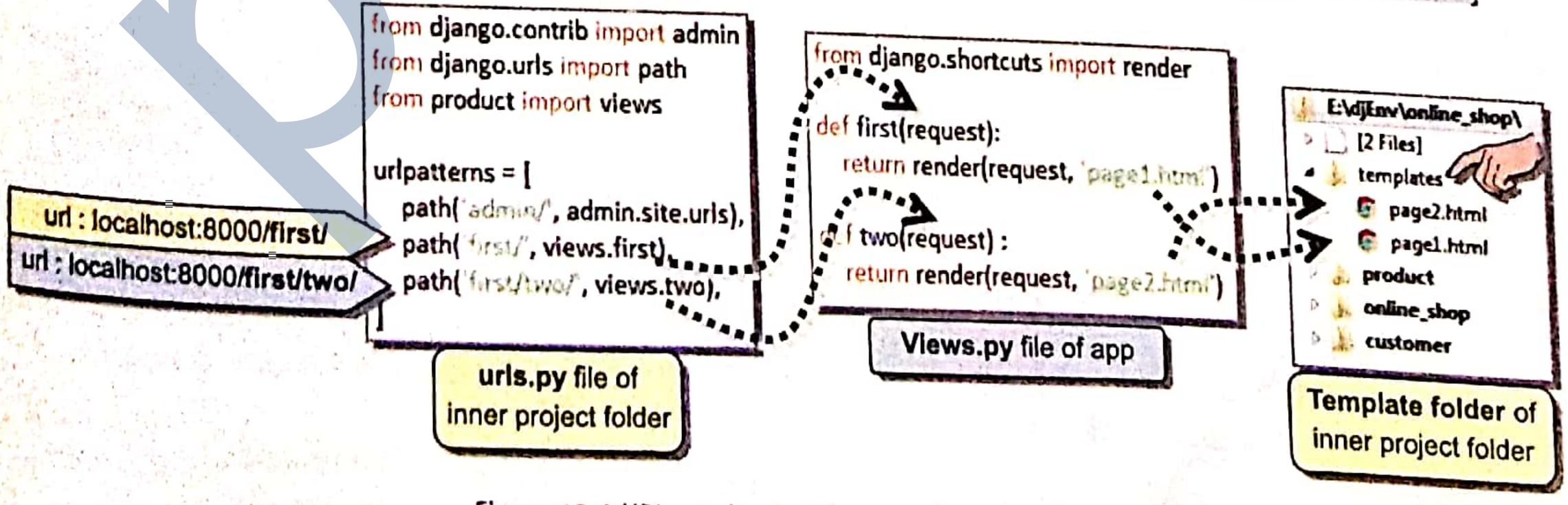


Figure 15.6 URL routing in Django web application.

Using the above discussed steps you can resolve an HTTP GET requests *i.e.*, display a webpage by specifying a URL. Our example application will be doing the same once again in section 15.11.

15.10 WRITING DICTIONARY DATA TO CSV AND TEXT FILES

You must be wondering why this section while we are doing Django? Well, the answer is that we need to store a form's data to a CSV or text file via a Django web application as per the syllabus. Thus, in this section, we shall briefly talk about how you can write data to CSV files using Python code and then we shall apply this for HTTP POST request processing in Django's **views.py** file. We shall cover only that much what is required without discussing the details and other concepts. To write to a CSV file, you need to import python's csv module :

```
import csv
```

For writing data, you need to open a file with .CSV extension in the write ("w") or append("a") mode (refer to Table 5.1 in chapter 5 for different file modes). Also, we shall only talk about how to write a dictionary's data onto a csv file because in views of Django, an html form's data is available as dictionary form through **request** object.

Say, you have data to be written in the form of a dictionary called **dict1**, then you need to write the following code, which is explained along with it.

Opens file 'data.csv' in append mode

```
with open('data.csv', 'a') as csvfile :
    wrt = csv.writer(csvfile)
    for key, value in dict1.items():
        wrt.writerow([key, value])
```

*csvfile is the name of file-object which will be used inside the **with open file** construct (Notice all file handling statements within **with open** file constrict are indented inside **with open** file construct.)*

*The **wrt** links the fileobject with csv module's **writer( )***

*Using wrt now we can write to csv files using **writerrow( )** method*

*This **for** loop writes [key, value] from dictionary **dict1** to csv file linked to **wrt***

To write a dictionary's data in a text file, simply open a **binary file** in *write/append* mode and write into this file as you have learnt in files.

We shall be using the similar functionality to write the form data to process POST request in Django web-application.

Some Commonly used Commands for Django Projects

1. virtualenv venv — start/activate virtual environment
2. django-admin startproject <project> — create a Django project
3. python manage.py startapp <appname> — create an app within a Django project
4. python manage.py runserver — Starts a lightweight development Web server on the local machine. By default, the server runs on port 8000 on the IP address 127.0.0.1. You can pass in an IP address and port number explicitly.

15.11 PRACTICALLY PROCESSING GET AND POST REQUEST

Now you know all the required knowledge to create a minimal Django web application that parses a GET and POST request and responds accordingly. For this we shall be creating the following minimal Django application.

Django example web application : **Register Stream Application**

Your school wants to create a simple website where students who have passed X from your school, can register and specify their choice of streams for classes XI and XII. The opening page displays information about your school (GET request). The student details are stored in a CSV file (POST request)

Let us create our example Django web application by following the steps given in section 15.8.

Step 1 *Create Project and Apps, and register them*

Example web application **Step 1.1 :**

(*i*) Name the project and its apps.

Project : Stream Registration (project name : st_reg_site)

Apps : 1. **School Details** (application name : sch_details)

Displays information about the school and the streams that are available for classes XI and XII.

2. **Register Student** (application name : register)

Students can enter their details such as ***name, enrolment number*** and the ***stream choice***. The data entered by the students is saved in a CSV *file* namely ***stream.csv***.

