# LEARN DJANGO IN 24 HOURS

## FOR BEGINNERS

SIMPLE, CONCISE & EASY GUIDE TO BUILDING WEB APPLICATION USING

- HTML
- CSS
- JAVASCRIPT
- PYTHON
- DJANGO WEB FRAMEWORK

S. BASU

DJA
IN

# LEARN
# DJANGO
# IN 24 HOURS

A beginner's guide to building Web Application using HTML, CSS, Javascript, Python and Django Web Framework

**Copyright © 2021 S Basu**

**All rights reserved.**

# *Disclaimer:*

The information and materials presented here are for educational purposes only. Every effort has been made to make this book as complete and as accurate as possible but no warranty or fitness is implied. The information provided is on an "as is" basis. The ideas and opinions expressed are of the author's own imagination and the author is not affiliated to any organization, school or educational discipline and will not be held accountable or liable for any inadvertent misrepresentation.

# Contents


Chapter 1: HTML and CSS
- 1.1: Introduction to HTML and DOM
- 1.2: Introduction to Cascading Style Sheets (CSS)
  - *1.2.1: Internal CSS*
  - *1.2.2: External CSS*
  - *1.2.3: In-Line CSS*

Chapter 2: Introduction to Javascript
- 2.1: Javascript variable
  - *2.1.1: Difference between Var, Let and Const keyword*
- 2.2: Javascript Data Types
- 2.3: Javascript Function
- 2.4: Javascript Events

Chapter 3: How to Mobile Optimize a website
- 3.1: Three rules to Mobile Optimize a website

Chapter 4: Introduction to Python Programming.
- 4.1: Python Installation
- 4.2 : Python variables, Datatypes & Operators
  - 4.2.1 : Python String
  - 4.2.2 : Python List
  - 4.2.3 : Python Tuple
  - 4.2.4 : Python Dictionary
  - 4.2.5 : Python Operators
- 4.3 : Python Control Statements
  - 4.3.1 : Python If ..Elif ..Else
  - 4.3.2 : Python For Loop
  - 4.3.3 : Python While loop

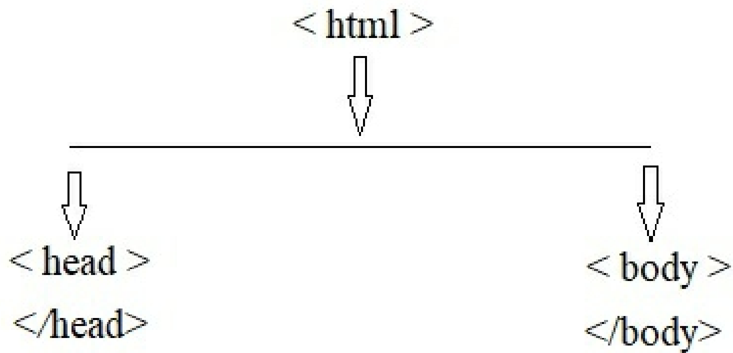
4.3.4 : Python Break and Continue
4.4 : Python functions.
4.5 : Python class
Chapter 5: Django
5.1 : Django Installation
5.2 : Django App
*5.2.1 : Django Architecture*
*5.2.2 : Hello World Example*
5.3 : Django Models
5.4 : Django Admin App
5.5 : How to navigate to different pages of our project using Django
What is {% …. %}?
5.6 : How to load static files like CSS & Images into our Django App
5.7 : Django Form
What is {{ … }}?
5.8 : Django Form with CSS & Rendering form fields manually
5.9 : Log In Page Validation process
5.10 : Displaying data from multiple tables or models into Web Page
5.11 : Django Email Form
What is Django send_mail function?
Important Info & Conclusion:

# Chapter 1: HTML and CSS

## 1.1: Introduction to HTML and DOM

**What is HTML?**

- HTML stands for Hyper Text Markup Language.
- HTML elements help to design a web page and they are represented with open and close tags. Example: (open tag) < *element_name* > **……** </ *element_name* > (close tag)



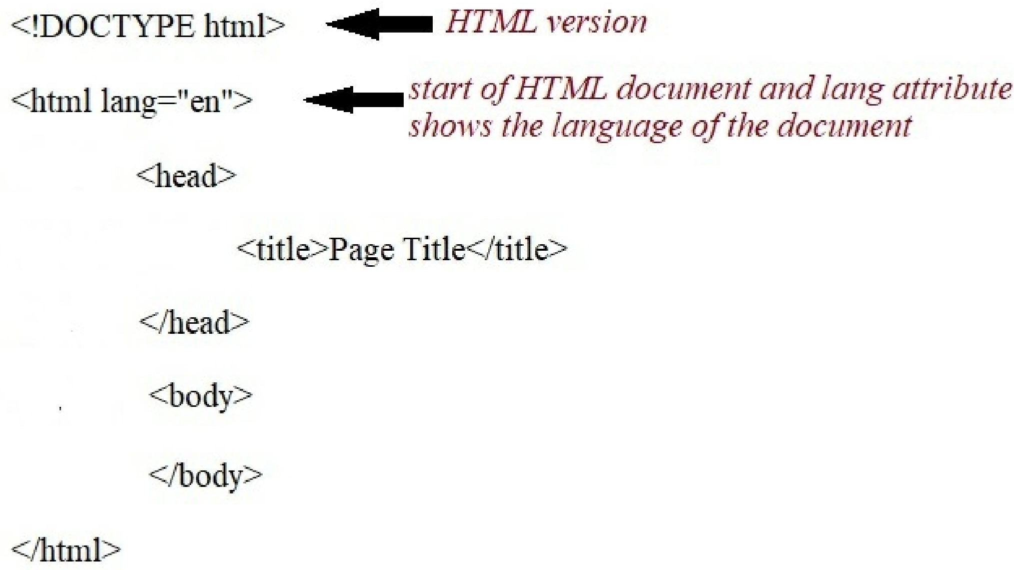

HTML document is divided into **head** and the **body** .

- The **<head>** contains all information about a web page. Example: CSS, Javascript, meta data
- The **<body>** contains the main content of the web page which will get displayed to the user.

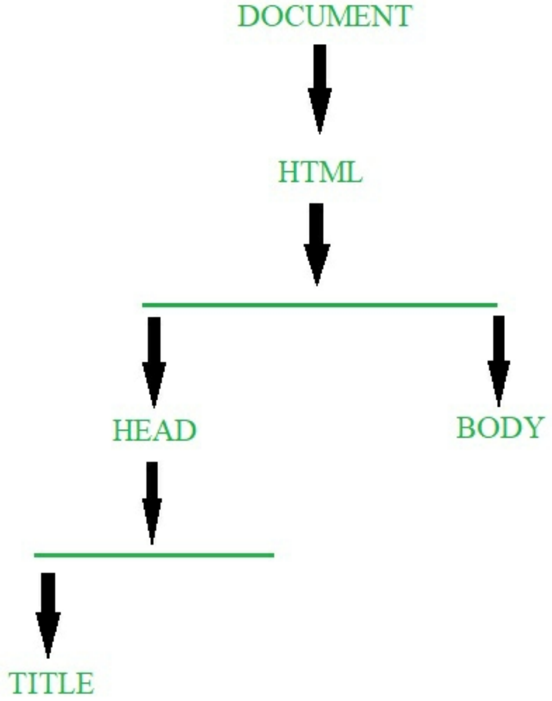
**What is DOM?**

- DOM stands for Document Object Model which defines the logical structure of a document.

Let’s look into a basic DOM structure below




The DOM of above HTML code is:



There are hundreds of HTML tags present. We will look into few important ones as we proceed further and start creating our web project.

# 1.2: Introduction to Cascading Style Sheets (CSS)

CSS stands for Cascading Style Sheets and is mainly used to make a HTML document pretty and presentable. If there are multiple pages using the same styling information, CSS helps us to save time by preventing repetitive work.

CSS is of three types:

1. Internal CSS

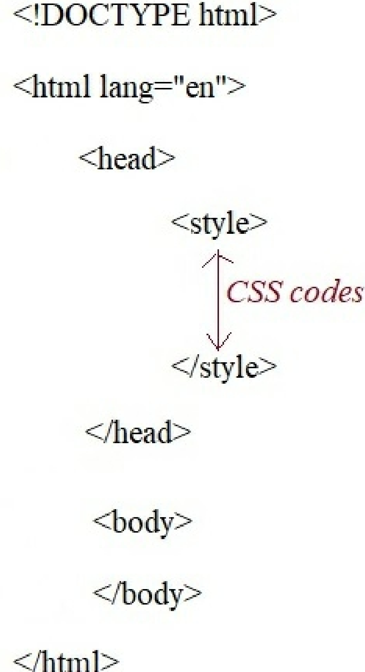
2. External CSS

3. Inline CSS

## *1.2.1: Internal CSS*

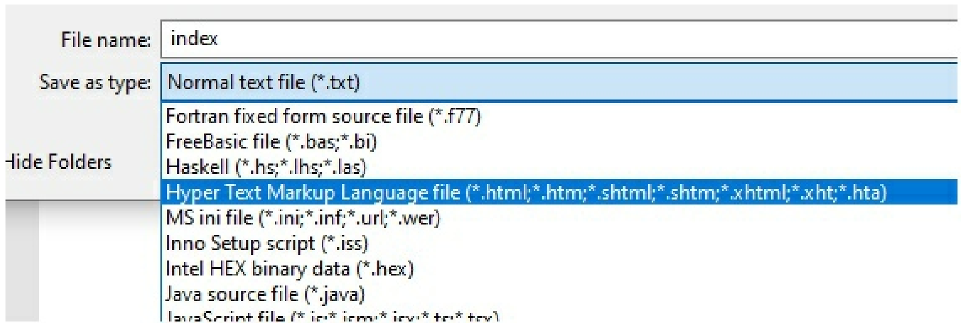
In Internal CSS, the CSS codes and main content of the web page are present within the same HTML document and it provides styling information exclusive for that page.

The CSS codes are written within the **<style>** tags in the **<head>** section of a HTML file.




Now let’s start coding..

We will be creating a HTML document with Internal CSS. Open **Notepad** or **Notepad++** -> Create a New File and save it as HTML. I named my file ***index.html***




In this book, I will be creating a project called ***Doggy Day Care System*** . It is a simple project which will contain a ***Home Page*** , a ***Log In*** , ***Create a new profile*** , ***About Us*** and ***Contact Us*** pages. As we proceed further, we will be converting this project into a **Django Web Application** .

Now let’s design our ***Home page*** .

***<u>Index.html</u>***

```
<!DOCTYPE html>
<html>
<head>
<title>Scrappy-Doo Day Care Center</title>
<style>

    body{

        background-image:url("C:/Users/[redacted]/Picture4.jpg");
        background-repeat: repeat;
        width:100%;
        height:100%;

    }

    #homePageHeader{

    margin-top:50px;
    background-color:white;
    font-family:Kristen ITC;
    font-size:24px;
    text-align:right;
    width:100%;
    height:300px;

    }

</style>
</head>

<body>

    <div id = "homePageHeader">
    <h1>Scrappy-Doo Day Care Center</h1>
    </div>

</body>
</html>
```

In the above example, I created a web page and gave it a **title** of *Scrappy-Doo Day Care Center* . For now the web page only contains a heading which is present within a **<div>** container whose **id** is ***homePageHeader*** .

To add styling information to the **<body>** and the **<div>** container, go to the **<head>** section **->** within **<style>** tag call each HTML element by its name or **id** and give the desired styling information within the curly brackets **{** ….. }.

**What is Id in CSS?**

ID helps to uniquely identify certain HTML elements. In order to access the **id** of an HTML element within **<style** > tag, a hashtag ( **#** ) symbol is used. Example: **#** ***id_name***

**What is <div> tag?**

The **<div>** tag defines a division or a section in an HTML document. It is a good practice to divide different sections of a web page into different **<div>** containers.

- In the above screen shot, I added a **background image** to the <**body>** of the web page. Its CSS code is:

  **background - image : url ( "** *picture_location* **" ) ;**

- If we want the background image to **repeat** itself and fill the web page then the CSS code is:

  **background-repeat: repeat;**

- I have set the **height** and the **width** of the body of the web page to 100%.

**NOTE:** It is very important to set **height** and **width** of the body of a web page. This acts as a parent container and helps to prevent layout shifts of different elements present within the web page.

- To access the **id** of the **<div>** element within the **<style>** tag, **#** symbol is used.

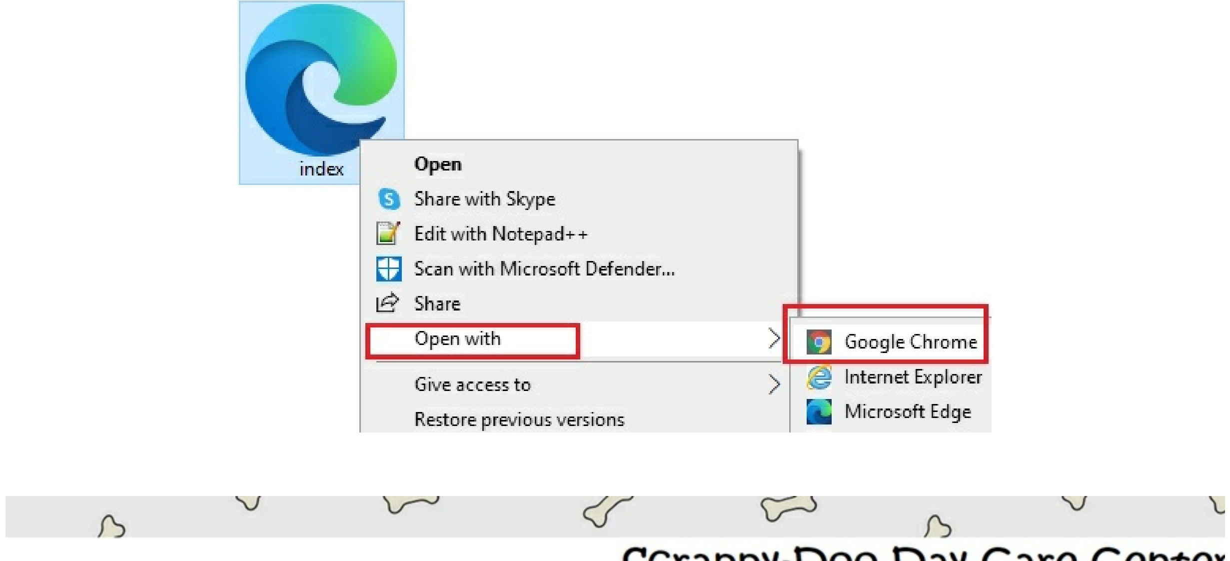
  **margin - top** sets the top margin to the **<div>** container to *50px* .

  **background - color** sets the color of the background of **<div>** container to *white* .

  **font - family** sets the font used within the **<div>** container to *Kristen ITC* .

**text - align** sets the text used within the **<div>** container to the *right side* of the container.

Now let’s save ***index.html*** and open with Google Chrome.




Everything looks perfect, but now I would like to set the heading of my web page at the **bottom right corner** of the **<div>** container. To do that, let’s

make some minor changes into our ***index.html*** file.

```
#homePageHeader{

margin-top:50px;
background-color:white;
font-family:Kristen ITC;
font-size:24px;
text-align:right;
width:100%;
height:300px;
position:relative;

}

h1{
position:absolute;
bottom:0px;
right:20px;
}

</style>
</head>

<body>

    <div id = "homePageHeader">
    <h1>Scrappy-Doo Day Care Center</h1>
    </div>
```

- I declared the **<div>** element with **id** ***homePageHeader*** as the parent container and set its **position** to **relative** .

  **position : relative**

- Then I declared the **<h1>** tag as child element and set its **position** to **absolute** .

  **position : absolute**

  Now the **<h1>** tag (*child element* ) is positioned relative to the parent element. We set the **bottom** of **<h1>** to *0px* and **right** to *20px* .

> **NOTE:** In order to move a child element to the **bottom** , **top** , **left** or **right** we need to declare a parent element and a child element by using **position properties** .

Now let’s save everything and open our web page

Everything looks perfect.

Now let’s proceed to adding a logo picture to our website .

```
}

h1{
position:absolute;
bottom:0px;
right:20px;
}

#logoPic{
float:left;
}


</style>
</head>

<body>

    <div id = "homePageHeader">
    <img id = "logoPic" src = "C:\[redacted]\Picture1.jpg" alt="Logo Pic">
    <h1>Scrappy-Doo Day Care Center</h1>
    </div>
```

The syntax for adding any image is :

**< img src = "** *image_location* **" alt= "** *image_name* **" >**

- **alt** attribute is used to set an alternate text for the image in case the image fails to load.
- In the above piece of code we see that an **id *logoPic*** is assigned to **<image>** tag and the image is placed within the parent container with **id *homePageHeader*** .
- I would like to set the logo picture on the **left** hand side of its parent container and in order to do that we use CSS **float property** .

  **float : left**

**What is float property?**

The CSS float property is used to place an element to the **left** or **right** of its parent container.

It looks perfect.

Now let's create a list of links which will enable a user to navigate to different pages. For now I will be creating two links, one will lead us to a ***Log In*** page and the other link will lead us to a ***Contact Us*** page.

**HTML**

```
<div id = "homePageLinks">
<ul>
<li><a href = "C:\ ............................................................................ \logInPage.html">Log In</a></li>
<li><a href = "C:\ ............................................................................ \contactUsPage.html">Contact Us</a></li>
</ul>
</div>
```

**CSS**

```
#homePageLinks{
background-color:#e68a00;
width:145px;
height:50px;
position:relative;
}

ul{
position:absolute;
text-align:center;
}

li{
display: inline;
font-family:Verdana;
}

a{
color:white;
font-weight:bold;
}
a:hover{
color:black;
}
```

In the above HTML code, we created another **<div>** parent element with **id *homePageLinks*** which will store the list of links. To create any **list** , **<ul>** and **<li>** tags are needed.

The syntax is:

```
<ul>
<li> item one </li>
<li> item two </li>
…..
</ul>
```

**<ul>** stands for **unordered list** and each items within the list should be present within **<li>** tags.

In order to make the links functional, **<a> href** attribute is needed. The syntax is:

**< a href = "** *..url…* **">** *link_name* **</a>**

**What is <a> tag and its href attribute?**

**<a>** tag is used to define a hyperlink and its **href** attribute specifies the **url** of the page the link goes to.

In the above CSS screen shot,

- I declared ***homePageLinks*** as my parent container and set its **position relative** . I also gave a **height** and **width** to the container and set a **background color** .

**background - color :** *color_name* **;**

- I would like **<li>** tag within the **<ul>** tag to be displayed side by side instead of one after another. In order to do that I set its **display** property to **inline** .

**display : inline;**

- For the **<a>** tag links, I set its color and made the text bold.

**color :** *color_name* **;**

**font – weight : bold;**

**:hover** selector is used to change color when a user hovers over that element.

The code *a* **: hover { color :** *black* **; }** specifies that turns the color of ***Log In*** text *black* when a user hovers the mouse over it.

Everything looks good. For now I will be displaying only one link (*Log In* ) and moved the other link (*Contact Us* ) to a drop down menu.

Let's create a drop down menu which will be present in the middle of the *home page* . The syntax for creating a drop down menu is :

```
<select>
<option value = " val1 "> Name_1 </option>
<option value = " val2 "> Name_2 </option>
……
</select>
```

HTML

```
<div id = "homePageDropdownMenu">

<label for="dropDown"></label>
<select id ="dropDown" onchange="window.location.href = this.value">

<option selected disabled hidden value=''></option>

<option class = "optionVal"
value="C:\[illegible]\HTML\aboutUs.html">
About Us</option>

<option class = "optionVal"
value="C:\[illegible]\HTML\createNewProfile.html">
Create a New Profile</option>

<option class = "optionVal"
value="C:\[illegible]\HTML\contactUsPage.html">
Contact Us</option>

</select>

</div>
```

CSS

```
#homePageDropdownMenu{
background-color:#bfbfbf;
border: 15px solid white;
margin-top:50px;
width:50%;
height:400px;
margin-left:auto;
margin-right:auto;
position:relative;
}

#dropDown{
position:absolute;
top:40%;
left:25%;
width:500px;
padding:20px;
font-size:24px;

}

.optionVal{
 font-size:24px;
}
```

In the above HTML screen shot

- I created a **<div>** section and gave an **id** of ***homePageDropdownMenu*** . Now within that **<div>** container I created a drop down menu with **id *dropDown*** which will help us to navigate to different pages when its value is selected.
    - The **<option>** within the **<select>** tag belongs to a CSS **class *optionVal*** .

**Difference between id and class in CSS?**

1. A **class** selector is a name preceded by a full stop (.) , whereas an **id** is preceded by hash sign (#).

2. An **Id** is unique and it only refers to one HTML element. But a **class** can refer to multiple elements.

In the above CSS screen shot

- I declared ***homePageDropdownMenu*** as a parent element and set its **position** property to **relative** . I would like my parent element aligned to the center of the page, in order to do that **margin-left** and **margin-right** should be set to **auto** .

  **margin - left : auto;**

  **margin - right : auto;**

  Now, I positioned **<select>** tag with **id *dropDown*** relative to its parent element.

  **position : absolute;**

- In the above HTML code, you will notice the following line:

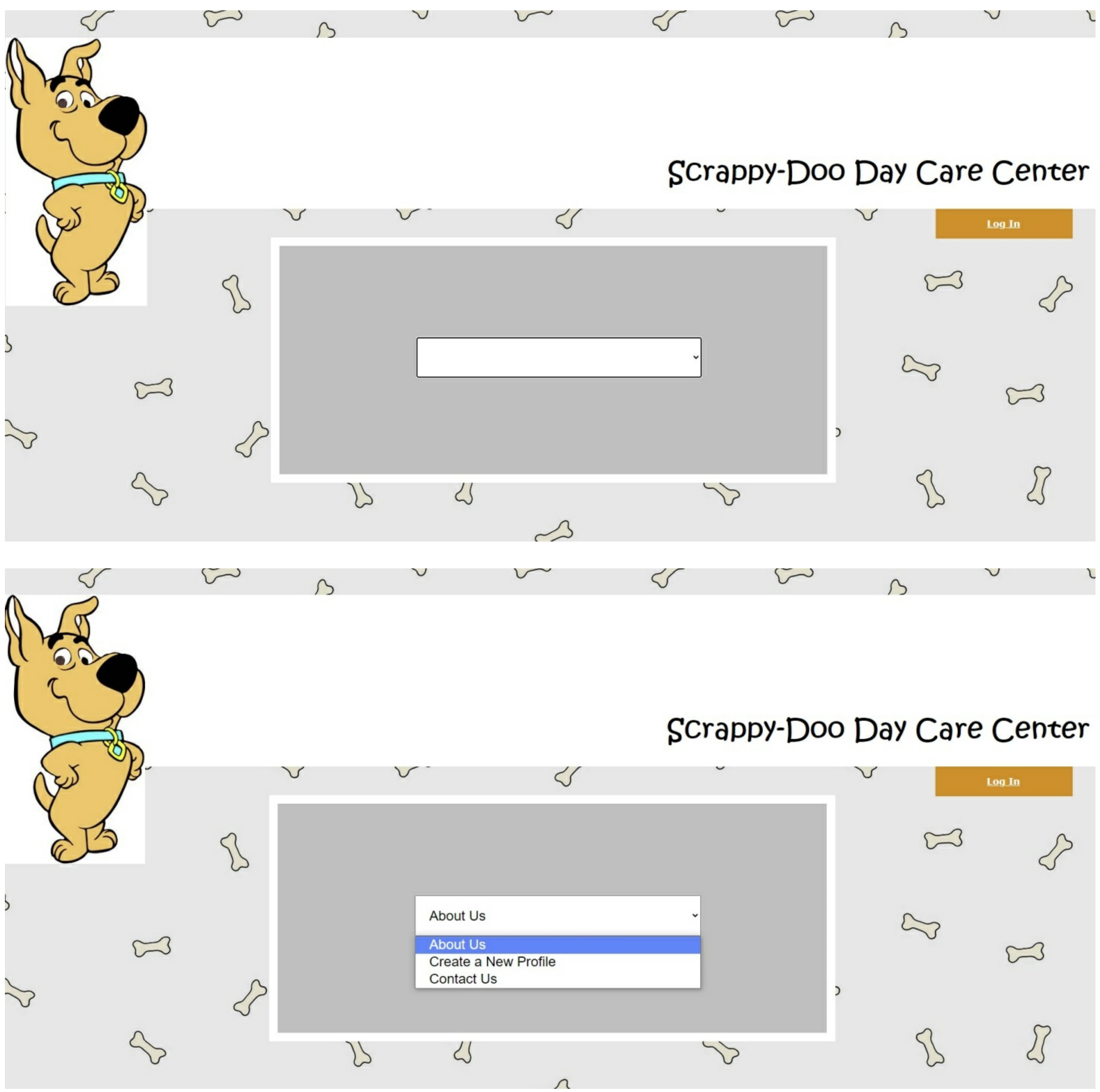
```
<option selected disabled hidden value=''></option>
```

    - **selected** attribute is used to display the default item which will appear in the drop down menu when the page loads.
    - **disabled** attribute makes that item disable and unclickable.

- **hidden** attribute is used to hide the element.

Due to this line of code, a default blank item will appear in our drop down menu when the page loads .

Now let's save everything and open ***index.html*** with Google Chrome.




Everything look good, but still we are unable to navigate to different pages which I select the values from the drop down menu. In order to do that

Javascript is needed (*please refer to section 2.4 of Chapter 2* )

Now let's proceed to making our ***Log In*** , ***About Us*** , ***Create a New Profile*** and ***Contact Us*** HTML pages.

### *1.2.2: External CSS*

Important points to note in External CSS are:

- In External CSS, we write our CSS code in a separate file and link that file with our HTML document using **<link>** tag.
- By separating the CSS code from HTML codes, makes the HTML document look cleaner and less complicated.
- External CSS saves time and prevents repetitive work of writing the same CSS code for multiple pages.

Now let's create a new file and I named my CSS file ***commonCss.css*** . This CSS file will contain the styling information used by all four pages (*Log In, Create a New Profile, About Us and Contact Us page* ).

***commonCss.css***

```
body{
    background-image:url("C:/Users/[redacted]/Picture4.jpg");
    background-repeat: repeat;
    width:100%;
    height:100%;
}
#containerLogIn{
    margin-top:150px;
    background-color:white;
    width:50%;
    height:500px;
    margin-left:auto;
    margin-right:auto;
    font-family:Verdana;
    font-size:18px;
    position:relative;
}

#submitButtonLogIn{
   position:absolute;
   bottom:70px;
   right:70px;
   width:100px;
   height:50px;
}

#submitButtonLogIn:hover{
   background-color:orange;
}

.fieldStyle{
   width:300px;
   height:30px;
}
```

To incorporate this styling information into our pages, we need **<link>** tags and this tag should be present within the **<head>** section of a HTML document.

```
<!DOCTYPE html>
<html>
<head>
<title>Log In - Scrappy-Doo Day Care Center</title>
<link rel="stylesheet" href="commonCss.css">
</head>
```

Now let’s create the ***Log In*** page. The ***Log In*** page will have two input fields, one for ***username*** and one for ***password*** .

- The **<input>** syntax which will take only text value is:

**<input type = "text" placeholder = "** *default_name* **" name = "** *name* **" id = “** *id* **”>**

- The **<input>** syntax which will take only password value is:

**<input type = "password" placeholder = "** *default_name* **" name = "** *name* **" id = “** *id* **”>**

- The **<input>** syntax for submit button :

**<input type = "submit" name = “** *name* **”>**

- The **<input>** syntax for radio button :

**<input type = "radio" name = "** *name* **">**

***logInPage.html***

```
<!DOCTYPE html>
<html>
<head>
<title>Log In - Scrappy-Doo Day Care Center</title>
<link rel="stylesheet" href="commonCss.css">
<style>

td{
   padding:50px;
}

</style>
</head>
<body>

<div id = "containerLogIn">

<table>
<tr>
<td><label for="uname">Username</label></td>
<td><input id = "uname" class = "fieldStyle" type = "text"
placeholder = "Enter Username" name = "username"></td>
</tr>


<tr>
<td><label for="pwd">Password</label></td>
<td><input id = "pwd" class = "fieldStyle" type = "password"
placeholder = "Enter password" name = "password"></td>
</tr>

<tr><input id = "submitButtonLogIn" type = "submit"
name = "submit"></tr>
</table>

</div>

</body>
</html>
```

From the above HTML code you will notice that the input fields are present within a HTML table structure.

Syntax for creating a table is :

**<table>**

**<th>…</th>**

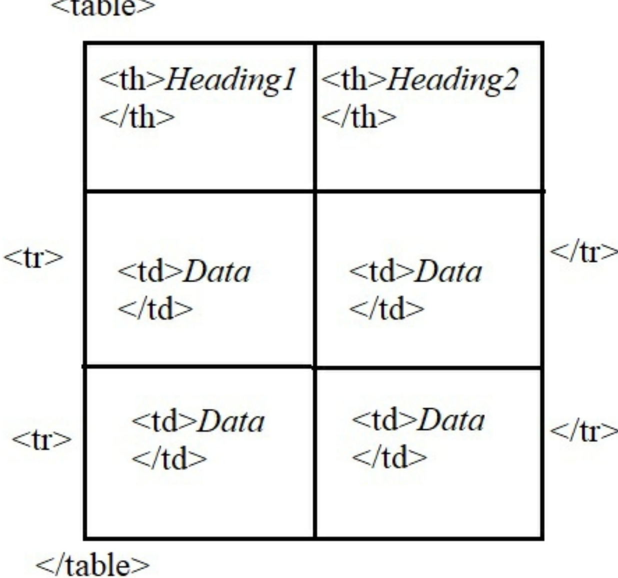
**<th>.. </th>**

**<tr>…<td> ..</td>  ..</tr>**

**<tr>..<td>….</td>..</tr>**

**……**

**</table>**




- **<th>** stands for table header
- **<tr>** stands for table row
- **<td>** stands for table data.

