MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
A Constituent Institution of Manipal University

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CERTIFICATE

This is to certify that Ms./Mr. …………………...……………………………………

Reg. No. …..…………………… Section: ……………… Roll No: ………………... has

satisfactorily completed the lab exercises prescribed for WEB PROGRAMMING

LAB [CSE 3243] of Third Year B. Tech. (Computer Science and Engineering) Degree at

MIT, Manipal, in the academic year 2024-2025.

Date: ……..................................

<div align="right">

Signature
Faculty in Charge

</div>

CONTENTS

Course Objectives

- Acquire in-depth understanding of web application architecture.
- Understand techniques to improve user experience in web applications.
- Gain knowledge about how to interact with database.

Course Outcomes

At the end of this course, students will be able to

- Develop a basic website using a modern web development tool.
- Design websites with a better look and feel.
- Create real-world web applications that interact with database.

Evaluation plan

- Internal Assessment Marks: 60%

  ➢ Continuous Evaluation: 25%

    Continuous evaluation component (for each evaluation):3 marks
    The assessment will depend on punctuality, program execution, and the
    ability to upload the solution on time.

  ➢ Project Evaluation: 25%

  ➢ Viva Voce: 10%

- End semester assessment of two-hour duration: 40 %
- Total (Internal assessment + End semester assessment): 100 marks

INSTRUCTIONS TO THE STUDENTS

Pre- Lab Session Instructions

1. Students should carry the Lab Manual Book and the required stationery to every lab session
2. Be in time and follow the institution dress code
3. Must Sign in the log register provided
4. Make sure to occupy the allotted seat and answer the attendance
5. Adhere to the rules and maintain the decorum
6. Students must come prepared for the lab in advance

In- Lab Session Instructions

- Follow the instructions on the allotted exercises
- Show the program and results to the instructors on completion of experiments
- On receiving approval from the instructor, copy the program and results in the Lab record
- Prescribed textbooks and class notes can be kept ready for reference if required

General Instructions for the exercises in Lab

- Implement the given exercise individually and not in a group.
- The programs should meet the following criteria:
  - Programs should be interactive with appropriate prompt messages, error messages if any, and descriptive messages for outputs. o Observation book should be complete with program, proper input output clearly showing the parallel execution in each process.
- Plagiarism (copying from others) is strictly prohibited and would invite severe penalties in evaluation.
- The exercises for each week are divided into three sets:
  - Solved example
  - Lab Assignments - to be completed during lab hours
  - Home Assignments - to be completed outside the lab or in the lab to enhance the skill
- In case a student misses a lab class, he/ she must ensure that the experiment is completed during the repetition lab with the permission of the faculty concerned but credit will be given only to one day's experiment(s).
- Questions for lab tests and examinations are not necessarily limited to the questions in the manual but may involve some variations and/or combinations of the questions.

- A sample note preparation is given as a model for observation.

THE STUDENTS SHOULD NOT

- Bring mobile phones or any other electronic gadgets to the lab.
- Go out of the lab without permission.

Lab No: 1   HTML and CSS BASICS

LAB ASSIGNMENTS:

1. Write a simple poem and represent it as a web page. Give a title for the poem. Make rhyming words in the poem as bold.

2. Assume you have brought a new car. Write down the list of additional accessories you need for the car as an unordered list in HTML. Also, list the travel plans on the car as an ordered list.

3. Complete the following website name "Share your Winter Vacation Videos". The required video files are located inside the compressed folder "Week 1 Assignment Files/Media/".
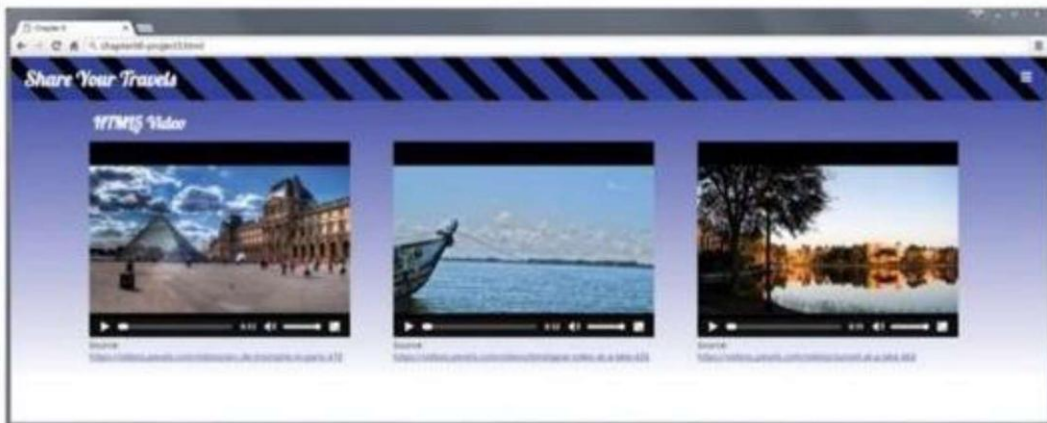
Hint: Add a <video> element to a <figure> element that will either play   paris.mp4, paris.webm or paris.ogv in the element. Test in different browsers

HOME ASSIGNMENTS:

1. Create the following output in HTML
2. Create an array of JavaScript objects for the data in question 1 and display each row in the table form through JavaScript code

Country	Population (In Crores)	
INDIA	1998	85
	1999	90
	2000	100
USA	1998	30
	1999	35
	2000	40
UK	1998	25
	1999	30
	2000	35

Lab No: 2   JQuery

Objectives:

In this lab, student will be able to

1. Develop responsive web pages using jquery
2. Familiarize with DOM manipulation and animations

jQuery

jQuery is a fast and concise JavaScript library to develop web based application.

Here is the list of important core features supported by jQuery −

- *DOM manipulation* − The jQuery made it easy to select DOM elements, negotiate them and modifying their content by using cross-browser open source selector engine called Sizzle.

- *Event handling* − The jQuery offers an elegant way to capture a wide variety of events, such as a user clicking on a link, without the need to clutter the HTML code itself with event handlers.

- *AJAX Support* − The jQuery eases developing a responsive and feature rich site using AJAX technology.

- *Animations* − The jQuery comes with plenty of built-in animation effects which you can use in your websites.

- *Lightweight* − The jQuery is very lightweight library - about 19KB in size (Minified and gzipped).

- *Cross Browser Support* − The jQuery has cross-browser support, and works well in IE 6.0+, FF 2.0+, Safari 3.0+, Chrome and Opera 9.0+

- *Latest Technology* − The jQuery supports CSS3 selectors and basic XPath syntax.

You can download jQuery library from https://jquery.com/download/ on your local machine and include it in your HTML code.
Solved Example:
<html>

```
<head>
  <title>The jQuery Example</title>
  <script type = "text/javascript" src = "jquery-3.4.1.js">
  </script>
  <script type = "text/javascript" language = "javascript">
    $(document).ready(function() {
      $("div").click(function() {alert("Hello, world!");});
    });
  </script>
</head>
<body>
  <div id = "mydiv">
    Click on this to see a dialogue box.
  </div>
</body>
</html>
```

A good rule of thumb is to put your JavaScript programming (all your <script> tags) after any other content inside the <head> tag, but before the closing </head> tag. The *$(document).ready()* function is a built-in jQuery function that waits until the HTML for a page loads before it runs your script.

When a web browser loads an HTML file, it displays the contents of that file on the screen and also the web browser remembers the HTML tags, their attributes, and the order in which they appear in the file—this representation of the page is called the *Document Object Model*, or DOM for short.

Selector: jQuery offers a very powerful technique for selecting and working on a collection of elements—CSS selectors. The basic syntax is like this:
$('selector') use a CSS class
selector like this:
$('.submenu')

```
    <script type = "text/javascript" language = "javascript">
      $(document).ready(function() {
        $("p").css("background-color", "yellow");
        $("#myid").css("background-color", "red");
      });
    </script>
  </head>
```

```
   <body>
     <div>
        <p class = "myclass">This is a paragraph.</p>
        <p id = "myid">This is second paragraph.</p>
        <p>This is third paragraph.</p>
     </div>
</body>
```
We can select tag available with the given class in the DOM. For example $('.someclass')
selects all elements in the document that have a class name as some-class.

Get And Set Atrributes:
```
<script type = "text/javascript" language = "javascript">
      $(document).ready(function()        {
var title = $("p").attr("title");
        $("#divid").text(title);
        $("#myimg").attr("src", "/jquery/images/jquery.jpg");
      });
    </script>
  </head>
  <body>
    <div>
      <p title = "Bold and Brave">This is first paragraph.</p>
      <p id = "myid">This is second paragraph.</p>
      <div id = "divid"></div>
      <img id = "myimg" alt = "Sample image" />
    </div>
</body>
</html>
```
 You can replace a complete DOM element with the specified HTML or DOM elements.
selector.replaceWith(content)
```
<script type = "text/javascript" language = "javascript">
      $(document).ready(function() {
        $("div").click(function () {
          $(this).replaceWith("<h1>JQuery is Great</h1>");
        });
      });
    </script>
```

Events

To make your web page interactive, you write programs that respond to events.
Mouse events: click, dblclick, mousedown, mouseup, mouseover, etc
Document/Window Events: load, resize, scroll, unload etc
Form Events: submit, reset, focus, and change

```
<script type = "text/javascript" language = "javascript">
     $(document).ready(function() {
       $('#button').click(function() {
         $(this).val("Stop that!");
           }); // end click
     });
   </script>
 </head>
 <body>
   <div id = "mydiv">
     Click on this to see a dialogue box.
        <input type="button" id="button">
   </div>
 </body>
```