Example web application **Step 1.2 :**

(*ii*) Go to desired folder and start/activate virtual environment as explained in section 15.6.

```
E:\djEnv>virtualenv venv
Using base prefix 'c:\\program files\\python37-32'
New python executable in E:\djEnv\venv\Scripts\python.exe
Installing setuptools, pip, wheel...
done.

E:\djEnv>venv\scripts\activate

(venv) E:\djEnv>_
```

(*iii*) Create Django project by issuing command : **django-admin startproject <projectname>**
So we shall write command with project name **st_reg_site** (the name of our project) :

```
django-admin startproject st_reg_site
```

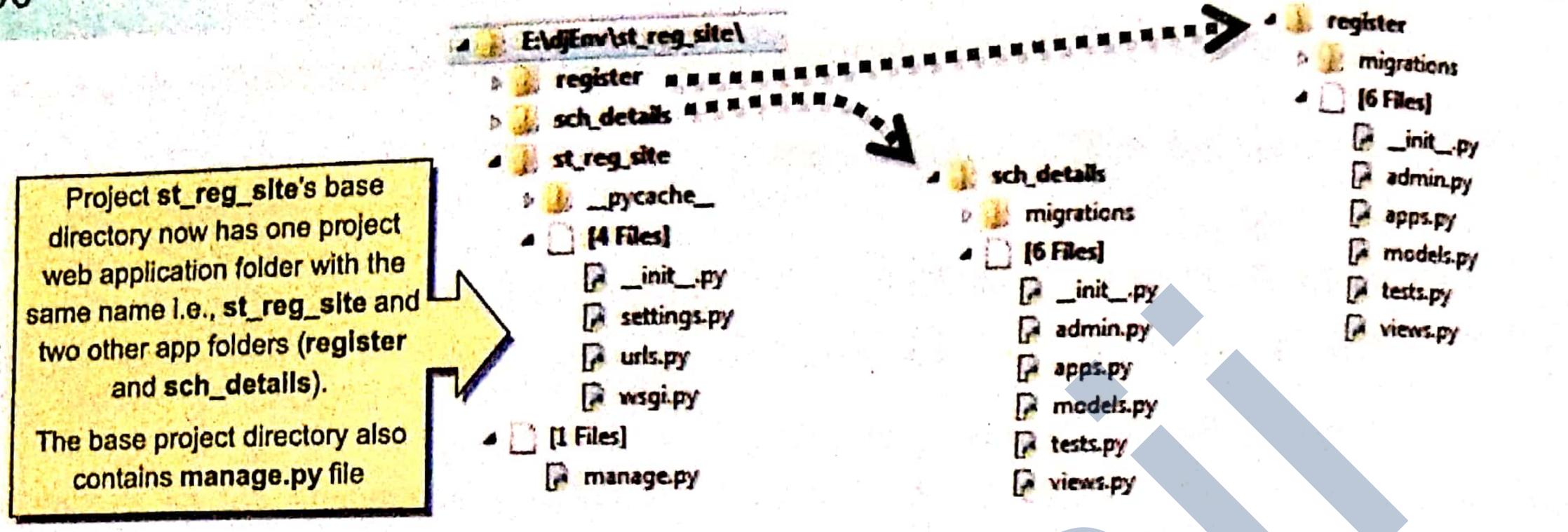
```
(venv) E:\djEnv>django-admin startproject st_reg_site
(venv) E:\djEnv>_
```

(*iv*) Create the *two* apps by giving following commands inside outer **st_reg_site** folder (the BASE DIRECTORY folder)

```
python manage.py startapp sch_details
python manage.py startapp register
```

```
(venv) E:\djEnv\st_reg_site>python manage.py startapp sch_details
(venv) E:\djEnv\st_reg_site>python manage.py startapp register
```

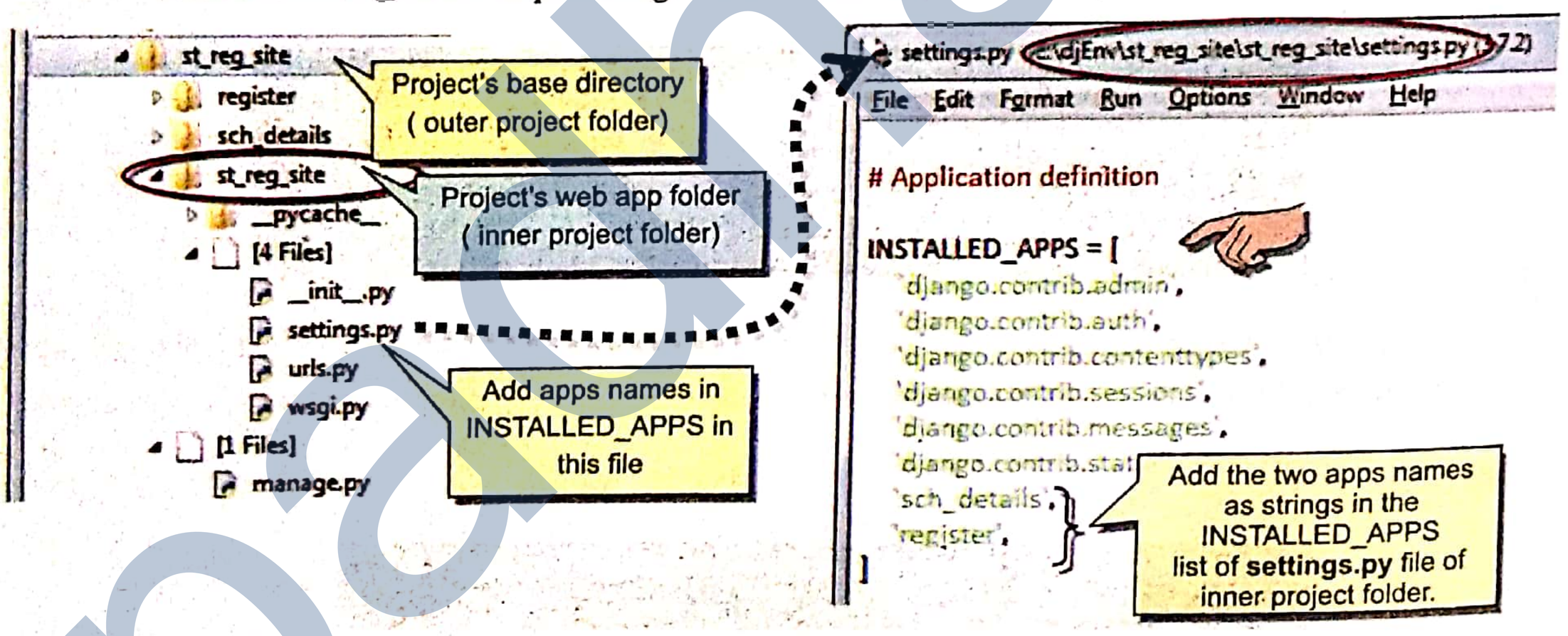
Now your outer project folder (base directory of the project) contains *three* folders and a file **manage.py**. These *three* folders are : web-application folder having the project name (**st_reg_site** – recall that inner project name folder is also a web application folder) and two app folders having names ***register*** and ***sch_details***