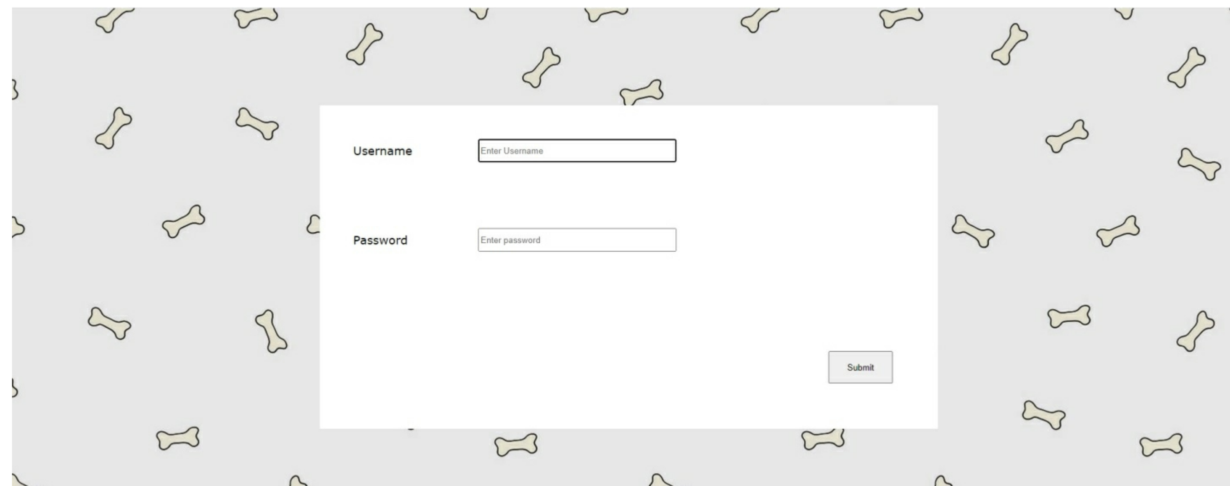
In the above HTML code, you will also notice the presence of **<label>** tag.

**What is <label> tag?**

**<label>** tag makes the cursor appear automatically in the input field when an users click on the text or some area around the text.

To connect **<label>** tag with its corresponding **<input>** tag, the input field's **id** must match with the **for** attribute of **<label>** tag

Now let's save the file and open it with Google Chrome browser.




Everything looks perfect.

**Please note:** We will be creating rest of the pages of our project in Chapter 2.

### *1.2.3: In-Line CSS*

Few important points to note in In-Line CSS are:

- In In-Line CSS, the styling information is added with the HTML element by using the **style** attribute. Example:
  **<h1 style = "** *font-family : Verdana;* **">** *Hello World* **</h1>**
- Developers usually avoid In-line CSS because:
  1. It makes the HTML code look messy and big. The best practice is to separate the content and the design portion.
  2. As our HTML document starts to become big and complex, In-line CSS becomes confusing and harder to maintain.
  3. It is time consuming.

# Chapter 2: Introduction to Javascript

Few important points to note about Javascript are:

1. Javascript is an **Event Driven Programming** language. When an **event** happens like a button is clicked or a value is selected then the Javascript codes get triggered and they perform a certain task. The task is usually written within a **function** .

2. Javascript is mainly used to depict the behavior of a web page.

3. Javascript codes should be written with the **<script>** tag and the **<script>** tag can be present within the **<head>** section or the **<body>** section of a HTML document.

The best practice is to write Javascript codes separately at the end just before closing the **<body>** tag.

**Please note:** In this chapter, I will be covering few important Javascript topics essential for the development of our project. To gain in-depth and through knowledge of all Javascript topics, please visit website https://www.w3schools.com/

## 2.1: Javascript variable

There are three ways a variable in Javascript can be declared:

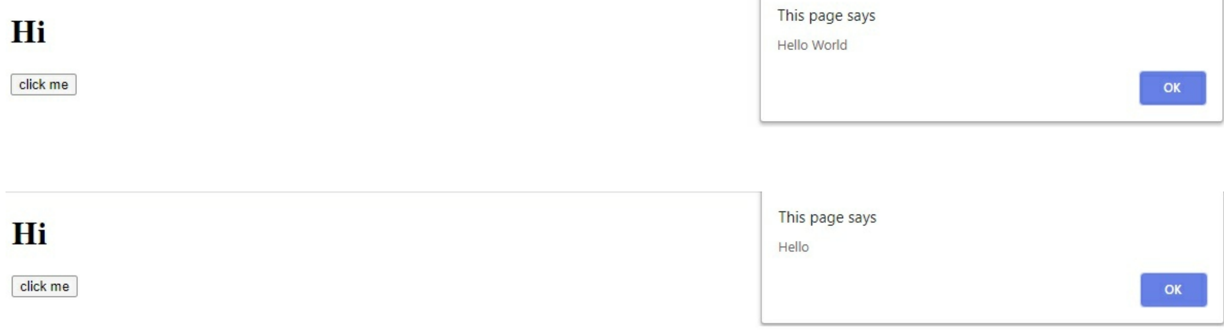
1. **Var** keyword
2. **Let** keyword
3. **Const** keyword.

Of the above three, **Let** and **Const** is more preferable to use by developers.

## *2.1.1: Difference between Var, Let and Const keyword*

```
<!DOCTYPE html>
<html>
<head>
<title>Demo</title>
<script>


function hello(){
    var x = 5;
    var word1 = "Hello";

    if(x<6){
    var word2 = "Hello World";
    alert(word2);
    }
    alert(word1);
    alert(word2);
}




</script>
</head>
<body>
<h1>Hi</h1>
<button onclick="hello()">click me</button>
</body>
</html>
```

## **Output**



**Hi**

click me

This page says

Hello World

OK

**Code Explanation:**

- In the above piece of code, you will notice that the **body** of the HTML document contains a simple button named ***click me*** . With the help of Javascript **onclick** event, the **function *hello( )*** runs when the button is clicked.

The syntax for creating a button is:

**<button type="button">** *Click Me* **</button>**

> **What is Javascript onclick event?**
>
> Onclick event occurs when a user clicks on an element.
>
> **What is Javascript alert( ) function?**
>
> **alert( )** function is used to display a message in a small pop up box.

- The ***hello( )* function** is present within the **<script>** tag of **head** section of HTML document. In this function, we declared two variables ***x*** and ***word1*** .

Within the **if loop** we declared another variable ***word2*** .
Now when we run our code, we get three outputs as shown in the screen shots above, one from ***word1*** and others from ***word2*** .

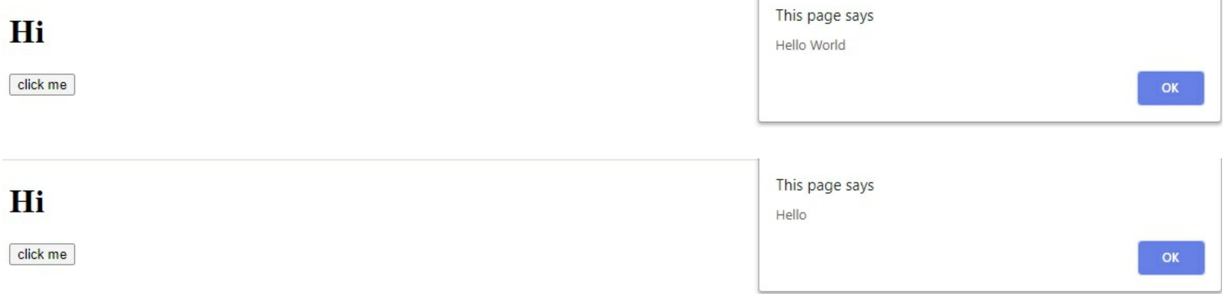
> **NOTE:** Code written within curly brackets **{ }** is referred to as **Block** . ***word2*** is declared within the **if** block of codes and we see that it is

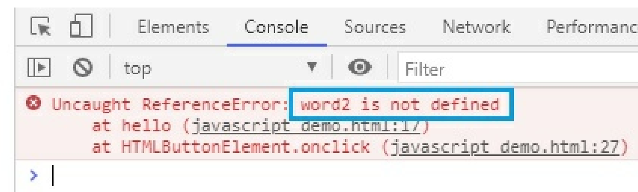
accessible from outside the **if** block of codes. This means that variables declared with **var** keyword does not have any **Block Scope** .

Now let's run the same code by using **let** keyword.

```
<!DOCTYPE html>
<html>
<head>
<title>Demo</title>
<script>


function hello(){
    let x = 5;
    let word1 = "Hello";

    if(x<6){
    let word2 = "Hello World";
    alert(word2);
    }
    alert(word1);
    alert(word2);
}




</script>
</head>
<body>
<h1>Hi</h1>
<button onclick="hello()">click me</button>
</body>
</html>
```

## Output






**NOTE:** When we tried to access the ***word2*** variable from outside the **if** block of codes, it returned an error saying *" word2 is not defined "* . We were unable to access ***word2*** from outside the **if** block of codes. This means that variables declared with **let** keyword have **Block Scope** .

**The difference between var, let, const is:**

**var** variables can be accessed from outside a block of codes written within curly braces.

**let** variables cannot be accessed from outside a block of codes written within curly braces.

Any variable declared with **const** keyword means that its value is fixed or constant.

## 2.2: Javascript Data Types

Important points to note are:

- In Javascript, the data types are:

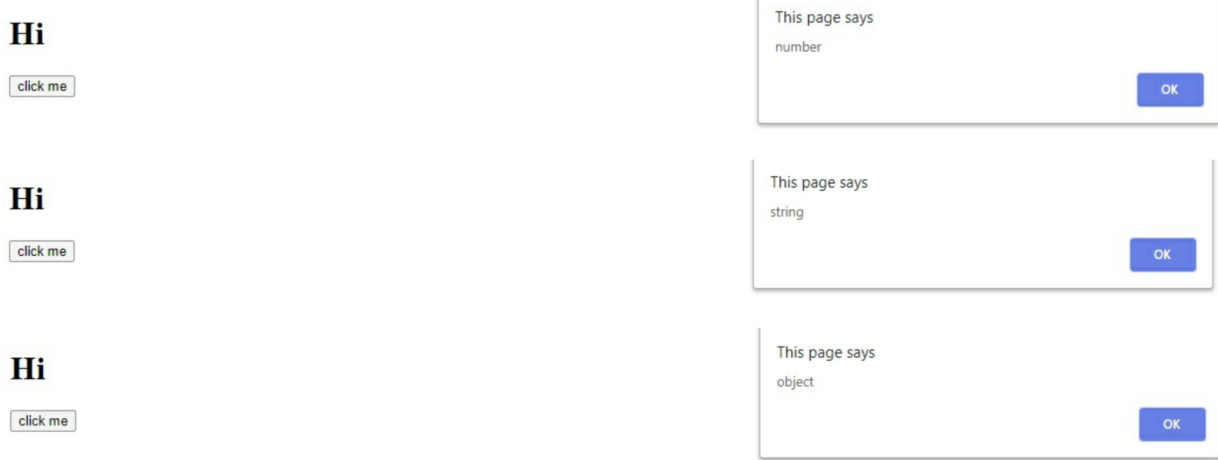
  1. Number
  2. String
  3. Object

- In Javascript, we do not have to declare a variable's data type like **int** *x = 10* or **string** *x = "John"* .

  By simply assigning a value into a variable, Javascript will know its data type.

```html
<!DOCTYPE html>
<html>
<head>
<title>Demo</title>
<script>


function hello(){
   let x = 5;

   let y = "Hello World";

   let z = {Name:"Nick", Age:90};

   let a = ["Toyota","Ford"];

   alert(typeof(x));

   alert(typeof(y));

   alert(typeof(z));

   alert(typeof(a));
}


</script>
</head>
<body>
<h1>Hi</h1>
<button onclick="hello()">click me</button>
</body>
</html>
```

## Output



**Hi**

click me

This page says

object

OK

**What is Javascript typeof operator?**

**typeof** operator is used to get the data type of a variable.

In the above code, we see Javascript shows the datatype of:

***x*** as **number** ,

***y*** as **string** ,

***z*** as **object**

***a*** as also **object** .

**What is Javascript Object?**

Javascript Objects are written with curly brackets **{ }** and its properties are written as **name : value** pairs, separated by commas. Example:

*employee* = **{**
*emp_id* **:** *123* **,**
*emp_name* **:** **“** *John* **”**
**};**

## 2.3: Javascript Function

- Javascript function is a block of code performing some task. The syntax is :

**function** *function_name* **( ) {**
**…………………..**
**}**

- The code inside a function will get called when a Javascript **event** occurs, like some button is clicked or some item from a list is selected.

Now let's create the ***createNewProfile.html*** file of our *Doggy Day Care Center project* .

***createNewProfile.html***

```
<body>

<form onsubmit = "createNewProfile()">

<div class = "containerCreateNewProfile">

<table id = "createNewProfile">

<tr>
<td><label for="fName">First Name</label></td>
<td><input id = "fName" class = "fieldStyle" type = "text"
placeholder = "Enter First Name" name = "firstName"></td>
</tr>

<tr>
<td><label for="lName">Last Name</label></td>
<td><input id = "lName" class = "fieldStyle" type = "text"
placeholder = "Enter Last Name" name = "lastName"></td>
</tr>

<tr>
<td><label for="gender">Gender</label></td>
<td><label for="gender">Male</label>
<input id = "gender" type = "radio" name = "male" value = "male"></td>
<td><label for="gender">Female</label>
<input id = "gender" type = "radio" name = "female" value = "female"></td>
</tr>
```

```
<tr>
<td><label for="usn">Username</label></td>
<td><input id = "usn" class = "fieldStyle" type = "text"
placeholder = "Enter username" name = "username"></td>
</tr>

<tr>
<td><label for="pwd">Password</label></td>
<td><input id = "pwd" class = "fieldStyle" type = "password"
placeholder = "Enter password" name = "password"></td>
</tr>

<tr>
<td><label for="address">Address</label></td>
<td><input id = "address" class = "fieldStyle" type = "text"
placeholder = "Enter address" name = "address"></td>
</tr>

<tr>
<td><label for="state">State</label></td>
<td><select id = "state" class = "createPageDropDowns">
<option value = "california">California</option>
<option value = "new york">New York</option>
<option value = "texas">Texas</option></select></td>
</tr>
```

```
<tr>
<td><label for="dogNum">Number of dogs</label></td>
<td><select id = "dogNum" class = "createPageDropDowns">
<option value = "0">0</option>
<option value = "1">1</option>
<option value = "2">2</option>
<option value = "3">More than 2</option>
</select></td>
</tr>

<tr>
<td><label for="monthlyFee">Monthly Fee : Please select a package</label></td>
<td><select id = "monthlyFee" class = "createPageDropDowns">
<option value = "0">0</option>
<option value = "basic">$50/dog : Basic</option>
<option value = "silver">$100/dog : Silver</option>
<option value = "gold">$150/dog : Gold</option>
<option value = "platinum">$200/dog : Platinum</option>
</select></td>
</tr>
```

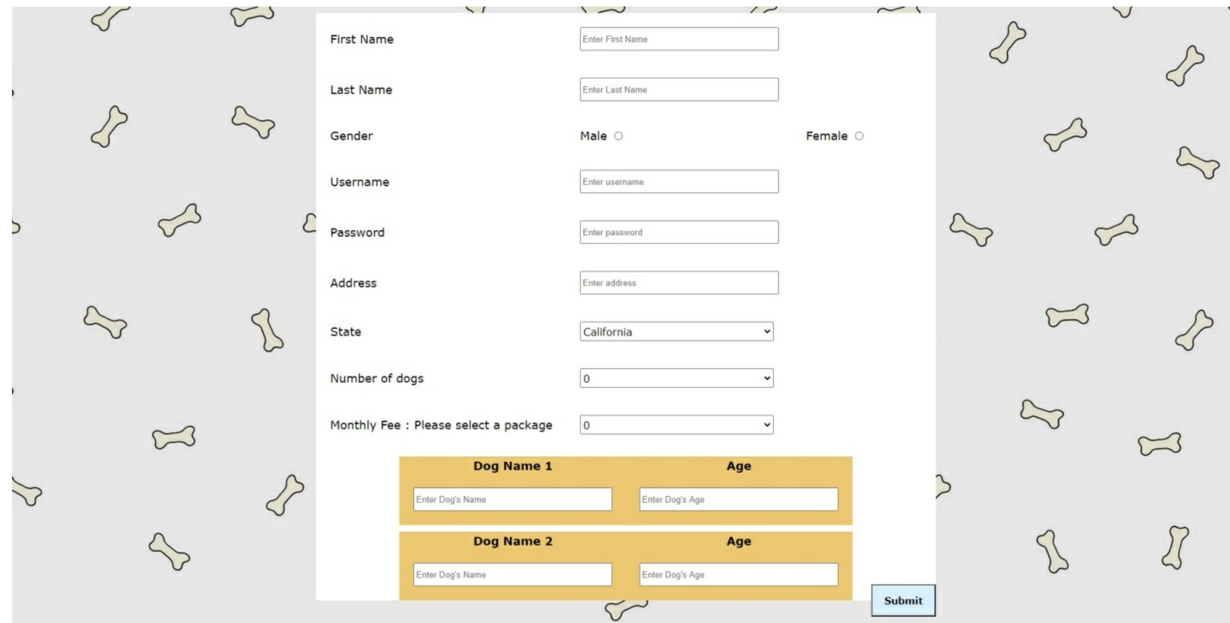
```
<tr>
<table class = "dogInfoTable">
<tr>
<th><label for="dog1">Dog Name 1</label></th>
<th><label for="dogAge1">Age</label></th>
</tr>
<tr><td>
<input id = "dog1" class = "fieldStyle" type = "text"
placeholder = "Enter Dog's Name" name = "dog1">
</td>
<td>
<input id = "dogAge1" class = "fieldStyle" type = "number"
placeholder = "Enter Dog's Age" name = "dogAge1">
</td>
</tr>
</table>
</tr>

<tr>
<table class = "dogInfoTable">
<tr>
<th><label for="dog2">Dog Name 2</label></th>
<th><label for="dogAge2">Age</label></th>
</tr>
<tr><td>
<input id = "dog2" class = "fieldStyle" type = "text"
placeholder = "Enter Dog's Name" name = "dog2">
</td>
<td>
<input id = "dogAge2" class = "fieldStyle" type = "number"
placeholder = "Enter Dog's Age" name = "dogAge2">
</td>
</tr>
</table>
</tr>
<tr><input id = "submitButtonCreateNewProfile" type = "submit" name = "submit"></tr>
</table>
</div>
</form>
```

## CSS code

```
<style>

.dogInfoTable{
background-color:#ffc266;
margin-top:10px;
margin-left:auto;
margin-right:auto;
}

td{
   padding:20px;
}

.createPageDropDowns{
   width:300px;
   height:30px;
   font-size: 16px;
   font-family:Verdana;
}

</style>
```

```
.containerCreateNewProfile{
     margin-top:10px;
     background-color:white;
     width:50%;
     height:auto;
     margin-left:auto;
     margin-right:auto;
     font-family:Verdana;
     font-size:18px;
     position:relative;

}

#submitButtonCreateNewProfile{
    font-family:Verdana;
    font-size:15px;
    font-weight:bold;
    background-color:#ccf2ff;
    position:absolute;
    bottom:-25px;
    right:0px;
    width:100px;
    height:50px;
}

#submitButtonCreateNewProfile:hover{
    background-color:orange;
}
```

Let’s save everything and open the HTML file with Google Chrome.




**Code Explanation:**

We have discussed about **<input>** tags, **<label>** tags in pervious sections (*chapter 1, section 1.2.2* ). Now let's discuss about **<form>** tag.

**What is <form> tag?**

The **<form>** tag is used to create an HTML form where a user can enter data.

The most important attribute of a **<form>** tag is its **action** attribute. The action attribute contains a **url** and once the **form** is submitted, the form data gets send to that specified **url** .

The two important **form** methods to process data from one page to another are: **GET** and **POST**

**GET** – This methods appends **form** data into the URL name. This makes the entire process risky because if an user sends confidential information it will appear on the URL.

**POST** – This method appends **form** data with the HTTP request and not with the URL.

In the above HTML code, within the **<form>** tag, I have called an **onsubmit event** and passed a **function *createNewProfile( )*** into it.

```
<form onsubmit = "createNewProfile()">
```

The **function *createNewProfile( )*** executes once the **form** is submitted.

**What is Javascript onsubmit event?**

**Onsubmit** event triggers when a form is submitted.

Now let's create the **function *createNewProfile*** .

Javascript **functions** should be written within **<script>** tags and the best practice is to write the code separately at the end just before closing of the **<body>** tag.

```
<script>
function createNewProfile(){

let x = document.getElementById("usn").value;
if(x =="" || x ==null){
   alert("Please enter a username");
        }

let y = document.getElementById("pwd").value;
if(y =="" || y ==null){
    alert("Please enter a password");
        }

let z = document.getElementById("dogNum").value;
let zInt = parseInt(z);
if(zInt < 1){
    alert("Please enter the number of dogs");
         }

if(zInt > 2){
    alert("Sorry, we can register at most 2 dogs per customer");
          }

let a = document.getElementById("monthlyFee").value;
if(a =="0"){
    alert("Please select a monthly fee package");
          }
}

</script>
```

This **function** simply checks whether there is values present in ***Username*** , ***Password*** , ***Number of dogs*** and ***Monthly Fee*** fields or not ( *please refer to the above HTML code for these fields* ) . If no value is present then it throws an **alert** .

Few important points to note are:

- In order to access value of an input field by its **id** , the Javascript syntax is :

  **document . getElementById (“** *id_name* **“) . value**

- In order to access value of an input field by its **class** , the Javascript syntax is :

  **document . getElementsByClassName (“** *class_name* **“) . value**

- In order to convert a string value to integer valu e , **parseInt( ) function** is used.

**What is the difference between = = and = = = comparison operators?**

= = is used to compare values between two variables.

= = = is used to compare both values and its data type.

Now let's create our ***Contact Us Page*** of our project.

***contactUs.html***

```
<!DOCTYPE html>
<html>
<head>
<title>Contact Us - Scrappy-Doo Day Care Center</title>
<link rel="stylesheet" href="commonCss.css">
<style>
td{
padding:20px;
}

#message {
border: 1px solid black;
}
</style>
</head>
<body>
<div id = "containerContactUs">
<table>
<tr>
<td><label for="name">Name:</label></td>
<td><input id = "name" class = "fieldStyle" type = "text"
placeholder = "Enter Name" name = "name"></td>
</tr>
<tr>
<td><label for="email">From:</label></td>
<td><input id = "email" class = "fieldStyle" type = "email"
placeholder = "Enter email" name = "email"></td>
</tr>
</table>
<br>
<textarea id = "message" name="Text1" cols="100" rows="30"
placeholder = "Message"></textarea>
<input id = "submitButtonContactUs" type = "submit"
name = "submit">
</div>
</body>
</html>
```

## **CSS code:**

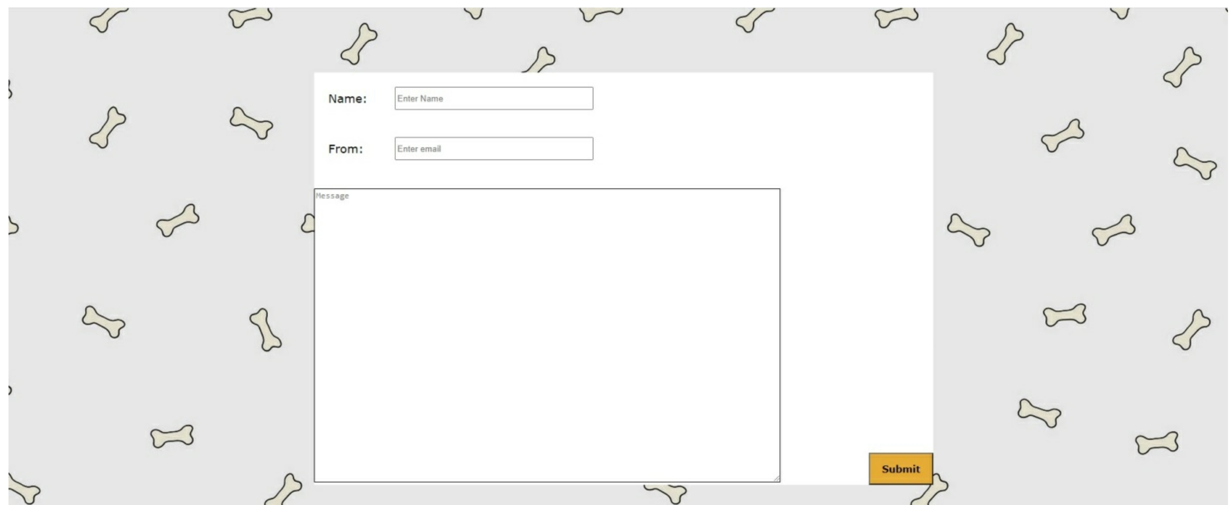
```
#containerContactUs{
    margin-top:100px;
    background-color:white;
    width:50%;
    height:auto;
    margin-left:auto;
    margin-right:auto;
    font-family:Verdana;
    font-size:18px;
    position:relative;

}

#submitButtonContactUs{
   font-family:Verdana;
   font-size:15px;
   font-weight:bold;
   background-color:orange;
   position:absolute;
   bottom:0px;
   right:0px;
   width:100px;
   height:50px;
}

#submitButtonContactUs:hover{
   background-color:#ccf2ff;
}
```

In the above HTML, we have used a new piece of code and that is:

**<textarea id = “** *message* **” name=”** *Text1* **” cols=”** *100* **” rows=”** *30* **” placeholder = “** *Message* **”> </textarea>**

**<textarea>** tags are used to define multi-line text . Its attribute **cols** is used to give width to the text area and **rows** is used to give the number of lines in the text area.

Now let’s save the HTML file and open with Google Chrome.




Now let’s create a simple ***About Us*** page of our project.

***<u>aboutUS.html</u>***

```
<!DOCTYPE html>
<html>
<head>
<title>About Us - Scrappy-Doo Day Care Center</title>
<link rel="stylesheet" href="commonCss.css">
<style>

p{
padding: 30px;
color:orange;
}

.backButton{
   font-family:Verdana;
   font-size:15px;
   position:absolute;
   bottom:0px;
   right:0px;
   width:100px;
   height:50px;

}

</style>
</head>
<body>

<div id = "container">

<p>We are the most reputed Doggy Day Care System where we provide
excellent care to your beloved pooch while you are away. </p>

<input class = "backButton" type="button" value="BACK" onclick="history.back()">
</div>

</body>
</html>
```

Save the file and open it with Google chrome

In the above HTML code,

- you will notice the text is written within **<p>** tag and **p** stands for paragraph .

- In ***About Us*** page, we created an BACK button

```
<input class = "backButton" type="button"
value="BACK" onclick="history.back()">
```

The above piece of code means when the **button** is clicked go back to the previous page

**onclick = “history.back( )”** .

- The **history** object contains the list of previous **urls** visited by the user.

- The **back( )** method loads the previous **url** in the history list.

Now we have finished creating all HTML pages of our ***Doggy Day Care Center*** project. The next step is mobile optimization, which we will do in Chapter 3.

## 2.4: Javascript Events

Javascript is an **Event Driven Programming** language and its codes will run only if some **event** occurs. There are hundreds of Javascript **events** , but the most commonly used **events** are:

1. **onsubmit** (*discussed in section 2.3)*
2. **onclick** (*discussed in section 2.1)*

3. **onchange** – As you may recall in chapter 1, section 1.2.1, while designing our ***Home Page (*** *index.html* ***)*** for the ***Doggy Day Care Center*** project, we created a drop down menu with the help of **<select>** tag. The menu contained the paths to different HTML pages given within the **<option>** tag, but the links did nothing when its values were selected. In order to make the links functional, Javascript **onchange event** is needed (*please refer to the HTML code of drop down menu present in section 1.2.1, chapter 1* ) .

```
onchange="window.location.href = this.value">
```

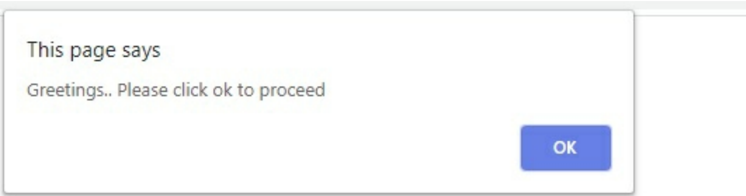
   - **window** refers to the browser window.
   - **location** object represents the **url** of a page.
   - **href** property is used to navigate to the provided **url** .
   - **this** keyword refers to the current object which is executing.

   It is the Javascript way of saying that if the value within the **<select>** field changes then trigger the Javascript **onchange event.**
   **Onchange event** will take that **url** value (*written within* ***<option>*** *tag* ) and returns that page to the user.

4. **onload – onload event** is an event which triggers when a page or an image or a frame loads. Example:

```
<!DOCTYPE html>
<html>
<head>
</head>

<body onload = "function1()">
Doggy Day Care

<script>
function function1(){
 alert("Greetings.. Please click ok to proceed")
}
</script>

</body>
</html>
```

## Output



# Chapter 3: How to Mobile Optimize a website

It is very important to make a website responsive, so that it loads perfectly in all devices.

Now let's open the ***Home page*** of our ***Doggy Day Care Center*** project and shrink the window size to check how it will appear on a mobile screen.

The above ***Home page*** not only looks messy but it is also unreadable. So for this reason it is very important to make any website responsive.

## 3.1: Three rules to Mobile Optimize a website

We can make a website responsive by following three simple rules:

**Rule 1:** Add responsive meta tag within **<head>** section of a HTML document.

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

- **viewpoint** is the visible area of a web page on any device.
- **width = device-width** means set the width of the web page to the device screen width.
- **initial-scale = 1.0** sets the initial zoom level to 1.0 when the page is first loaded.

**Rule 2:** Set the **height** and **width** of the entire web page or any section with the web page in percentage format like 100%, 80%, 50% or 20%. Try to avoid hardcoding height and width values.

**Rule 3:** Use media queries. The syntax is:

**@media (min-width :** *600px* **) {**

*…write CSS properties within this media query when the minimum width of the device is 600px….*

**}**

**@media (max-width :** *600px* **) {**

*…write CSS properties within this media query when the max width of the device is 600px….*

**}**

**NOTE:** Media query should be written within **<style>** tags, which in turn should be present within the **<head>** section of a HTML document.

Now let's apply these three rules into our web pages of ***Doggy Day Care Center project*** .

The screen shots below shows the updated CSS code of ***index.html (*** *Home page* ***)*** file which we created in Section 1.2.1 of Chapter 1

```
#homePageHeader{
margin-top:50px;
background-color:white;
text-align:right;
width:100%;
height:300px;
position:relative;
}

h1{
font-family:Kristen ITC;
font-size:50px;
position:absolute;
bottom:0px;
right:20px;
}

#logoPic{
float:left;
}
```

```
#homePageLinks{
background-color:#e68a00;
width:145px;
height:50px;
position:relative;
}

ul{
position:absolute;
text-align:center;
}

li{
display: inline;
font-family:Verdana;
}

a{
color:white;
font-weight:bold;
}

a:hover{
color:black;
}


#homePageDropdownMenu{
height:400px;
width:50%;
margin-left:auto;
margin-right:auto;
position:relative;
}

#dropDown{
position:absolute;
top:40%;
padding:20px;

}

.optionVal{
 font-size:24px;
}
```

Now let’s create the Media Queries when max width of the device is 600px:

- I would like to add some additional properties into elements with **ID** s ***logoPic*** , ***homePageLinks*** and ***dropDown*** which will get

displayed only if the max width of the device is 600px.

```
@media (max-width:600px){

 #logoPic{
 width:180px;
 height:300px;
 opacity:0.1;
 }

#homePageLinks{
margin-left:auto;
margin-right:auto;
}

#dropDown{
left:0px;
width:100%;
font-size:16px;
}

}
```

In the above piece of code, we made our logo pic with **id** ***logoPic*** transparent with the help of **opacity** property . This property is used to make an image transparent. Lesser the value more transparent the image will be.