   ☐ The hover(over, out) method simulates hovering (moving the mouse on, and off,
     an object).

```
<script type = "text/javascript" language = "javascript">
     $(document).ready(function() {
       $('div').hover(
         function () {
           $(this).css({"background-color":"red"});
         },
         function () {
           $(this).css({"background-color":"blue"});
         }
       );
     });    </script>
```
The bind() method is a more flexible way of dealing with events than jQuery's event
specific functions like click() or mouseover(). It not only lets you specify an event and a

function to respond to the event, but also lets you pass additional data for the event-handling function to use.

```
$('#theElement').bind('click', function() {
// do something interesting
}); // end bind
```

  checked selector selects all checked check-boxes or radio buttons. Let us understand this with an example.

```
<html>
<head>
  <title></title>
  <script src="jquery-1.11.2.js"></script>
  <script type="text/javascript">
    $(document).ready(function                    ()                    {
$('#btnSubmit').click(function () {
        var  result  =  $('input[type="radio"]:checked');
if (result.length > 0) {
        $('#divResult').html(result.val() + " is checked");
      }
else {
        $('#divResult').html("No radio button checked");
      }
      });
    });
  </script>
</head>
<body           style="font-family:Arial">
Gender :
  <input type="radio" name="gender" value="Male">Male
  <input type="radio" name="gender" value="Female">Female
   <input id="btnSubmit" type="submit" value="submit" />
    <div id="divResult">
  </div>
</body>
</html>
```

  The each() method in jQuery is used to execute a function for each matched element.

```
<html>
```

```
<head>
   <title></title>
   <script src="jquery-1.11.2.js"></script>
   <script type="text/javascript">
      $(document).ready(function                ()                {
$('#btnSubmit').click(function () {
          var result = $('input[type="checkbox"]:checked');
if (result.length > 0) {
             var resultString = result.length + " checkboxe(s) checked<br/>";
result.each(function () {
                resultString += $(this).val() + "<br/>";
             });
             $('#divResult').html(resultString);
          }                    Lab No:1
          else {
             $('#divResult').html("No checkbox checked");
          }
        });
      });
   </script>
</head>
<body style="font-family:Arial">
   Skills :
   <input type="checkbox" name="skills" value="JavaScript" />JavaScript
   <input type="checkbox" name="skills" value="jQuery" />jQuery
   <input type="checkbox" name="skills" value="C#" />C#
   <input type="checkbox" name="skills" value="VB" />VB
<br /><br />
   <input id="btnSubmit" type="submit" value="submit" />
   <br /><br />
   <div id="divResult">
   </div>
</body>
</html>
```

11

The animate() Method

The jQuery animate() method is used to create custom animations.

$(*selector*).animate({*params*},*speed*,*callback*);

The required params parameter defines the CSS properties to be animated.

The optional speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.

The optional callback parameter is a function to be executed after the animation completes.

$("button").click(function(){

$("div").animate({left:'250px'});

});

LAB ASSIGNMENTS:

1. Write a web page which contains table with 3 X 3 dimensions (fill some data) and one image. Style the rows with alternate color. Move the table and image together from right to left when button is clicked.
2. Design a calculator to perform basic arithmetic operations. Use textboxes and buttons to design web page.
3. Create a web page to design a birthday card shown below.

HOME ASSIGNMENTS:

1. Design a webpage. The page contains:
   a. Dropdown list with HP, Nokia, Samsung, Motorola, Apple as items.
   b. Checkbox with Mobile and Laptop as items. □ Textbox where you enter quantity.
   c. There is a button with text as 'Produce Bill'.

On Clicking Produce Bill button, alert should be displayed with total amount.

2. Implement the bouncing ball using animate() function.
3. Write a web page which displays image and show the sliding text on the image.

Lab No:3   BOOTSTRAP

Objectives:

In this lab, student will be able to:

1. Develop web pages using design templates
2. Learn how to use bootstrap elements ***What is Bootstrap?***

1. Bootstrap is a free front-end framework for faster and easier web development
2. Bootstrap includes HTML and CSS based design templates for typography, forms, buttons, tables, navigation, modals, image carousels and many other, as well as optional JavaScript plugins
3. Bootstrap also gives you the ability to easily create responsive designs(automatically adjust themselves to look good on all devices)

Bootstrap Containers are used to pad the content inside of them, and there are two container classes available:

- The .container class provides a responsive fixed width container
- The .container-fluid class provides a full width container, spanning the entire width of the viewport Example:

<!DOCTYPE html>

<html lang="en">

<head>

  <title>Bootstrap Example</title>

  <meta charset="utf-8">

```html
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css">

    <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>

    <script
src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.16.0/umd/popper.min.js"
></script>

    <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.min.js"></script>

</head>

<body>

<div class="container">

    <h1>My First Bootstrap Page</h1>

    <p>This part is inside a .container class.</p>

    <p>The .container class provides a responsive fixed width container.</p>

    <p>Resize the browser window to see that its width (max-width) will change at
different breakpoints.</p>

</div>

</body></html>
```

Bootstrap Tables

The .table-striped class adds zebra-stripes to a table.

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Bootstrap Example</title>
  <meta charset="utf-8">
```

Firstname	Lastname	Email
John	Doe	john@example.com
Mary	Moe	mary@example.com
July	Dooley	july@example.com

```
 <meta name="viewport" content="width=device-width, initial-scale=1">
 <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css">
 <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>
 <script
src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.16.0/umd/popper.min.js"></scrip
t>  <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.min.js"></script>
</head>
<body>

<div class="container">
 <h2>Dark Striped Table</h2>
 <p>Combine  .table-dark  and  .table-striped  to  create  a  dark,  striped  table:</p>
<table class="table table-dark table-striped">
   <thead>
    <tr>
     <th>Firstname</th>
     <th>Lastname</th>
     <th>Email</th>
    </tr>
   </thead>
   <tbody>
    <tr>
     <td>John</td>
     <td>Doe</td>
     <td>john@example.com</td>
    </tr>
   </tbody>
  </table>
 </div>
```

```
</body>
</html>
<button type="button" class="btn btn-outline-primary">Primary</button>
<button type="button" class="btn btn-outline-secondary">Secondary</button>
<button type="button" class="btn btn-outline-success">Success</button>
<button type="button" class="btn btn-outline-info">Info</button>
<button type="button" class="btn btn-outline-warning">Warning</button>
<button type="button" class="btn btn-outline-danger">Danger</button>
<button type="button" class="btn btn-outline-dark">Dark</button>
<button type="button" class="btn btn-outline-light text-dark">Light</button>
```

Button Styles

Bootstrap 4 provides different styles of buttons:

Basic Primary Secondary Success Info Warning Danger Dark Light Link

LAB ASSIGNMENTS:

1.   Design the student bio-data form using button, label, textbox, radio button, table and checkbox.
2.   Design a web page which shows the database-oriented CRUD operation. Consider Employee data.
3.   Create a web page using bootstrap as mentioned. Divide the page into 2 parts top and bottom, then divide the bottom into 3 parts and design each top and bottom part using different input groups, input, badges, buttons and button groups. Make the design more attractive.
4.   Design your class timetable using bootstrap table and carousel.

HOME ASSIGNMENTS:

1. Design an attractive 'train ticket booking form' using different bootstrap elements.
2. Design an attractive 'magazine cover page' using different bootstrap elements.

Lab No:4   Python Basics

Objectives:

In this lab, student will be able to

1. Familiarize with the python programming language
2. Understand the usage of python primitives, data structures and functions

Guido van Rossum created Python in the early 90s. It is now one of the most popular languages in existence.

\# Single line comments start with a number symbol.

""" Multiline strings can be written
using three "s, and are often used   as
documentation.
"""

1. Primitive Datatypes and Operators

```
# You have numbers
3  # => 3

# Math is what you would expect
1 + 1   # => 2
8 - 1   # => 7
10 * 2  # => 20
35 / 5  # => 7.0

# Integer division rounds down for both positive and negative numbers. 5
// 3      # => 1
```

```
-5 // 3     # => -2
5.0 // 3.0   # => 1.0 # works on floats too
-5.0 // 3.0  # => -2.0

# The result of division is always a float
10.0 / 3  # => 3.3333333333333335

# Modulo operation
7 % 3  # => 1

# Exponentiation (x**y, x to the yth power)
2**3  # => 8

# Enforce precedence with parentheses
1 + 3 * 2  # => 7
(1 + 3) * 2  # => 8

# Boolean values are primitives (Note: the capitalization)
True  # => True
False  # => False

# negate with not not
True   # => False
not False  # => True

# Boolean Operators
# Note "and" and "or" are case-sensitive
True and False  # => False
False or True   # => True

# True and False are actually 1 and 0 but with different keywords
True + True # => 2
True * 8    # => 8
False - 5   # => -5

# Comparison operators look at the numerical value of True and False
0 == False  # => True 1
== True   # => True
```

```
2 == True   # => False
-5 != False # => True


# Using boolean logical operators on ints casts them to booleans for evaluation, but their
non-cast value is returned
# Don't mix up with bool(ints) and bitwise and/or (&,|) bool(0)
# => False
bool(4)     # => True
bool(-6)    # => True 0
and 2      # => 0
-5 or 0     # => -5


# Equality is ==
1 == 1  # => True
2 == 1  # => False


# Inequality is !=
1 != 1  # => False
2 != 1  # => True


# More comparisons
1 < 10  # => True
1 > 10  # => False
2 <= 2  # => True
2 >= 2  # => True


# Seeing whether a value is in a range
1 < 2 and 2 < 3  # => True
2 < 3 and 3 < 2  # => False
# Chaining makes this look nicer
1 < 2 < 3  # => True
2 < 3 < 2  # => False


# (is vs. ==) is checks if two variables refer to the same object, but == checks
# if the objects pointed to have the same values. a = [1, 2, 3, 4]  # Point a at a
new list, [1, 2, 3, 4] b = a          # Point b at what a is pointing to b is a
# => True, a and b refer to the same object b == a          # => True, a's and
b's objects are equal b = [1, 2, 3, 4]  # Point b at a new list, [1, 2, 3, 4]
```

```python
b is a          # => False, a and b do not refer to the same object b
== a            # => True, a's and b's objects are equal

# Strings are created with " or '
"This is a string."
'This is also a string.'

# Strings can be added too! But try not to do this.
"Hello " + "world!"  # => "Hello world!"
# String literals (but not variables) can be concatenated without using '+'
"Hello " "world!"    # => "Hello world!"

# A string can be treated like a list of characters
"This is a string"[0]  # => 'T'

# You can find the length of a string
len("This is a string")  # => 16

# You can also format using f-strings or formatted string literals (in Python 3.6+) name
= "Reiko"
f"She said her name is {name}." # => "She said her name is Reiko"
# You can basically put any Python statement inside the braces and it will be output in
the string.
f"{name} is {len(name)} characters long." # => "Reiko is 5 characters long."