Example web application **Step 1.3 :**

(*i*) Register the apps with the project so that Django recognises them as part of the web application. For this purpose, you need to open **settings.py of project's web-application folder** (*inner project folder*) and add the names of apps to the INSTALLED_APPS list.

For this open **settings.py** in any editor (you can open it in *Spyder* or *Python IDLE* also the way you open other python files) and then add the two apps' names in INSTALLED_APPS as shown below. Just add the two names at the end of all entries of INSTALLED_APPS separating with commas (see below)[2]



Step 2 Determine MTV and HTTP request type

2.1 HTTP request type : Determining the request type for both the apps of the project :

App name	*HTTP request type*	*Remarks*
sch_details	GET	no data is being sent to web server
register	POST	students' data from HTML form will be sent to web server

2.2. Determine the Model, Template and Views required

Model. Since both the apps of our Django application won't interact with database, we need not create any model for these apps. The **register** app, however, will store an html form's data in a **csv** file (a standalone file) but we need not create a *model* for this as this does not require any database interaction.

2. You may also add app entry in INSTALLED APPS as **<appname>.apps.<App>Config**, *e.g.*, if app name is **list** then the entry will be **list.apps.ListConfig** (this is a Django way of qualifying the name of an app). For app namely register, the entry will be **register.apps.RegisterConfig**. This is done in detailed and advanced web applications.

Step 3 After determining, create MTV and **URL Conf** in this step

3.1 Template. To create template for our apps :

(*i*) First of all, we need to create **templates** folder inside the project's BASE DIRECTORY folder (outer project name folder).

(*ii*) Then we need to create two html files stored inside this templates folder :

- For *sch_details* app, we need to display school information about streams in an html file. For this let us create an html file namely **strdetails.html** as shown here:

strdetails - Notepad

strdetails.html

File Edit Format View Help

```
<HTML>
<BODY>

<H2> Welcome to YOUR School</H2>
Dear students,

For classes XI and XII, you need to register your choice of
streams from Medical, Non-Medical, Commerce, Humanities.
<P>
Please register your choice before April 15.

</BODY>
</HTML>
```

- For **register** app, we need to create an HTML form (**strchoice.html** inside the ***templates*** folder of outer project-name folder) that takes student details and sends this data for processing with **post** method. (*Please note that here we assume that you know how to create HTML forms. If not, you can refer to any HTML tutorial or book for the same.*)

IMPORTANT. For Django framework's middleware to access and process html form's data, you need to add following to your html file :

```
{% csrf_token %}
```

Also, set the <form> tag's attributes as follows :

- **action** attribute as "#", which means the html form will be processed within and not from any outside file/url.
- **method** = "post" as data of the form is to be sent to the web server.

So the strchoice.html file looks like the one shown here :

strchoice - Notepad

strchoice.html

File Edit Format View Help

Make sure to add this

```
<HTML>
<BODY>
<FORM action="#" method = "post">
{% csrf_token %}

Roll No.: <input type="text" name="rollno"  > <br>
Name :    <input type="text" name="name"  > <br>

<input type="radio" name="stream" value="Medical" checked> Medical
<input type="radio" name="stream" value="Non-Medical"> Non-medical
<input type="radio" name="stream" value="Commerce" checked> Commerce
<input type="radio" name="stream" value="Humanities"> Humanities<br>
<input type="submit" value="Submit">

</FORM>
</BODY>
</HTML>
```

(*iii*) Register the **templates** folder of BASE DIRECTORY (outer project-name folder) in the TEMPLATES setting of **settings.py** file of your project. Recall that the **settings.py** file of a Django project resides in the inner project-name folder. Open this file and add templates folder's name inside the **DIRS** list of **TEMPLATES** setting.

Do not change any other code/settings.

settings.py - E:\djEnv\st_reg_site\st_reg_site\settings.py (3.7.2)

File Edit Format Run Options Window Help

```
ROOT_URLCONF = 'st_reg_site.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': ['templates'],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

3.2 Views. Now we need to create views for our project.

(*i*) For **sch_details app**, open its **views.py** file and add following *view function* to it so that it is linked to **strdetails.html**. The view name defined is **details (views.details)**. Refer to previous section (15.9) to know how to create view functions.

views.py - E:\djEnv\st_reg_site\sch_details\views.py (3.7.2)

File Edit Format Run Options Window Help

```
from django.shortcuts import render

def details(request):
    return render(request, 'strdetails.html')
```

(*ii*) Since we are not using separate data models for apps, we can can add following *view function* for register app to the same views file of *sch_details app* to keep the view resolving simple. In this view function, link it to **strchoice.html**. The view name defined is **studata(views.studata)**. Here you will write functionality to write to csv file. Make sure to import csv module of Python (see below) :

views.py - E:\djEnv\st_reg_site\sch_details\views.py (3

File Edit Format Run Options Window Help