Now let’s create the Media Query when min width of the device is 600px :

- In this, we added some properties into elements with **ID** s ***homePageLinks*** , ***dropDown*** and ***homePageDropdownMenu*** and this will get displayed only if the min width of the device is 600px.

```
@media (min-width:600px){

#homePageLinks{
float:right;
margin-right:50px;
}

#dropDown{
left:25%;
width:500px;
font-size:24px;
}

#homePageDropdownMenu{
margin-top:50px;
background-color:#bfbfbf;
border: 15px solid white;
}
}
```

Now let’s save everything and open the HTML file (***index.html*** ) with Google Chrome.

The ***home page*** is responsive and it looks perfect.

**Please Note:** Chrome developer tools can also be used to simulate page views in mobile device. More details can be found here- https://developers .google.com/web/tools/chrome-devtools/device-mode

# Chapter 4: Introduction to Python Programming.

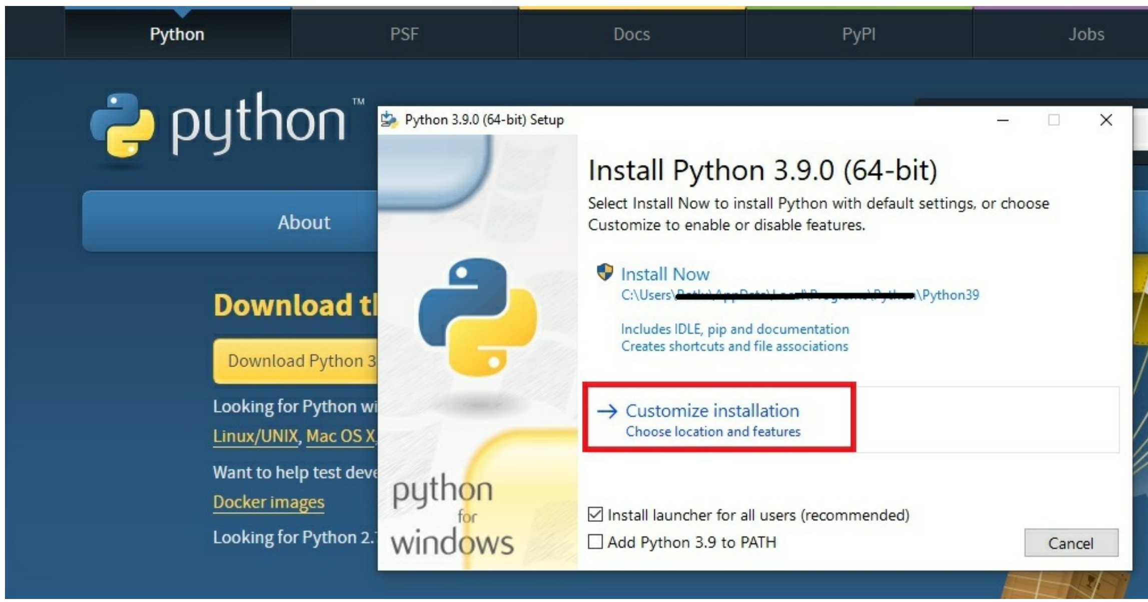
Python is the most popular and most widely used **scripting language** . Its syntax is easy to use and is written to make coding easy.

## 4.1: Python Installation

**Step 1:** Go to https://www.python.org/downloads/ and download the latest Python version available.

**Step 2:** Click on Customize installation.

For your Python download, it is important to give your own desirable path location which is short and easy to remember. This is because we will be using this path location again for **Django installation** .




**Step 3:** Click Next

**Step 4:** Then browse for your install location.

Python 3.9.0 (64-bit) Setup

Advanced Options

- [ ] Install for all users
- [x] Associate files with Python (requires the py launcher)
- [x] Create shortcuts for installed applications
- [ ] Add Python to environment variables
- [ ] Precompile standard library
- [ ] Download debugging symbols
- [ ] Download debug binaries (requires VS 2017 or later)

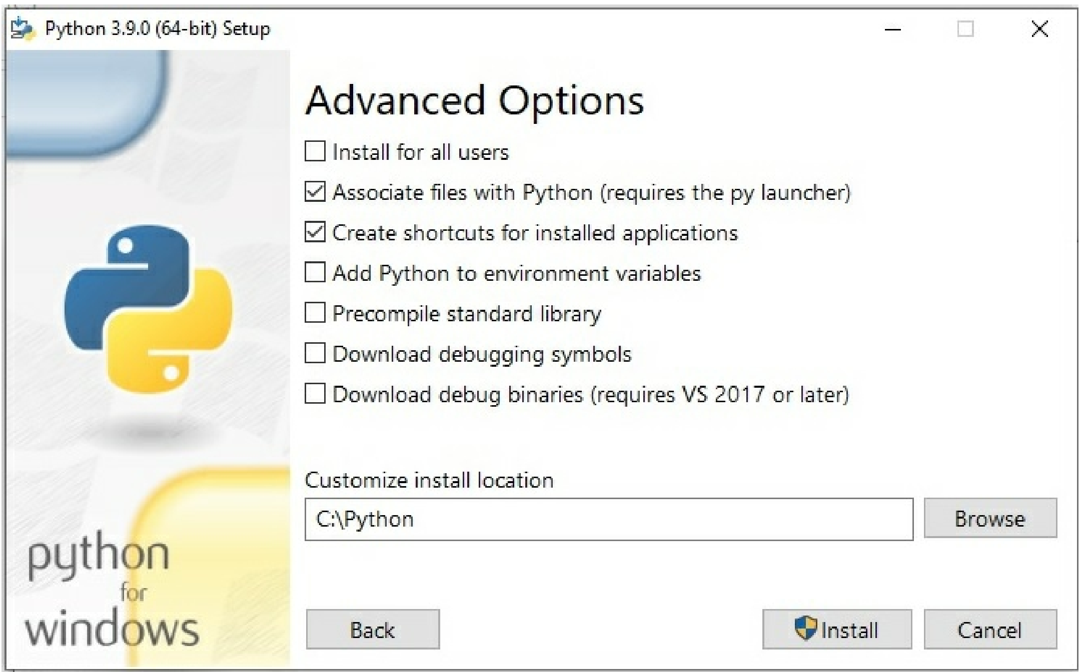
Customize install location

C:\Users\ \Python\Python39 Browse

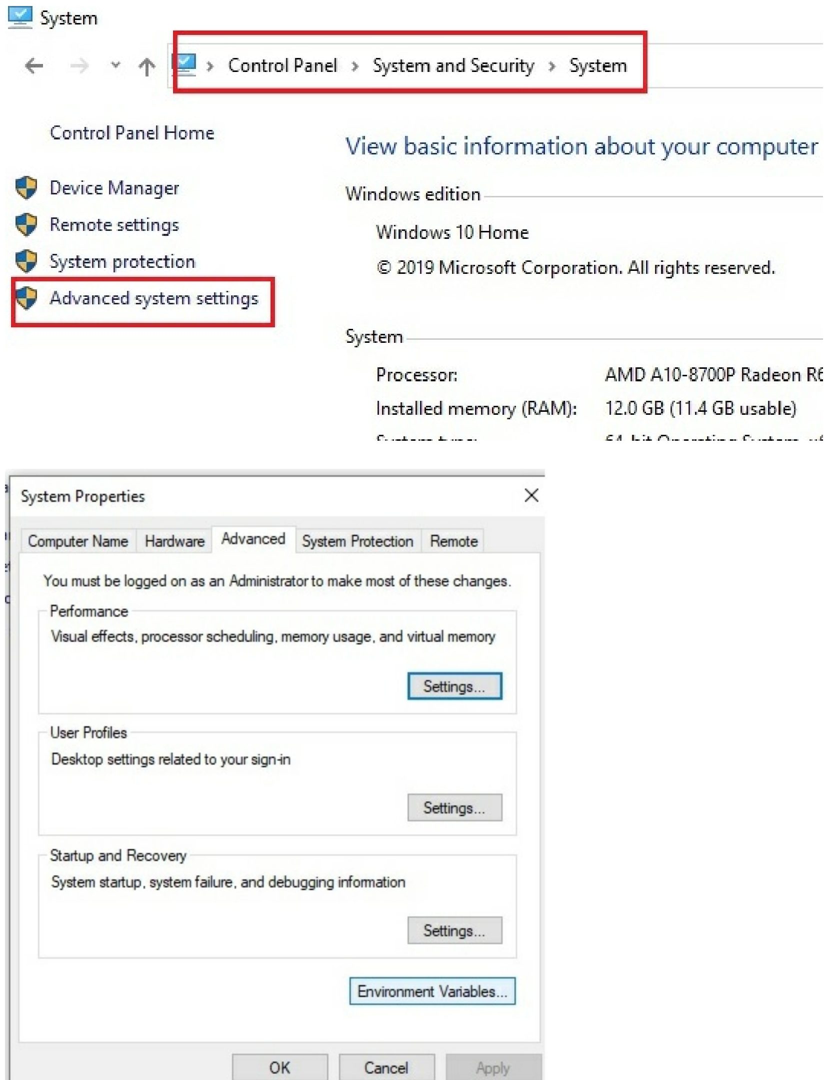
Back Install Cancel

python for windows

**Step 5: C** lick install




**Step 6:** Now go to **Control panel** -> **System and Security** -> **System** -> **Advanced system settings** -> **Environment Variables** -> Add Python Installation location into your PATH variables.

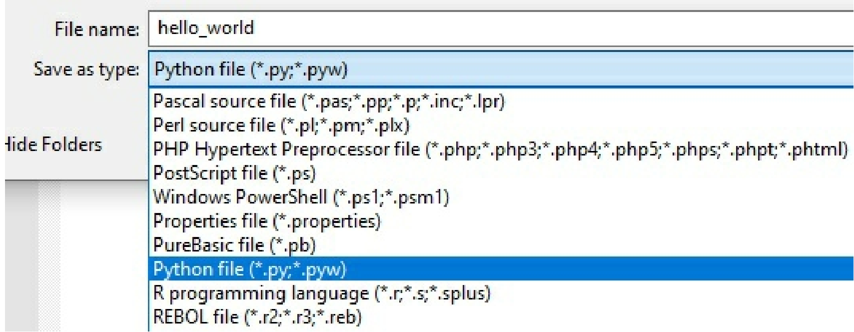



**Step 7:** Open command prompt and with the help of **python** command check whether installation is done correctly or not.

```
C:\>python
Python 3.9.0 (tags/v3.9.0:9cf6752, Oct  5 2020, 15:34:40) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> _
```

Now let's code..

Open **Notepad++** and create a new file (*hello_world* ) and save the file with **.py** extension




Inside ***hello_world.py file*** , write one line of code:

```
1   print("Hello World")
```

**What is print( ) function in Python?**

**Print( )** function displays the output to the screen

Now let’s run the Python file.

Open command prompt -> Go the ***hello_world.py*** file location and write the following command:

**python** *file_name* **.py**

```
C:\>cd C:\Users\[redacted]\Python_script

C:\Users\[redacted]\Python_script>python hello_world.py
Hello World

C:\Users\[redacted]\Python_script>_
```

# 4.2 : Python variables, Datatypes & Operators

Important points to note are:

- Python variables are used to hold data.

- In most programming languages, you need to declare the data type of a variable. For example: **int** x = 5, **String** y = “John” etc, where **int** refers to an integer value and **String** refers to a sequence of character value.

But in Python, you do not have to declare the data type of a variable. Python understands the data type of any variable simply from its value.

Let us consider the example below:

```
1  x = 5
2  y = 5.5
3  z = "Hello World"
4  print(type(x))
5  print(type(y))
6  print(type(z))
```

**Output**

```
C:\Users\[redacted]\Python_script>variables.py
<class 'int'>
<class 'float'>
<class 'str'>
```

When we run the above piece of code, Python shows the data types of variable ***x*** , ***y*** and ***z*** .

**What is type( ) function?**

**type( )** function is used to determine the data type of a variable.

Apart from data types like **int** , **float** and **string** , Python also has **list** , **tuple** and **dictionaries** .

## 4.2.1 : Python String

Important points to note are:

- Python string is a sequence of characters.
- Python string is immutable meaning that once a string is created it cannot be modified.

**What is String Slicing in Python?**

Python String slicing is the process of obtaining a sub string of the given string.

Syntax:

string **[** index_number_start *(including its value)* **:** index_number_stop *(excluding its value)* **]**

Example:

Open **Notepad++** and create a new python file and write the following lines of code.

```
1  message = "Health is wealth"
2  print(message[2:12])
3  print(message[:6])
4  print(message[7:])
5  print(message[-1])
6  print(message[-5])
```

Now run the above piece of code:

```
C:\Users\[redacted]\Desktop\Project >python test.py
alth is we
Health
is wealth
h
e
```

**Code explanation:**

Let's look into the message *"Health is wealth"*.

Index number always starts from 0. Letter *H* is present at index number 0, letter *e* is present at index number 1, letter *a* is present at index number 2, letter *l* is present at index number 3, letter *t* is present at index number 4, letter *h* is present at index number 5. Then whitespace is present at index number 6. Letter *i* is present at index number 7, letter *s* is present at index number 8. Then again whitespace is present at index number 9. Letter *w* is present at index number 10, Letter *e* is present at index number 11, Letter *a* is present at index number 12, Letter *l* is present at index number 13, Letter *t* is present at index number 14, Letter *h* is present at index number 15.

In the above piece of code, the output of ***message[2:12]*** is : print values from 2 *(including its value at index position 2)* to 12 *(excluding its value at index position 12)* **alth is we**

The output of ***message[:6]*** is : print values from start *(index position 0, including its value)* to 6 *(excluding its value at index position 6)* **Health**

The output of ***message[7:]*** is : print values from 7 *(including its value at index position 7)* to end **is wealth**

To print letters or values from the **end** of a string, minus sign is used. For example: ***message[-1]*** will return **h** , ***message[-2]*** will return **t** and so on.

**Important and commonly used String methods**

1. **len( )** function returns the length of the string.

Syntax: len(string)

2. **lower( )** function returns the string with lower case characters.

Syntax: string.lower()

3. **upper( )** function returns the string with upper case characters.

Syntax: string.upper()

4. **strip( )** function gets rid of left or right whitespace.

Syntax: string.strip()

5. **lstrip( )** function gets rid of left whitespace.

Syntax: string.lstrip()

6. **rstrip( )** function gets rid of right whitespace.

Syntax: string.rstrip()

7. **index( )** function returns the index number of the given element.

Syntax: string.index(*element* )

```
car = "kia toyota"

index_num = car.index("t")

print(index_num)
```

**Output:**

```
C:\Users\...\Desktop\Project >python test.py
4
```

8. **split( )** function returns the **list** of substrings separated by whitespace.

Syntax: string.split( )

```
fruit = "orange banana pear"
fruit = fruit.split()
print(fruit)
```

**Output:**

```
C:\Users\...\Desktop\Project ..>python test.py
['orange', 'banana', 'pear']
```

9. **replace(old,new)** function replaces the old with new.

Syntax: string.replace(old,new)

```
message = "apples are tasty"
message = message.replace("tasty","yummy")
print(message)
```

**Output:**

```
C:\Users\...\Desktop\Project ..>python test.py
apples are yummy
```

10. **join( )** string method returns a string by joining all the elements of an iterable (lists, tuples and string) with the delimiter.

Syntax: delimiter.join(iterable)

```
fruit = "orange"
fruit = "#".join(fruit)
print(fruit)
```

**Output:**

```
C:\Users\. _ ' _\Desktop\Project  `>python test.py
o#r#a#n#g#e
```

## String formatting

**format( )** method is used to format a string.

**Rules for string formatting:**

1. Create a placeholder with the help of curly brackets {}.
2. Call the **format** method and pass variables as parameters. When program executes, the values passed to these parameters will replace the curly brackets placeholders.

**Example 1:**

```
animal = "dog"

legs = 4

print("{} has {} legs".format(animal,legs))
```

In the above example, the values passed to variable ***animal*** will fill the first placeholder and the value passed to variable ***legs*** will fill the second placeholder.

```
print("{} has {} legs".format(animal,legs))
```

first placeholder    second placeholder

**Output:**

```
C:\Users\¯˙ '¸\Desktop\Project ,>python test1.py
dog has 4 legs
```

**Example 2:**

If you want to display a float number with two digits after the decimal dot, formatting expression **{:.2f}** is used.

```
salary = 537.7476
name = "John"
print("{}: ${:.2f}".format(name,salary))
```

**Output:**

```
C:\Users\¯˙ '¸\Desktop\Project `>python test1.py
John: $537.75
```

## 4.2.2 : Python List

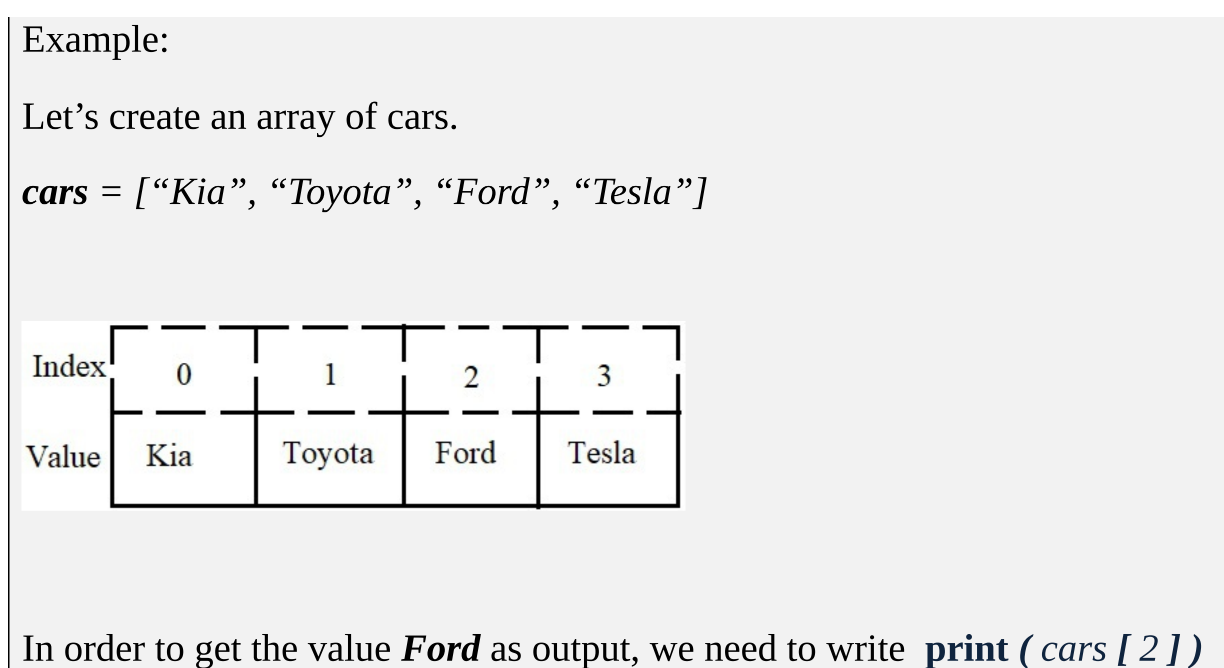
- A Python **List** is very similar to an **Array** . It contains a list of elements separated by commas and its elements are written with square brackets **[** …. **]** .
- Python list are mutable meaning that we can modify list elements.
- A value from a **List** can be accessed from its index value.

  Syntax for accessing the values from a List is:

  *List_name* **[** *index_num* **]**

**What is an Array?**

An Array is a collection of items or elements all having the same datatype. An element from an array can only be accessed from its index value.



Example:

Let’s create an array of cars.

***cars** = [“Kia”, “Toyota”, “Ford”, “Tesla”]*

| Index | 0 | 1 | 2 | 3 |
| --- | --- | --- | --- | --- |
| Value | Kia | Toyota | Ford | Tesla |

In order to get the value ***Ford*** as output, we need to write **print *( cars [ 2 ] )***

**Example:**

Create a new Python file (***variables.py*** ) and write the following lines of code:

```
1  fruits  =  ["apple","orange","banana","mango","strawberry","peaches"]
2  x  =  "this apple is tasty"
3  print(len(fruits))
4  print(fruits[2:4])
5  print(x[2:8])
```

After running the above piece of code, we get an output of:

```
C:\Users\[redacted]\Desktop\Python_script>variables.py
6
['banana', 'mango']
is app
```

**Code explanation:**

- In the above piece of code, ***fruits*** is a **List** containing 6 elements. The value at position 0 is “*apple* ”, at position 1 is “*orange* ” and so on.

- ***x*** is a variable holding a string value. The value at position 0 is "*t*" , at position 1 is "*h* " and so on.
- **len( )** function is used to get the length of the list .
- In Line 4 and 5 we are performing **Slicing operation** .
    - ***print ( fruits [ 2 : 4] )*** means print elements from position 2 (*including its value* ) to position 4 (*excluding its value* ).

```
                 0          1          2        3          4            5
fruits  =  ["apple","orange","banana","mango","strawberry","peaches"]
```

**Output: *['banana', 'mango']***

- ***print ( x [ 2 : 8 ] )*** means print the values of ***x*** ("*this apple is tasty* ") from position 2 (*including its value* ) to position 8 (*excluding its value* ).

```
      0 1 2 3 4 5 6 7 8 9 ..........................
x = this apple is tasty
```

**Output: *is app***

**<u>Important and commonly used List methods</u>**

**1. append( )** method inserts the element at the end of the list.

Syntax: list.append(*element* )

```
animals = ["lion", "monkey"]
animals.append("tiger")
print(animals)
```

**Output:**

```
C:\Users\.....\Desktop\Project :>python test.py
['lion', 'monkey', 'tiger']
```

**2. insert( )** method inserts an element at the specified index number.

Syntax: list.index(*index_num* , *element* )

```
fruits = ["apple","orange"]
fruits.insert(1,"banana")
print(fruits)
```

**Output:**

```
C:\Users\. . ' .\Desktop\Project ^>python test.py
['apple', 'banana', 'orange']
```

**3. remove( ) method** removes the first occurrence of element specified.

Syntax: list.remove(*element* )

```
planets = ["earth","mars","pluto"]
planets.remove("pluto")
print(planets)
```

**Output:**

```
C:\Users\. . ' .\Desktop\Project ^>python test.py
['earth', 'mars']
```

## 4.2.3 : Python Tuple

- A **Tuple** is very much like a **List** . It contains elements separated by commas within open and close parenthesis **( …. )** .
- Python tuple are immutable meaning once a tuple is declared, it cannot be modified.

### Difference between Tuple and List

| | |
|---|---|

| Tuple | List |
|---|---|
| Tuple contains elements within (… ) parenthesis | List contains elements within [ …] square brackets |
| Once a tuple is created, no item can be added, updated or deleted | List gives us the ability to add, update and delete items. |

**Example:**

```
1  fruits  =  ["apple","orange","banana","mango","strawberry","peaches"]
2  car  =  ("toyota","ford","volvo","tesla")
3  del(fruits[1])
4  print(fruits)
5  del(car[2])
6  print(car)
```

After running the above piece of code, we get an output of:

```
C:\Users\[redacted]\Python_script>variables.py
['apple', 'banana', 'mango', 'strawberry', 'peaches']
Traceback (most recent call last):
  File "C:\Users\[redacted]\Desktop\Python_script\variables.py", line 5, in <module>
    del(car[2])
TypeError: 'tuple' object doesn't support item deletion
```

**Code explanation:**

**del ( ) function** is used to delete element from a certain position. When applying delete function to ***fruits* list** , the value at position 1 (*orange* ) was deleted successfully. But in case of **tuple** , when we tried to delete an element at position 2, it threw an error.

**What is unpacking a tuple mean?**

When we create a tuple and assign new values into it. This is called packing a tuple.

When we extract those values and store them into a variable. This is called unpacking a tuple.

```
cars  =  ("toyota","tesla")
first_car,second_car  =  cars
print("{} is expensive".format(second_car))
print("I like {}".format(first_car))
```

**Output:**

```
C:\Users\. _. ` ,\Desktop\Project `>python test1.py
tesla is expensive
I like toyota
```

## 4.2.4 : Python Dictionary

- Python dictionaries are used to store **key - value** pairs where the key should be a unique value.
- A dictionary is written within { … } curly braces.
- Python dictionary are mutable meaning that its content can be modified.

**Example:**

```
1  score  =  {
2  "John"  :  95,
3  "Rita"  :  90,
4  "Sam"  :  80
5  }
6  print("John scored",  score['John'])
```

After running the above piece of code, we get an output of:

```
C:\Users\[redacted]\Python_script>variables.py
John scored 95
```

## Important Dictionary operations and methods

1. Now let’s **insert** some new records.

To insert new records into the dictionary, the syntax is:

*dictionary_name* **[** *key* **]** = *value*

```
score = {
"John" : 95,
"Rita" : 90,
"Sam" : 80
}
score["Ram"] = 96
score["Sid"] = 82
print(score)
```

**Output:**

```
C:\Users\ \Desktop\Project .>python test1.py
{'John': 95, 'Rita': 90, 'Sam': 80, 'Ram': 96, 'Sid': 82}
```

2. **Update a record**

To update a record, the syntax is same as insert a new record.

```
score  =  {
"John"  :  95,
"Rita"  :  90,
"Sam"  :  80
}
score["John"]  =  70
print(score["John"])
```

**Output:**

```
C:\Users\~` '_\Desktop\Project `>python test1.py
70
```

In the above piece of code, we see the new value (*70* ) assigned to key ***John*** replaces the old value (*95* ).

### 3. Delete a record

To delete a record from a dictionary, **del** keyword is used. Syntax:

**del** *dictionary_name* **[** *key* **]**

**Important Dictionary Methods:**

1. To iterate through dictionary, **items( )** method is used. **Syntax: dictionary.items( )** *(example present in chapter 5* )

2. To display only dictionary keys, **key( )** method is used.

**Syntax: dictionary.keys( )**

3. To display only dictionary values, **values( )** method is used.

**Syntax: dictionary.values( ).**

## 4.2.5 : Python Operators

**Commonly used Arithmetic operators**

| Operator | Description |
|---|---|
| + | Addition |
| - | Subtraction |
| * | Multiplication |
| / | Division |
| % | Modulus. This sign returns the remainder. |
| ** | Exponentiation. For example: 2 ** 3 means 2 to the power of 3 and that is equal to 8 |
| // | Floor division. This operator returns only the whole integer number. For example: 23 // 2 will give 11 |

**Commonly used Comparison Operators**

| Operator | Description |
|---|---|
| = = | Equals: x = = y |
| != | Not Equals: x != y |
| < | Less than: x < y |
| <= | Less than or equal to: x <= y |
| > | Greater than: x > y |
| >= | Greater than or equal to: x >= y |

**Commonly used Logical Operators**

| Operator | Description |
|---|---|
| and | Example:<br>x = 123 **and** y = “John” executes certain block of code if **both** statements are TRUE |
| or | Example:<br>x = 123 **or** y = “John” executes certain block of code if **one** of the statements is TRUE |
| in | Example:<br>x = [ 123 , 345 ]<br>if 123 **in** x returns TRUE, then executes certain block of code |
| not in | Example:<br>x = [ 123 , 345 ]<br>if 12 **not in** x returns TRUE, then executes certain block of code |

**Commonly used Assignment Operators**

| Operator | Description |
|---|---|
| = | Example: x = 2, this means value 2 is assigned to x |
| +=<br>(x += 1) | This is same as x = x + 1. If value of x = 3, then the new value of x is 3 + 1 = 4 |
| -=<br>(x -= 2) | This is same as x = x – 2. If value of x = 5, then the new value of x is 5 – 2 = 3 |
| *=<br>(x *= 5) | This is same as x = x * 5. If value of x = 2, then the new value of x is 2 * 5 = 10 |
| /=<br>(x /= 2) | This is same as x = x / 2. If value of x is 10, then the new value of x is 10 / 2 = 5 |
| %=<br>(x %= 2) | This is same as x = x % 2. If value of x is 10, then the new value of x is 10 % 2 = 0 |

# 4.3 : Python Control Statements

The concept of control statements in Python is similar to any other programming languages but it has some minor differences. They are:

1. The syntax is different.
2. In other programming languages, the control statements block of codes is written within curly braces { …}, whereas in Python, indentation marks the beginning and end of a block of code.

## 4.3.1 : Python If ..Elif ..Else

- **if else** block of code is used to check whether certain condition is true or false. If the condition is satisfied the **if** block of code is executed and if not satisfied the **else** block of code is executed.
- Python **elif** means else if

The syntax is:

**if** *condition* **:**

**……..**

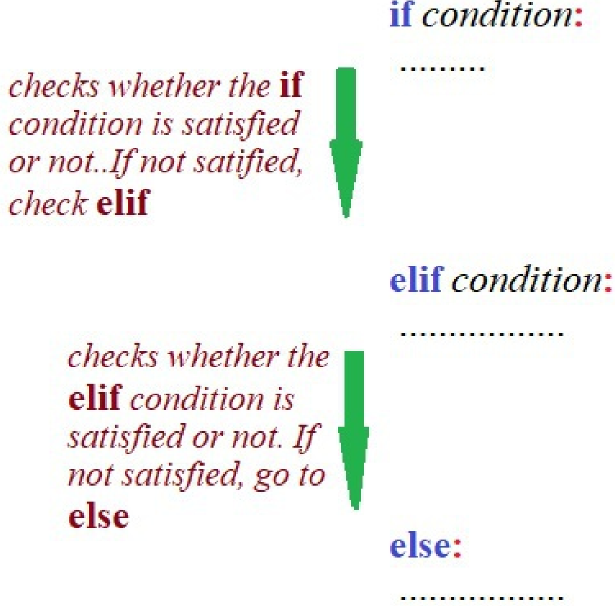
**elif** *condition* **:**

**……..**

**else :**

**…………**

**Execution flow:**




**Example 1:**

Create a new Python file (***test.py*** ) and write the following lines of code:

```
1  fruits = ["banana","orange","apple","grapes"]
2  if fruits[1]=="orange":
3    print("Output is Orange")
4  elif fruits[2] == "apple":
5    print("Output is apple")
6  else:
7    print("could not find orange or apple")
```

**Python Indentation Explanation:**

- colon( **: )** marks the beginning of the block of codes.