# None is an object
None  # => None

# Don't use the equality "==" symbol to compare objects to None #
Use "is" instead. This checks for equality of object identity.
"etc" is None  # => False
None is None   # => True

# None, 0, and empty strings/lists/dicts/tuples all evaluate to False.
# All other values are True
bool(0)   # => False
bool("")  # => False bool([])
```

```
# => False bool({})  # =>
False
bool(())  # => False
```

2. Variables and Collections

```
# Python has a print function print("I'm Python. Nice to meet you!")  # =>
I'm Python. Nice to meet you!

# By default the print function also prints out a newline at the end.
# Use the optional argument end to change the end string.
print("Hello, World", end="!")  # => Hello, World!

# Simple way to get input data from console input_string_var = input("Enter
some data: ") # Returns the data as a string # Note: In earlier versions of
Python, input() method was named as raw_input()

# There are no declarations, only assignments. #
Convention is to use lower_case_with_underscores
some_var = 5 some_var  # => 5

# Accessing a previously unassigned variable is an exception.
# See Control Flow to learn more about exception handling.
some_unknown_var  # Raises a NameError

# if can be used as an expression
# Equivalent of C's '?:' ternary operator
"yahoo!" if 3 > 2 else 2  # => "yahoo!"

# Lists store sequences li
= []
# You can start with a prefilled list other_li
= [4, 5, 6]

# Add stuff to the end of a list with append
li.append(1)    # li is now [1] li.append(2)
# li is now [1, 2] li.append(4)    # li is now
[1, 2, 4] li.append(3)    # li is now [1, 2, 4,
3] # Remove from the end with pop
```

```python
li.pop()        # => 3 and li is now [1, 2, 4]
# Let's put it back li.append(3)    # li is
now [1, 2, 4, 3] again.

# Access a list like you would any array
li[0]   # => 1
# Look at the last element
li[-1]  # => 3

# Looking out of bounds is an IndexError li[4]
# Raises an IndexError
# You can look at ranges with slice syntax.
# The start index is included, the end index is not # (It's a
closed/open range for you mathy types.) li[1:3]   # Return list
from index 1 to 3 => [2, 4] li[2:]    # Return list starting from
index 2 => [4, 3] li[:3]    # Return list from beginning until
index 3  => [1, 2, 4] li[::2]   # Return list selecting every
second entry => [1, 4] li[::-1]  # Return list in reverse order
=> [3, 4, 2, 1]
# Use any combination of these to make advanced slices
# li[start:end:step]

# Make a one layer deep copy using slices li2 = li[:]  # => li2 =
[1, 2, 4, 3] but (li2 is li) will result in false.

# Remove arbitrary elements from a list with "del"
del li[2]  # li is now [1, 2, 3]

# Remove first occurrence of a value li.remove(2)
# li is now [1, 3]
li.remove(2)  # Raises a ValueError as 2 is not in the list

# Insert an element at a specific index
li.insert(1, 2)  # li is now [1, 2, 3] again

# Get the index of the first item found matching the argument li.index(2)
# => 1
li.index(4)  # Raises a ValueError as 4 is not in the list
```

```python
# You can add lists
# Note: values for li and for other_li are not modified. li
+ other_li  # => [1, 2, 3, 4, 5, 6]

# Concatenate lists with "extend()" li.extend(other_li)
# Now li is [1, 2, 3, 4, 5, 6]

# Check for existence in a list with "in"
1 in li  # => True

# Examine the length with "len()"
len(li)  # => 6

# Tuples are like lists but are immutable.
tup = (1, 2, 3) tup[0]      # => 1
tup[0] = 3  # Raises a TypeError

# Note that a tuple of length one has to have a comma after the last element but
# tuples of other lengths, even zero, do not. type((1))   # => <class 'int'>
type((1,))  # => <class 'tuple'> type(())    # => <class 'tuple'>

# You can do most of the list operations on tuples too len(tup)
# => 3
tup + (4, 5, 6)  # => (1, 2, 3, 4, 5, 6) tup[:2]
# => (1, 2)
2 in tup        # => True

# You can unpack tuples (or lists) into variables a, b, c =
(1, 2, 3)  # a is now 1, b is now 2 and c is now 3
# You can also do extended unpacking
a, *b, c = (1, 2, 3, 4)  # a is now 1, b is now [2, 3] and c is now 4
# Tuples are created by default if you leave out the parentheses d,
e, f = 4, 5, 6  # tuple 4, 5, 6 is unpacked into variables d, e and f
# respectively such that d = 4, e = 5 and f = 6 #
Now look how easy it is to swap two values e,
d = d, e  # d is now 5 and e is now 4
```

```python
# Dictionaries store mappings from keys to values empty_dict
= {}
# Here is a prefilled dictionary
filled_dict = {"one": 1, "two": 2, "three": 3}

# Note keys for dictionaries have to be immutable types. This is to ensure that #
the key can be converted to a constant hash value for quick look-ups.
# Immutable types include ints, floats, strings, tuples.
invalid_dict = {[1,2,3]: "123"}  # => Raises a TypeError: unhashable type: 'list'
valid_dict = {(1,2,3):[1,2,3]}   # Values can be of any type, however.

# Look up values with [] filled_dict["one"]
# => 1 # Get all keys as an iterable with
"keys()". We need to wrap the call in list()
# to turn it into a list. We'll talk about those later.  Note - for Python
# versions <3.7, dictionary key ordering is not guaranteed. Your results might #
not match the example below exactly. However, as of Python 3.7, dictionary #
items maintain the order at which they are inserted into the dictionary.
list(filled_dict.keys())  # => ["three", "two", "one"] in Python <3.7
list(filled_dict.keys())  # => ["one", "two", "three"] in Python 3.7+


# Get all values as an iterable with "values()". Once again we need to wrap it
# in list() to get it out of the iterable. Note - Same as above regarding key #
ordering. list(filled_dict.values())  # => [3, 2, 1]  in Python <3.7
list(filled_dict.values())  # => [1, 2, 3] in Python 3.7+

# Check for existence of keys in a dictionary with "in"
"one" in filled_dict  # => True
1 in filled_dict     # => False

# Looking up a non-existing key is a KeyError filled_dict["four"]
# KeyError

# Use "get()" method to avoid the KeyError
filled_dict.get("one")     # => 1 filled_dict.get("four")
# => None
# The get method supports a default argument when the value is missing
filled_dict.get("one", 4)   # => 1
```

```python
filled_dict.get("four", 4)  # => 4

# "setdefault()" inserts into a dictionary only if the given key isn't present
filled_dict.setdefault("five", 5)  # filled_dict["five"] is set to 5
filled_dict.setdefault("five", 6)  # filled_dict["five"] is still 5

# Adding to a dictionary
filled_dict.update({"four":4})  # => {"one": 1, "two": 2, "three": 3, "four": 4}
filled_dict["four"] = 4          # another way to add to dict

# Remove keys from a dictionary with del
del filled_dict["one"]  # Removes the key "one" from filled dict

# From Python 3.5 you can also use the additional unpacking options {'a':
1, **{'b': 2}}  # => {'a': 1, 'b': 2}
{'a': 1, **{'a': 2}}  # => {'a': 2}

# Sets store ... well sets empty_set
= set()
# Initialize a set with a bunch of values. Yeah, it looks a bit like a dict. Sorry. some_set
= {1, 1, 2, 2, 3, 4}  # some_set is now {1, 2, 3, 4}

# Similar to keys of a dictionary, elements of a set have to be immutable.
invalid_set = {[1], 1}  # => Raises a TypeError: unhashable type: 'list' valid_set
= {(1,), 1}

# Add one more item to the set filled_set
= some_set
filled_set.add(5)  # filled_set is now {1, 2, 3, 4, 5}
# Sets do not have duplicate elements
filled_set.add(5)  # it remains as before {1, 2, 3, 4, 5}

# Do set intersection with & other_set
= {3, 4, 5, 6}
filled_set & other_set  # => {3, 4, 5}

# Do set union with |
filled_set | other_set  # => {1, 2, 3, 4, 5, 6}
```

```python
# Do set difference with -
{1, 2, 3, 4} - {2, 3, 5}  # => {1, 4}

# Do set symmetric difference with ^
{1, 2, 3, 4} ^ {2, 3, 5}  # => {1, 4, 5}

# Check if set on the left is a superset of set on the right
{1, 2} >= {1, 2, 3} # => False

# Check if set on the left is a subset of set on the right
{1, 2} <= {1, 2, 3} # => True

# Check for existence in a set with in
2 in filled_set   # => True
10 in filled_set # => False

# Make a one-layer deep copy
filled_set = some_set.copy()  # filled_set is {1, 2, 3, 4, 5} filled_set
is some_set        # => False
```

3. Control Flow and Iterables

```python
# Let's just make a variable
some_var = 5

# Here is an if statement. Indentation is significant in Python!
# Convention is to use four spaces, not tabs. # This prints
"some_var is smaller than 10" if some_var > 10:
print("some_var is totally bigger than 10.") elif some_var <
10:   # This elif clause is optional.     print("some_var is
smaller than 10.") else:            # This is optional too.
print("some_var is indeed 10.")