```
from django.shortcuts import render
import csv

def details(request):
    return render(request, 'strdetails.html')

def studata(request) :
    if request.method == 'POST' :
        dict1 = request.POST
        with open('studata.csv', 'a') as csvfile :
            wrt = csv.writer(csvfile)
            for key, value in dict1.items():
                wrt.writerow([key, value])

    return render(request, 'strchoice.html')
```

This is imported by default.

Since in html form, the action attribute is '#', it means post request will be processed within

Inside the view function, check if request type is POST. If it is then write code for writing the POST data in a csv file as explained earlier.

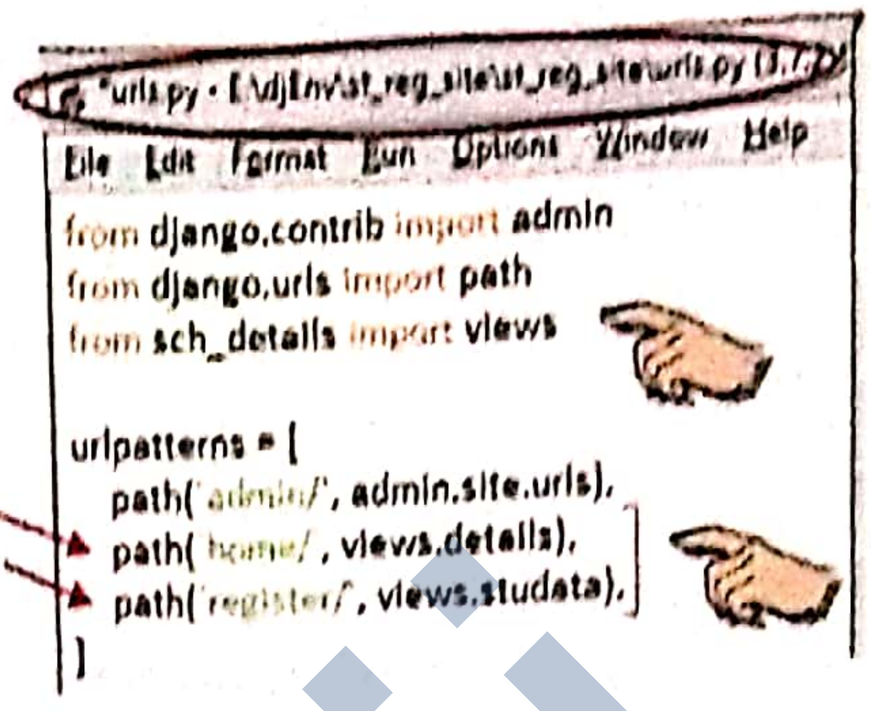
The POST data is available in dictionary form from POST attribute of request argument of view function (request.POST)

Else for initial URL request render the form page (strchoice.html)

3.3 URL CONFiguration. Since all our views are in sch_details app's views.py, we just need to import only this file inside the **urls.py** of your inner project-name folder and then add following URLS' configuration :

```
localhost:8000/home/  routes to views.details
localhost:8000/register/  routes to views.studata
```

(refer to section 15.9 for URL routing basics)



Step 4 *Run your project*

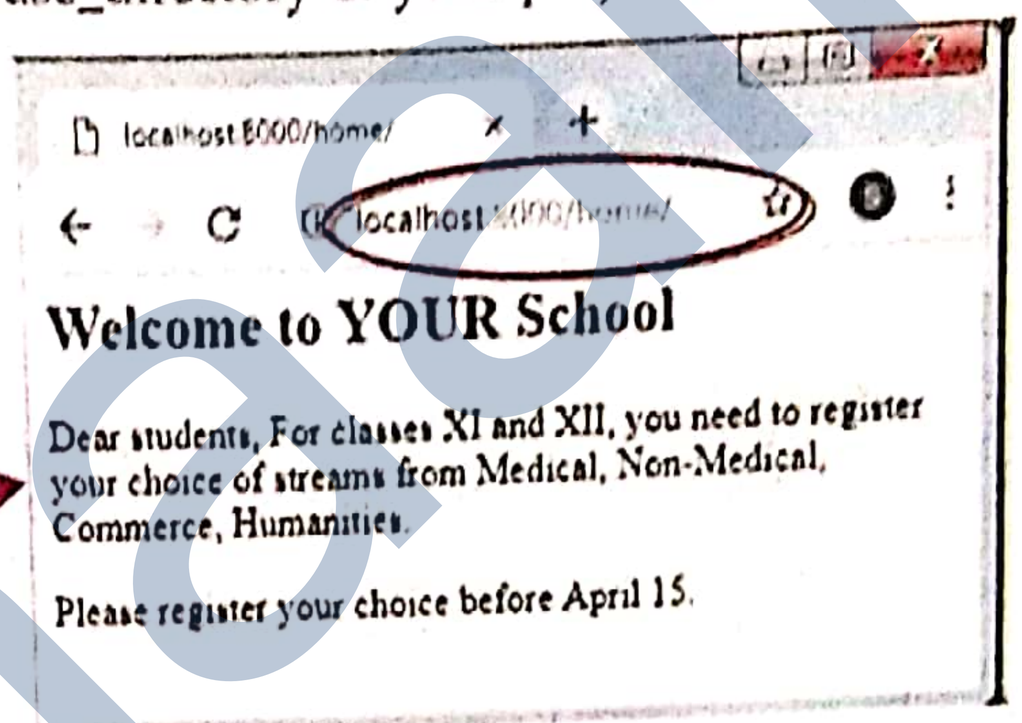
4.1. With active virtual environment, go to the base_directory of your project (outer project name folder) and run server as :