- You will notice that the above piece of code does not contain any curly brackets. In Python indentation marks the beginning and end of a block of code.

- In the above example, after stating the **if** line of code (**line 2** ), we gave an indentation and wrote the line ***print("Output is Orange")*** . This indentation signifies that the **print** statement belongs to the **if** block of code. Then after stating the **elif** line of code (**line 4** ), we gave an indentation and wrote the line ***print("Output is apple")*** . This one indentation again signifies that the **print** statement belongs to the **elif** block of code. Then again we followed the same process for **else** block of code.

- Once the **if** condition is satisfied, the line ***print("Output is Orange")*** executes. If for some reason, the **if** condition is not satisfied, then **elif** condition is tested. If the **elif** condition is also not satisfied then **else** block of code executes.

Now let's run the above piece of code.

Open command prompt -> navigate to the Python file location and run the script using **python** command.

**Output**

```
C:\Users\[redacted]\Desktop\Python_script\PYTHON>python test.py
Output is Orange
```

In the above Python code, ***fruits*** is a **List** of 4 elements**.** Since the value at position 1 is indeed "*orange* ", the **if** statement is satisfied and the **elif** and **else** block of code are not checked anymore.

**Example 2:**

```
1  stocks = ["Walmart", "Apple", "NIO", "Tesla"]
2  if stocks:
3    print(stocks)
4  else:
5    print("hello")
```

**Code explanation:**

The **Line 2** of the above piece of code checks whether there is value present in ***stocks*** **List** or not. If TRUE, execute the **if** block of code, else execute the **else** block of code.

**Output**

```
C:\Users\[redacted]\Desktop\Python_script\PYTHON>python test.py
['Walmart', 'Apple', 'NIO', 'Tesla']
```

## 4.3.2 : Python For Loop

Important points to note are :

- Python **for** loop is used to iterate through an **array** , **list** , **tuple** , **dictionary** or **sequence of characters** .

- **for** loop continues till the length of a given variable. Example:

*cars = [“Kia”, “Toyota”, “Ford”]*

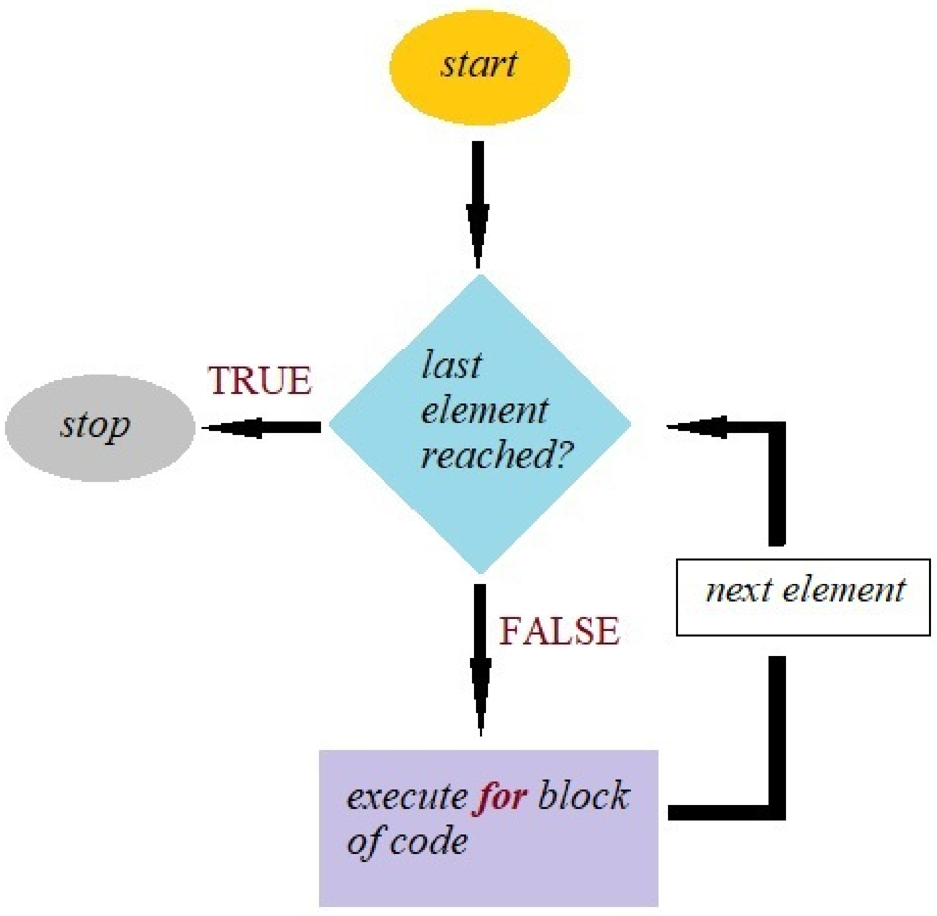
The length of the above ***cars*** **list** is 3, so the loop will go on for 3 times. First it will take the element at index number 0 (“*Kia* ”) and perform the checking operation by traversing down the loop -> then it will check the element at index number 1 (“*Toyota* ”) and perform the checking operation by traversing down the loop -> then it will check the element at index number 2 (“*Ford* ” ).

The syntax is:

**for** *each_element* **in** *elements* **:**

**………**

**Execution flow of for loop:**




**Example:**

Create a new Python file (***loops.py*** ) and write the following lines of code:

```
car = ["toyota","ford","tesla","truck"]
for x in car:
  if x == "tesla":
    print("I wish to own tesla one day")
  elif x == "truck":
    print("But trucks are the best")
  else:
    print("I am fine with any car")
```

After running the above piece of code, we get an output of:

```
C:\Users\[redacted]\Python_script>loops.py
I am fine with any car
I am fine with any car
I wish to own tesla one day
But trucks are the best
```

**Important points to note are:**

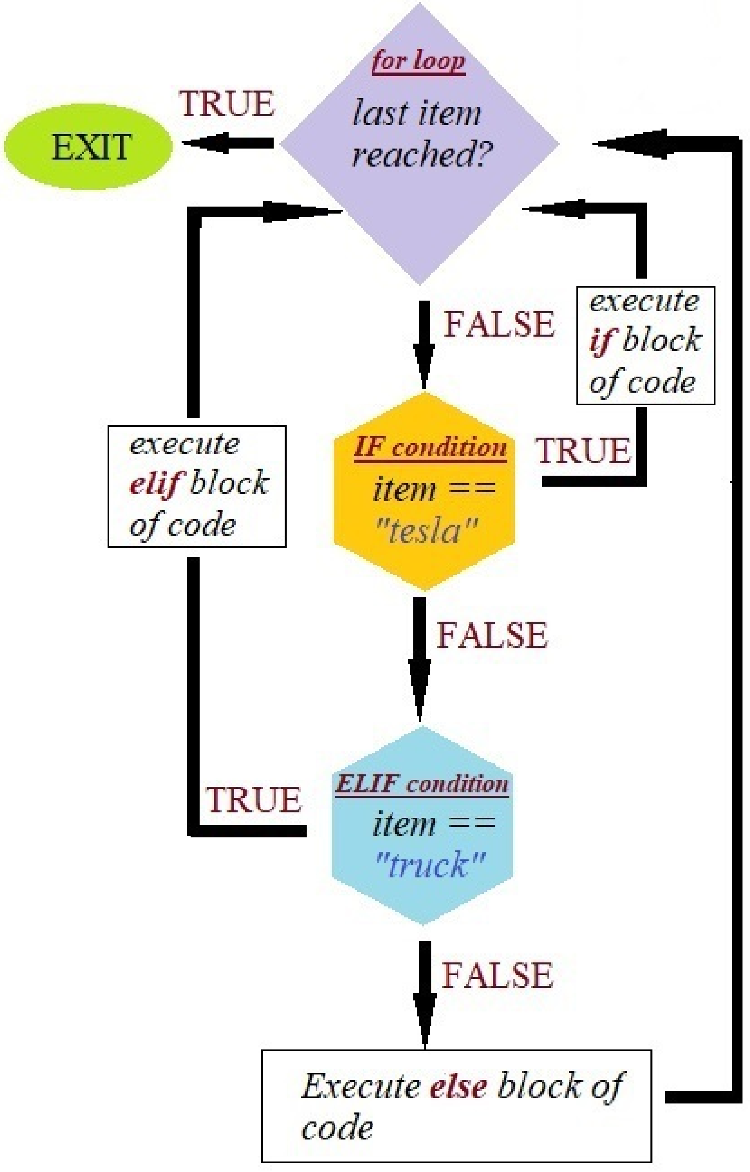
1. colon( **:** ) marks the beginning of the block of codes.
2. After initiating the **for** loop, we gave an indentation and then started our **if** loop. That one indentation signifies that the **if** block of code belongs to the **for** loop.

Now within **if** block of code, I gave two indentation to signify that the line *print("I wish to own tesla one day")* belongs to **if** statement which in turn belongs to the **for** loop.

We followed the same above process for **elif** and **else** block of codes.

3. The loop will go on till the length of the **List** ***car*** . The length of the above **List** is 4, so the loop will happen 4 times.

**Execution flow of the above piece of code:**




## Nested for loop

**Nested for** loop means that a **for** loop is present within a **for** loop.

```
groceries = [["apple","orange"],["potato","onion"]]
for record in groceries:
  result = ""
  get_index = groceries.index(record)
  for grocery_item in record:
    result = result + grocery_item + " "
  print("Grocery items at index {} : {}".format(str(get_index),result.strip()))
```

**Output:**

```
C:\Users\...\Desktop\Project >python test.py
Grocery items at index 0 : apple orange
Grocery items at index 1 : potato onion
```

**Code explanation:**

In the above piece of code we created a list of groceries and stored it in a variable ***groceries*** . The list is a collection of mini lists.

In line ***for record in groceries:*** , we capture the mini lists. Then in line ***for grocery_item in record:*** we loop and get the grocery items present in the mini list.

## 4.3.3 : Python While loop

**while** loop keeps on executing a block of code as long as the condition is true.

The syntax is :

**while** *condition* **:**

**……..**

**Example:**

```
1  x = 10
2  while x<=100:
3    print(x)
4    x=x+10
```

**Code explanation:**

- In **Line 1** , we declared the value of variable ***x*** .
- In **Line 2** , the **while** loop is initiated.
  Since the value of ***x*** is less than 100, then the **while** loop condition is satisfied and the **print** statement at **Line 3** executes.
- In **Line 4** , we increment the value of ***x*** by 10 and assign that increment value back to variable ***x*** . The **while** loop starts again with the incremented value and goes on unless and until the value of ***x*** becomes equal to or less than 100.

Now let's run above piece of code.

```
C:\Users\[redacted]\Desktop\Python_script\PYTHON>python test.py
10
20
30
40
50
60
70
80
90
100
```

**NOTE:** In while loop it is very important to initialize a variable and then increment the variable as we did in the above example. If proper initialization and incrementation is not done, the while loop will keep on looping and will not stop.

## 4.3.4 : Python Break and Continue

Python **Break** and **Continue** statements are used to alter the normal flow of a loop.

### Break

**Break** statements are used to break out of a loop if certain condition is satisfied.

**Example:**

```
#created LIST of stocks
stocks = ["Walmart", "Apple", "NIO", "Tesla"]
#initiated FOR loop
for s in stocks:
# IF s equals Apple, print 'Apple is the best stock' and
# break out of the loop
  if s == "Apple":
    print("Apple is the best stock")
    break
  else:
    print("Trading stocks")
```

In the above example, the **for** loop will go on until the **if** condition is satisfied. Once the **if** condition is satisfied the execution will stop because it will encounter the **break** statement.

Now let’s run the above piece of code,

```
C:\Users\[redacted]\Desktop\Python_script\PYTHON>python test.py
Trading stocks
Apple is the best stock
```

## Continue

When a certain condition executes the **Continue** statement, the control returns to the beginning of the loop and skips the rest of the code.

Note that the loop does not get terminated.

**Example:**

```
1  #created LIST of stocks
2  stocks = ["Apple", "Tesla", "Walmart", "NIO"]
3  #initiated FOR loop
4  for s in stocks:
5  # IF s equals Tesla, stop printing the lines below and
6  # go back to the for loop
7      if s == "Tesla":
8          continue
9      print("hello ",s)
```

In the above piece of code, you will notice that once the **continue** statement was encountered, the code at **Line 9** was not printed and the control moved back to the beginning of the loop.

Now let’s run the above piece of code,

```
C:\Users\[redacted]\Desktop\Python_script\PYTHON>python test.py
hello  Apple
hello  Walmart
hello  NIO
```

Except “*Tesla*” , all other values were printed.

# 4.4 : Python functions.

Functions are block of codes performing certain task. The syntax for declaring a function is:

**def** *function_name* **( ) :**

**……..**

```
1  username = "John123"
2  def validate_username(uname):
3    if uname == "John123":
4      print("Hello John")
5    else:
6      print("Wrong username")
7  validate_username(username)
```

Code Explanation:

- In the above piece of code, we created a **function** ***validate_username*** with parameter ***uname.*** If the argument is "*John123* ", the **function** prints "*Hello John* " or else it will print "*Wrong username* ".

- To call any function, we simply have to write the function name followed by parenthesis. Within that parenthesis, we passed the value of ***username*** which is "*John123* "

**Output**

```
C:\Users\[redacted]\Python_script>functions.py
Hello John
```

# 4.5 : Python class

- Python is an **object oriented programming language** .
- A **class** is a "blueprint" for creating an **object** and like any other programming language Python **class** contains attributes and methods.

- The syntax for creating a class is:

**class** *class_name* **:**
**def __init__(self) :**
**….**
**def** *method1* **:**
**….**

**What is def __init__ (self)?**

**"__init__"** is a reserved method in **Python** classes initializing the **object** 's state. In Object Oriented concepts, it refers to the **constructor** of the **class** . This method initializes the attributes of a **class** when an **object** is created.

```
1  class Employee:
2    def __init__(self,id_num,name,age,address,salary):
3      self.id_num = id_num
4      self.name = name
5      self.age = age
6      self.address = address
7      self.salary = salary
8    def validation(self):
9      if (str(self.id_num) == "123" and self.name =="John"):
10       print("This is Jon and his age is",self.age,
11       ",his address is",self.address,
12       "and his salary is", self.salary)
13     else:
14       print("Not John")
15 e = Employee(123,"John",25,"Texas",1000)
16 e.validation()
```

Output:

```
C:\Users\[redacted]\Python_script>classes.py
This is Jon and his age is 25 ,his address is Texas and his salary is 1000
```

**Important points to note from the above piece of code are:**

1. A Python **Class** name should start with a capital letter.
2. Within the **__init__**method we declared different attributes of our **class *Employee*** and passed those values to ***self . id_num*** , ***self . name*** and so on.
3. In our ***validation ( )*** method we pass the argument **self** which helps to access all the attributes of the **class *Employee*** .
4. **str( )** function helps to convert an integer value to a string value.
5. **and** keyword is the **logical and operator** in Python.
6. In line 15 of our above code, we declared an **object *e*** and passed values into it.

**Please note:** In this chapter, I have covered only few important Python topics essential for the development of our Django project. To gain in-depth and through knowledge of all Python topics, please visit website https://www.w3schools.com/

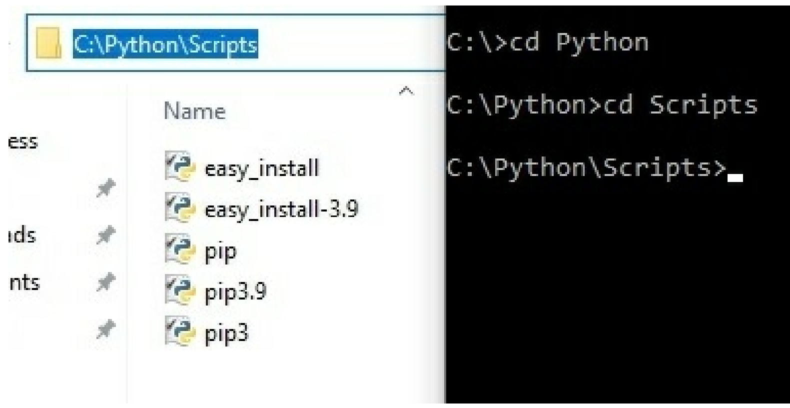
# Chapter 5: Django

Django is a free and open source **high-level Python web framework** that helps in the development of a website.

## 5.1 : Django Installation

**Step 1:** Open command prompt -> Go to the file location where **Python** folder is present (*Python installed in Chapter 4, section 4.1* ) -> Go to **Scripts** folder

*My Python folder is present in C drive. C drive -> Python folder -> Script.*




**Step 2:** Check for the presence of **pip** application into the **Scripts** folder. Let's display the **pip** version by using command **pip –version**

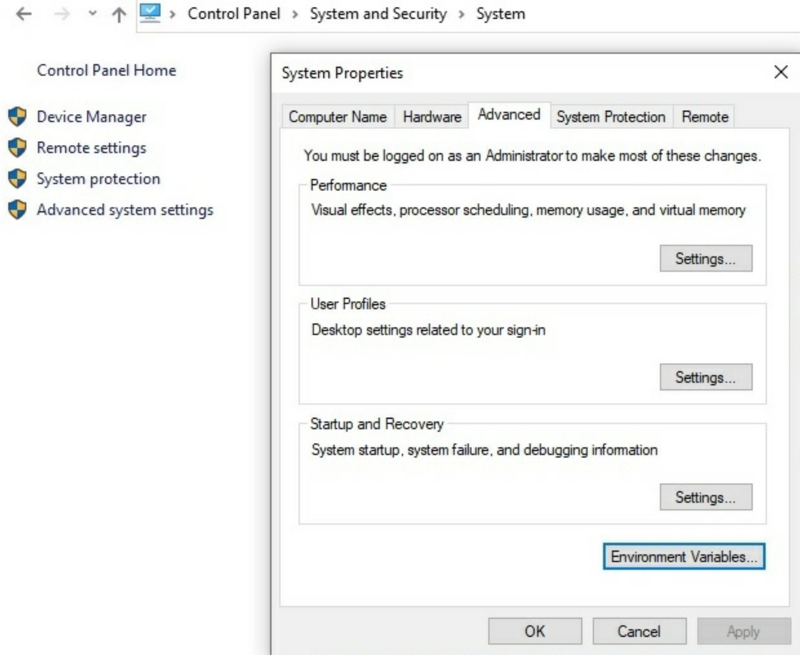
```
C:\Python\Scripts>pip --version
pip 20.2.3 from c:\python\lib\site-packages\pip (python 3.9)
```

**Step 3:** To install Django, type the command **py –m pip install Django**

```
C:\Python\Scripts>py -m pip install Django
Collecting Django
  Downloading Django-3.1.2-py3-none-any.whl (7.8 MB)
     |████████████████████████████████| 7.8 MB 311 kB/s
Collecting pytz
  Downloading pytz-2020.1-py2.py3-none-any.whl (510 kB)
     |████████████████████████████████| 510 kB 504 kB/s
Collecting sqlparse>=0.2.2
  Downloading sqlparse-0.4.1-py3-none-any.whl (42 kB)
     |████████████████████████████████| 42 kB 86 kB/s
Collecting asgiref~=3.2.10
  Downloading asgiref-3.2.10-py3-none-any.whl (19 kB)
Installing collected packages: pytz, sqlparse, asgiref, Django
  WARNING: The script sqlformat.exe is installed in 'C:\Python\Scripts' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
  WARNING: The script django-admin.exe is installed in 'C:\Python\Scripts' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
```

During the installation process, I encountered a warning saying *" Consider adding this directory to PATH "* . So let's fix this warning..

Go to **Environment Variables** -> Now add the **pip** file location into PATH variables.



```
C:\Python\Scripts>py -m pip install Django
Collecting Django
  Downloading Django-3.1.2-py3-none-any.whl (7.8 MB)
     |████████████████████████████████| 7.8 MB 311 kB/s
Collecting pytz
  Downloading pytz-2020.1-py2.py3-none-any.whl (510 kB)
     |████████████████████████████████| 510 kB 504 kB/s
Collecting sqlparse>=0.2.2
  Downloading sqlparse-0.4.1-py3-none-any.whl (42 kB)
     |████████████████████████████████| 42 kB 86 kB/s
Collecting asgiref~=3.2.10
  Downloading asgiref-3.2.10-py3-none-any.whl (19 kB)
Installing collected packages: pytz, sqlparse, asgiref, Django
  WARNING: The script sqlformat.exe is installed in 'C:\Python\Scripts' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
  WARNING: The script django-admin.exe is installed in 'C:\Python\Scripts' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
Successfully installed Django-3.1.2 asgiref-3.2.10 pytz-2020.1 sqlparse-0.4.1
```

Now it says *"Successfully installed Django"*

**Step 4:** Now let's set our **virtual environment** . To do that let's follow the steps from Django Documentation: **https://docs.djangoproject.com/en/3.1/howto/windows/**

To create a **virtual environment** Django documentation says the following:

To create a virtual environment for your project, open a new command prompt, navigate to the folder where you want to create your project and then enter the following:

```
...\> py -m venv project-name
```

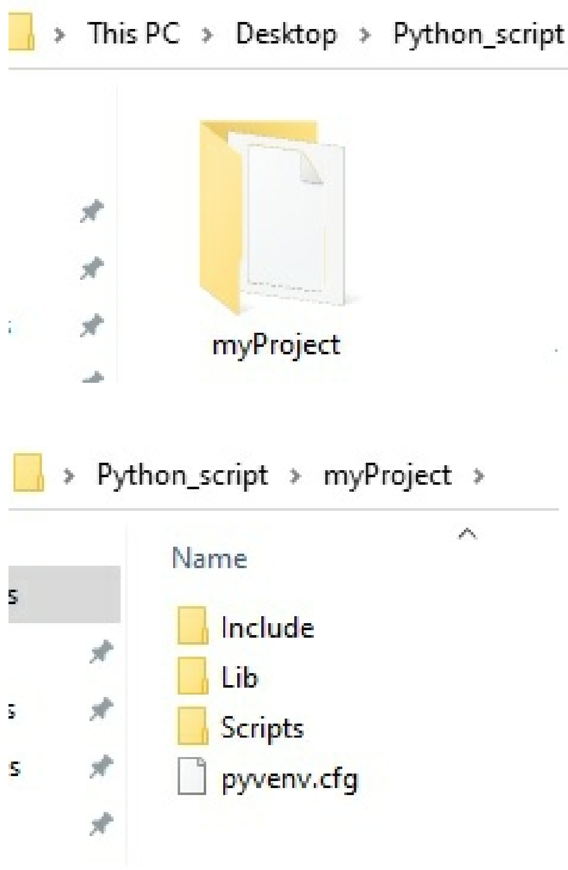
So let's create a folder *(I named my folder **Python_script)*** .

Open Command prompt -> navigate to the folder location (***Python_script)*** then write the following command:

**py –m venv *project_name***

```
C:\Users\[redacted]\Desktop\Python_script>py -m venv myProject
```

We see Django created the folder for us.




Now Django documentation says:

This will create a folder called 'project-name' if it does not already exist and setup the virtual environment. To activate the environment, run:

```
...\> project-name\Scripts\activate.bat
```

The virtual environment will be activated and you'll see "(project-name)" next to the command prompt to designate that. Each time you start a new command prompt, you'll need to activate the environment again.

So let's activate the virtual environment. Open Command prompt -> Navigate to **Scripts** folder -> run the following command:

```
activate.bat
```

```
C:\Users\[illegible]\Desktop\Python_script>cd myProject

C:\Users\[illegible]y\Desktop\Python_script\myProject>cd Scripts

C:\Users\[illegible]y\Desktop\Python_script\myProject\Scripts>activate.bat

(myProject) C:\Users\[illegible]y\Desktop\Python_script\myProject\Scripts>
```

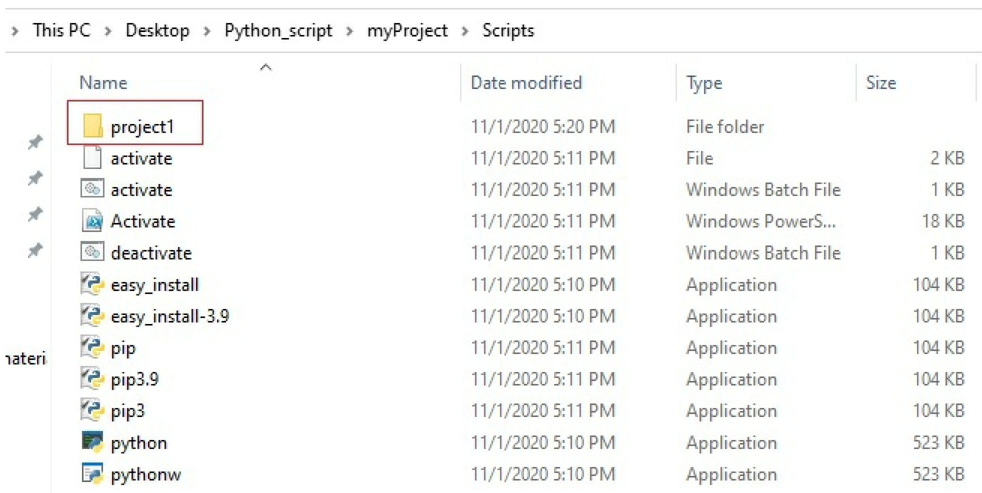
**Step 5:** After setting up the virtual environment, open command prompt -> navigate to above **Scripts** folder -> create the Django project by using the following command:

**django-admin startproject** *project_name*

*(I named my project* ***project1*** *).*

```
(myProject) C:\Users\[illegible]y\Desktop\Python_script\myProject\Scripts>django-admin startproject project1

(myProject) C:\Users\[illegible]y\Desktop\Python_script\myProject\Scripts>
```

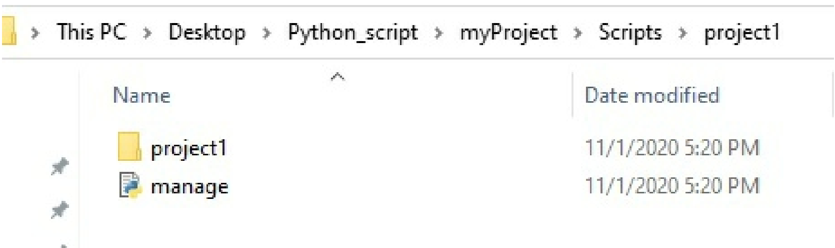
We see that Django created the project folder (***project1*** ) for us.




**NOTE:** All Django main projects reside within the **Scripts** folder

**Step 6:** Open the project folder (***project1*** ) and you will see the presence of **manage.py** file. This file is the most important file and is needed to run a

Django project.



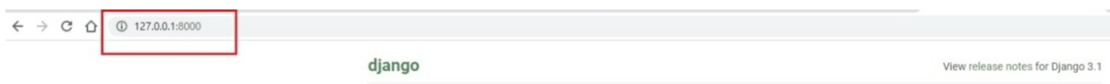

Now let’s **run** our first Django project (***project1*** ).

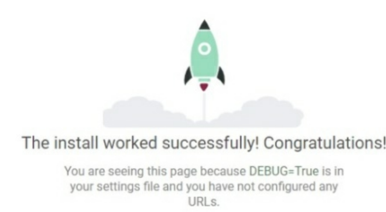
Open command prompt -> navigate to the ***project1*** folder location where **manage.py** file is present -> run the following command:

**python manage.py runserver**

```
C:\>cd C:\Users\[illegible]\Desktop\Python_script\myProject\Scripts\project1

C:\Users\[illegible]\Desktop\Python_script\myProject\Scripts\project1>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly until you apply the
 auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
November 01, 2020 - 17:40:20
Django version 3.1.2, using settings 'project1.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Open web browser and go to site highlighted in the screen shot above.




The installation works perfectly.

## 5.2 : Django App

App stands for Application.

**What is the difference between a Django project and a Django app?**

A **project** refers to the entire application all together, while an **app** refers to a sub module of the main project.

Let's take Amazon for example. Amazon is a huge company and it may be divided into multiple departments. One department may handle the Amazon Marketplace, other department may handle the Amazon KDP and so on. So in this case, Amazon Marketplace and Amazon KDP become **apps** or sub divisions of the big **project** Amazon.

Now let's create an **app** and connect it with our main project (***project1*** *)*

.

**Step 1:** Open command prompt -> since the **app** will reside within the main project (***project1*** *)*, we need to navigate to the project location -> now create

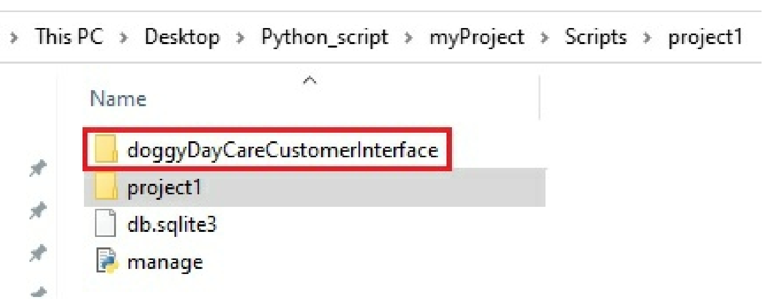
a new **app** with the help of the command:

**python manage.py startapp** *app_name*

```
C:\>cd C:\Users\[illegible]\Desktop\Python_script\myProject\Scripts\project1

C:\Users\[illegible]\Desktop\Python_script\myProject\Scripts\project1>python manage.py startapp doggyDayCareCustomerInterface
```

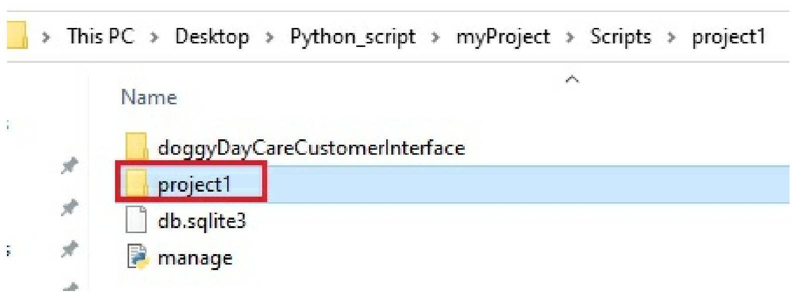
*(I gave the name of my* ***app doggyDayCareCustomerInterface*** *).*

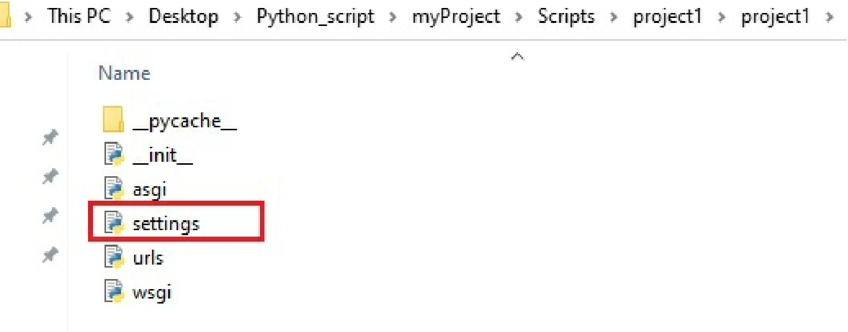
We see Django automatically creates the new **app** folder within the main project (***project1*** ).




**Step 2:** Now let's connect our newly created **app** (***doggyDayCareCustomerInterface)*** with the main project (***project1*** ) .

Open the ***project1*** folder which contains all the important **Python** files like **settings.py** , **urls.py** -> open **settings.py** file






In **settings.py** , add the newly created **app** (***doggyDayCareCustomerInterface)*** into the list of **INSTALLED_APPS** and click save.