"""
For loops iterate over lists prints:
    dog is a mammal
cat is a mammal
mouse is a mammal
```

```
""" for animal in ["dog", "cat",
"mouse"]:
    # You can use format() to interpolate formatted strings
print("{} is a mammal".format(animal))

"""
```

"range(number)" returns an iterable of numbers
from zero to the given number prints:
    0
    1
    2
    3

```
""" for i in
range(4):
    print(i)

"""
```

"range(lower, upper)" returns an iterable of numbers
from the lower number to the upper number prints:
    4
    5
    6
    7 """ for i in

```
range(4, 8):
    print(i)

"""
```

"range(lower, upper, step)" returns an iterable of numbers from
the lower number to the upper number, while incrementing by
step. If step is not indicated, the default value is 1. prints:
    4
    6 """ for i in

```
range(4, 8, 2):
    print(i)

"""
```

To loop over a list, and retrieve both the index and the value of each item in the list
prints: 0 dog
1 cat

```python
2 mouse """ animals = ["dog",
"cat", "mouse"] for i, value in
enumerate(animals):
    print(i, value)
```

"""
While loops go until a condition is no longer met.
prints:
    0
    1
    2

```python
3 """ x = 0 while x < 4:    print(x)
    x += 1  # Shorthand for x = x + 1
```

```python
# Handle exceptions with a try/except block try:
    # Use "raise" to raise an error    raise IndexError("This is an index error") except
IndexError as e:    pass             # Pass is just a no-op. Usually you would do recovery
here. except (TypeError, NameError):    pass             # Multiple exceptions can be
handled together, if required. else:             # Optional clause to the try/except block.
Must follow all except blocks    print("All good!")   # Runs only if the code in try raises
no exceptions
finally:             #  Execute under all circumstances
print("We can clean up resources here")
```

```python
# Instead of try/finally to cleanup resources you can use a with statement
with open("myfile.txt") as f:    for line in f:        print(line)
```

```python
# Writing to a file contents = {"aa":
12, "bb": 21} with open("myfile1.txt",
"w+") as file:
    file.write(str(contents))        # writes a string to a file
```

```python
with open("myfile2.txt", "w+") as file:
file.write(json.dumps(contents)) # writes an object to a file
```

```python
# Reading from a file with open('myfile1.txt', "r+") as
file:    contents = file.read()         # reads a string from
a file print(contents)
# print: {"aa": 12, "bb": 21}
```

```python
with open('myfile2.txt', "r+") as file:
    contents = json.load(file)      # reads a json object from a file print(contents)
# print: {"aa": 12, "bb": 21}
```

```python
# Python offers a fundamental abstraction called the Iterable.
# An iterable is an object that can be treated as a sequence. #
The object returned by the range function, is an iterable.

filled_dict = {"one": 1, "two": 2, "three": 3} our_iterable = filled_dict.keys()
print(our_iterable) # => dict_keys(['one', 'two', 'three']). This is an object that implements
our Iterable interface.

# We can loop over it. for i in
our_iterable:    print(i) # Prints
one, two, three

# However we cannot address elements by index. our_iterable[1]
# Raises a TypeError

# An iterable is an object that knows how to create an iterator. our_iterator
= iter(our_iterable)

# Our iterator is an object that can remember the state as we traverse through it.
# We get the next object with "next()". next(our_iterator)
# => "one"

# It maintains state as we iterate. next(our_iterator)
# => "two"
next(our_iterator)  # => "three"

# After the iterator has returned all of its data, it raises a StopIteration exception
next(our_iterator)  # Raises StopIteration

# We can also loop over it, in fact, "for" does this implicitly!
our_iterator = iter(our_iterable)
for i in our_iterator:    print(i) #
Prints one, two, three
```

```python
# You can grab all the elements of an iterable or iterator by calling list() on it.
list(our_iterable)  # => Returns ["one", "two", "three"]
list(our_iterator)  # => Returns [] because state is saved
```

The indention of each line controls whether it is within a loop, if statement, etc. -- there are no { } to define blocks of code. This use of indentation in Python is unusual, but it's logical and you get used to it.

4. Functions
```python
# Use "def" to create new functions def
add(x, y):    print("x is {} and y is
{}".format(x, y))
    return x + y  # Return values with a return statement


# Calling functions with parameters
add(5, 6)  # => prints out "x is 5 and y is 6" and returns 11


# Another way to call functions is with keyword arguments add(y=6,
x=5)  # Keyword arguments can arrive in any order.


# You can define functions that take a variable number of
# positional arguments def
varargs(*args):
    return args


varargs(1, 2, 3)  # => (1, 2, 3)


# You can define functions that take a variable number of
# keyword arguments, as well def
keyword_args(**kwargs):
    return kwargs


# Let's call it to see what happens
keyword_args(big="foot", loch="ness")  # => {"big": "foot", "loch": "ness"}
```

```
# You can do both at once, if you like def
all_the_args(*args, **kwargs):
    print(args)
print(kwargs)
"""
all_the_args(1, 2, a=3, b=4) prints:
    (1, 2)
    {"a": 3, "b": 4}
"""


# When calling functions, you can do the opposite of args/kwargs!
# Use * to expand tuples and use ** to expand kwargs.
args = (1, 2, 3, 4) kwargs = {"a": 3, "b": 4} all_the_args(*args)
# equivalent to all_the_args(1, 2, 3, 4) all_the_args(**kwargs)
# equivalent to all_the_args(a=3, b=4)
all_the_args(*args, **kwargs)  # equivalent to all_the_args(1, 2, 3, 4, a=3, b=4)


# Returning multiple values (with tuple assignments) def swap(x, y):
return y, x  # Return multiple values as a tuple without the parenthesis.
# (Note: parenthesis have been excluded but can be included)


x = 1 y = 2 x, y = swap(x, y)    # =>
x = 2, y = 1
# (x, y) = swap(x,y)  # Again parenthesis have been excluded but can be included.


# Function Scope
x = 5


def set_x(num):
    # Local var x not the same as global variable x
x = num    # => 43
    print(x)   # => 43


def set_global_x(num):
    global x
print(x)   # => 5
    x = num    # global var x is now set to 6
    print(x)   # => 6
```

```
set_x(43)
set_global_x(6)


# Python has first class functions
def create_adder(x):
def adder(y):
return x + y
    return adder


add_10 = create_adder(10)
add_10(3)   # => 13

# There are also anonymous functions
(lambda x: x > 2)(3)              # => True
(lambda x, y: x ** 2 + y ** 2)(2, 1)  # => 5

# There are built-in higher order functions list(map(add_10,
[1, 2, 3]))        # => [11, 12, 13] list(map(max, [1, 2, 3], [4,
2, 1]))  # => [4, 2, 3]

list(filter(lambda x: x > 5, [3, 4, 5, 6, 7]))  # => [6, 7]

# We can use list comprehensions for nice maps and filters
# List comprehension stores the output as a list which can itself be a nested list
[add_10(i) for i in [1, 2, 3]]       # => [11, 12, 13]
[x for x in [3, 4, 5, 6, 7] if x > 5]  # => [6, 7]

# You can construct set and dict comprehensions as well.
{x for x in 'abcddeef' if x not in 'abc'}  # => {'d', 'e', 'f'}
{x: x**2 for x in range(5)}  # => {0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
```

5. Modules

```
# You can import modules import
math
print(math.sqrt(16))  # => 4.0
```

```python
# You can get specific functions from a module
from math import ceil, floor print(ceil(3.7))   #
=> 4.0
print(floor(3.7))  # => 3.0

# You can import all functions from a module.
# Warning: this is not recommended from
math import *

# You can shorten module names import
math as m
math.sqrt(16) == m.sqrt(16)  # => True

# Python modules are just ordinary Python files. You #
can write your own, and import them. The name of the #
module is the same as the name of the file.

# You can find out which functions and attributes
# are defined in a module. import math
dir(math)

# If you have a Python script named math.py in the same
# folder as your current script, the file math.py will # be
loaded instead of the built-in Python module. # This
happens because the local folder has priority # over
Python's built-in libraries.
```

Example:

Write a python program to display content of a file.

```python
#!/usr/bin/python -tt

"""
-tt flag above detects space/tab indent problems
"""

# sys is one of many available modules of library code, import to use.
```

```python
# sys.argv is the list of command line arguments.
import sys

# defines a global variable a
= 123

# defines a 'cat' function which takes a filename def
cat(filename):
  """Given filename, print its text contents."""
  print  filename,  '======='
f = open(filename, 'r')
  for line in f:  # goes through a text file line by line     print line,
# trailing comma inhibits the ending print-newline   #
alternative, read the whole file into a single string:   # text =
f.read()   f.close()

def main():
  # sys.argv contains command line arguments.
  # This assigns a list of all but the first arg into a local 'args' var.   args
= sys.argv[1:]

  # important syntax -- loop of variable 'filename' over the args list.   for
filename in args:
    # detect scary filenames: if/else and/or/not       if
filename == 'voldemort' or filename == 'vader':
      print    'this    file    is    very    worrying'
cat(filemane, 123, bad_variable)
      # important point: errors in above line only caught if it is run     else:
      #         regular         case
cat(filename)
  print 'all done'  # this print is outside the loop, due to its indentation


# Standard boilerplate at end of file to call main() function. if
__name__ == '__main__':
  main()
```

Defining a Class in Python

Like function definitions begin with the [def](#) keyword in Python, class definitions begin with a [class](#) keyword.

The first string inside the class is called docstring and has a brief description about the class. Although not mandatory, this is highly recommended.

Here is a simple class definition.

```python
class MyNewClass:
    '''This is a docstring. I have created a new class'''
    pass
```

A class creates a new local [namespace](#) where all its attributes are defined. Attributes may be data or functions.

There are also special attributes in it that begins with double underscores __. For example, __doc__ gives us the docstring of that class.

As soon as we define a class, a new class object is created with the same name. This class object allows us to access the different attributes as well as to instantiate new of class.

```python
class Person:
    "This is a person
class"    age = 10

    def greet(self):
        print('Hello')


# Output: 10
print(Person.age)

# Output: <function Person.greet>
print(Person.greet)
```

```python
# Output: 'This is my second class'
print(Person.__doc__)
```

Output

```
10
<function Person.greet at 0x7fc78c6e8160>
This is a person class
```

Creating an Object in Python

We saw that the class object could be used to access different attributes.

It can also be used to create new object instances instantiation) of that class. The procedure to create an object is similar to a [function](#) call

```
>>> harry = Person()
```

This will create a new object instance named harry. We can access the attributes of objects using the object name prefix.

Attributes may be data or method. Methods of an object are corresponding functions of that class.

This means to say, since Person.greet is a function object (attribute of class), Person.greet will be a method object.