```
python manage.py runserver
```

HTTP GET REQUEST PROCESSING

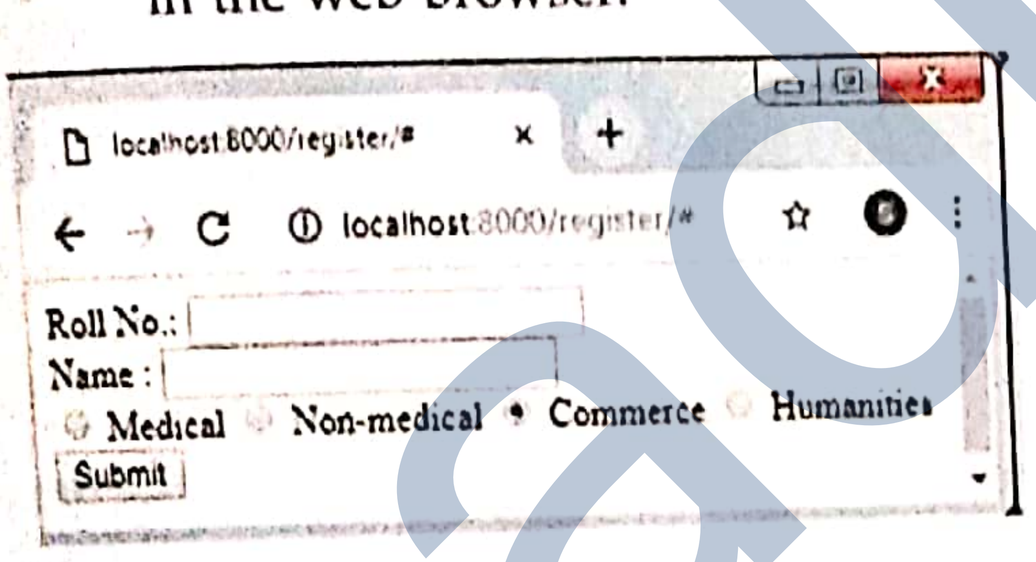
4.2 Now in web browser, give URL as **localhost : 800/home/**

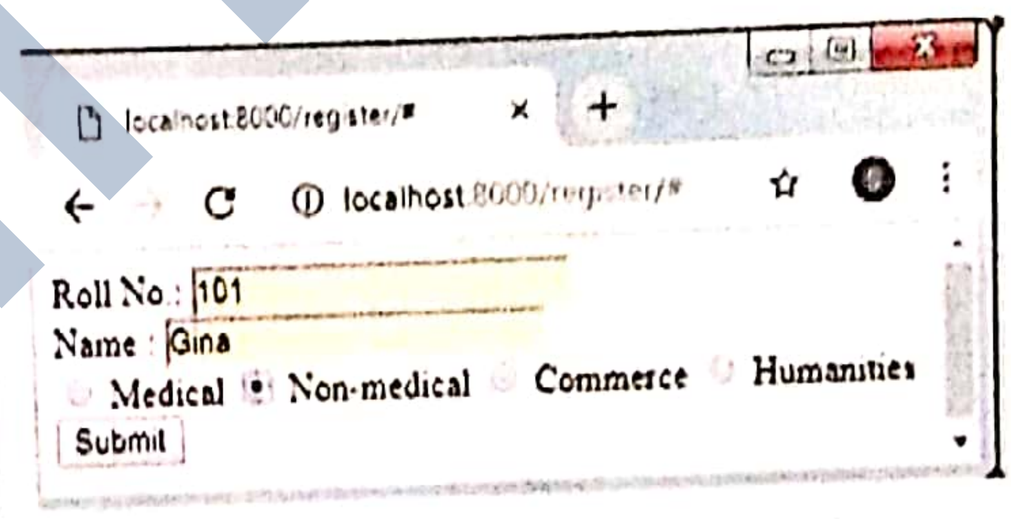
See what has happened →



HTTP POST REQUEST PROCESSING

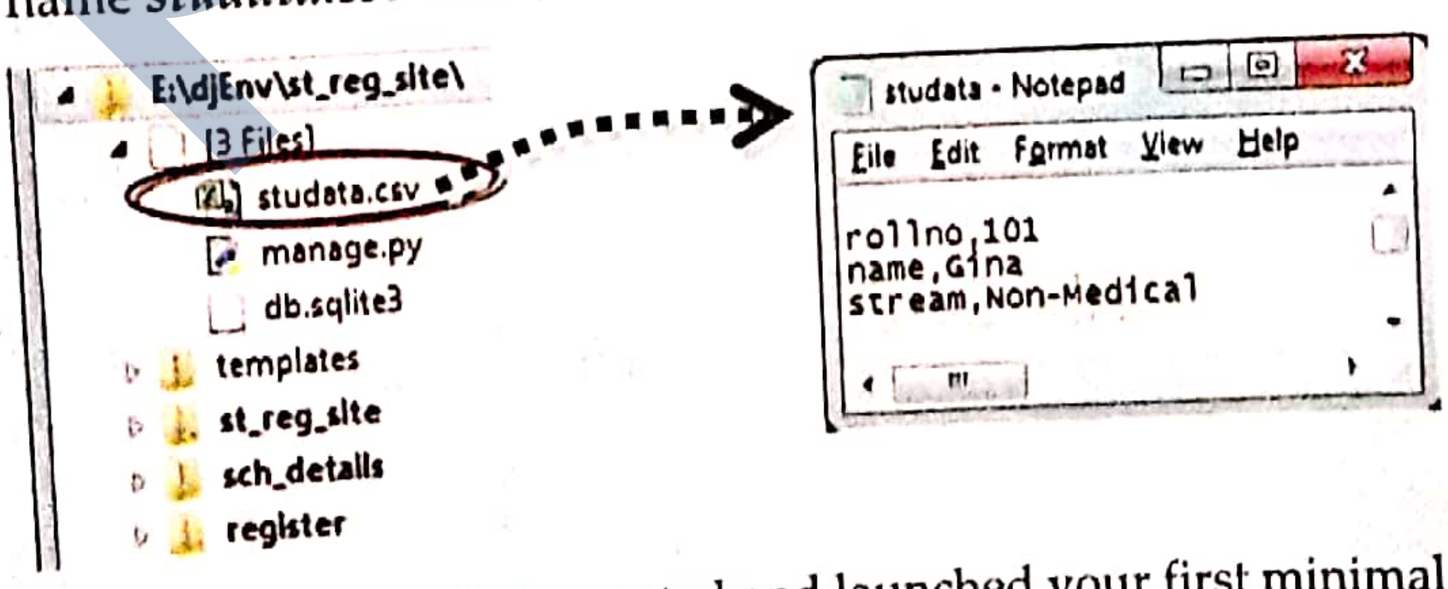
4.3 Now issue URL as **localhost:8000/register** in the web browser.





After entering data in the form when you click **Submit** button, the form's data will be sent for processing in form of a POST request. As per your post request processing view function, you wrote the POST data in **studata.CSV** file.

Now if you check the base directory folder of your project, you will find a file by the name *studata.csv* there, and if you open it with Notepad, you will find your data in it ☺



YIPPIE! You just successfully created and launched your first minimal Django web application that can parse GET and POST requests.

Figure 15.7 lists all the steps to create and run a Django application.

To create a text file from the retrieved data (in dictionary form), simply create a BINARY FILE in write mode and write the dictionary in it as you write in files.

(INSIDE A VIRTUAL ENVIRONMENT)

Step 1 (*a*) CREATE PROJECT : django-admin start project <projectname>

- In the folders created, the outerprojectname folder is the BASE DIRECTORY.
- The inner projectname folder is the settings directory for the project.

(*b*) CREATE APPS : python manage.py startapp <appname>

- You must be inside BASE DIRECTORY of your project while issuing above command.

(*c*) REGISTER CREATED APPS in file *settings.py*

- In the **settings.py** in the inner project folder, add app names in the **INSTALLED_APPS** list.

(*d*) CREATE a *Templates folder* inside BASE DIRECTORY.

- All html files for your project have to stored inside this templates folders.

(*e*) REGISTER Templates folder in file *settings.py*

- Add the name of Templates folder in the **DIRS** list inside **TEMPLATES** setting of **settings.py** file, which is in inner project folder.

Step 2 CREATE MTV for all APPs

Create model, template, view for each created app :

(*a*) Create **model** in **models.py** file of app's own folder.

(*b*) Create html files for each app and store in registered **template** folder.

(*c*) Create View function(s) for app – one view function for each URL in the **views.py** file in app folder.

A View function links to the html file through **render( )**.

Step 3 CREATE URL Conf in **urls.py** of BASE DIRECTORY

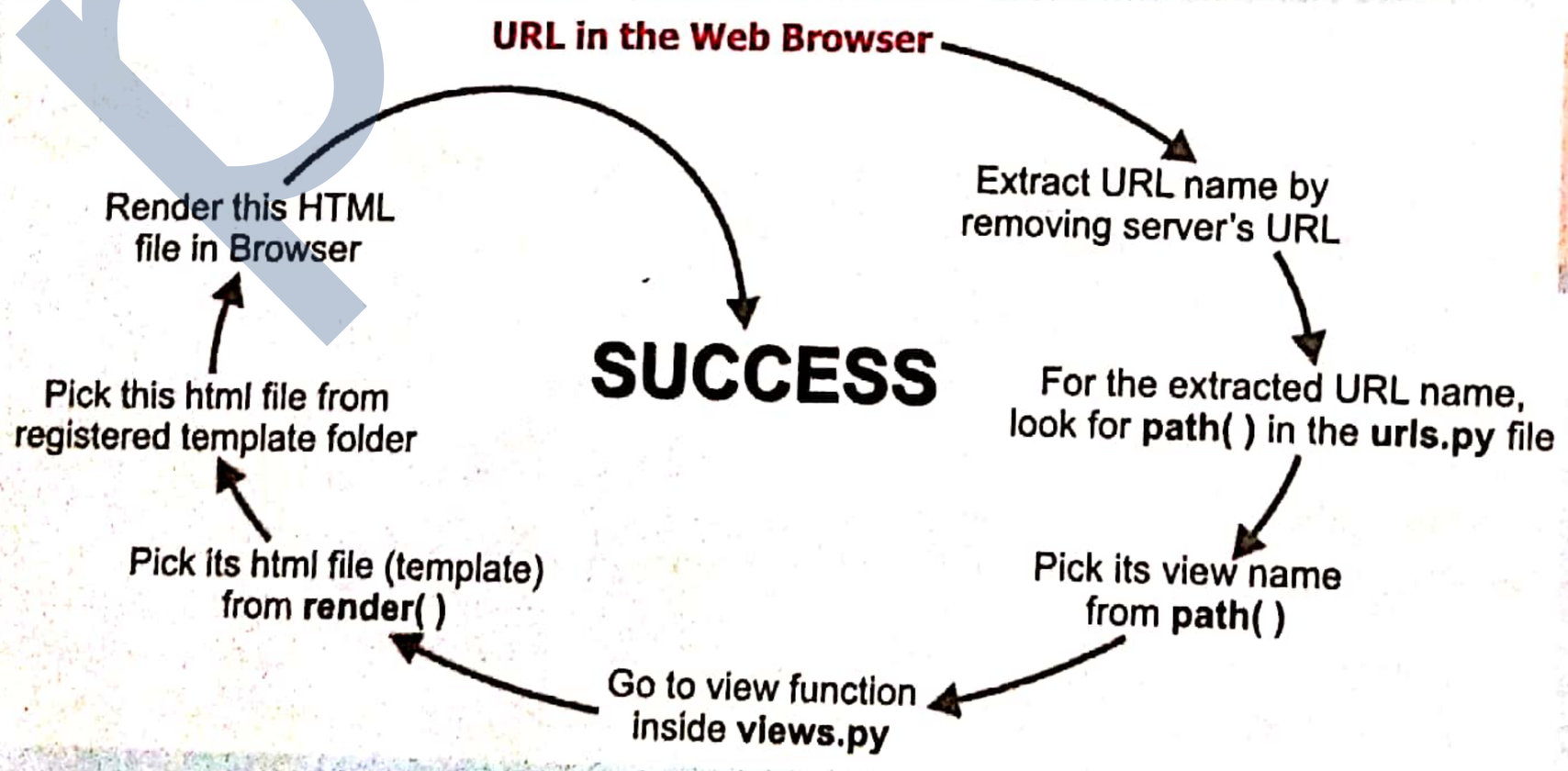
- For each URL you created in previous step, you need to create a URL Conf.
- In the ***urls.py***,
  - import the **views.py** module created in Step 2(*c*).
  - make changes in **urlpatterns** setting as :
    - add a path( ) for each URL.
    - the path( ) links the URL to a view function.

Step 4 RUN BUILT-IN WEBSERVER : **python manage.py runserver**

Step 5 In a web browser, type the urls of your project as **localhost:8000/<url>/** and see them loaded.

- The server's URL is **localhost:8000/**

How URLs get resolved internally ? (URL routing)



You create Django apps in MTV order but at the time of rendering, Django resolves it in MVT order.

Figure 15.7 Steps to CREATE AND RUN A DJANGO APPLICATION.

Check Point 15.2

6. What is the purpose of settings.py?
   (a) To configure settings for the Django project
   (b) To configure settings for an app
   (c) To set the date and time on the server
   (d) To sync the database schema