```
28  ALLOWED_HOSTS = []
29
30
31  # Application definition
32
33  INSTALLED_APPS = [
34          'doggyDayCareCustomerInterface',
35          'django.contrib.admin',
36          'django.contrib.auth',
37          'django.contrib.contenttypes',
38          'django.contrib.sessions',
39          'django.contrib.messages',
40          'django.contrib.staticfiles',
41  ]
42
43  MIDDLEWARE = [
44          'django.middleware.security.Securi
45          'django.contrib.sessions.middlewa1
```

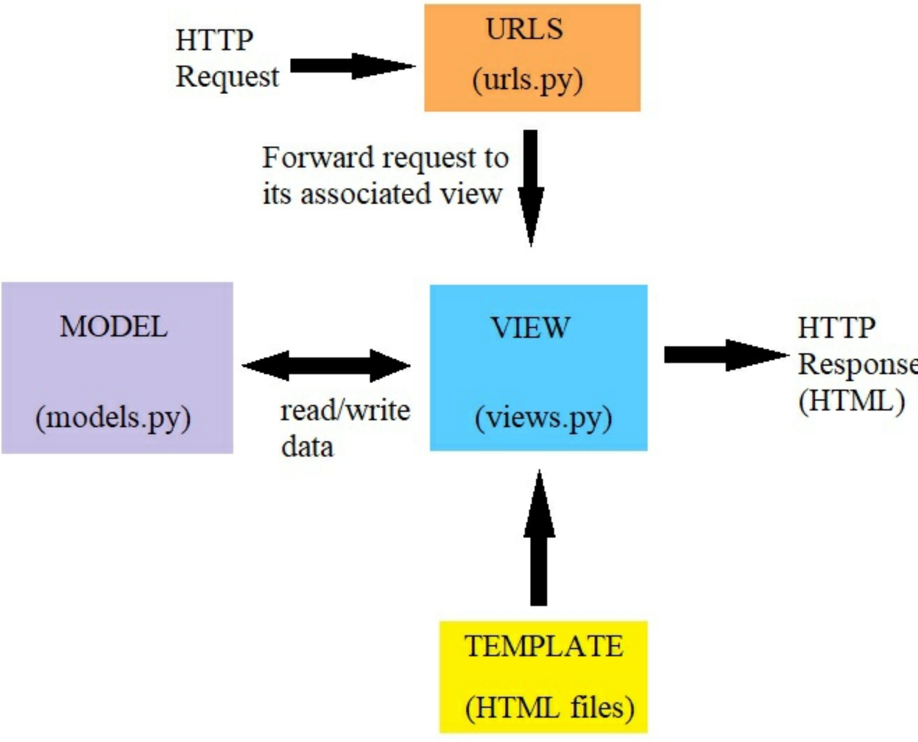
The connection between the **app** (***doggyDayCareCustomerInterface)*** and the **project** (***project1*** *)* is now established.

### *5.2.1 : Django Architecture*

We have successfully created our new **app** (***doggyDayCareCustomerInterface).***

Each **app** must contain the following:

1. **urls.py file** – It contains the list of all **urls** present within the **app** .
2. **views.py file** – It contains all the **functions** and **classes** which will perform tasks.
3. **models.py file** – It contains the **models** or the database tables.
4. **admin.py file** - It helps to register the **models** into **Django Admin App** .
5. **templates folder** – It contains all the HTML files.
6. **static folder** – It contains all the **static** files like CSS, Images and Javascripts.




The Django process is very simple:

1. A **HTTP request** from a user first goes to **urls.py** file where it looks for its associated **url name** and with that name finds its associated **function** which is present in **views.py** file.

2. The function in **views.py** file throws the **HTTP response** (HTML file from **templates** folder) to the user.

- **views.py** file also has the ability to access **models.py** file to read or write data.
- **models.py** contains all the **models** or database tables

Now let's create a simple Django ***Hello World* app** to understand the entire process.

### *5.2.2 : Hello World Example*

**Step 1:** Let's create a new **app** within ***project1*** and name it ***helloWorld*** .

```
C:\Users\[redacted]\Desktop\Python_script\myProject\Scripts\project1>python manage.py startapp hello_world
```

**Step 2:** Now let's connect our newly created **app** (***helloWorld)*** with the main project (***project1 )*** **.**

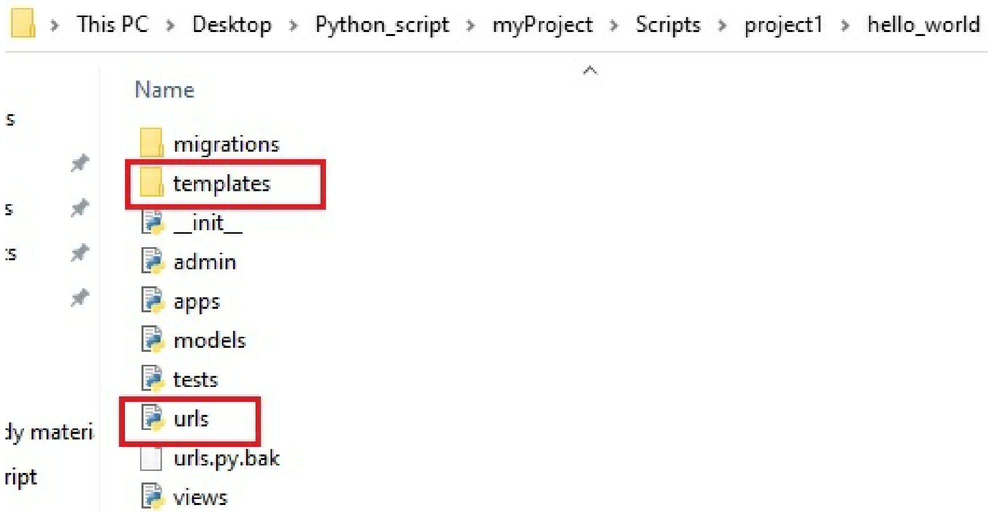
Open folder ***project1*** -> Open **settings.py** file -> Add ***hello_world*** within **INSTALLED_APPS** and click save.

```
33  INSTALLED_APPS = [
34          'hello_world',
35          'doggyDayCareCustomerInterface',
36          'django.contrib.admin',
37          'django.contrib.auth',
38          'django.contrib.contenttypes',
39          'django.contrib.sessions',
40          'django.contrib.messages',
41          'django.contrib.staticfiles',
42  ]
43
44  MIDDLEWARE = [
45          'django.middleware.security.Secur
```

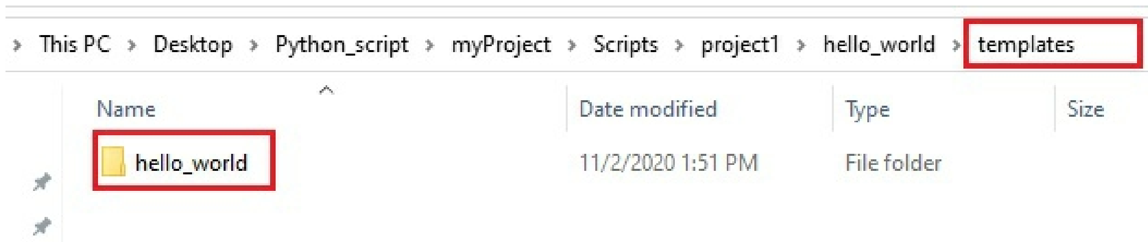
**Step 3:** Now open ***hello_world*** **app** folder and check for the presence of **urls.py, views.py** and **models.py** file. If **urls.py** is not present then create a new **Python** file and name it **urls.py** .

**Please note:** I am using Notepad++ to write all my codes.

**Step 4:** Within ***hello_world*** **app** , create a new folder and name it **templates.**




**Step 5:** Inside **templates** folder create another folder with the same name as the **app** . This new folders **templates** -> ***hello_world*** will contain all HTML files used exclusively by ***hello_world* app** .




Create a simple HTML file (*I named my file **index.html*** ) and save it inside **templates** -> ***hello_world*** folder.

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <style>
5  h1{
6  font-family: Lucida Handwriting;
7  font-size:100px;
8  font-weight:bold;
9  text-align:center;
10 color:green;
11 }
12 </style>
13 </head>
14 <body>
15 <h1>Hello World</h1>
16 </body>
17 </html>
```

> This PC > Desktop > Python_script > myProject > Scripts > project1 > hello_world > templates > hello_world

Name

index

**Step 6:** Open **urls.py** file and write the following piece of code shown in the screen shot below

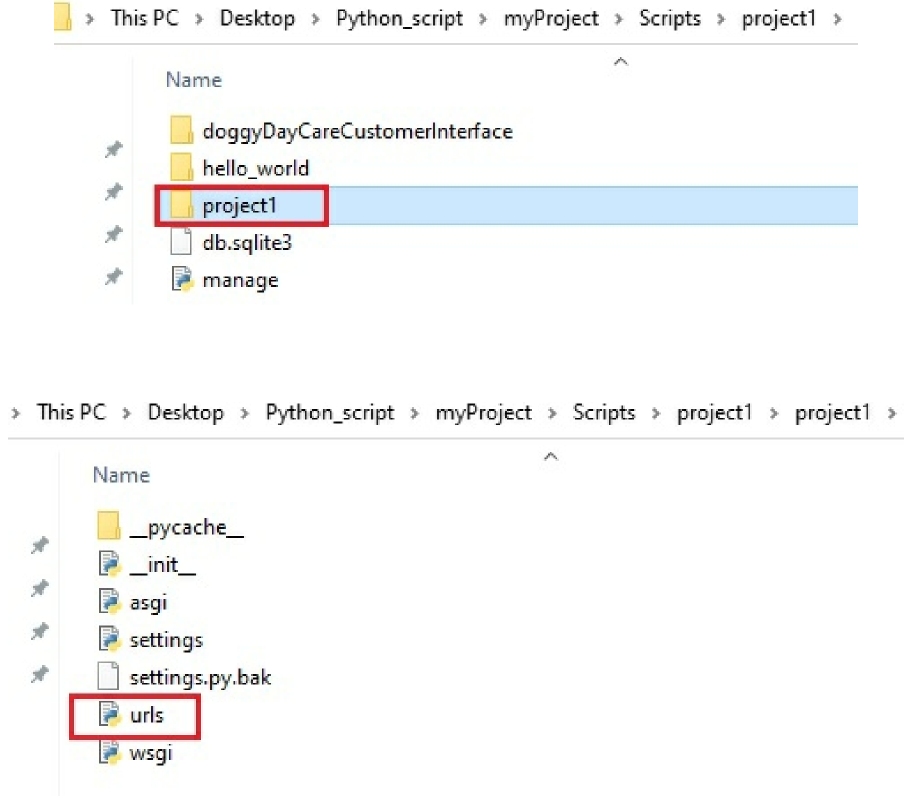
```
1  from django.urls import path
2  from . import views
3  urlpatterns = [
4          path("home", views.index, name="home")
5  ]
6
```

- **urlpatterns** is a **tuple** which contains the paths to each **function** written in **views.py** file. It defines the connection between the **urls** and the **views** .

- **path( )** function is used for routing **url** s to the appropriate **view functions** .

**Step 7:** Connect our **app** 's (***hello_world)*** **urls.py** file with our main project's (***project1*** ) **urls.py** file. To do that:

- Open project folder (***project1)*** where important files like **settings.py** and **urls.py** files are present -> Open **urls.py** file



- Into our main project's **urls.py** file, create a path that will lead to us to ***hello_world*** **app** and its corresponding **url** s and click save.

```
16  from django.contrib import admin
17  from django.urls import include, path
18
19  urlpatterns = [
20          path('admin/', admin.site.urls),
21          path('hello_world/',include("hello_world.urls"))
22  ]
```

**NOTE:** To use **include ( ) function** , we need to import it from **django.urls**

**Step 8:** Open app's (***hello_world)*** **views.py** file and create the ***index*** **function** which is called from app's (***hello_world)*** **urls.py** file.

```
1  from django.shortcuts import render
2  from django.http import HttpResponse
3  # Create your views here.
4  def index(request):
5    return render(request, "hello_world/index.html")
6
```

The ***index*** **function** takes the **HTTP request** and with the help of **render( )** function gives a HTTP response which is the ***index.html*** file stored in **templates** -> ***hello_world*** folder.

**What is render function?**

**Render( )** function is used to return an **Http response** and to use this **function** we need to import **HttpResponse** as shown in the screen shot above.

**Step 9:** Save everything and run the project.

Open command prompt -> navigate to the main project (***project1*** ) location where the **manage.py** file is present -> run the following command:

**python manage.py runserver**

This PC > Desktop > Python_script > myProject > Scripts > project1

Name

doggyDayCareCustomerInterface

hello_world

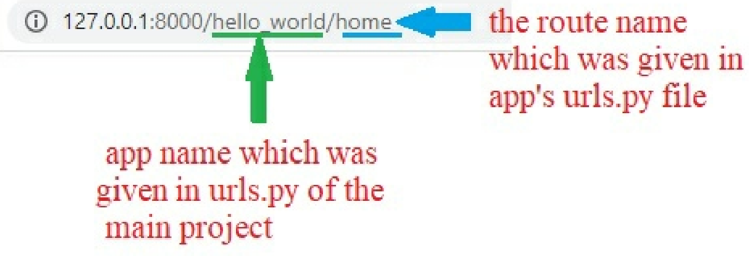
project1

db.sqlite3

manage

```
C:\Users\[illegible]\Desktop\Python_script\myProject\Scripts\project1>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly until you apply the
 auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
November 02, 2020 - 15:26:16
Django version 3.1.2, using settings 'project1.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Open your web browser -> go to [**http://127.0.0.1:8000/**](http://127.0.0.1:8000/) and type the following **url** depicted in the screen shot below.




127.0.0.1:8000/hello_world/home

Hello World

Let’s summarize the entire Django process.

1
- Create the main **project**. The most important files of our main project are: **settings.py** file and **urls.py** file.

2
- The main project is divided into many sub projects or **apps**. Create the **apps**. The most important files of each app are: **urls.py**, **views.py** and **models.py**

3
- Establish connection between the **app** and the main **project** with the help of **settings.py** file and **urls.py** file

4
- In **app**'s **urls.py** file, create a list of all **urls**. The list should contain path to all **url**'s corresponding **functions** present in **views.py** file

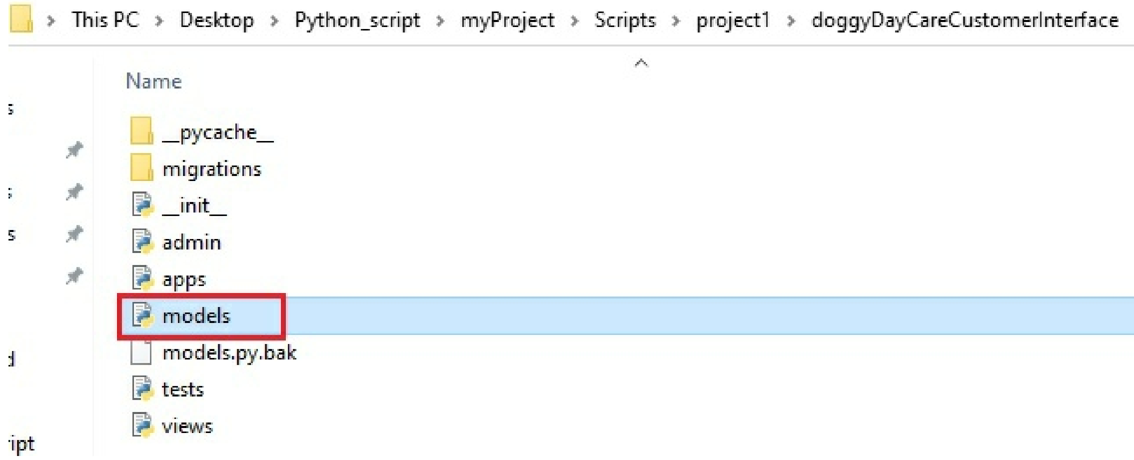
5
- In **views.py** file, the main task is performed and the **HTTP response** is generated. NOTE : **views.py** file has the ability to access the **templates** folder and the **models.py** file

# 5.3 : Django Models

Django Models are more like database tables where data are stored.

Let’s create some **models** or tables for our **app** ***doggyDayCareCustomerInterface*** *(created in section 5.2)* .

**Step 1:** Inside ***doggyDayCareCustomerInterface*** **app** folder look for **models.py** file.




**Step 2:** Inside **models.py** we will create two **model classes** , ***customer_information*** and ***dog_information*** .

```
1   from django.db import models
2
3   # Create your models here.
4   class customer_information(models.Model):
5     customer_id = models.IntegerField()
6     username = models.CharField(max_length=64)
7     password = models.CharField(max_length=20)
8     firstName = models.CharField(max_length=64)
9     lastName = models.CharField(max_length=64)
10    gender = models.CharField(max_length=1)
11    address = models.CharField(max_length=64)
12    state = models.CharField(max_length=64)
```

```
14  class dog_information(models.Model):
15    dog_id = models.ForeignKey(customer_information,on_delete=models.CASCADE)
16    dog_name1 = models.CharField(max_length=64)
17    dog_name2 = models.CharField(max_length=64)
18    dog_age1 = models.IntegerField()
19    dog_age2 = models.IntegerField()
```

- **models.IntegerField( )** creates a column to store integer values.

  **models.CharField(max_length= n)** creates a column to store character values, where **n** specifies the max length.

**Please Note :** There are multiple model field, please visit Django documentation for Model field reference https://docs.djangoproject.com/en/3.2/ref/models/fields/

- To establish relationship between the two **models** or tables, we used **Foreign Key.**

```
dog_id = models.ForeignKey(customer_information,on_delete=models.CASCADE)
```

  **on_delete = models.CASCADE** means that if the reference object is deleted, then also delete its associated records pointing to the reference object.

**NOTE :** To add primary key to a particular field, set **primary_key** to **True** .

Example : name = models.CharField(max_length=50, primary_key=True)

**Step 3:** Start the **migration process** of creating the tables.

Open command prompt -> navigate to main project folder (***project1*** ) location where **manage.py** file is present -> run the following command:

**python manage.py makemigrations**

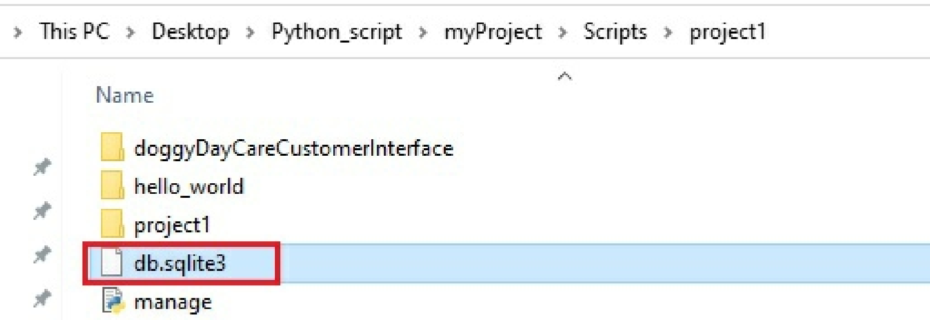
```
C:\Users\[redacted]\Desktop\Python_script\myProject\Scripts\project1>python manage.py makemigrations
Migrations for 'doggyDayCareCustomerInterface':
  doggyDayCareCustomerInterface\migrations\0001_initial.py
    - Create model customer_information
    - Create model dog_information
```

For applying the migration to Django database run the following command:

**python manage.py migrate**

```
C:\Users\[illegible]\Desktop\Python_script\myProject\Scripts\project1>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, doggyDayCareCustomerInterface, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
```

**Step 4:** Open the **SQLITE file** present in the main project folder (***project1*** ) location and notice the presence of newly created **models** in that file.




**Step 5:** Now let's add some data into our **models** .

Open command prompt -> navigate to the main project folder (***project1*** ) location where **SQLITE file** and **manage.py** file is present and enter the following command:

**python manage.py shell**

```
C:\Users\[illegible]\Desktop\Python_script\myProject\Scripts\project1>python manage.py shell
Python 3.9.0 (tags/v3.9.0:9cf6752, Oct  5 2020, 15:34:40) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>>
```

In order to work with models, we need to import them first.

- **from** *doggyDayCareCustomerInterface* **.models import** *customer_information*

this line means from model.py file present in doggyDayCareCustomerInterface app import the table or model customer_information

```
>>> from doggyDayCareCustomerInterface.models import customer_information
>>> x = customer_information(customer_id = 123, username = "John",password = "John",firstName = "John Jim",lastName = "K
im", gender = "M",address = "Cedar Park",state = "Texas")
>>> x.save()
>>>
```

The syntax for inserting data is :

**model_name ( column1 = “** *data* **”, column2 = “** *data* **”, column3 =** *123* **……… )**

- Inserted some data into ***customer_information*** model or table:
  *x = customer_information (customer_id = 123, username = “John”, password = “John”, firstName = “John Jim”, lastName = “Kim”, gender = “M”, address = “Cedar Park”, state = “Texas”)*

- Save the data
  *x .* ***save( )***

We have successfully inserted a new record into our ***customer_information*** model or table. Now let’s access that record from **Django admin app** .

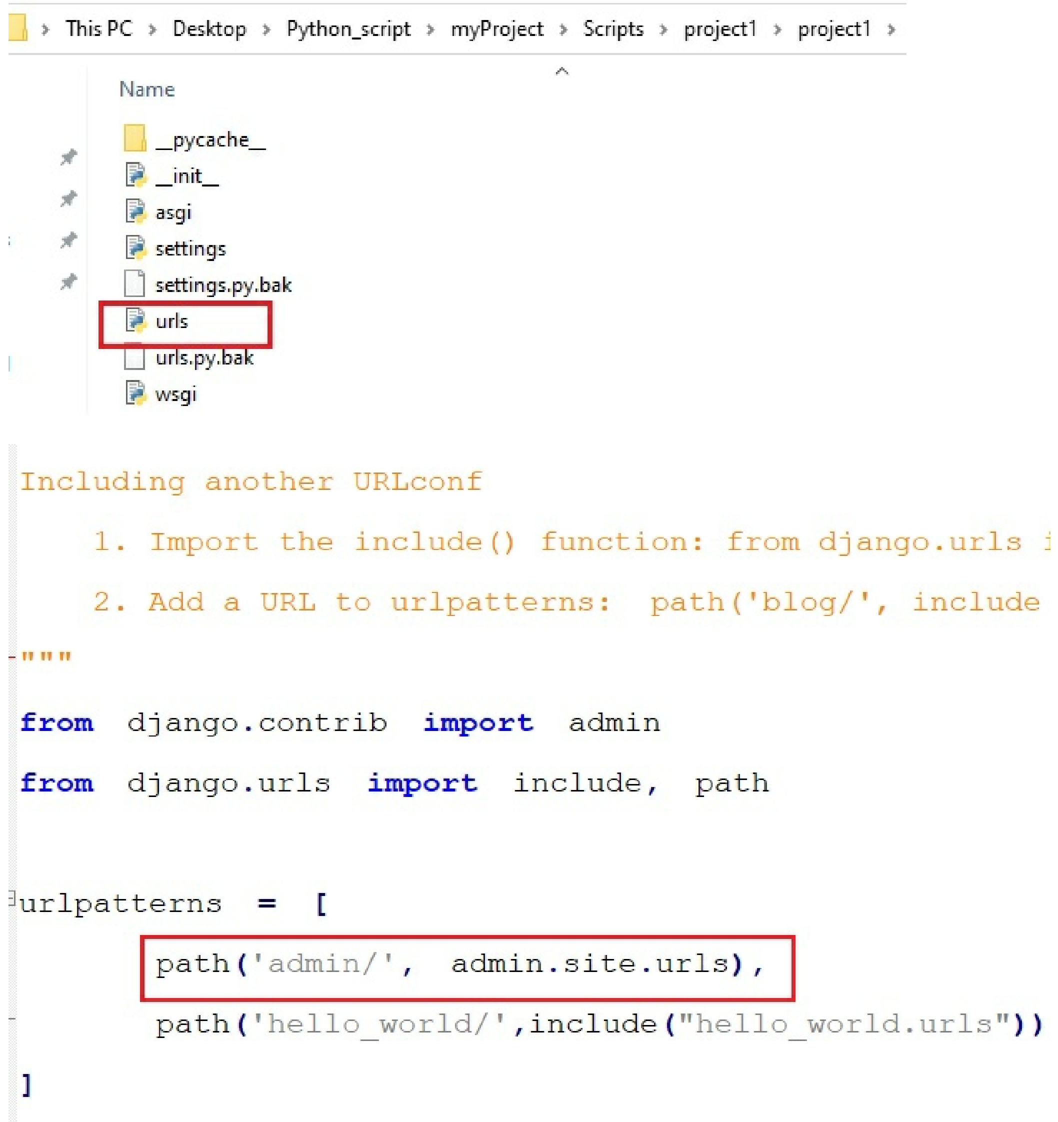
## 5.4 : Django Admin App

Django has already created an **app** for us for data manipulation and it is called the **Admin App** . This **app** gives us the ability to **insert** , **update** or **delete** data without going through the hassle of writing multiple database queries.

Steps to access the Django **Admin App** are:

**Step 1:** Open the project folder (***project1*** ) which contains important files **settings.py** and **urls.py** - > Open the **urls.py** file and check for the presence of path to the **admin app** :

- **path('admin/', admin.site.urls)**



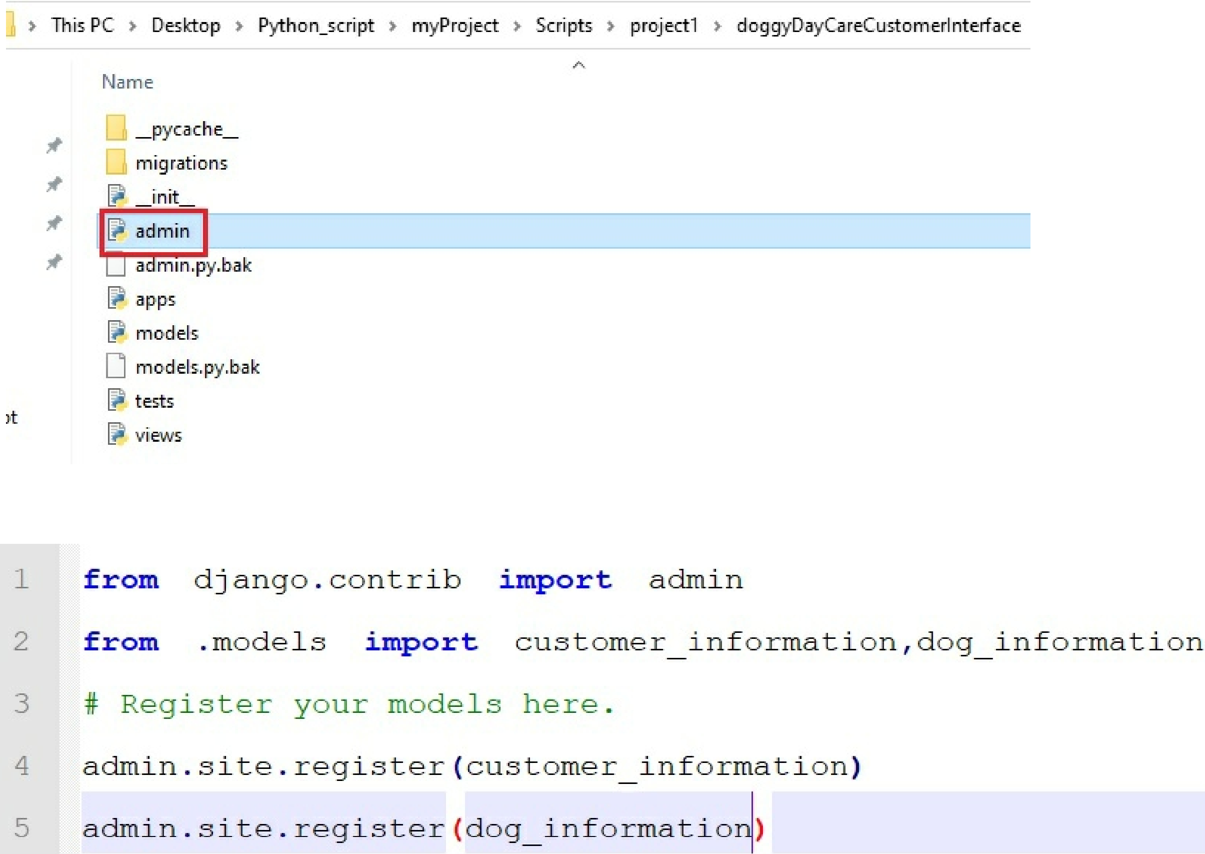

**Step 2:** Now open command prompt - > navigate to the main project (***project1*** ) folder location.

To access the **admin app** , we need to create our administrative account which can be done with the help of the following command:

## python manage.py createsuperuser