```
class Person:
    "This is a person class"
age = 10

    def greet(self):
print('Hello')


# create a new object of Person class
harry = Person()

# Output: <function Person.greet>
print(Person.greet)

# Output: <bound method Person.greet of <__main__.Person object>>
print(harry.greet)

# Calling object's greet() method
# Output: Hello
harry.greet()
```

Output

```
<function Person.greet at 0x7fd288e4e160>
<bound method Person.greet of <__main__.Person object at 0x7fd288e9fa30>>
Hello
```

You may have noticed the self parameter in function definition inside the class but we

called the method simply as harry.greet() without any arguments. It still worked.

This is because, when `harry.greet()` translates into `Person.greet(harry)`, self is passed as the first argument. So,   .

In general, calling   a method with a list of n arguments is equivalent to calling the corresponding function with an argument list that is created by inserting the method's object before the first argument.

For these reasons, the first arg self nt of the function in class must be the object itself. This is conventionally called   . It can be named otherwise but highly recommend to follow the convention.

Another Example:

```python
# We use the "class" statement to create a class

class Human:
    # A class attribute. It is shared by all instances of this class
    species = "H. sapiens"

    # Basic initializer, this is called when this class is instantiated.
    # Note that the double leading and trailing underscores denote objects
    # or attributes that are used by Python but that live in user-controlled
    # namespaces. Methods(or objects or attributes) like: __init__, __str__,
    # __repr__ etc. are called special methods (or sometimes called dunder methods)
    # You should not invent such names on your own.     def __init__(self, name):

        # Assign the argument to the instance's name attribute
        self.name = name
```

```python
        # Initialize property
        self._age = 0

    # An instance method. All methods take "self" as the first argument
    def say(self, msg):
        print("{name}: {message}".format(name=self.name, message=msg))

    # Another instance method
    def sing(self):
        return 'yo... yo... microphone check... one two... one two...'

    # A class method is shared among all instances
    # They are called with the calling class as the first argument
    @classmethod
    def get_species(cls):
        return cls.species

    # A static method is called without a class or instance reference
    @staticmethod
    def grunt():
        return "*grunt*"

    # A property is just like a getter.
    # It turns the method age() into an read-only attribute of the same name.
    # There's no need to write trivial getters and setters in Python, though.
    @property
    def age(self):
```

```python
        return self._age

    # This allows the property to be set
    @age.setter
def age(self, age):
self._age = age

    # This allows the property to be deleted
    @age.deleter
def age(self):
del self._age

# When a Python interpreter reads a source file it executes all its code. # This
__name__ check makes sure this code block is only executed when this #
module is the main program.

if __name__ == '__main__':
    # Instantiate a class
i = Human(name="Ian")
    i.say("hi")                 # "Ian: hi"
j = Human("Joel")
    j.say("hello")              # "Joel: hello"
    # i and j are instances of type Human, or in other words: they are Human objects


    # Call our class method
    i.say(i.get_species())      # "Ian: H. sapiens"
```

```python
    # Change the shared attribute

    Human.species = "H. neanderthalensis"

    i.say(i.get_species())        # => "Ian: H. neanderthalensis"

    j.say(j.get_species())        # => "Joel: H. neanderthalensis"

    # Call the static method

print(Human.grunt())             # => "*grunt*"

    # Cannot call static method with instance of object

    # because i.grunt() will automatically put "self" (the object i) as an argument

    print(i.grunt())              # => TypeError: grunt() takes 0 positional arguments but 1
was given


    # Update the property for this instance

    i.age = 42     #

Get the property

    i.say(i.age)                  # => "Ian: 42"

    j.say(j.age)                  # => "Joel: 0"

# Delete the property     del i.age

    # i.age                       # => this would raise an AttributeError
```

Example: Write a Python program to convert an integer to a roman numeral.

```python
class py_solution:     def
int_to_Roman(self, num):
    val = [
        1000, 900, 500, 400,
        100, 90, 50, 40,
        10, 9, 5, 4,
        1
```

```python
        ]
    syb = [
        "M", "CM", "D", "CD",
        "C", "XC", "L", "XL",
        "X", "IX", "V", "IV",
        "I"
        ]
    roman_num = ''
    i = 0        while  num > 0:
for _ in range(num // val[i]):
            roman_num += syb[i]
            num -= val[i]
i += 1
    return roman_num

print(py_solution().int_to_Roman(1)) print(py_solution().int_to_Roman(4000))
```

LAB ASSIGNMENTS:

1. Write a python program to reverse a content a file and store it in another file.
2. Write a python program to implement binary search with recursion.
3. Write a python program to sort words in alphabetical order.
4. Write a Python class to get all possible unique subsets from a set of distinct integers Input:[4,5,6]
   Output : [[], [6], [5], [5, 6], [4], [4, 6], [4, 5], [4, 5, 6]]

5. Write a Python class to find a pair of elements (indices of the two numbers) from a given array whose sum equals a specific target number.
   Input: numbers= [10,20,10,40,50,60,70],   target=50
   Output: 3, 4.
6. Write a Python class to implement pow(x, n).
7. Write a Python class which has two methods get_String and print_String. The get_String accept a string from the user and print_String print the string in upper case.

HOME ASSIGNMENTS:

1. Write a python program to select smallest element from a list in an expected linear time.
2. Write a python program to implement bubble sort.
3. Write a python program to multiply two matrices
4. Write a Python class to find validity of a string of parentheses, '(', ')', '{', '}', '[' and ']. These brackets must be close in the correct order, for example "()" and "()[]{}" are valid but "[)", "({[)]" and "{{{" are invalid.
5. Write a Python class to reverse a string word by word.
6. Write a Python class named Circle constructed by a radius and two methods which will compute the area and the perimeter of a circle.

Lab No: 5

Developing Web Application using Django - Part I

Objectives:

In this lab, student will be able to

1. Understand the fundamentals of web forms creation
2. Learn to create views and templates
3. Design Django web applications using views and templates

First step need to be followed:

python3 -m venv djangoproject

source djangoproject/bin/activate

pip3 install django

1. **Build Your First Web Application in Django**

For creating a web application, first create a directory, say *PythonProject* where you would like to share your code, and then run the following command from the created directory using VScode Terminal:

django-admin startproject myproject

Myproject is the name of the project created. The following list of files are created inside the directory.
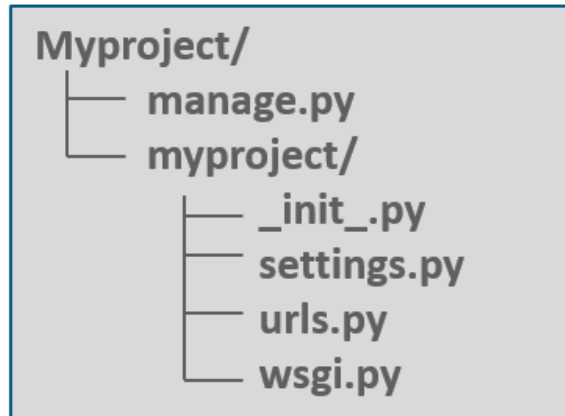
manage.py – It is a command-line utility that allows to interact with the Django project in various ways.

myproject/ – It is the actual Python package for the project. It is used to import anything, say – myproject.urls.

init.py – Init just tells the python that this is to be treated like a python package.

settings.py – This file manages all the settings of the project. **urls.py** – This is

the main controller which maps it to the website. **wsgi.py** – It serves as an entry

point for WSGI compatible web servers.

Now to create the application, type the below command in Powershell from the created project folder (i.e., *myproject)*.

```
python3 manage.py startapp webapp
```

Now the *'webapp'* directory will have some extra things from the original *myproject*. It includes model, test which are related to the backend databases.

It is important to import your application manually inside the project settings. For that, open *myproject/settings.py* and add your app manually:

```
INSTALLED_APPS = (
                'webapp',
                'django.contrib.admin',
```

'django.contrib.auth',
                    'django.contrib.contenttypes',
                    'django.contrib.sessions',
                    'django.contrib.messages',
                    'django.contrib.staticfiles',
)

Now to create a view, open *webapp/views.py* and put the below code in it:

```
from django.shortcuts import render from
django.http import HttpResponse def
index(request):
        return HttpResponse("<H2>HEY! Welcome to Edureka! </H2>")
```

The above code creates a view which returns HttpResponse. Now this view is to be mapped to a URL. So create a new python file *"urls.py"* inside the *webapp* folder. In *webapp/urls.py* include the following code:

```
from django.urls import re_path
from . import views
urlpatterns = [
            re_path(r'^$', views.index, name='index'),
]
```

In the above code, a view is referenced which will return index (defined in views.py file). The url pattern is in regular expression format where ^ stands for beginning of the string and $ stands for the end.

The next step is to point the root URLconf at the webapp.urls module. Open myproject/urls.py file and write the below code:

```
from django.conf.urls import include, url from
django.contrib import admin
urlpatterns = [
            re_path (r'^admin/', admin.site.urls),
            re_path (r'^webapp/', include('webapp.urls')),
]
```

In the above code, *webapp* and the *webapp.urls* are included. Now import *django.conf.urls.include* and insert an *include()* in the urlpatterns list. The *include()* function allows referencing other URLconfs.

Note that the regular expression doesn't have a '$' but rather a trailing slash, this means whenever Django encounters *include()*, it chops off whatever part of the URL matched up to that point and sends the remaining string to include URLconf for further processing.

To start the server, type the below command:

E:\MyFolder\myproject> python3 manage.py runserver

After running the server, go to **http://localhost:8000/webapp/** in your browser, and you should see the text "*HEY! Welcome to Edureka!*", which is defined in the index view(Fig 5.1).
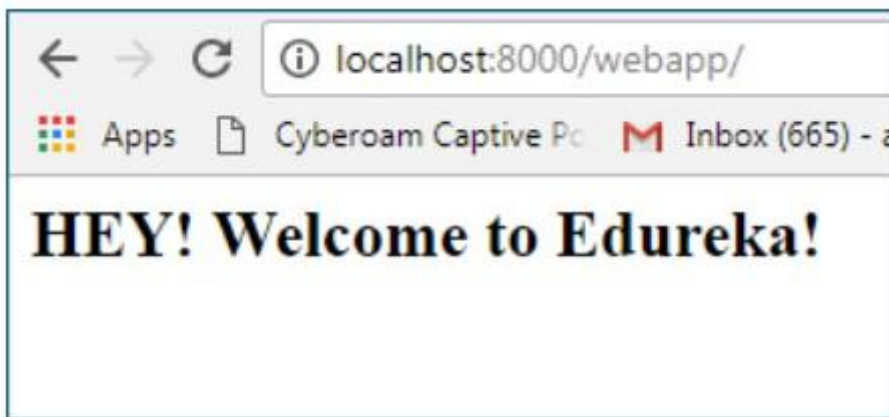
Fig 6.1

2. Creating a View

Django's views are the information brokers of a Django application. A view sources data from your database (or an external data source or service) and delivers it to a template. The view makes decisions on what data gets delivered to the template—either by acting on input from the user, or in response to other business logic and internal processes. Each Django view performs a specific function and has an associated template.

Modify webapp/views.py and put the below code in it:

\webapp\views.py