7. What is the Django shortcut method to more easily render an html response?
   (a) render_to_html
   (b) render_to_response
   (c) response_render
   (d) render
8. What is the default port used by built in webserver of Django?
   (a)800 (b) 8000
   (c) 8800 (d) 127.0.0.1:8000
9. What is the default URL of your Django project when you run it on builtin server?
   (a) localhost:8000
   (b) localhost
   (c) 127.0.0.1
   (d)127.0.0.1:8000
10. What is the name of the administrative task management file in Django projects?
    (a) manage.py
    (b) urls.py
    (c) views.py
    (d) models.py
11. What is the name of the app module in Django projects that stores the view functions?
    (a) manage.py
    (b) urls.py
    (c) views.py
    (d) models.py
12. What is the name of the module in Django projects that stores URL Confs?
    (a) manage.py
    (b) urls.py
    (c) views.py
    (d) models.py

LET US REVISE

- *A web framework is a software tool that provides a way to build and run dynamic websites and web-enabled applications.*
- *An HTTP GET Request refers to a way of retrieving information from the given server using a given URL over web.*
- *An **HTTP POST request** is a way to send data to the server, (e.g., data filled in an online form such as student information, file upload, etc.) using HTML forms over web.*
- *Django is a Python based free and open source web application framework.*
- ***Virtualenv** is a useful tool which creates isolated Python environments that take care of interdependencies and let us develop different applications in isolation. Each isolated environment has different set of libraries unlike a common global set of libraries.*
- *A Django **project** refers to an entire application and an **app** is a submodule of the project caters to one specific functionality.*
- *You can create a Django project by issuing **django-admin startproject <projectname>** command.*
- *There are two directories/folders created with the project name and one is a subfolder of other.*
- *The outer project name folder is the base directory of the project and holds all other app folders and setting modules of the project.*
- *The inner project folder is the project web-application folder and holds all the setting modules for the project.*
- *Command **python manage.py startapp <appname>** creates an app within a Django project.*
- *Django project architecture is based on MTV (Model Template View) architecture. Earlier it was also known as MVC (Model View Controller) architecture.*
- ***Model** in Django project is the data management component(s).*
- ***Template** is responsible for presentation logic for a Django project.*
- ***View** is the handling logic or business logic of a Django project that links the URLs and templates through URL routing.*
- ***Logic tier** provided by Django framework is the **controller**.*
- *Steps to create a Django project are : (i) determine project and its apps (ii) determine for each app the http request type, **model, template, view** and **url**, (iii) create **model, template, views** for the apps as per step (ii) and map **urls** to the templates through **URL Confs**. And (iv) run server and test your project.*
- *The mapping between URL path to the views of Django Project are called URL Confs (URL configurations). The views of Django project are python functions.*
- *Django project views are Python functions that get HTTP request type in the form of mandatory argument **request** (HTTP request type).*