```
C:\Users\[redacted]\Desktop\Python_script\myProject\Scripts\project1>python manage.py createsuperuser
Username (leave blank to use '[redacted]'):
Email address: [redacted]
Password:
Password (again):
The password is too similar to the username.
This password is too short. It must contain at least 8 characters.
This password is too common.
Bypass password validation and create user anyway? [y/N]: y
Superuser created successfully.
```

**Step 3:** Go to **app** (***doggyDayCareCustomerInterface*** ) -> Open its **admin.py** file -> Import and Register the **models** (*created in section 5.3* ) -> click save

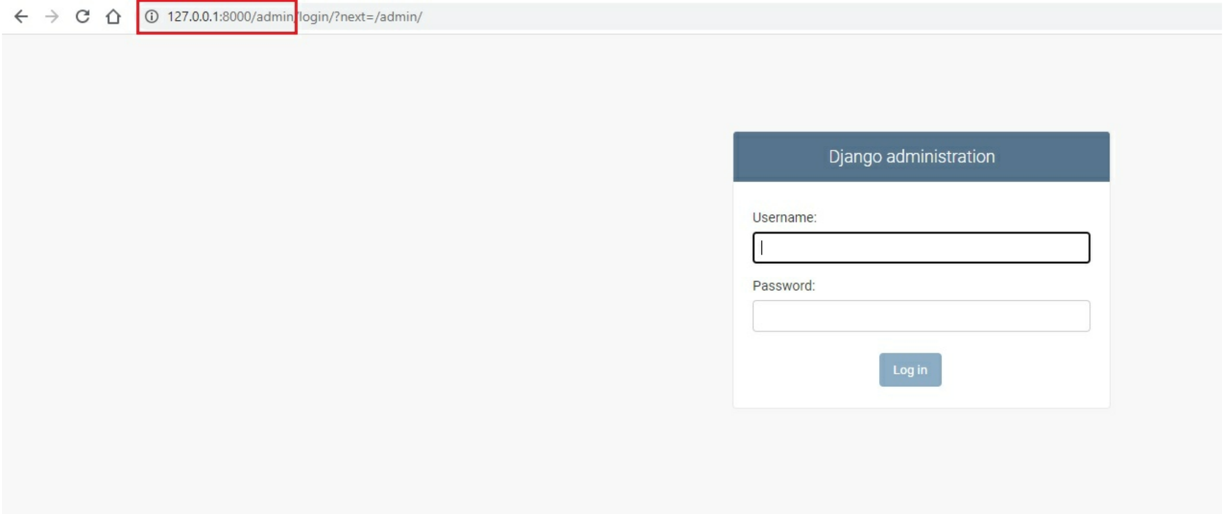



**Step 4:** Open command prompt -> navigate to the main project (***project1*** ) folder location where the **manage.py** file is present -> run the project using the following command

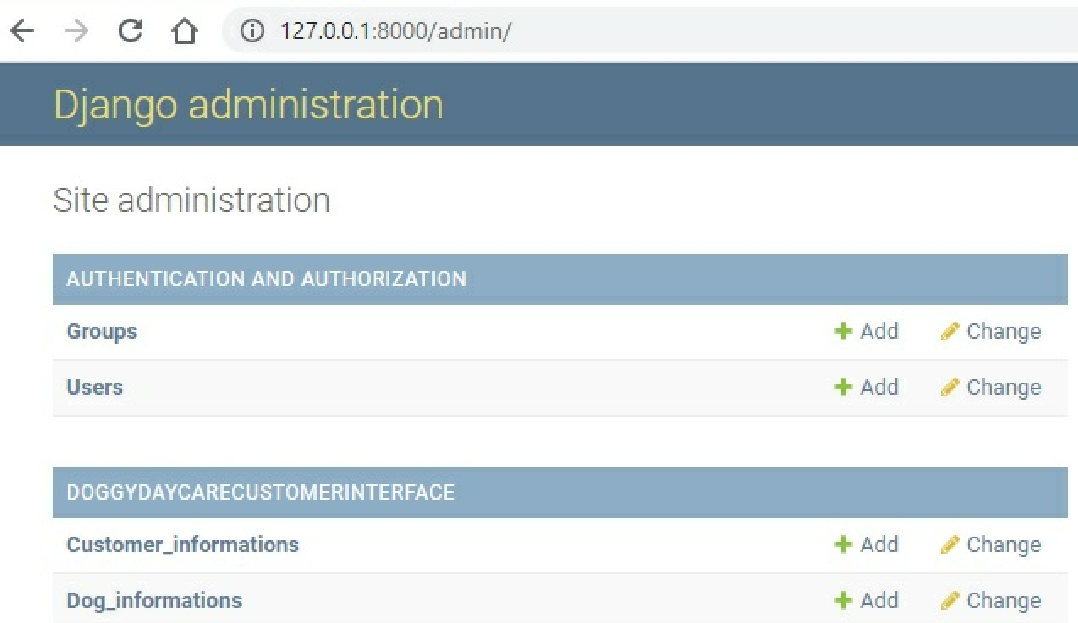
## python manage.py runserver

```
C:\Users\[illegible]\Desktop\Python_script\myProject\Scripts\project1>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
November 03, 2020 - 12:20:01
Django version 3.1.2, using settings 'project1.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

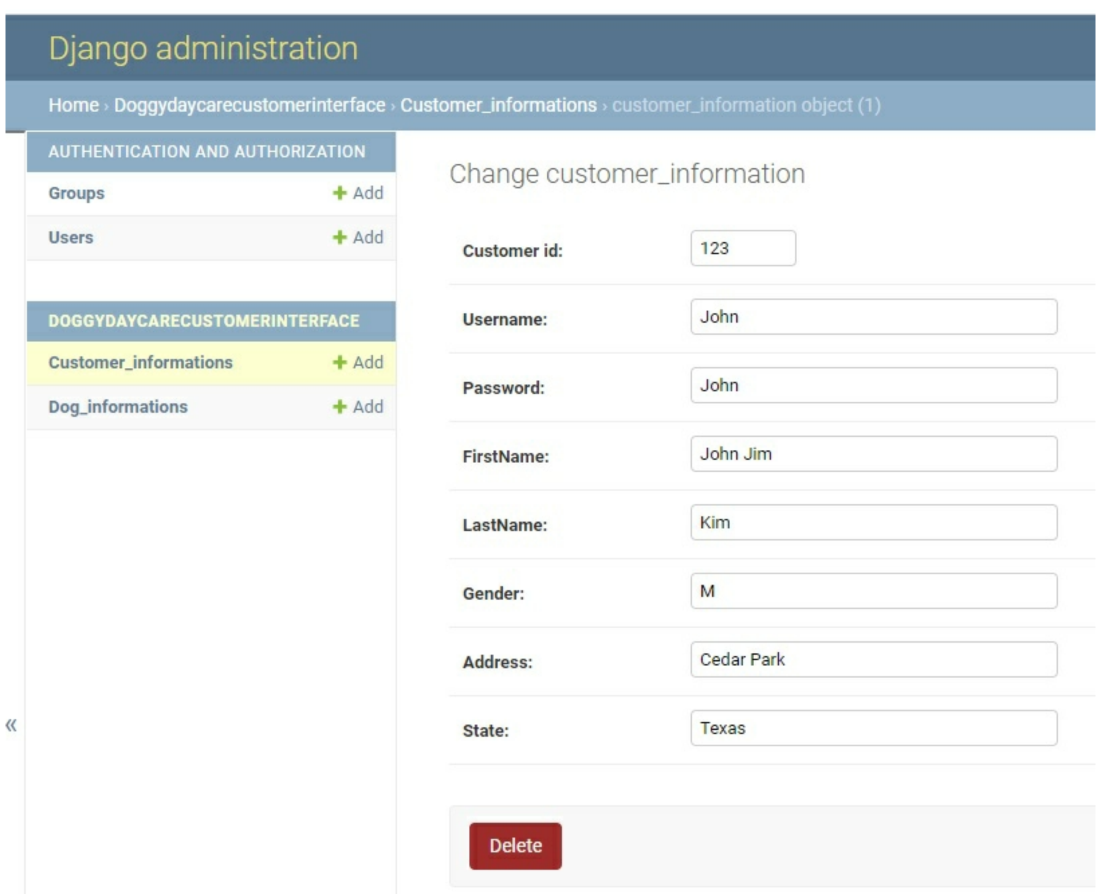
Open web browser -> type **[http://127.0.0.1:8000/ admin](http://127.0.0.1:8000/admin)**



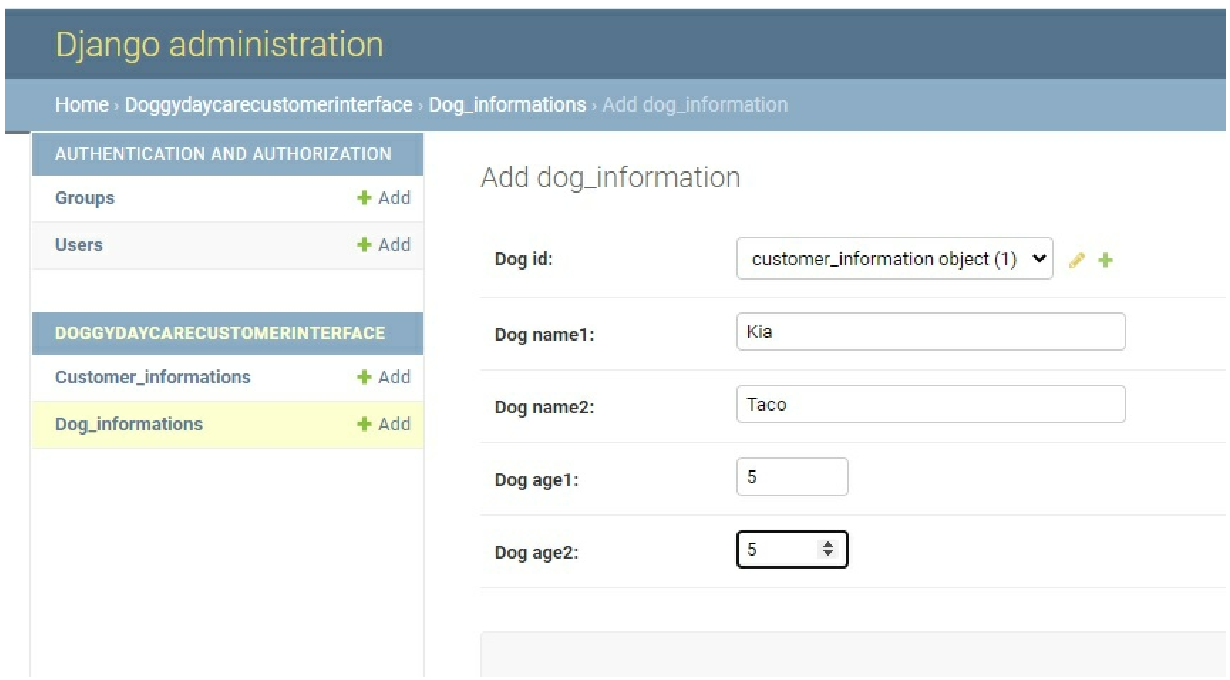

Enter username and password -> click Log In

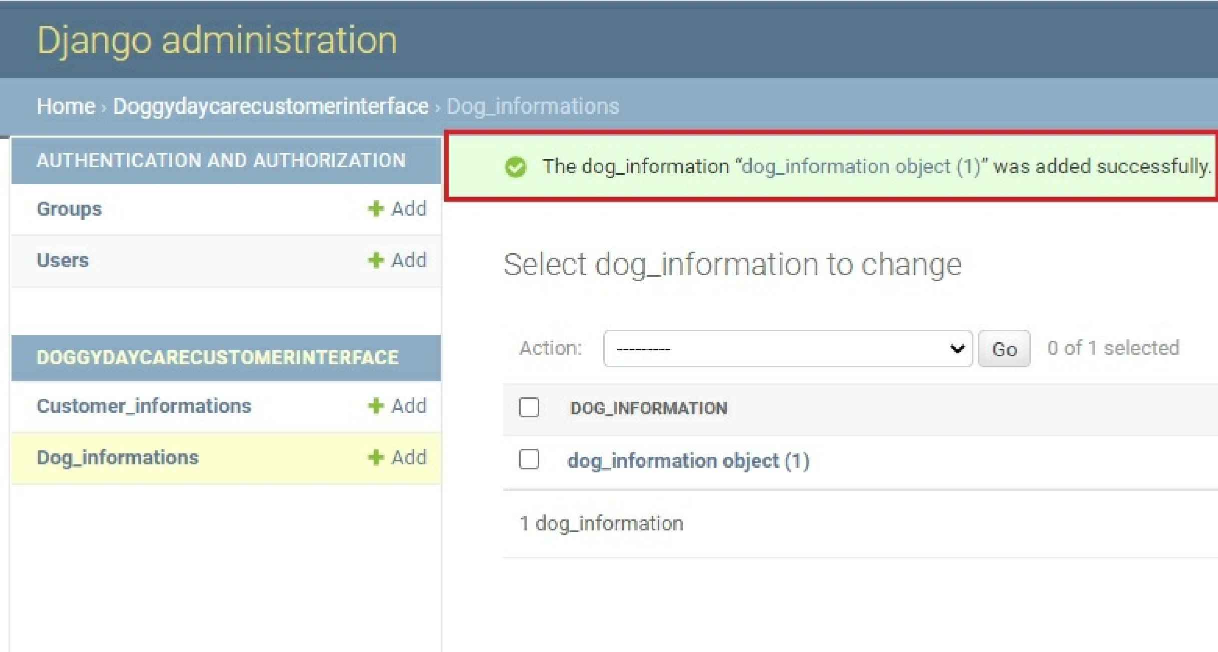



The **admin app** shows the **models** or tables with records (*created in section 5.3* )



Add some associated dog information into ***dog_information* model** or table and click save.




## 5.5 : How to navigate to different pages of our project using Django

In section 5.2, we have created our **app *doggyDayCareCustomerInterface*** and connected it with our main project ***project1*.**

In section 5.3 we created its **models** or database tables.

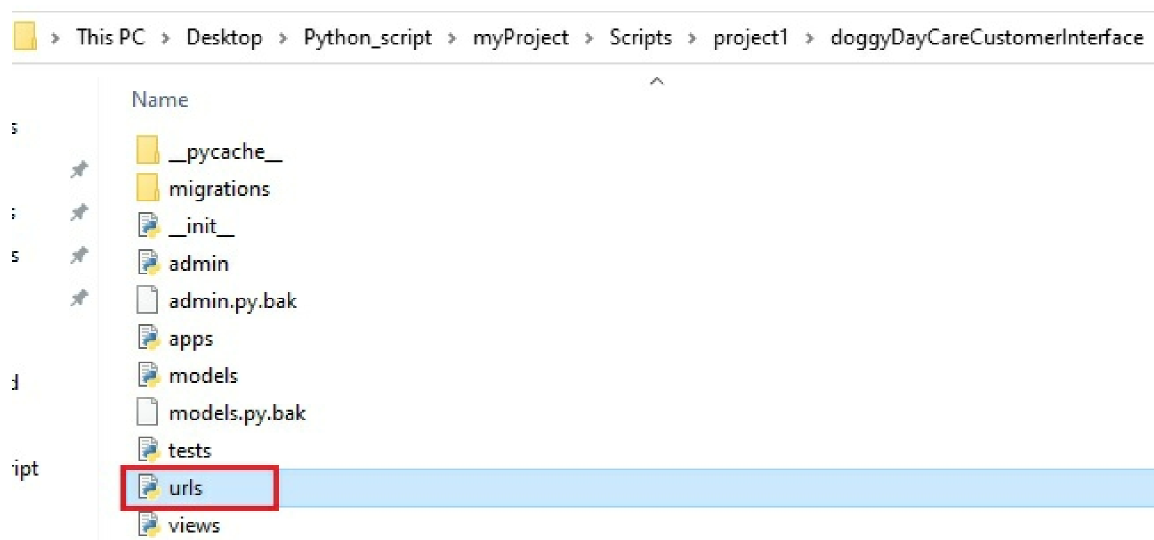
Now let's follow the steps from ***Hello World*** Example (*section 5.2.2* ) and make our ***doggyDayCareCustomerInterface* app** functional**.**

> **Please Note: *doggyDayCareCustomerInterface* app** is the continuation of the ***Doggy Day Care Center*** project and it will contain all the HTML files which we created in Chapter 1 and 2.

**Step 1:** Established connection between the main project and ***doggyDayCareCustomerInterface* app** through **settings.py** file (*already done in section 5.2* ).

**Step 2:** Now let's include the **app** 's path and all its **url** s into main project's (***project1*** *)* **urls.py** file.

- Create a new **urls.py** file into our **app** (***doggyDayCareCustomerInterface)*** folder.



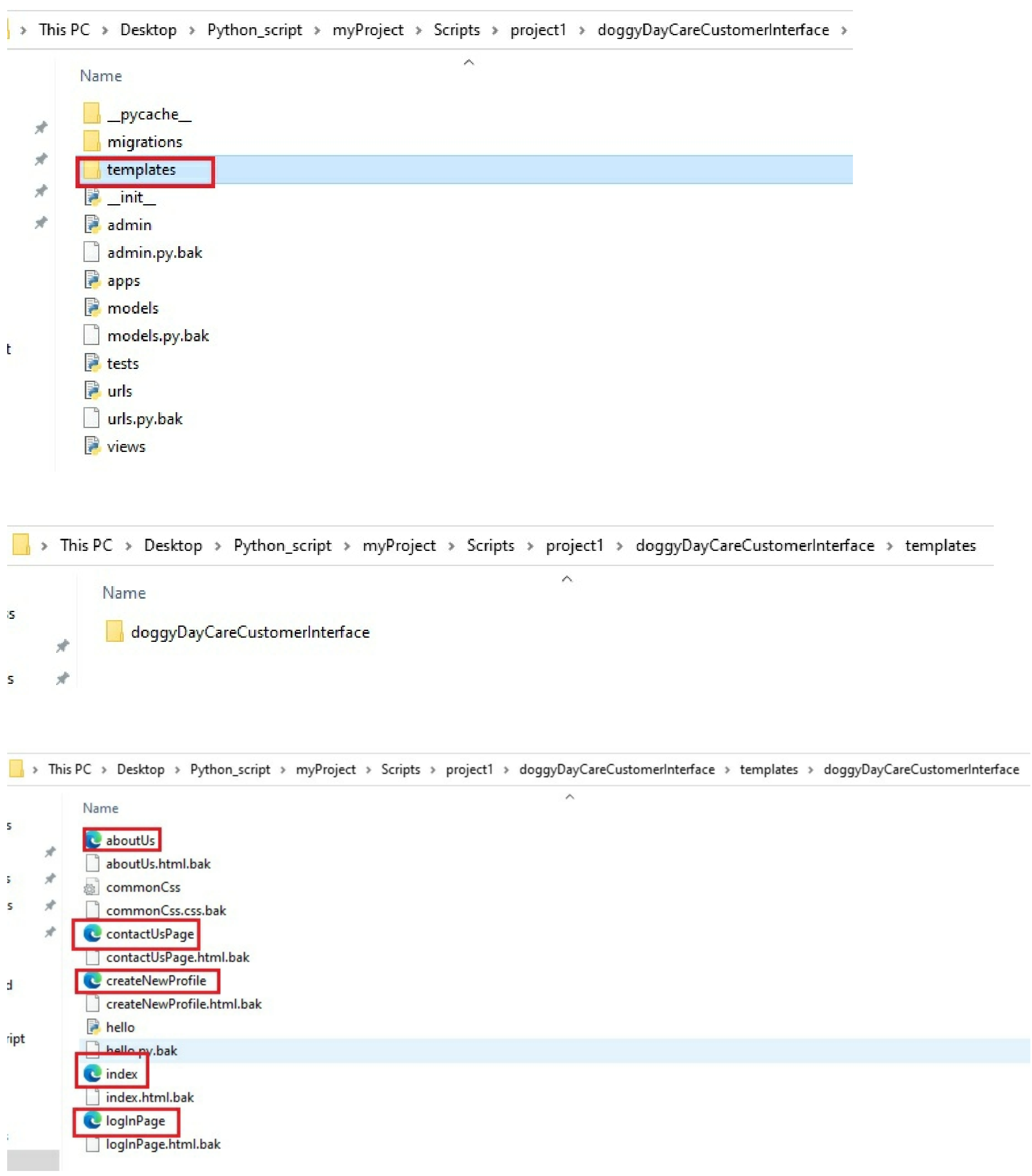

- Connect the **app** 's **urls.py** file with our main project **urls.py** file (*as we did in hello world example in section 5.2.2* )

This PC > Desktop > Python_script > myProject > Scripts > project1 > project1 >

Name
- __pycache__
- __init__
- asgi
- settings
- settings.py.bak
- urls
- urls.py.bak
- wsgi

```
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import include, path

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

**Step 3:** Create the **templates** folder -> Create another folder within **templates** folder with the same name as the **app** and paste all the ***Doggy Day Care*** project’s HTML files into it.




**Step 4:** Now let's go to the **app** 's (***doggyDayCareCustomerInterface)*** **urls.py** file and add the following lines of code:

**urls.py**

```
from django.urls import path
from . import views
urlpatterns = [
    path("", views.index, name="index"),
    path("logInPage", views.logInPage, name="logInPage")
]
```

**Step 5:** Open **app** 's (***doggyDayCareCustomerInterface)*** **views.py** file and add the following lines of code:

**views.py**

```
from django.shortcuts import render
from django.http import HttpResponse
# Create your views here.
def index(request):
  return render(request, "doggyDayCareCustomerInterface/index.html")

def logInPage(request):
  return render(request, "doggyDayCareCustomerInterface/logInPage.html")
```

**Code Explanation:**

- In **urls.py** file, ***path("", views.index, name="index")*** does not have any routing name. This means when there is no routing name simply proceed to ***index*** **function** present in **views.py** file. The ***index*** **function** returns ***index.html*** (*Home page of Doggy Day Care project* ) file to the user.

- In **urls.py** file, ***path("logInPage", views.logInPage, name="logInPage")*** have a routing name and **url** name of ***logInPage*** . This means when the routing name is ***logInPage*** proceed to ***logInPage*** **function** present in **views.py** file. The ***logInPage*** **function**

returns ***logInPage.html*** (*Log In page of Doggy Day Care project* ) to the user.

**Step 6:** Open ***index.html*** file of the ***Doggy Day Care Project (*** *created in section 1.2.1, Chapter 1)* -> Notice there was a hardcoded path given to the HTML **href attribute** which would lead us to ***Log In*** HTML page.

index.html

```
<div id = "homePageLinks">
<ul>
<li><a href =
"C:\Users\[illegible]\HTML\logInPage.html">
Log In</a></li>
</ul>
</div>
```

Now delete the old path and in that **href attribute** give the **url name** ***logInPage*** from **urls.py** file.

```
<div id = "homePageLinks">
<ul>
<li><a href = "{% url 'logInPage' %}">Log In</a></li>
</ul>
</div>
```

The syntax for adding the **url name** from **urls.py** file into a HTML file is:

**{% url ‘** *url_name’* **%}**

### What is {% …. %}?

{% … %} are called **Template Tags** in Django. They are mainly used within HTML document to:

1. Load some external information.

2. Perform loops or logic.

When user click on the ***Log In*** button of the ***Home Page*** , the **href attributes** leads us to **url** name ***logInPage*** . The request proceeds to **urls.py** file, where it looks for the **path** with **url** name ***logInPage*** . Now with the help of this name, it goes to its associated **function** ***logInPage*** present in **views.py** file. **Function** ***logInPage*** simply returns the ***Log In*** page to the user (*please refer to codes above* ).

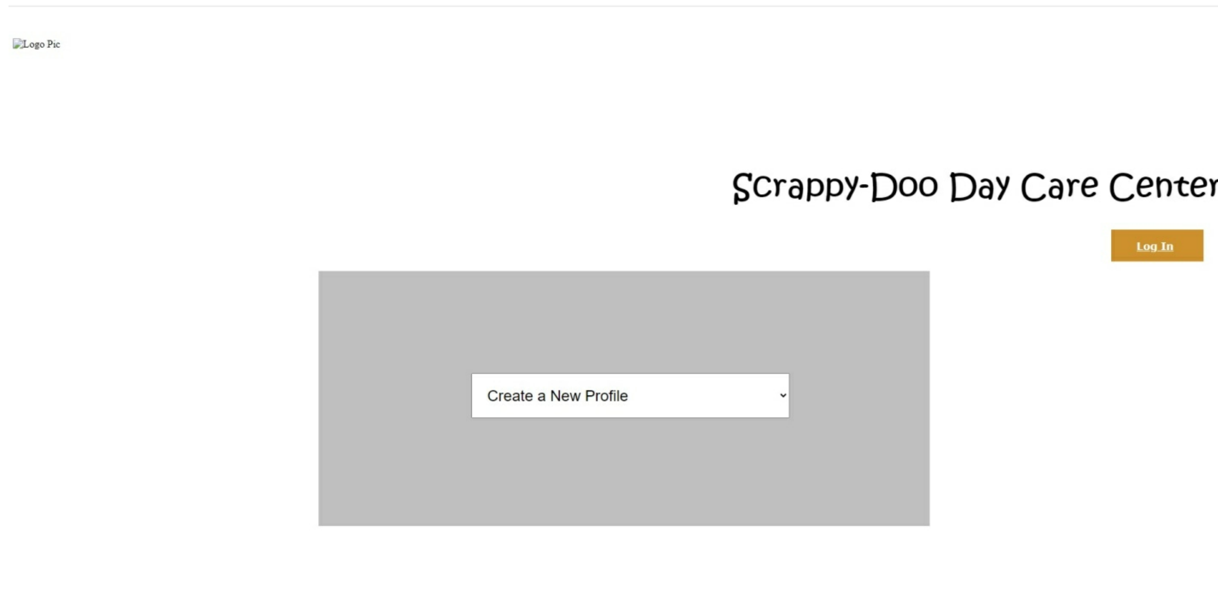
**Step 7:** Let’s run our project and check whether the navigation from ***Home page*** to ***Log In page*** is performing correctly or not.

Open command prompt -> navigate to the main project (***project1*** ) location where **manage.py** file is present and run the following command:

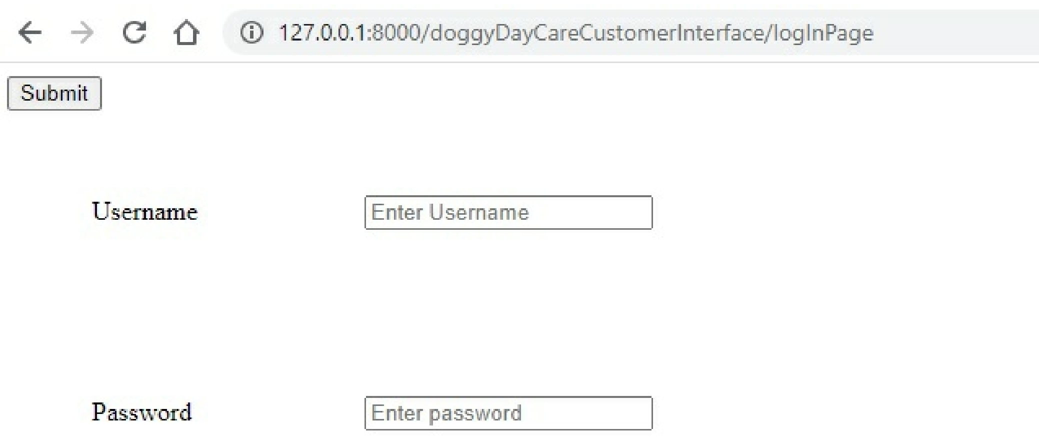
**python manage.py runserver**

```
C:\Users\[illegible]\Desktop\Python_script\myProject\Scripts\project1>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
November 03, 2020 - 13:50:46
Django version 3.1.2, using settings 'project1.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Open web browser -> go to **http://127.0.0.1:8000/doggyDayCareCustomerInterface/**




Now click on Log In button to check the navigation works properly or not.




The navigation works perfectly. Now follow the steps above and replace all hardcoded paths present in ***index.html*** files with **url names** .

Everything works fine but you will notice one BIG flaw. You will notice that the pages loaded without any images and CSS. So let’s fix this issue in

the next section.

## 5.6 : How to load static files like CSS & Images into our Django App

In section 5.2, we have created our **app *doggyDayCareCustomerInterface*** and connected it with our main project ***project1.***

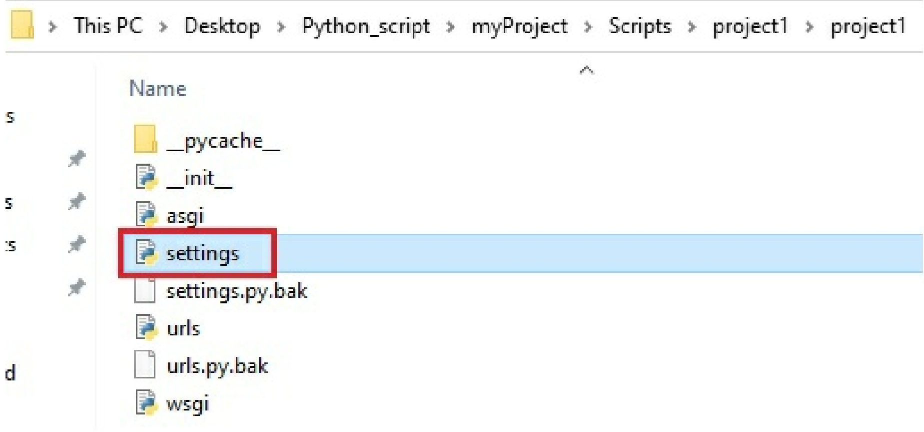
In section 5.3, we created its **models** or tables

In section 5.5, we have learnt how to navigate to different pages.