```
1  from django.shortcuts import render

2  from django.http import HttpResponse

3  from datetime import date

4  import calendar

5  from calendar import HTMLCalendar

8     def index(request,year,month):

9     year = int(year)

10    month = int(month)

11    if year < 1900 or year > 2099: year = date.today().year

12    month_name = calendar.month_name[month]

13    title = "MyClub Event Calendar - %s %s" % (month_name,year)

14    cal = HTMLCalendar().formatmonth(year, month)

15    return HttpResponse("<h1>%s</h1><p>%s</p>" % (title, cal))
```

Modify webapp/urls.py and put the below code in it

\webapp\urls.py

```
1 from django.urls import path, re_path

2 from . import views

3

4    urlpatterns = [
```

```
5    path(' ', views.index, name='index'),

6    re_path(r'^(?P<year>[0-9]{4})/(?P<month>0?[1-9]|1[0-2])/',views.index,
     name='index'),

7    ]
```

After running the server, go to http://localhost:8000/2019/03 in your browser, and the screen appears as shown in Fig 5.2.
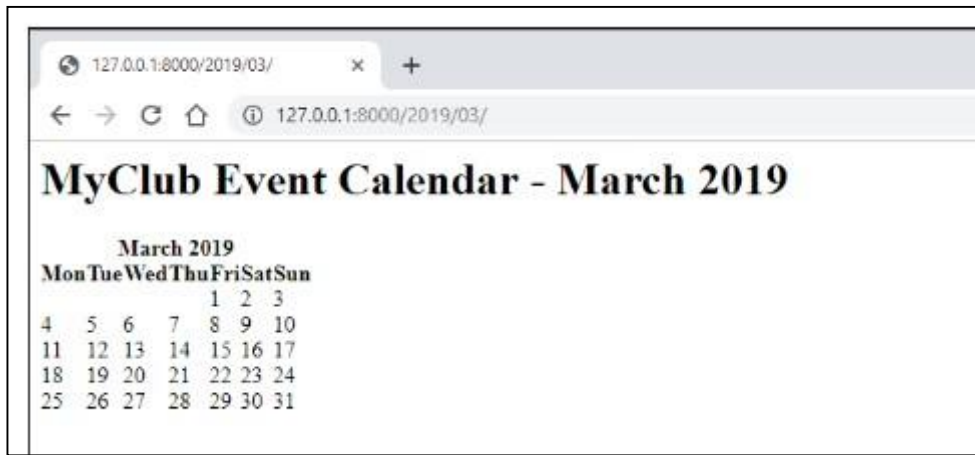
Fig 6.2

Creating a Site Template

All modern websites have a site template; a common look or branding that is duplicated across every page on the website.

The most common place for storing site template files in Django is in the website app that Django created automatically when *startproject* command is executed. Django didn't create the templates folder, so go ahead and create that folder. The folder structure should look like this:

\webapp

\templates
            __init__.py
        ...

As the website app is not in INSTALLED_APPS, Django won't automatically look for templates in the \webapp\templates folder. So tell Django where to look by adding a path to the DIRS setting. Modify settings.py (changes in bold):

TEMPLATES = [{
                'BACKEND': 'django.template.backends.django.DjangoTemplates',
                'DIRS': [**os.path.join(BASE_DIR, 'webapp/templates')**)],

        'APP_DIRS': True,
        # ...
This looks complicated, but is easy to understand—os.path.join is a Python command to create a file path by joining strings together (concatenating). In this example, webapp/templates is joined to the project directory to create the full path to the templates directory, i.e., <project path>/myproject/webapp/templates.

Now that the template folder is created and the folder path is listed, Django can find the site template. Now to create a simple template, create a html file base.html:

\webapp\templates\base.html

1   <!doctype html>
2   <html>
3   <head>
4   <meta charset="utf-8">
5   <title>Basic Site Template</title>
6   </head>
7
8   <body>
9   <h1>{{ title }}</h1>
10   <p>{{ cal }}</p>
11   </body>
12   </html>

3. **Displaying a Template**

Now that the template is created, tell Django to use the new base template when displaying content on the site. This is done in views.py file. Make the following changes to the index view (changes in bold):

\webapp\views.py

```
1  from django.shortcuts import render
2  # from django.http import HttpResponse
3  from datetime import date
4  import calendar
5  from calendar import HTMLCalendar
8      def index(request, year=date.today().year, month=date.today().month):
9      year = int(year)
10     month = int(month)
11     if year < 1900 or year > 2099: year = date.today().year
12     month_name = calendar.month_name[month]
13     title = "MyClub Event Calendar - %s %s" % (month_name, year)
14     cal = HTMLCalendar().formatmonth(year, month)
15     # return HttpResponse("<h1>%s</h1><p>%s</p>" % (title, cal))
16     return render(request, 'base.html', {'title': title, 'cal': cal})
```

For the new view, replace the call to HttpResponse() with a call to render(). render() is a special Django helper function that creates a shortcut for communicating with a web browser. When Django receives a request from a browser, it finds the right view and the view returns a response to the browser.

When we wish to use a template, Django first must load the template, create a context—which is basically a dictionary of variables and associated data that is passed back to the browser—and then return a HttpResponse. Django's render() function provides a shortcut that provides all three steps in a single function.

When the original request, the template and a context is supplied directly to render(), it returns the appropriately formatted response without having to code the intermediate steps.

In the modified views.py, the original request object is returned from the browser, the name of the site template and a dictionary (the context) containing the title and cal variables from the view.

Once views.py file is modified, save it and fire up the development server. Navigate to http://127.0.0.1:8000/, to see your simple new site template.

The calendar will be rendered as plain text, not as HTML. To get Django to render the HTML correctly, turn off autoescape for the calendar code. As this is a common task, the Django developers created the autoescape tag to make life easier. Make the following changes to the base.html file (changes in bold):

\webapp\templates\base.html

```
1       <!doctype html>
2       <html>
3       <head>
4       <meta charset="utf-8">
5       <title>Basic Site Template</title>
6       </head>
7
8       <body>
9       <h1>{{ title }}</h1>
10      <p>{% autoescape off %}{{ cal }} {% endautoescape %}</p> 11     </body>
12 </html>
```

Now, when you refresh your browser, the site homepage should look like Fig 5.3.

Fig 6.3

Solved Excersise:

Develop a simple Django web application to accept two numbers from user and add them up.

A new project named MyForm is created which has the manage.py file. Inside MyForm a new app named formapp is created which contains all the application related files.

 # MyForm/settings.py
```
INSTALLED_APPS = [
    'formapp',
    'django.contrib.admin',
    'django.contrib.auth',
]
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR,'formapp/templates')],
        'APP_DIRS': True,
},
]
```

 # MyForm/urls.py from
django.contrib import admin from
django.urls import path
from django.conf.urls import include, url

```
urlpatterns = [      path(r'^admin/',
admin.site.urls),
    url('', include('formapp.urls')),
]
```

 # formapp/urls.py from
django.conf.urls import url
from . import views

```
urlpatterns = [
url(' ', views.index, name='index'),
]
```

 # formapp/views.py from
django.shortcuts import render #
Create your views here.

```
def index(request):
        return render(request, 'basic.html')
```

formapp/templates/basic.html

```html
<!doctype html>
 <html>
    <head>
        <meta charset="utf-8">
```
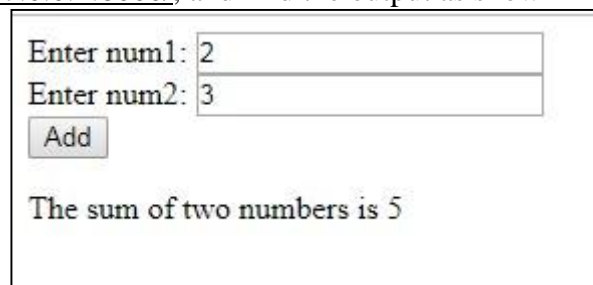
```
    <title>App to add two Nos</title>