- *URL Routing is a process of resolving and mapping URLs through URL Confs in **urls.py** to actual templates which can be displayed in the web browser as a web page.*

SOLVED PROBLEMS

1. *What is Django ?*

   Solution. Django is a free and open source web application framework, written in Python. It is a server-side web framework that provides rapid development of secure and maintainable websites.

2. *Explain Django Architecture.*

   Solution. Django runs on MVT/MVC architecture. Following are the components that make up Django architecture :

   Models Models are data management components such as database schema and relationships among data.

   Views Views define the business logic component that controls what is to be shown to end-user by linking templates to URLs through resolving and mapping URLs.

   Templates Templates deal with the presentation layer of Django, *i.e.*, in what format the information is to be shown.

   Controller It refers the logic tier of the Django framework the binds and works with all other components of Django project.

3. *What are Django Templates?*

   Solution. Django templates make up the presentation tier of Django. A template can use any presentation format such as XML, HTML etc. Templates can also include logic to fill in data or values coming from Django's files and databases.

4. *What are Django views?*

   Solution. Views are the business logic layer of Django. They take up HTTP request in form a URL and then map URLs using *URL Confs* onto linked templates and produce a web page as HTTP response. Views in Django contain Python functions that render result in terms of some template file. Views are also associated with *URL Confs* so that templates and URLs get mapped (URL routing).

5. *What are two types of HTTP requests?*

   Solution. HTTP requests are used on web by web clients for communicating with webservers. HTTP requests are made in two forms:

   (*i*) **GET request.** This HTTP request is made to request data from the server. This request sends a URL to the webserver and the webserver returns the asked webpage's html, which the web client displays in the browser as a webpage.

   (*ii*) **POST request.** This HTTP request is made to submit data to be processed to the webserver. This request is made by the web clients when it has data (filled via forms or for upload), which the webserver has to process. The POST request carries the data from web client to the web server.

6. *There are two folders created with the project name when you create a Django project. How are these two folders different from one another, and what are their roles in a Django project?*

   Solution. The two folders that have the same name as that of the project are created as parent and child folders. The outer projectname folder is the BASE DIRECTORY of the Django project. It contains *manage.py* file and also contains *templates* in it. All the *app folders* are also created in the outer projectname folder (the base directory folder).

   The inner projectname folder is the project's web application folder. It holds all the settings, views and models for the project.

7. *Identify the URLs formed by following URL Confs in* **urls.py**, *considering the default URL with default port of built-in webserver.*

(i) `path('home/', views.main)`

(ii) `path('home/check/', views.newone)`

(iii) `path('check/', views.onenew)`

(iv) `path('check/home/', views.core)`

Solution. (i) localhost:8000/home/

(ii) localhost:8000/home/check

(iii) localhost:8000/check/

(iv) localhost:8000/check/home/