Now let's learn how to load images and CSS into our ***doggyDayCareCustomerInterface*** **app .**

Before we begin, please note in Chapter 1, we have done **Internal CSS** for ***index.html*** file and **External CSS** for the rest of the pages. The name which we gave to our **External CSS** file was ***commonCss.css***

**Step 1:** Open **settings.py** file of the main project (***project1*** ).

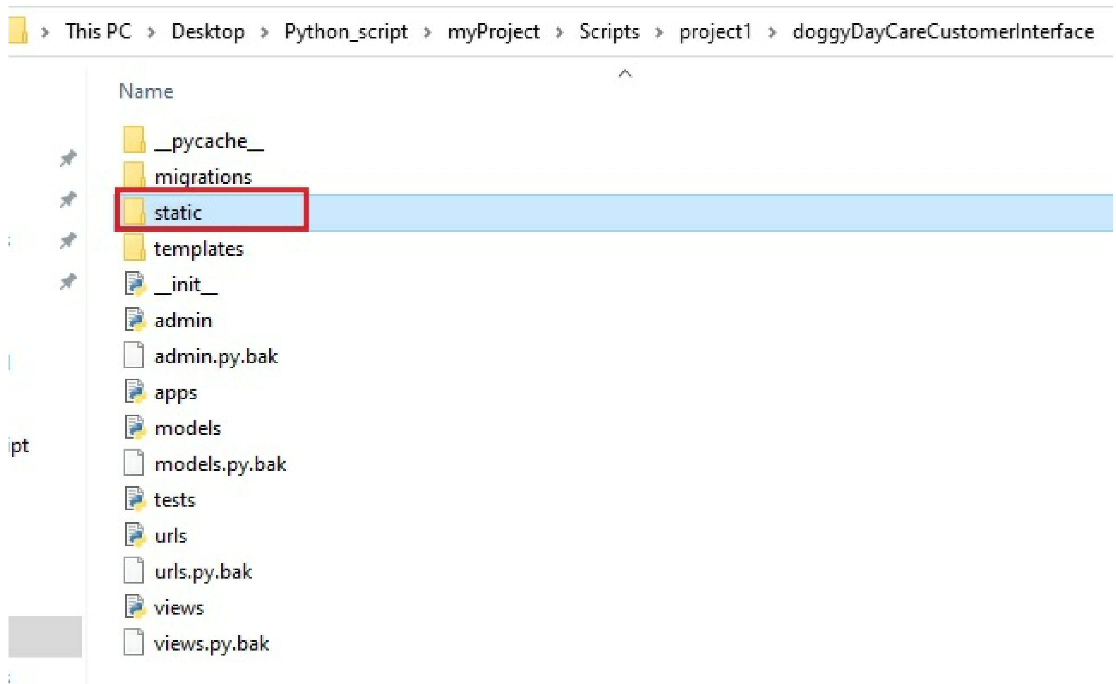



Scroll down and look for the **STATIC_URL** . It says all static files like CSS, Images, Javascript will be stored in **static** .

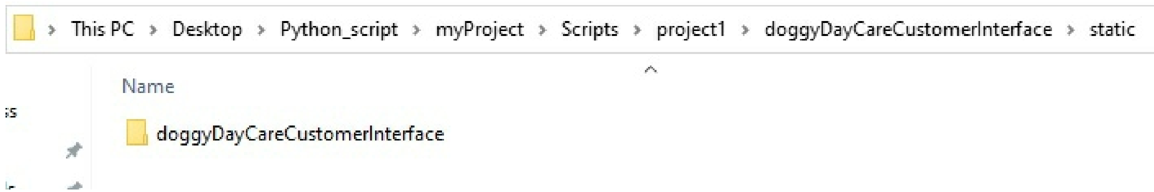
```
115
116  USE_TZ = True
117
118
119  # Static files (CSS, JavaScript, Images)
120  # https://docs.djangoproject.com/en/3.1/howto/static-files/
121
122  STATIC_URL = '/static/'
123
```

So let’s create our **static** folder.

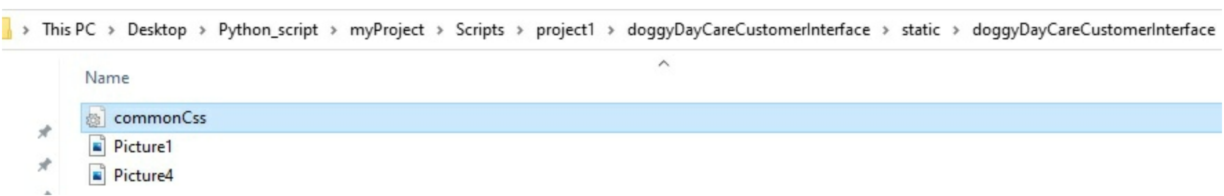
**Step 2:** Open **app** folder (***doggyDayCareCustomerInterface*** ) -> create a new folder within it and name it **static**




**Step 3:** Inside **static** folder create another folder and its name should be same as the **app** ’s name. This folder will contain all **static** files used exclusively by ***doggyDayCareCustomerInterface* app .**




**Step 4:** Add all the images and CSS files into **static** -> ***doggyDayCareCustomerInterface*** folder.




**Step 5:** Open ***index.html*** **(** *Doggy Day Care Project Home Page* **)** file -> in **<head>** section of the HTML document, write the following line of code:

**{% load static %}**

***index.html***

```
<!DOCTYPE html>
<html>
<head>
{% load static %}
<meta name="viewport" content="widtl
<title>Scrappy-Doo Day Care Center</

<style>

    body{
    background-image:url("C:/Users/I
    background-repeat: repeat;
    width:100%;
    height:100%;
```

Then add the static files with the help of syntax :

**{% static ‘ *subdirectory/static_file_name’* %}**

```
body{
background-image:url("{% static "doggyDayCareCustomerInterface/Picture4.jpg" %}");
background-repeat: repeat;
width:100%;
height:100%;
}
```

```
<div id = "homePageHeader">
<img id = "logoPic"
src = "{% static "doggyDayCareCustomerInterface/Picture1.jpg" %}" alt="Logo Pic">
<h1>Scrappy-Doo Day Care Center</h1>
</div>

<div id = "homePageLinks">
<ul>
<li><a href = "{% url 'logInPage' %}">Log In</a></li>
</ul>
</div>
```

**Step 6:** Now let’s load **External CSS** file ***commonCss.css*** into our ***logInPage.html ( ****Log In page* ) file.

**{% load static %}**

**{% static “ *doggyDayCareCustomerInterface/commonCss.css”* %}**

***<u>logInPage.html</u>***

```
<!DOCTYPE html>
<html>
<head>
{% load static %}
<title>Log In - Scrappy-Doo Day Care Center</title>
<link rel="stylesheet" href="{% static "doggyDayCareCustomerInterface/common Css.css" %}">
<style>
```

**How to access a static file (image) from another static file (CSS)?**

**1** . Since both are **static** files are in the same location, we simply have to give the **static** file name (*image name* ) into the CSS **url ( ) function** .

**2** . We do not have to use the following line of code:

{% **load static** %}

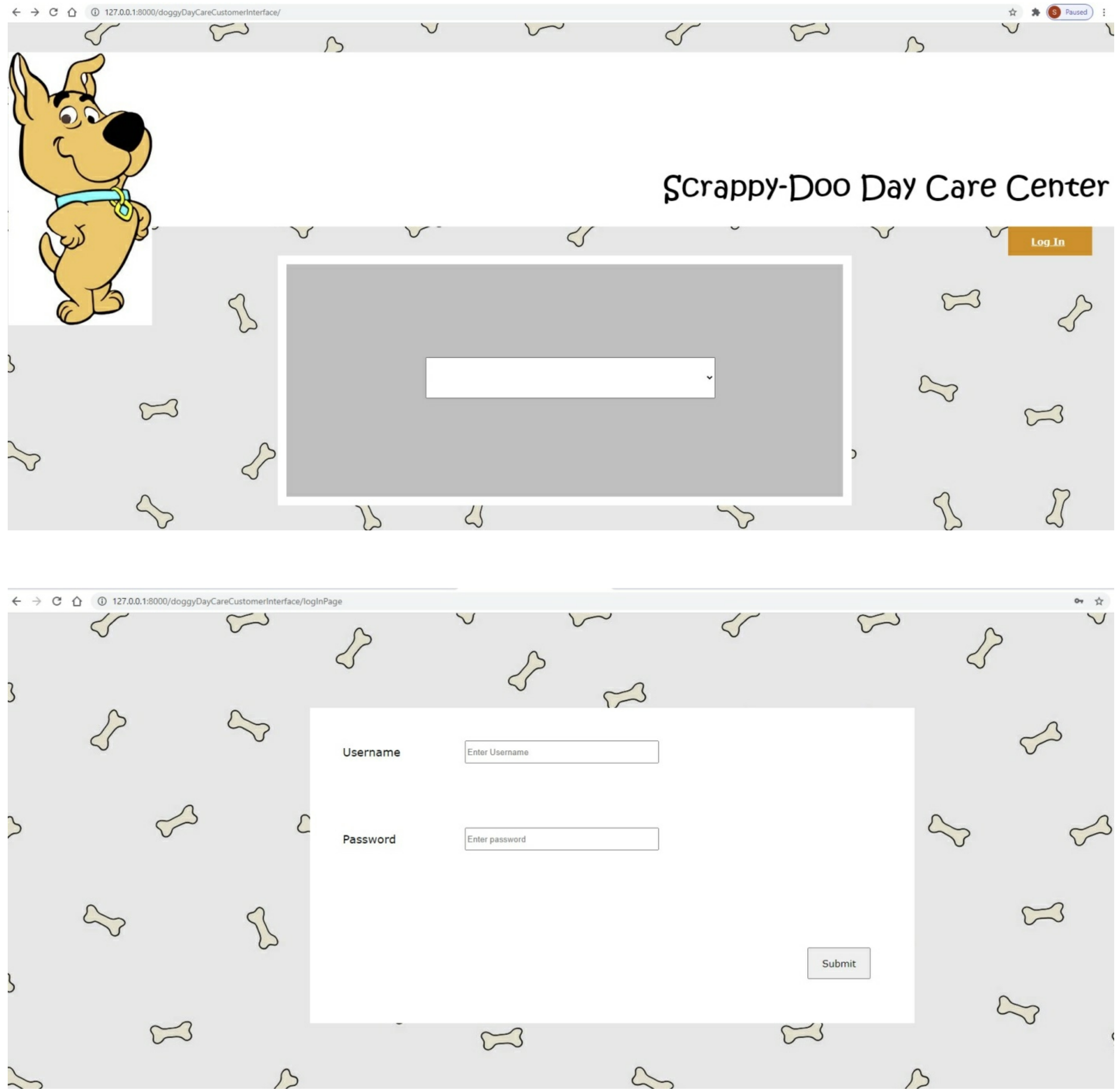
```
body{
    background-image: url("Picture4.jpg");
    background-repeat: repeat;
    width:100%;
    height:100%;
}
```

Now let's run our project. Open command prompt -> navigate to the main project (***project1*** ) location where **manage.py** file is present and run the following command:

**python manage.py runserver**

```
C:\Users\[redacted]\Desktop\Python_script\myProject\Scripts\project1>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
November 03, 2020 - 12:20:01
Django version 3.1.2, using settings 'project1.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Open web browser -> Go to **http://127.0.0.1:8000/doggyDayCareCustomerInterface/**




The images and CSS show perfectly.

**How to add Javascript static file into our HTML document?**

Similar to **External CSS** file, we need to create an **External Javascript** file and save it with Javascript extension **.js** .

Then add the Javscript file into our HTML document. The syntax for this:

```
<script src="{% static 'file_name .js' %}">  </script>
```

## 5.7 : Django Form

Working with Django **form** can be little bit complicated and requires a lot of steps. Well I will try my best to simplify the steps for you all.

Before we go into our ***Log In*** page of ***Doggy Day Care project*** , let’s first understand how Django **form** actually works with a simple example.

Let’s create a simple HTML file within our ***doggyDayCareCustomerInterface*** **app** and name it ***test123.html.***

In ***test123.html,*** I created an input field ***Name*** and a submit button. When the **form** is submitted, I would like to show another HTML file as response which will contains the submitted ***Name*** from the **form** .

***test123.html***

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>

<form>

<label for="name">Name</label>
<input id = "name" type = "text"
placeholder = "Enter name" name = "name">

<br>

<input type = "submit"
name = "submit">
</form>

</body>
</html>
```

The above piece of code shows a HTML **form** with an input field and a submit button. But in Django, this simple **form** will work little differently. So let's follow the steps below of creating a Django **form**

**Step 1:** Open ***doggyDayCareCustomerInterface* app** 's **urls.py** file -> create a path with route name and **url** name ***test*** and which will lead us to a **function *test123*** present in **views.py** file**.**

```
from django.urls import path
from . import views
urlpatterns = [
    path("", views.index, name="index"),
    path("logInPage", views.logInPage, name="logInPage"),
    path("contactUsPage", views.contactUsPage, name = "
    path("test", views.test123, name="test"),
    path("mail", views.mail, name="mail"),
    path("customer", views.customerProfile, name = "cus
```

**Step 2:** Open **views.py** file -> created a **form class** and named it ***test123_validation*** . Inside this **class** , all the **form objects** with its data type are declared.

**Form objects** are simply the input fields present within the **<form>** tag of a HTML document.

**Please Note :** There are multiple form fields present, please visit Django documentation https://docs.djangoproject.com/en/3.2/ref/forms/fields/ for Form fields reference.

```
from django import forms

class test123_validation(forms.Form):
  name = forms.CharField(label = 'Name')

def test123(request):
  return render(request, "doggyDayCareCustomerInterface/test123.html",
  {"form" : test123_validation()
  })
```

**Please note:** Our ***test123.html*** file contains only one input file ***Name*** . So in **class *test123_validation*** , I created only one input form field ***name*** which will hold character values.

**Step 3:** In **views.py** file create the **function *test123*** . Inside the function, all the **form objects** from **class *test123_validation*** are passed into a variable ***form*** and this variable ***form*** is used by ***test123.html*** file**.**

***test123* function** returns the ***test123.html*** file with all the **form objects.**


```
from django import forms

class test123_validation(forms.Form):
  name = forms.CharField(label = 'Name')

def test123(request):
  return render(request, "doggyDayCareCustomerInterface/test123.html",
  {"form" : test123_validation()
  })
```


**Step 4:** Inside ***test123.html*** , we simply have to plug in the ***form*** variable within template tag **{{ … }}** and Django will automatically generate the input fields with its corresponding **<label>** tags

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>

<form>

{{form}}

<input type = "submit"
name = "submit">
</form>

</body>
</html>
```

**What is {{ … }}?**

{{ … }} is a template tag and is used to plug in variables into a HTML document.

**Step 5:** Now let’s add the **action attribute.** Once the **form** is submitted, I would like to navigate to **url** name ***greetings*** . (*In section 5.5, we have learned how to navigate to different pages)*

```html
<!DOCTYPE html>
<html>
<head>
</head>
<body>

<form action = "{% url 'greetings'  %}" method = "post">

{{form}}

<input type = "submit"
name = "submit">
</form>

</body>
</html>
```

**Step 6:** Open **urls.py** file -> create a new path with **url name *greetings*** which will lead us to ***greetings* function** present in **views.py** file.

```python
from django.urls import path
from . import views
urlpatterns = [
    path("", views.index, name="index"),
    path("logInPage", views.logInPage, name="logInPage"),
    path("greetings", views.greetings, name="greetings"),
]
```

**Step 7:** Open **views.py** file and create our ***greetings* function** which will handle the **request** from the user.

```
def  greetings(request):
  if  request.method  =="POST":
    form  =  test123_validation(request.POST)
    if  form.is_valid():
      obj  =  form.cleaned_data["name"]
    return  render(request,
    "doggyDayCareCustomerInterface/greetings.html",
    {"form"  :  obj
    })
```

## Code Explanation:

```
if  request.method  =="POST":
```

*..................if method is post......................*

```
form  =  test123_validation(request.POST)
```

*......take all the values from the form and pass it to a variable named **form**......................................*

```
    if  form.is_valid():
```

*...........**is_valid( )** is used to validate data and returns a boolean value................................*

```
      obj  =  form.cleaned_data["name"]
```

*...........**cleaned_data** attribute can only be used after **is_valid( )** is used and it helps to access values. That value is passed to variable **obj** ...............*

In short it means:

- If the method is **POST** , take the values from the **form** field which belongs to **class** ***test123_validation*** and pass it into a variable ***form.*** If value is present, take that value and store it in variable ***obj*** .

- Now in the last line of the **function *greetings,*** we simply pass the value of ***obj*** into a variable ***form*** and this variable is accessed by ***greetings.html.***
- In ***greetings.html*** , we simply plug in the ***form*** variable.

| **NOTE: cleaned_data** attribute is used to access the value from the input field. |
|---|

***greetings.html***

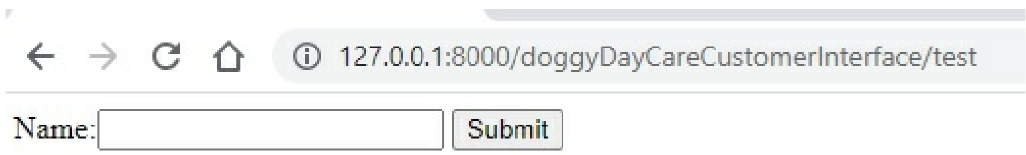
```
<!DOCTYPE html>
<html>
<head>
</head>
<body>

<h1>hello,</h1>

{{form}}

</body>
</html>
```

Now let's run our project. Open command prompt -> navigate to the main project (***project1*** ) folder location where **manage.py** file is present -> run the project using the following command

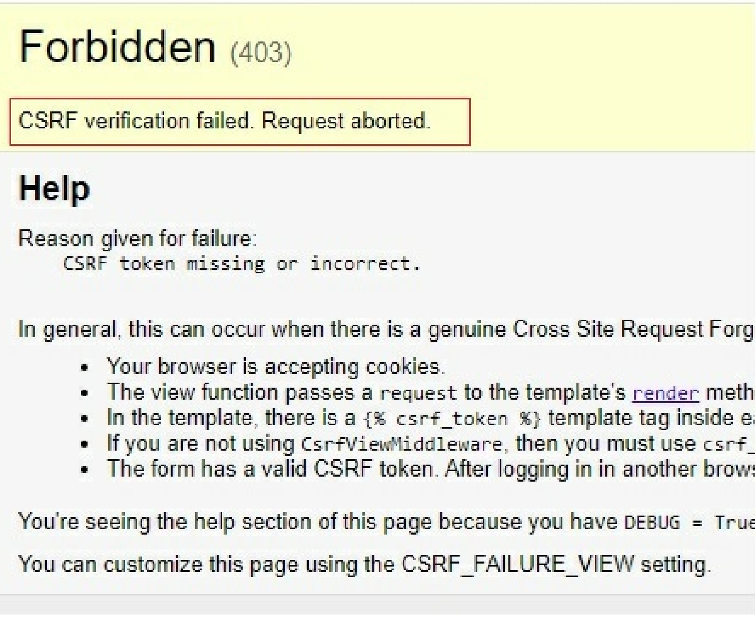
**python manage.py runserver**

```
C:\Users\[redacted]\Desktop\Python_script\myProject\Scripts\project1>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
November 05, 2020 - 11:23:52
Django version 3.1.2, using settings 'project1.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Open web browser -> go to **http://127.0.0.1:8000/doggyDayCareCustomerInterface/test**




After I entered the name and clicked the submit button, an error shows up.




This error appeared because I forgot to add **csrf token** within the **<form>** tag of ***test123.html*** .

**What is csrf token?**

A Django csrf token is a unique value generated exclusively for authenticated users. It MUST be used with all Django forms because it

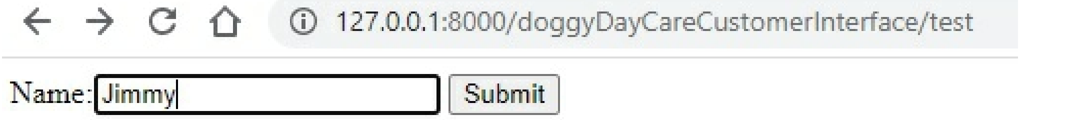
helps to protect the user's information and prevent any kind of malicious activities.

So let's add the **csrf token** within **<form>** tag of ***test123.html*** file and save it**.**

***test123.html***

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>

<form action = "{% url 'greetings'  %}" method = "post">
{% csrf_token %}
{{form}}

<input type = "submit"
name = "submit">
</form>

</body>
</html>
```

Run the project.



127.0.0.1:8000/doggyDayCareCustomerInterface/greetings

**hello,**

Jimmy

Everything works perfectly.

**Let’s summarize the entire process:**

- To work with Django **form** , we need to create the **form class** (***test123_validation)*** .
- This **form class** contains all the input fields of our HTML document (***test123.html*** ).
- From **views.py** -> **function *test123,*** the **form class** is passed into a variable ***form*** which is accessed by ***test123.html*** and in that HTML page we simply plug in the variable. Once the variable is plugged in, Django automatically generates all the input fields.
- Once the **form** is submitted, it takes us to **url** name ***greetings.*** The **request** then proceeds to **urls.py** file **->** ***greetings*** **function** in **views.py** file**.**
- ***greetings*** **function** takes the value from the **form** input field entered by user and displays it in ***greetings.html*** .

In real world, no pages look simple as the example above. Web pages are far more complex and contain CSS, Javascript and lot more. In the next section we will learn how to manually insert Django **form fields** into our HTML file without disturbing the original **DOM** structure.

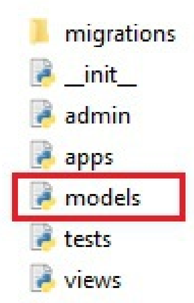
**Example :**

In this example, we will create a database model and insert some data into it. Then we will print out the data from the model into a HTML file.

- Let’s create a new **app** within our main project (***project1*** ). *(I named my new app **cars**)* . Then connect the newly created app (***cars*** ) with the main project (***project1*** ) through **settings.py** and **urls.py** file.

- Setup the **urls.py** file, **views.py** file and **templates** folder.

- Open **models.py** file and create a new model *(I named my model **Cars** ).*




models.py

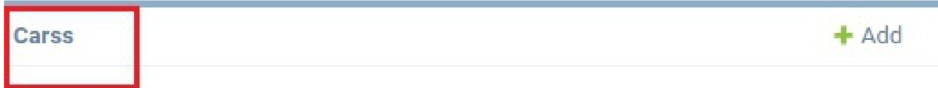
```
from django.db import models

# Create your models here.
class Cars(models.Model):
    car_model = models.CharField(max_length=30)
    car_make = models.CharField(max_length=30)
```

- Start the migration process *(discussed in section 5.3)* .

- Now register the models in **admin.py** file.

```
from django.contrib import admin
from .models import Cars

# Register your models here.
admin.site.register(Cars)
```

- Run the project and visit [http://127.0.0.1:8000/admin/](http://127.0.0.1:8000/admin/) . The newly created ***Cars*** model should be present as shown in the screen shot below.




Let's add some data into it and display the data in a HTML document.

In **views.py** file add the following lines of code *(code discussed in previous sections)* .

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


# Create your views here.

def cars(request):
    car = Cars.objects.all()
    return render(request, "Cars/cars.html", {'car' : car})
```

In ***cars.html*** file display the records using **for loop** .

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Hello</title>
    <style>
    th,td{
    border: 1px solid black;
    padding:10px;
    }
    </style>
</head>
<body>
{% csrf_token %}
   {% for car in car %}
      <table>
        <th>Make</th>
        <th>Model</th>
        <tr>
        <td>{{car.car_make}}</td>
        <td>{{car.car_model}}</td>
        </tr>
      </table>
    {% endfor %}

</body>
</html>
```

**Please Note :** In Django if you start a loop in a template, you need to end the loop.

Example: {% for … %} ….. {% endfor %} ,

{% if … %} ….. {% endif %}

Run the project.

| Make | Model |
| --- | --- |
| Toyota | Corolla |

| Make | Model |
| --- | --- |
| Ford | Expedition |

| Make | Model |
| --- | --- |
| Kia | K900 |

## 5.8 : Django Form with CSS & Rendering form fields manually

Now let’s open our ***doggyDayCareCustomerInterface*** **app** -> go to **templates** -> ***doggyDayCareCustomerInterface*** and open our ***logInPage.html*** file

(*section 5.5 has the steps of creating the* ***templates*** *folder* ).

***logInPage.html***

```
<!DOCTYPE html>
<html>
<head>
<title>Log In - Scrappy-Doo Day Care Center</title>
<link rel="stylesheet" href="commonCss.css">
<style>

td{
   padding:50px;
}

</style>
</head>
<body>

<div id = "containerLogIn">

<table>
<tr>
<td><label for="uname">Username</label></td>
<td><input id = "uname" class = "fieldStyle" type = "text"
placeholder = "Enter Username" name = "username"></td>
</tr>


<tr>
<td><label for="pwd">Password</label></td>
<td><input id = "pwd" class = "fieldStyle" type = "password"
placeholder = "Enter password" name = "password"></td>
</tr>

<tr><input id = "submitButtonLogIn" type = "submit"
name = "submit"></tr>
</table>

</div>

</body>
</html>
```

You will notice two things after going through the code above and they are:

- The page is styled in a table format.
- The input fields belong to CSS **class** ***fieldStyle*** present in ***commonCss.css*** .

If we follow the Django **form** steps in section 5.7 and simply plug in a variable containing all **form** fields, you will notice none of the table format and the CSS will load. In order to prevent that from happening, let's manually and individually render Django **form** fields without disturbing the entire **DOM** structure of the HTML document.

Now we will create the Django **form** for ***Log In*** page.

**Step 1:** Create a path in **urls.py** file which will lead us to our ***Log In*** page. (*we already did this part in section 5.5* ).

**Step 2:** Open **views.py** file -> create a **class** containing all the **form** fields of ***Log In*** page.

```
class  logInPagee_validation(forms.Form):
  username  =  forms.CharField
  (widget=forms.TextInput(attrs={'class'  :  'fieldStyle'}))
  password  =  forms.CharField
  (widget=forms.PasswordInput(attrs={'class'  :  'fieldStyle'}))
```

- **widget** is a Django's representation of an HTML input element.

- **attrs** is used to store HTML attributes. Here, I passed an attribute stating that the **form** input field belongs to CSS **class** ***fieldStyle (*** *present in* ***commonCss.css).***

- **TextInput** allows entering any string value using an input box.

- **PasswordInput** allows entering any password value using an input box.

**Please Note:** If we want to add a **Javscript Event** into our Django **form field** , we can add that in attributes **attrs.** Example:

password = forms.CharField (widget=forms.PasswordInput (attrs = { 'class' : 'fieldStyle', 'onchange' : 'js_function_name( )' }))

Now in **function *logInPage*** , let's pass the above **form** fields into a variable ***form*** which is accessed by ***logInPage.html*** HTML file.

**views.py**

```
def  logInPage(request):
  return  render(request,  "doggyDayCareCustomerInterface/logInPage.html",
  {"form"  :  logInPagee_validation()
  })
```

**Step 3:** Open ***logIn.html*** file -> in place of **<input>** tags, I am rendering the Django **form** fields manually with the help of dot (.) operator.

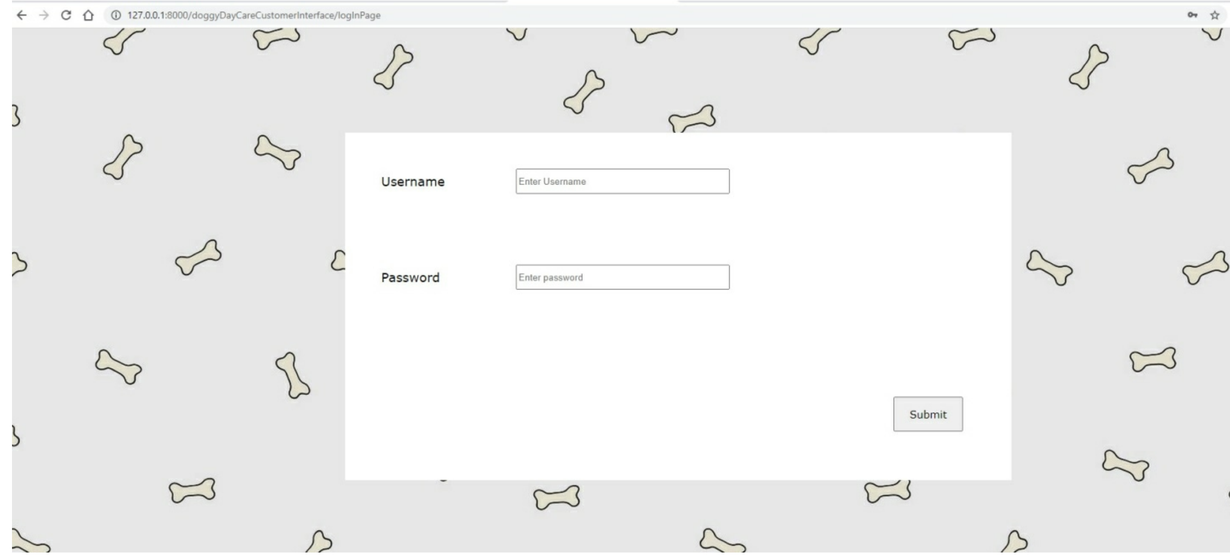
***logIn.html***

```
<form action = "{% url 'customer'  %}" method = "post">
{% csrf_token %}
<div id = "container">
<table>
<tr>
<td>
<label for="{{ form.username.id_for_label }}">Username:</label>
</td>
<td>
{{form.username}}
</td>
</tr>

<tr>
<td>
<label for="{{ form.password.id_for_label }}">Password:</label>
</td>
<td>
{{form.password}}
</td>
</tr>
<tr>
<input id = "submitButtonLogIn" type = "submit"
name = "submit"></tr>
</table>
</div>
</form>
```

**NOTE:** When **form fields** are rendered manually, Django will not generate its corresponding **<label>** tag automatically. We need to manually add the **label** for its corresponding field and to do that **Id_for_label** is used.

.**Id_for_label** is used if we are constructing labels for the fields manually .

Save everything and check whether the new changes made in our ***logInPage.html*** works properly or not**.**




Everything looks good.

Let’s proceed to performing the validation process in the next section.

# 5.9 : Log In Page Validation process

In section 5.3, we have created **models** or database tables for our ***doggyDayCareCustomerInterface*** **app**

In section 5.8, we have created the **form** for our ***Log In*** page**.**

Now let’s perform the validation process.

Please note when the **form** is submitted from ***logInPage.html (*** *created in section 5.8* ***),*** it leads us to **url** name ***customer*** as shown in the screen shot below.

```
<form action = "{% url 'customer'  %}" method = "post">
{% csrf_token %}
<div id = "container">
<table>
<tr>
<td>
```

**Step 1:** Open **urls.py** file -> create a path with **url** name, **route** name ***customer*** and which will lead to **function** ***customerProfile*** present in **views.py** file

```
from django.urls import path
from . import views
urlpatterns = [
    path("", views.index, name="index"),
    path("logInPage", views.logInPage, name="logInPage"),
    path("test", views.test123, name="test"),
    path("greetings", views.greetings, name="greetings"),
    path("customer", views.customerProfile, name = "customer")
]
```

**Step 2:** Open **views.py** file and let's create our **function** ***customerProfile*** . The function will take the **request** from the user, check whether the entered username and password matches with its corresponding database records or not and then throw a **response** .

```
from .models import customer_information
from django.http import HttpResponseRedirect
def customerProfile(request):
  if request.method =="POST":
    form = logInPagee_validation(request.POST)
    if form.is_valid():
      obj1 = form.cleaned_data["username"]
      obj2 = form.cleaned_data["password"]
      table_val=customer_information.objects.filter(username = obj1, password = obj2)
      if table_val:
        row = table_val.first()
        return render(request,
            "doggyDayCareCustomerInterface/customerProfile.html",
              {"form" : row})
      else:
        return HttpResponseRedirect("logInPage")
```

## Code Explanation:

In section 5.7, we have discussed lines of code from:

*if request.method =="POST"*

to

*obj2 = form.cleaned_data["password"] .*

Now let’s discuss the code from line:

*table_val = customer_information.objects.filter(username = obj1, password = obj2)*

.