  </head>
   <body>
    <script type="text/javascript">         function
myfunc(){        var n1=
document.getElementById("num1").value;        var n2=
document.getElementById("num2").value;
      var n3=parseInt(n1)+parseInt(n2);
      document.getElementById("para1").innerHTML="The sum of two numbers is
"+n3;
    }
    </script>
    Enter num1: <input type="text" id="num1"><br>
 Enter num2: <input type="text" id="num2"><br>
 <button onclick="myfunc()">Add</button><br\>
 <p id="para1"></p>
   </body>
 </html>
```

After editing all the above files save them and fire up the development server as shown below

E:\newdir\MyForm> python manage.py runserver

Navigate to http://127.0.0.1:8000/, and find the output as shown in Fig 5.4.

Fig 6.4

LAB ASSIGNMENTS:

1) Create a web form which allows the teacher to enter the student details like name, date of birth, address, contact number, email id and marks of English, Physics and Chemistry. After filling all the information and on clicking submit button, details should be added to a textarea displayed on the same page. Display the total percentage of marks obtained in a label.

2) Create a web form with employee ids in dropdown list. Use a textbox for accepting date of joining. Add a button named "Am I Eligible for Promotion". On clicking the button, if he has more than 5 years of experience, then display "YES" else "NO" in the label control.

HOME ASSIGNMENTS:

Develop a simple web page to reproduce the given Captcha. Upon match, suitable message has to be displayed. If there is a mismatch for more than 3 times, TextBox has to be disabled.

Lab No: 6

Developing Web Application using Django – Part II

LAB ASSIGNMENTS:

1) Develop a simple web page to perform basic arithmetic operations. Take two integer inputs from the user, select the operation to be performed using a dropdown. Include a button "Calculate" to perform the selected operation, and then display the result in the same web page.
2) Develop a simple web form that generates the front cover for a magazine. The form should provide the options for selecting the image, background color, changing font size, color etc. Input messages must be taken from the user so as to display it on the front cover with legible font family and font size. The front cover developed should be proportionate to the web page size. Place the css files inside static folder.

HOME ASSIGNMENTS:

1.   Design a simple web application to provide information about a book. The home page of the application should display the cover page of the book along with three hyperlinks: Metadata, Reviews, Publisher info. Give provision to revert to home page from any other page.

2.   Design a simple web application which will ask the user to input his name and a message, display the two items concatenated in a label, and change the format of the label using radio buttons and check boxes for selection, the user can make the label text bold, underlined or italic and change its color. include buttons to display the message in the label, clear the text boxes and label and exit

Enter Name: [   ]

☐ BOLD ☐ ITALIC ☐ UNDERLINE

○ RED ○ GREEN ○ PINK

Enter Message:

Label3 [Display]

Enter Name: [PRACHIT   ]

☑ BOLD ☐ ITALIC ☑ UNDERLINE

◉ RED ○ GREEN ○ PINK

We Can WE Will

Enter Message:

Name:PRACHIT
Message:We Can WE Will [Display]

Lab No: 7

Form Processing using Django – Part I

Objectives:

In this lab, student will be able to

1. Develop web forms using Form class in Django
2. Learn to use Form Widgets to enhance the web forms
3. Design Django web applications using session management techniques

Django Forms:

When one creates a **Form** class, the most important part is defining the fields of the form. Each field has custom validation logic. Forms are basically used for taking input from the user in some manner and using that information for logical operations on databases. For example, registering a user by taking input as his name, email, password, etc.

Django maps the fields defined in Django forms into HTML input fields. Django handles three distinct parts of the work involved in forms:

- preparing and restructuring data to make it ready for rendering
- creating HTML forms for the data
- receiving and processing submitted forms and data from the client **Syntax:** Django Fields have the following syntax:

field_name = forms.<u>FieldType</u>(**options)

Built in Django Form fields:

The **forms** library comes with a set of **Field** classes that represent common validation needs.

For each field, we describe the default widget used. We also specify the value returned when you provide an empty value.

BooleanField *class*

BooleanField(***kwargs***)**

- Default widget: **CheckboxInput**
- Empty value: **False**
- Normalizes to: A Python **True** or **False** value.

CharField *class*

CharField(***kwargs***)**

- Default widget: **TextInput**
- Empty value: Whatever you've given as **empty_value**.
- Normalizes to: A string.
- Uses arguments **max_length** or **min_length** (integer values),to ensure that the string is at most or at least the given length.

ChoiceField *class*

ChoiceField(***kwargs***)**

- Default widget: **Select**
- Empty value: **''** (an empty string) ☐   Normalizes to: A string.
- Validates that the given value exists in the list of choices.

DateField *class*

DateField(***kwargs***)**

- Default widget: **DateInput**
- Empty value: **None**
- Normalizes to: A Python **datetime.date** object.

- Validates that the given value is either a **datetime.date**, **datetime.datetime** or string formatted in a particular date format.

EmailField *class*

EmailField(***kwargs***)**

- Default widget: **EmailInput** ⬜ Empty value: **''** (an empty string) ⬜

  Normalizes to: A string.

- Uses **EmailValidator** to validate that the given value is a valid email address, using a moderately complex regular expression.

FileField *class*

FileField(***kwargs***)**

- Default widget: **ClearableFileInput**

- Empty value: **None**

- Normalizes to: An **UploadedFile** object that wraps the file content and file name into a single object.

IntegerField *class*

IntegerField(***kwargs***)**

- Default widget: **NumberInput** when **Field.localize** is **False**, else **TextInput**.

- Empty value: **None**

- Normalizes to: A Python integer.

Takes two optional arguments for validation:

- **max_value**
- **min_value**

These control the range of values permitted in the field.

URLField *class*

URLField(***kwargs***)**

- Default widget: **URLInput**

FIELD OPTIONS	DESCRIPTION
required	By default, each Field class assumes the value is required, so to make it not required you need to set required=False
label	The label argument lets you specify the "human-friendly" label for this field. This is used when the Field is displayed in a Form.
label_suffix	The label_suffix argument lets you override the form's label_suffix on a per- field basis.
widget	The widget argument lets you specify a Widget class to use when rendering this Field. See Widgets for more information.
help_text	The help_text argument lets you specify descriptive text for this Field. If you provide help_text, it will be displayed next to the Field when the Field is rendered by one of the convenience Form methods.

	The error_messages argument lets you override the default messages that the field will raise. Pass in a dictionary with keys matching the error messages you want to override.
error_messages	

validators	The validators argument lets you provide a list of validation functions for this field.
localize	The localize argument enables the localization of form data input, as well as the rendered output.
disabled.	The disabled boolean argument, when set to True, disables a form field using the disabled HTML attribute so that it won't be editable by users.

- Empty value: **''** (an empty string) ☐ Normalizes to: A string.

Takes the following optional arguments:

- **max_length**
- **min_length**

These are the same as **CharField.max_length** and **CharField.min_length**.

Core Field Arguments:

Core Field arguments are the arguments given to each field for applying some constraint or imparting a particular characteristic to a particular Field. For example, adding an argument required = False to CharField will enable it to be left blank by the user.

Creating a Django Form:

To use Django Forms, create a project and an app inside it. After you start an app, create a form in app/forms.py.

For creating a form in Django we have to specify what fields would exist in the form and of what type.

Let us create a form with CharField, IntegerField and BooleanField as follows:

 # app/forms.py from django

import forms   class

RegForm(forms.Form):   title =

forms.CharField()   description =

forms.CharField()   views =

forms.IntegerField()   available =

forms.BooleanField()

Rendering Django Forms:

Django form fields have several built-in methods to ease the work of the developer but sometimes one needs to implement things manually for customizing User Interface(UI). A form comes with 3 in-built methods that can be used to render Django form fields.

- {{ form.as_table }} will render them as table cells wrapped in <tr> tags
- {{ form.as_p }} will render them wrapped in <p> tags
- {{ form.as_ul }} will render them wrapped in <li> tags

 # app/views.py from
django.shortcuts import render
from .forms import RegForm #
creating a home view def
home_view(request):
   context = {}   form =
RegForm(request.POST or None)
context['form'] = form   return
render(request, "home.html", context)

#app/ templates/home.html

<html>

<body>

<form action="" method="POST">

  {{ form.as_p }}

  <input type="submit" value="Submit">

</form>

</body>

</html>

Use the command 'python manage.py runserver' to see the following output in the web page (Fig 6.1):
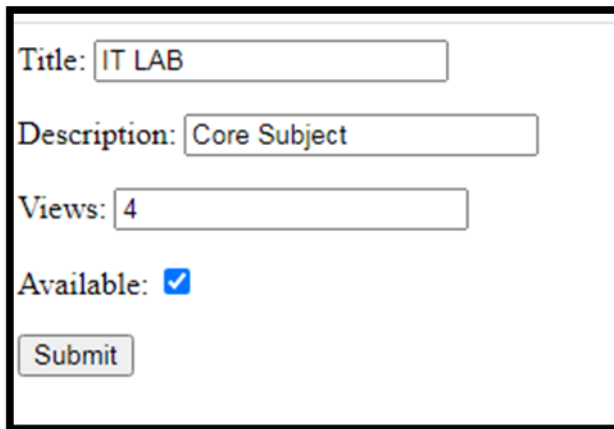
Fig 8.1

Widgets used in Django Forms:

A widget is Django's representation of an HTML input element. The widget handles the rendering of the HTML, and the extraction of data from a GET/POST dictionary that corresponds to the widget.

Whenever you specify a field on a form, Django will use a default widget that is appropriate to the type of data that is to be displayed.

However, if you want to use a different widget for a field, you can use the **widget** argument on the field definition. For example:

```
from django import forms class

CommentForm(forms.Form):

    name  =  forms.CharField()

url = forms.URLField()

    comment = forms.CharField(widget=forms.Textarea)
```

This would specify a form with a comment that uses a larger **Textarea** widget, rather than the default **TextInput** widget.

Widgets handling input of text

These widgets make use of the HTML elements **input** and **textarea**.

TextInput class

TextInput

- **input_type**: **'text'**
- **template_name**: **'django/forms/widgets/text.html'**
- Renders as: **<input type="text" ...>**

NumberInput class

NumberInput

- **input_type**: **'number'**
- **template_name**: **'django/forms/widgets/number.html'**
- Renders as: **<input type="number" ...>**

EmailInput class

EmailInput¶

- **input_type**: **'email'**
- **template_name**: **'django/forms/widgets/email.html'**

- Renders as: **\<input type="email" ...\>**

PasswordInput class

PasswordInput

- **input_type**: **'password'**
- **template_name**: **'django/forms/widgets/password.html'**
- Renders as: **\<input type="password" ...\>**

HiddenInput class

HiddenInput

- **input_type**: **'hidden'**
- **template_name**: **'django/forms/widgets/hidden.html'**
- Renders as: **\<input type="hidden" ...\>**

DateInput class

DateInput

- **input_type**: **'text'**
- **template_name**: **'django/forms/widgets/date.html'**
- Renders as: **\<input type="text" ...\>**

Textarea class

Textarea

- **template_name**: **'django/forms/widgets/textarea.html'**
- Renders as: **\<textarea\>...\</textarea\>**

CheckboxInput class

CheckboxInput

- **input_type**: **'checkbox'**
- **template_name**: **'django/forms/widgets/checkbox.html'**
- Renders as: **\<input type="checkbox" ...\>**

Select class

Select

- **template_name**: **'django/forms/widgets/select.html'**
- **option_template_name**: **'django/forms/widgets/select_option.html'** ☐
  Renders as: **<select><option ...>...</select>**

Sample code:

CHOICES= (('1','Choice1'), ('2','Choice2'), ('3','Choice3'),)   select =

forms.ChoiceField(widget=forms.Select, choices=CHOICES)

RadioSelect class

RadioSelect

- **template_name**: **'django/forms/widgets/radio.html'**
- **option_template_name**: **'django/forms/widgets/radio_option.html'**

Similar to **Select**, but rendered as a list of radio buttons within **<li>** tags:

<**li**><**input** type="radio" name="..."></**li**>

Sample Code:

YES_SMARTPHONE = 'Yes'
NO_SMARTPHONE = 'No'
SMART_PHONE_OWNERSHIP   =   ((YES_SMARTPHONE,   'Yes'),
(NO_SMARTPHONE, 'No'),)
smart_phone_ownership=forms.ChoiceField(widget=forms.RadioSelect(),
choices=SMART_PHONE_OWNERSHIP,   initial=   "",   label='Do   you   own   a
Smartphone?', required = False)

Custom Django Form field widgets:

We can override the default widget of each field for various purposes. To do so we need to
explicitly define the widget we want to assign to a field.
Make following changes to app/forms.py