8. *Identify view functions from the following URL confs and write their function headers.*

(i) `path('home/', views.main)`

(ii) `path('home/check/', views.newone)`

(iii) `path('check/', views.onenew)`

(iv) `path('check/home/', views.core)`

Solution. View functions will be inside views.py and their headers will be :

(i) `def main (request) :`

(ii) `def newone (request) :`

(iii) `def onenew (request) :`

(iv) `def core (request) :`

9. *Write a view function that can process a GET request and display "main.html" as template. Required file(s) and method(s) are already imported.*

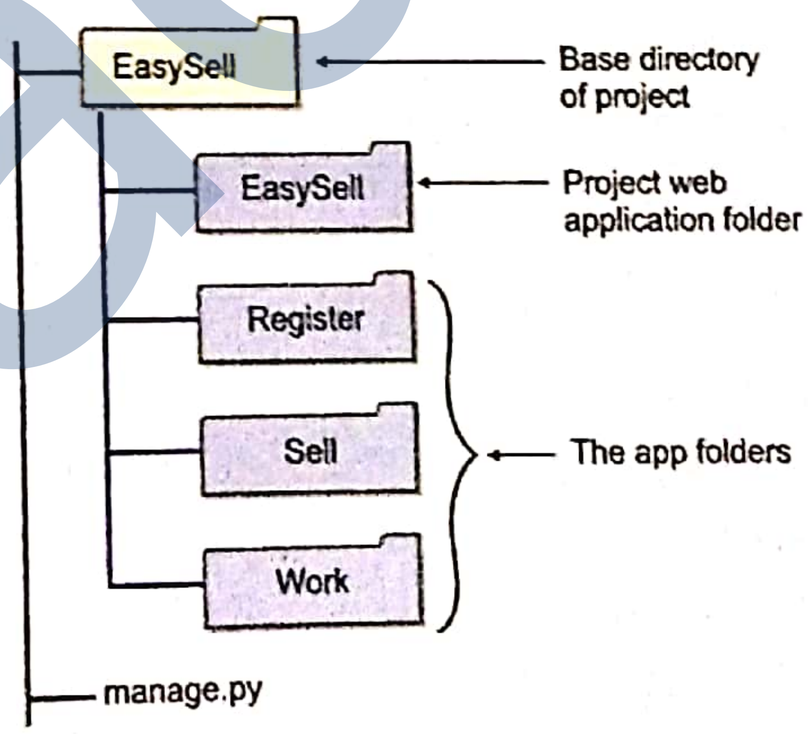
Solution.
```
def display(request) :
    return render(request, "main.html")
```

10. *You have created a Django project namely EasySell. It contains three apps in it :*

(i) *Register* (ii) *Sell* (iii) *Work*

What will be the contents of Django project folder. Only list the folders, do not list any file inside app folders.

Solution.



11. *Name the files that are found in project web application folder.*

Solution.

`__init__.py settings.py urls.py wsgi.py`

12. *Name the files that found in app folders by default.*

Solution.

`__init__.py admin.pyapps.py models.py tests.pyviews.py`

(e) The list app intends to show list of items available. For this, it has an html page that displays a list of items as shown below :

ITEMS LIST

1. Bath tubs
2. Battery Charger
3. Battery Eliminator
4. Beam Scales (upto 1.5 tons)
5. Belt leather & straps
6. Brass Wire
7. Bituminous Paints
8. Blotting Paper
9. Bolts & Nuts
10. Bolts Sliding
11. Bone Meal
12. Boot Polish

Create an html file namely ***list.html*** for this and store it under *templates* folder of your project.

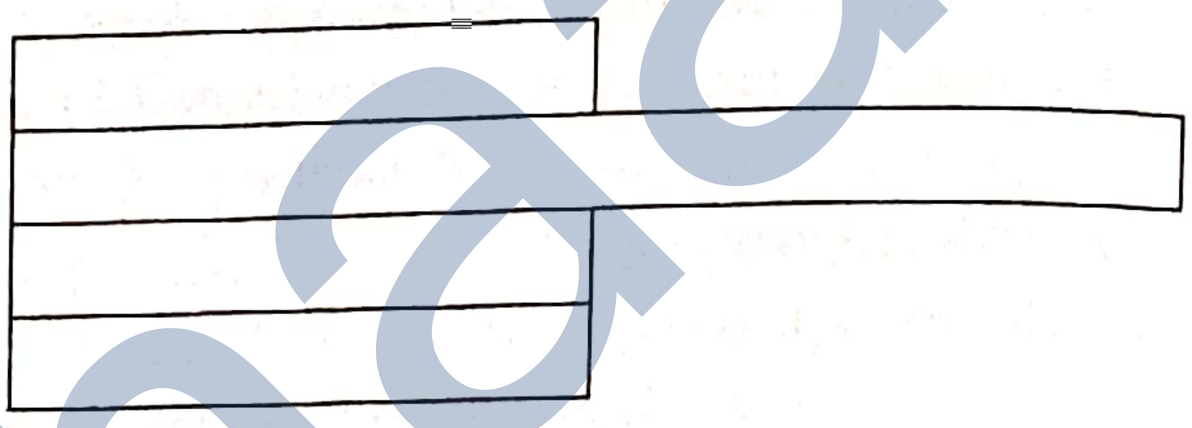
(f) The **order** app intends to display a form as shown below:

Customer name :

Customer address :

Order Item number :

Order item qty :



Create an html file namely **order.html** that displays this form and store it in the *templates* folder.

(g) Create a view function namely **mylist** for ***list app*** that renders ***list.html***(in ..list***views.py***).

(h) Create a view function namely **myorder** for ***order app*** that renders ***order.html***(in ..order ***views.py***). Add POST request processing to it so that it stores the received form data in a CSV file.

(i) In the **urls.py** file of your project's web-application folder (inner project name folder), import the two apps' **views.py** modules by adding following import commands :

```
from list import views as listviews
from order import views as ordviews
```

Recall that the default address of your local web server is ***localhost\8000****. We are referring to this address as* ***<server>*** *in questions below.*

(j) Create a URL Conf for **<server>/list/** URL that links to **mylist** view function created above. Use the name **listviews.mylist** for **mylist** function (as you imported **list\views.py** as *listviews*)

(k) Create a URL Conf for **<server>/order/** URL that links to **myorder** view function created above. Use the name **ordviews.myorder** for **myorder** function (as you imported **order\views.py** as **ordviews**).

(l) Run server for your project. And also open a web browser window.

(m) Type url as **localhost:8000/list/** in the web browser's address bar.

(n) Type url as **localhost:8000/order/** in the web browser's address bar.

(o) Check if your entered data has created any csv file in the base directory folder. If yes, show the contents of the csv file.