```
table_val=customer_information.objects.filter(username = obj1, password = obj2)
```

The above line of code is a database query where it looks for a record in ***customer_information*** **model** or table whose ***username*** = ***obj1*** and ***password*** = ***obj2*** .

We are all familiar with database **select** queries like:
**select * from** *table_name* **where** *column_name = “value”* **.**

But in Django the syntax of the above **select** query is little different. The Django equivalent of the above **select** query is:

*table_name* **. objects . filter (** *column_name = "value"* **)**

You can also use **get( )** method in place of **filter( )** method.

**NOTE :** The Django equivalent of **select * from** *table_name* is:

*table_name* **. objects . all ( )**

- In the above piece of code, the result from the query is stored into a variable ***table_val*** .

***customer_information*** is present in **models.py** file. To access the table in **views.py** file, first we need to import the table from the **models.py** file.

```
from .models import customer_information
```

Please follow the screen shot below for line by line code explanation.

```
if table_val:
```

*..........pythonic way of checking value is present or not. If value is present TRUE, if not then it returns FALSE..........*

```
    row = table_val.first()
```

*........first( ) returns the first row and stores the value in* ***row*** *.................*

```
    return render(request,
        "doggyDayCareCustomerInterface/customerProfile.html",
         {"form" : row})
```

*.......passed* ***row*** *into another variable* ***form*** *which will be used by customerProfile.html..........................*

In short it means,

Take the data from the above query and store it in variable ***table_val.*** If value is present, take the first row with the help of **first( ) method** and store

the data into another variable ***row.*** Now pass that ***row*** variable into another variable ***form*** which will be accessed by HTML file ***customerProfile.html*** .

The above steps are performed if the **form** is valid. Now if the **form** is not valid then **redirect** back to the **url** name ***logInPage*** .

```
else:
    return HttpResponseRedirect("logInPage")
```

To use **HttpResponseRedirect** , we need to import it first.

```
from django.http import HttpResponseRedirect
```

Now in ***customerProfile.html*** , let's display the data from ***customer_information*** **model** or table**.**

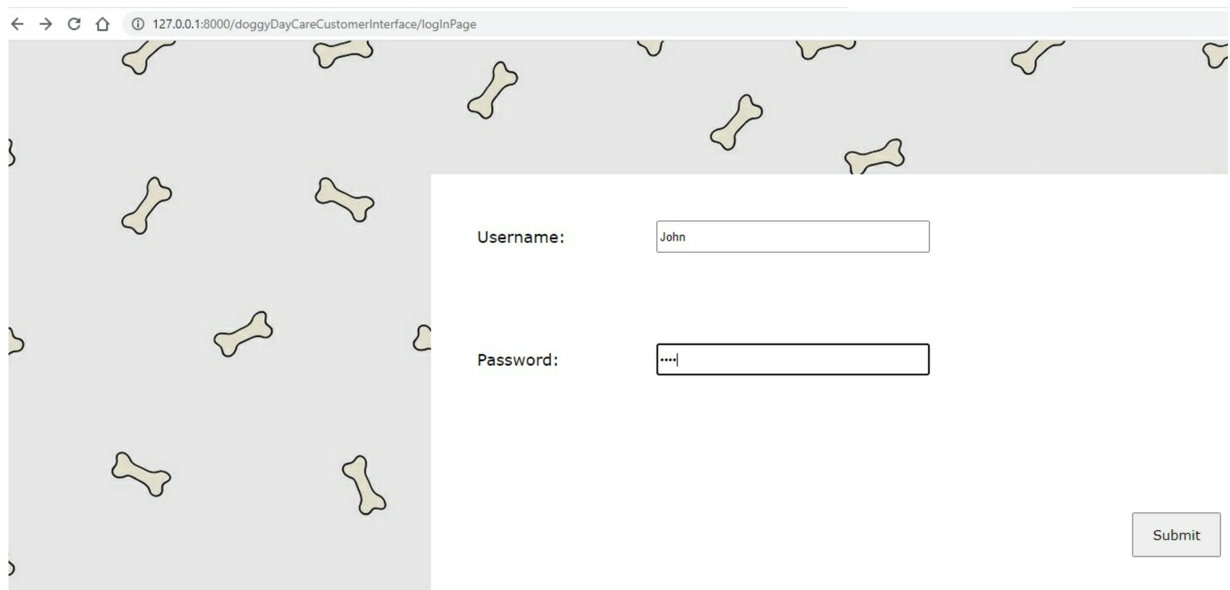
***<u>customerProfile.html</u>***

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>

<h1>hello,</h1>
{{form.customer_id}}
<br>
{{form.firstName}}
<br>
{{form.lastName}}
<br>
{{form.gender}}
<br>
{{form.address}}
<br>
{{form.state}}

</body>
</html>
```

Now run the project. Open command prompt -> navigate to the main project (***project1*** ) folder location where **manage.py** file is present -> run the project using the following command

**python manage.py runserver**

Open web browser -> go to **http://127.0.0.1:8000/doggyDayCareCustomerInterface/ logInPage**



127.0.0.1:8000/doggyDayCareCustomerInterface/customer

**hello,**

123
John Jim
Kim
M
Cedar Park
Texas

The validation works perfectly.

## 5.10 : Displaying data from multiple tables or models into Web Page

In section 5.3, we have created the **models** for our ***doggyDayCareCustomerInterface*** **app** .

In section 5.9, we have learned how to display information into our web page from one table or **model** .

Now in this section, we will learn how to display information from multiple tables or **models** .

Few important points to note are:

- We are all familiar with **SQL Join** . SQL Join helps to retrieve information from multiple tables. Django also gives us the ability

to retrieve data from multiple **models** or tables but the syntax is different.

- In **models.py** file of our ***doggyDayCareCustomerInterface* app** , the ***dog_information* model** or table is connected with ***customer_information* model** or table with the help of **foreign key** declared in column ***dog_id*** .

Due to this relationship, the ***dog_id*** column will display associated data from ALL columns of ***customer_information* model** . Instead of displaying data from all columns, I would like to display data from only one column ***customer_id*** and to do that we need the **__init__ function (*__init__ function* *discussed in chapter 4)*** .

So let's add the **__init__ function** into our ***customer_information* class**

- Open **models.py** file -> in **class *customer_information*** , declare the **__init__ function** which will return only the ***customer_id*** when the **object** of the **class** is created.

```
class  customer_information(models.Model):
    customer_id  =  models.IntegerField()
    username  =  models.CharField(max_length=64)
    password  =  models.CharField(max_length=20)
    firstName  =  models.CharField(max_length=64)
    lastName  =  models.CharField(max_length=64)
    gender  =  models.CharField(max_length=1)
    address  =  models.CharField(max_length=64)
    state  =  models.CharField(max_length=64)
    def  __str__(self):
      return  f"{self.customer_id}"
```

**What is f "…"?**

**f "…"** is a Python formatted string literal that contains expressions within { } and these expressions are replacement fields.

- Now let's open **views.py** file and go to **function *customerProfile( )*** which we created in the section 5.9.

In the same function, I added a new line of code:

```
def customerProfile(request):
 if request.method =="POST":
  form = logInPagee_validation(request.POST)
  if form.is_valid():
   obj1 = form.cleaned_data["username"]
   obj2 = form.cleaned_data["password"]
   table_val=customer_information.objects.filter(username = obj1, password = obj2)
   val2 = dog_information.objects.filter(dog_id__username = obj1)
   if table_val and val2:
    row1 = table_val.first()
    row2 = val2.first()
    return render(request,
        "doggyDayCareCustomerInterface/customerProfile.html",
         {"form1" : row1, "form2" : row2 })
   else:
    return HttpResponseRedirect("logInPage")
```

In the above highlighted query I am performing a **Join** , I access the ***username*** field (*present in **customer_information models*** ) from ***dog_id*** column and look for a record whose ***username*** = ***obj1*** . Then I store the result of the query in ***val2.***

*Code explanation of **function customerProfile( )** is present in sections 5.9.*

Now let's update our ***customerProfile.html*** file and plug in the variables.

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>

<h1>hello,</h1>


ID: {{form2.dog_id}}

<br><br>

Username: {{form1.username}}

<br><br>

State: {{form1.state}}

<br><br>

{{form2.dog_name1}} and age is {{form2.dog_age1}}

<br><br>

{{form2.dog_name2}} and age is {{form2.dog_age2}}


</body>
</html>
```

In ***ID*** , I am displaying the data from ***dog_id*** column which in turn will display its associated data from only one column ***customer_id*** and this was possible because of the presence of **__init__ function** .

127.0.0.1:8000/doggyDayCareCustomerInterface/customer

# hello,

ID: 123

Username: John

State: Texas

Kia and age is 5

Taco and age is 5

## 5.11 : Django Email Form

We created our ***Contact Us*** page in Chapter 2.

Now let's follow the steps below and make our ***contactUsPage.html*** file Django compatible.

**Step 1:** Open the ***doggyDayCareCustomerInterface* app** 's ***index.html*** (***Home page*** of ***Doggy Day Care project*** ) file and give an **url** name (*page navigation discussed in section 5.5* )

```
<option class = "optionVal"
value="{% url 'contactUsPage' %}">Contact Us</option>
```

**Step 2:** Open **urls.py** file and create a path that will lead us to ***contactUsPage*** function present in **views.py** file.

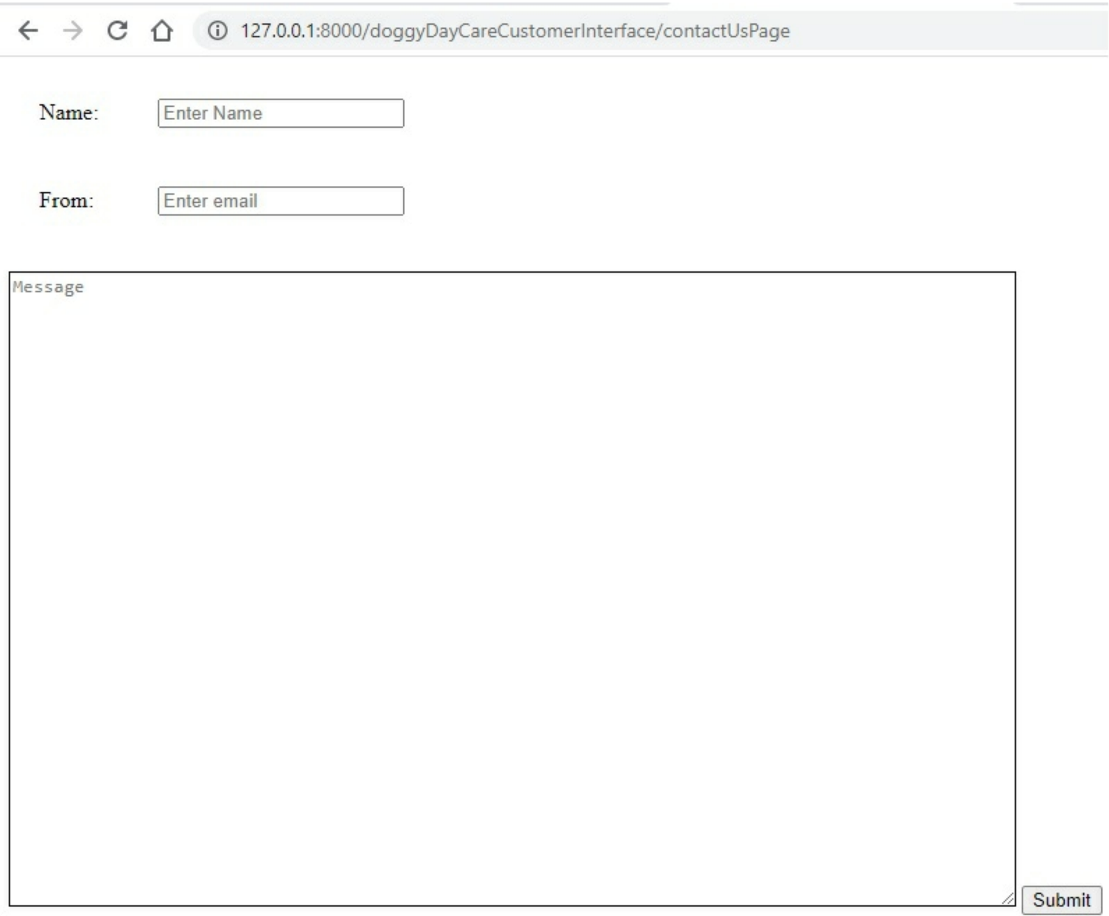
**urls.py**

```
from django.urls import path
from . import views
urlpatterns = [
    path("", views.index, name="index"),
    path("logInPage", views.logInPage, name="logInPage"),
    path("contactUsPage", views.contactUsPage, name = "contactUsPage"),
    path("test", views.test123, name="test"),
    path("greetings", views.greetings, name="greetings"),
    path("customer", views.customerProfile, name = "customer")


]
```

**Step 3:** Open **views.py** file and create the **function** ***contactUsPage( ).*** This function will simply return the ***contactUsPage.html*** page**.**

**views.py**

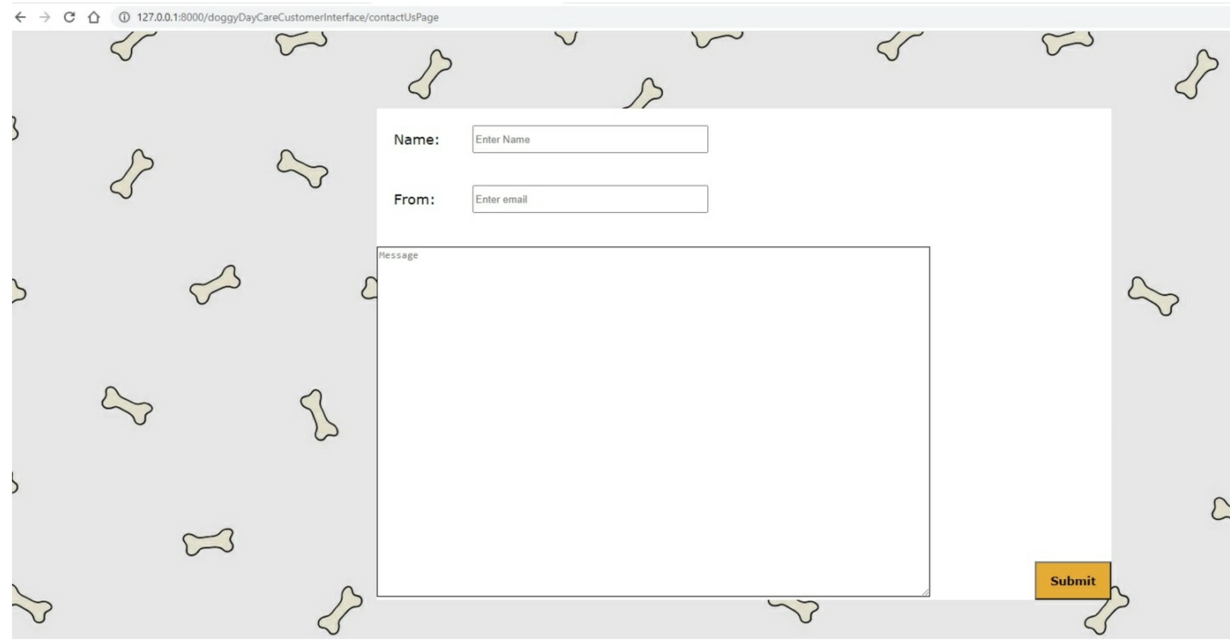
```
def contactUsPage(request):
  return render(request, "doggyDayCareCustomerInterface/contactUsPage.html")
```




**Step 4:** Our web page appeared without any styling information so let's load the styling information into our HTML file (*discussed in section 5.6* )

***contactUsPage.html***

```
<!DOCTYPE html>
<html>
<head>
{% load static %}
<link rel="stylesheet" href="{% static 'doggyDayCareCustomerInterface/common_Css.css' %}">

<title>Contact Us - Scrappy-Doo Day Care Center</title>
<link rel="stylesheet" href="commonCss.css">
<style>
td{
padding:20px;
}
```




**Step 5:** Now let's create the **form** . (*Django **form** discussed in section 5.7, 5.8 & 5.9* )

Open **views.py** file -> create a new **class** which will contain all the **form** fields of ***Contact Us*** page.

```
class contactUs(forms.Form):
  name = forms.CharField(widget=forms.TextInput(attrs={'class' : 'fieldStyle'}))
  sender = forms.CharField(widget=forms.TextInput(attrs={'class' : 'fieldStyle'}))
  message = forms.CharField(widget=forms.Textarea(attrs={'rows':15, 'cols':100}))
```

Now create the function ***contactUsPage*** and pass the **form** fields into our ***contactUsPage.html*** file.

```
def contactUsPage(request):
  return render(request, "doggyDayCareCustomerInterface/contactUsPage.html",
  {"form" : contactUs()
  })
```

**Step 6:** Open ***contactUsPage.html*** file and render the **form** fields manually (*discussed in section 5.8* ).

```
</head>
<body>
<form action = "{% url 'mail'  %}" method = "post">
{% csrf_token %}
<div id = "containerContactUs">
<table>
<tr>
<td><label for="{{ form.name.id_for_label }}">Name:</label></td>
<td>{{form.name}}</td>
</tr>
<tr>
<td><label for="{{ form.sender.id_for_label }}">From:</label></td>
<td>{{form.sender}}</td>
</tr>
</table>
<br>
{{form.message}}
<input id = "submitButtonContactUs" type = "submit"
name = "submit">
```

**Step 7:** In ***contactUsPage.html,*** add **action** attribute to **<form>** tag.

Once the **form** is submitted, it will take us to **url** name ***mail*** .

So let’s open our **urls.py** file and create a path with the **url** name and **route** name as ***mail*** and which will lead us to **function** ***mail*** present in **views.py** file.

**urls.py**

```
from django.urls import path
from . import views
urlpatterns = [
    path("", views.index, name="index"),
    path("logInPage", views.logInPage, name="logInPage"),
    path("contactUsPage", views.contactUsPage, name = "contactUsPage"),
    path("test", views.test123, name="test"),
    path("mail", views.mail, name="mail"),
    path("customer", views.customerProfile, name = "customer")

]
```

**Step 8:** Open **views.py** file and create the function ***mail( )*** .

## views.py

```
from django.core.mail import send_mail
from django.conf import settings
def mail(request):
  if request.method =="POST":
    form = contactUs(request.POST)
    if form.is_valid():
      name = form.cleaned_data["name"]
      email_from = settings.EMAIL_HOST_USER
      message = form.cleaned_data["message"]
      if name and message:
        send_mail(name, message, email_from, ['basu.sanjana1619@gmail.com'])
        return render(request,
                 "doggyDayCareCustomerInterface/mail.html")
      else:
        return render(request,
                 "doggyDayCareCustomerInterface/contactUsPage.html")
```

## Code Explanation:

We have already discussed lines of code from:

*if request.method =="POST"*

to

*name = form.cleaned_data["name"]* (*discussed in section 5.8, 5.9 and 5.10*).

Now let’s discuss the lines of code from:

*email_from = settings.EMAIL_HOST_USER*

But before we understand the above line of code, let’s first open the main project (***project1*** ) folder and go to **settings.py** file. In **settings.py** file, insert the following lines of code.

```
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_USE_TLS = True
EMAIL_USE_SSL = False
EMAIL_PORT = 587
EMAIL_HOST_USER = '[redacted]'
EMAIL_HOST_PASSWORD = '[redacted]'
```

- **EMAIL_BACKEND** contains the backend to use for sending emails. Its default value is 'django.core.mail.backends.smtp.EmailBackend'
- **EMAIL_HOST** contains the host to use for sending email. Its default value is 'localhost'
- **EMAIL_HOST_USER** contains the account name from which we will send an email
- **EMAIL_HOST_PASSWORD** contains the password of the email account that we are using to send emails.
- **EMAIL_PORT** contains the port used by the SMTP server.

> **What is SMTP server?**
>
> SMTP stands for Simple Mail Transfer Protocol and SMTP server helps to send and receive email between senders and receivers.

- **EMAIL_USE_TLS** contains a Boolean value true or false. This setting specifies whether to use an explicit TLS (secure) connection when talking to the SMTP server.

> **What is TLS?**
>
> TLS stands for Transport Layer Security that helps to encrypts data sent over the Internet to prevent malicious activities by hackers.

- **EMAIL_USE_SSL** contains a Boolean value true or false . This setting specifies whether to use an implicit TLS (secure) connection when talking to the SMTP server .

> **What is SSL?**
>
> SSL stands for Secure Sockets Layer. It helps to encrypt communication between a web browser and a web server.

> **Please Note:** Between **EMAIL_USE_TLS** and **EMAIL_USE_SSL** , only set one of those settings to **True** .

Now let’s proceed with our code discussion from line:

*email_from = settings.EMAIL_HOST_USER*

This line means from **settings.py** file access the **EMAIL_HOST_USER** value and store it in ***email_from*** variable.

Please follow the screen shot below for code explanation:

```
email_from = settings.EMAIL_HOST_USER
```

..........*take the host username and store it in* ***email_from***.........

```
if name and message:
  send_mail(name, message, email_from, ['[redacted]'])
  return render(request,
        "doggyDayCareCustomerInterface/mail.html")
```

.....*if values in* ***name***, ~~*email_from*~~ *and* ***message*** *is present then send the email using* ***send_mail*** *function and also return a* ***mail.html*** *page....*

***mail.html***

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>

mail send

</body>
</html>
```

If mail is send successfully, the **function** ***mail ( )*** will throw ***mail.html*** file to user

## What is Django send_mail function?

**send_mail function** is used to send mail. The syntax is:

**send_mail (**
*'Subject'* **,** *'Message '* **,** *'from@example.com'* **, [** *'to@example.com'* **]**

)

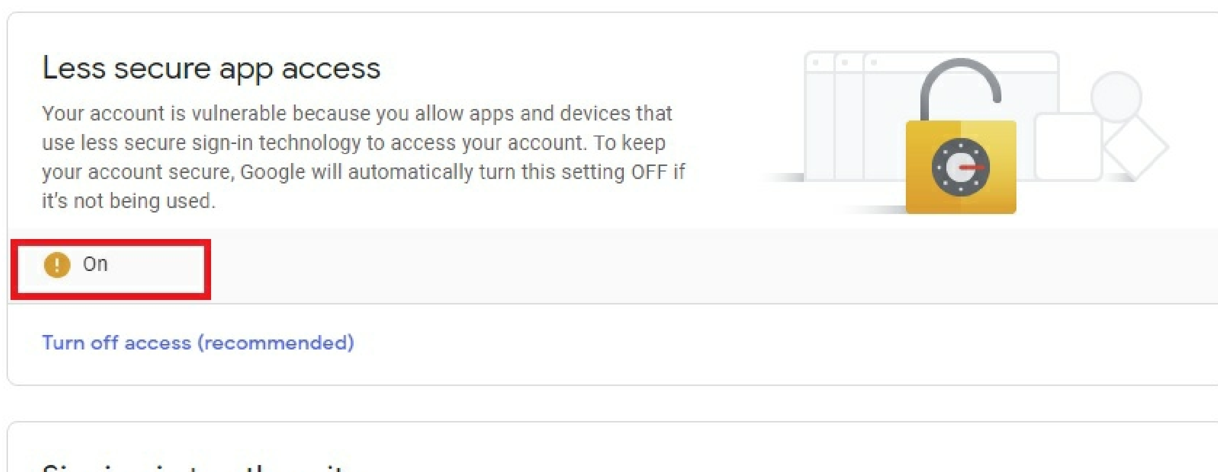
While using the Django **send_mail function** , it is very important to maintain the order of the syntax. The ***subject*** (a string value) should come first, then ***message*** *(* a string value*)* , then ***from _ email*** *(* a string value*)* , then ***recipient_email*** *(* a **list** of string value*)* .

Before using Django **send_mail** function we need to import it from **django.core.mail** module as shown in the screen shot below.

```
from django.core.mail import send_mail
from django.conf import settings
def mail(request):
  if request.method =="POST":
    form = contactUs(request.POST)
```

**Step 9:** I would like to send and receive emails using **Gmail** . Since our web application (***Doggy Day Care project*** ) is not a Google registered service so it falls in the category of less secured app. In order to access **Gmail** through my web application, we need to **turn on** the less secure app access.

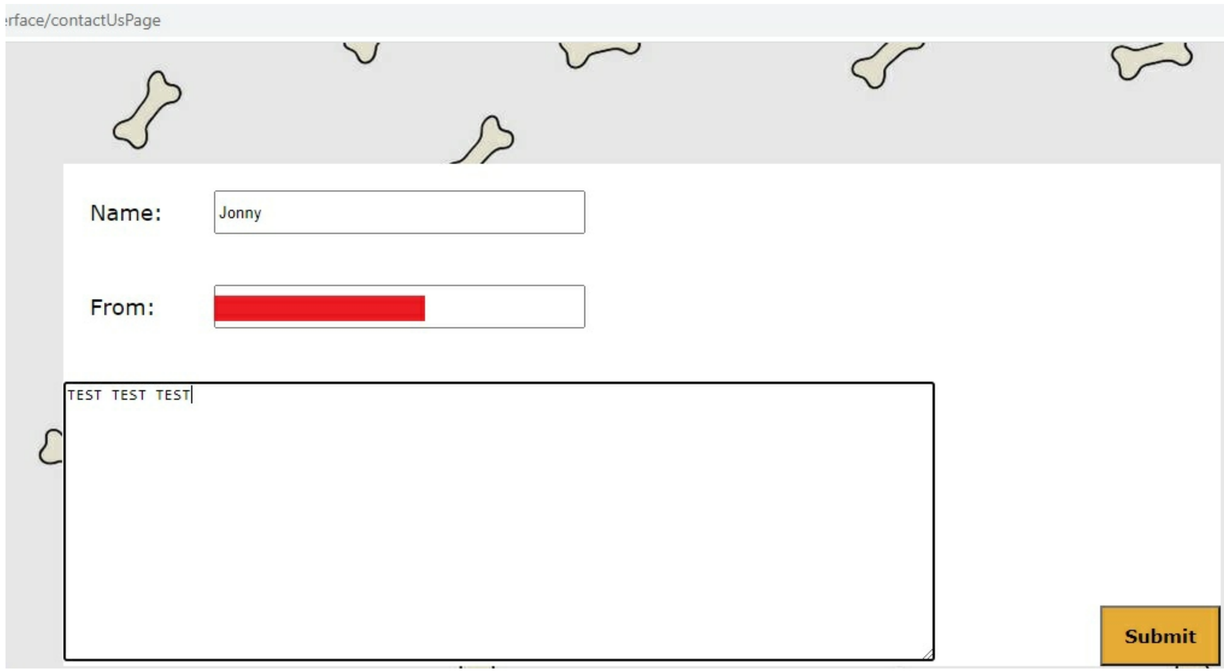
Go to your Google Settings [https://myaccount.google.com/security](https://myaccount.google.com/security) -> **turn on** the Less secure app access**.**



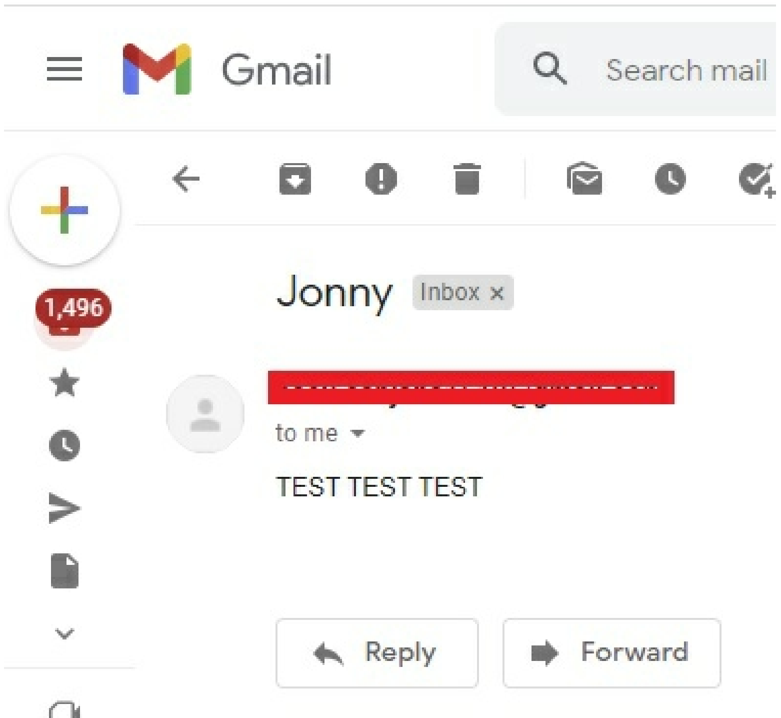

***Please note:*** *Since we are turning on the less secure app access. I recommended not using your main Gmail account for it.*

**Step 10:** Now let's run our project.

Now go to **http://127.0.0.1:8000/doggyDayCareCustomerInterface/contactUsPage**



The mail was sent successfully.



# Important Info & Conclusion:

Django can be bit confusing, but it becomes lot easier to understand through coding and lot of practice.

In this book, we have successfully converted our **Doggy Day Care Center project** into a **Django Web Application** . I have covered the **Home Page** , **Log In Page** and the **Contact Us** page and would like to leave the **Create a New profile page** for you all to practice. Well, I can give three hints:

- The syntax for inserting new data into a model is:

**model_name ( column1 = “** data **”, column2 = “** data **”, column3 =** 123 **……… )**

*(stated in section 5.3)*

- **cleaned_data** attribute is used to access values from a **form** input field.

(please refer to previous sections)

- As you may recall, we created few drop down lists with the help of **<select>** tag in **createNewProfile.html** (code present in chapter 2, section 2.3). To create a similar drop down list in **Django form** we should follow the steps below:

**Step 1:** Create a **list** of choices. This **list** is a collection of **tuples** and these tuple is made of **key-value pairs** .

Example:

```
STATE= [
    ('cal', 'California'),
    ('ny', 'New York'),
    ('tx', 'Texas'),
    ]
```

**Step 2:** Then we pass the **list** of **choices** into our **Django form field** for drop down menu.

Example:

```
state = forms.CharField(widget=forms. Select (choices=STATE))
```

- **Select** allows selecting any string value from a list of **choices** .

---------------------------------------

*Always remember, the most important learning is Self-Learning..*

*Wish you all the best and thank you very much for buying this book.*