```python
from django import forms

class GeeksForm(forms.Form):
    title = forms.CharField(widget = forms.Textarea)
    description = forms.CharField(widget = forms.CheckboxInput)
views = forms.IntegerField(widget = forms.TextInput)
available = forms.BooleanField(widget = forms.Textarea)
```

The output obtained will be as follows (Fig 6.2):

Fig 8.2

Solved Exercise:

A Sample program to demonstrate passage of multiple parameters from one page to another.

#loginapp/ forms.py

```python
from django import forms class
LoginForm(forms.Form):
```

```
    username = forms.CharField(max_length = 100)    contact_num =
forms.IntegerField()
```

#loginapp/views.py

```python
from django.shortcuts import render from
loginapp.forms import LoginForm def
login(request):    username = "not logged
in"    cn="not found"    if request.method
== "POST":
    #Get the posted form
        MyLoginForm    =    LoginForm(request.POST)
if MyLoginForm.is_valid():
        username = MyLoginForm.cleaned_data['username']            cn=
MyLoginForm.cleaned_data['contact_num']
        else:
        MyLoginForm = LoginForm()

  context = {'username': username,'contact_num':cn}

  return render(request, 'loggedin.html',context)
```

#loginapp/templates/login.html

```html
<html>
  <body>

    <form name = "form" action = "{% url 'login' %}"
    method = "POST" >{% csrf_token %}

      <div style = "max-width:470px;">
        <center>
          <input    type    =    "text"    style    =    "margin-left:20%;"
placeholder = "Identifiant" name = "username" />
        </center>
      </div>

      <br>
```

```html
      <div style = "max-width:470px;">
        <center>


          <input type = "number" style = "margin-left:20%;"
           placeholder = "contact_number" name = "contact_num" />            </center>
</div>

      <br>

      <div style = "max-width:470px;">
        <center>

          <button style = "border:0px; background-color:#4285F4; margin-top:8%;
height:35px; width:80%;margin-left:19%;" type = "submit"            value =
"Login" >            <strong>Login</strong>
          </button>

        </center>
      </div>

    </form>

  </body>
</html>
```

#loginapp/templates/loggedin.html

```html
<html>
  <body>
    You are : <strong>{{username}}</strong>
    Your number is : <strong>{{contact_num}}</strong>
  </body>   </html>
```

Output (Fig 6.3):

Command to be used: E:\MyFolder\FormProject > python manage.py runserver

Fig 8.3

Django Sessions:

Sessions are used to abstract the receiving and sending of cookies, data is saved on server side (like in database), and the client side cookie just has a session ID for identification. Sessions are also useful to avoid cases where the user browser is set to 'not accept' cookie

 Setting Up Sessions

In Django, enabling session is done in your project **settings.py**, by adding some lines to the **MIDDLEWARE_CLASSES** and the **INSTALLED_APPS** options. This should be done while creating the project, so **MIDDLEWARE_CLASSES** should have −

'django.contrib.sessions.middleware.SessionMiddleware'

And **INSTALLED_APPS** should have −

'django.contrib.sessions'

By default, Django saves session information in database (django_session table or collection), but we can configure the engine to store information using other ways like: in **file** or in **cache**.

When session is enabled, every request (first argument of any view in Django) has a session (dict) attribute.

Solved Exercise: #sessapp/forms.py

```python
from django import forms class LoginForm(forms.Form):
username = forms.CharField(max_length = 100)    password=
forms.CharField(widget= forms.PasswordInput())
```

#Sessapp/views.py

```python
from django.shortcuts import render from
sessapp.forms import LoginForm def
login(request):    username = 'not logged
in'      if request.method == 'POST':
         MyLoginForm     =     LoginForm(request.POST)
if MyLoginForm.is_valid():
         username = MyLoginForm.cleaned_data['username']
request.session['username'] = username        else:
       MyLoginForm = LoginForm()
  return render(request, 'loggedin.html', {"username" : username})

def formView(request):    if
request.session.has_key('username'):
username = request.session['username']
    return render(request, 'loggedin.html', {"username" : username})    else:
    return render(request, 'login.html', { })

def logout(request):    try:
    del request.session['username']
except:      pass
  return HttpResponse("<strong>You are logged out.</strong>")
```

#sessapp/templates/login.html

```html
<html>
  <body>

    <form name = "form" action = "{% url 'login' %}"
    method = "POST" >{% csrf_token %}

      <div style = "max-width:470px;">
        <center>
```

```html
        <input     type     =     "text"     style     =     "margin-left:20%;"
placeholder = "Identifiant" name = "username" />
        </center>
      </div>

      <br>

      <div style = "max-width:470px;">
        <center>
        <input     type     =     "password"     style     =     "margin-left:20%;"
placeholder = "password" name = "password" />
        </center>
      </div>

      <br>

      <div style = "max-width:470px;">
        <center>

        <button style = "border:0px; background-color:#4285F4; margin-top:8%;
height:35px; width:80%;margin-left:19%;" type = "submit"          value =
"Login" >          <strong>Login</strong>
        </button>

        </center>
      </div>

    </form>

  </body>
</html>
```

#sessapp/templates/loggedin.html

```html
    <html>

    <body>
      You are : <strong>{{username}}</strong>
    </body>
```

</html> **#sessapp/urls.py**

from django.conf.urls import url from
. import views

urlpatterns = [
  url(r'^connection/',views.formView, name = 'formView'),
  url(r'^login/', views.login, name = 'login'),
  url(r'^logout/', views.logout, name = 'logout'),
]

Output (Fig 6.4):

Commands to be used:

E:\MyFolder\SessProject> python manage.py migrate

E:\MyFolder\SessProject> python manage.py runserver

Fig 8.4

LAB ASSIGNMENTS:

1) Develop a web application using Django framework to demonstrate the transfer of multiple parameters between web pages. User should be presented with a dropdown list containing car manufacturers, a text box which takes model name of the manufacturer and a submit button. On submitting the web page, the user is forwarded to a new page. This new page should display the selected car manufacturer name and the model name.

2) Create a page firstPage.html with two TextBoxes [Name, Roll], DropDownList [Subjects], and a button. Create another page secondPage.html with a label and a button. When the user clicks the button in first Page, he should be sent to the second page and display the contents passed from first page in the label. The button in second page should navigate the user back to firstPage. Use Django sessions to transfer information.

HOME ASSIGNMENTS:

1) Develop a Web Application for Grocery Checklist Generation as shown in the figure below. It must have **checkboxes** which must be populated on page load listing grocery items. On clicking the **Add Item** button the selected Items and their prices have to be displayed in a Table. Set the borderstyle and border width for the table and its cells.

Select Item:
☑ Wheat
☐ Jaggery
☑ Dal

Item Name	Item Price
Wheat	40
Dal	80

Add Item

Lab No: 8

Form Processing using Django – Part II

LAB ASSIGNMENTS:

1) Create a Register page and Success page with the following requirements:
  i.   Register page should contain four input TextBoxes for UserName, Password, Email id and Contact Number and also a button to submit. Make the username as compulsory field and other fields as optional.
  ii.   On button click, Success page is displayed with message "Welcome {UserName}" and also his Email and Contact Number has to be displayed.
  iii.   Use secure technique to send details to the Success page (Hint: use csrftoken) 4) Design a website with two pages.

    2)   "How is the book ASP.NET with c# by Vipul Prakashan?" Give the user three choice: i)Good ii)Satisfactory iii)Bad. Provide a VOTE button. After user votes, present the result in percentage using labels next to the choices

    3)   Create a website with two pages. Page 1 has two TextBoxes (name and total marks) and one 'Calculate' Button as shown in the figure. On clicking the 'Calculate' Button, CGPA (total marks/50) along with the name should be displayed in the Page 2. Use Django sessions to store the information.

Name:	Alex		Welcome Alex
Total Marks:	450		Your CGPA is = 9
	Calculate	Page 1	Page 2

HOME ASSIGNMENTS:

a. Create a website with two pages.First page contains:

RadioButton with HP, Nokia, Samsung, Motorola, Apple as options.

CheckBox with Mobile and Laptop as items. TextBox to enter quantity.

There is a button with text as "Produce Bill".

On Clicking Produce Bill button, item should be displayed with total amount on another page**.**

2.   Write a program to display the following feedback form. The different options for the list box must be ASP-XML, DotNET, JavaPro and Unix,C,C++. When the Submit Form button is clicked after entering the data, a message as seen in the last line of the above figure must be displayed.

Lab No: 9

Databases – Part I

Objectives:

In this lab, student will be able to

1. Understand the MTV architecture
2. Create an App in Django and establish a connection with SQLite database
3. Set different privileges to different types of users.
4. Set the Django administrator account.

Django supports following databases

1. MySql
2. PostgreSql
3. Oracle
4. Sqllite

With the help of 3rd party backend Django supports following databases

1. SAP SQL Anywhere
2. IBM DB2
3. Microsoft SQL Server
4. Firebird
5. ODBC
6. ADSDB

Django abstracts the details of underlying database. One only need to specify the (models.py) python functions which will be converted into underlying database statements. Django supports CRUD operations. There are two way one can control the data on the website. First way is to use the admin interface second way is to use the forms.

Solved Exercise

Model is the name given to data abstraction part. To create the model you must first create an app. To create an app right click on the project→Add→DjangoApp
Let us name the app as "blog"

Step1: In settings.py add the app name (blog) under Installed_Apps as follows:

```
INSTALLED_APPS = [
    # Add your apps here to enable them
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'blog'
]
```
Under Templates provide the path of the template directory as follows

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR,'blog/templates/blog')],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

If you are using sqllite leave the default setting for database which will look as follows

```
DATABASES = {
 'default': {
    'ENGINE': 'django.db.backends.sqlite3',
    'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
 }
}
```

If you are using MySQL in that case modify the database entry as follows

```
DATABASES = {
      'default': {
                  'ENGINE':'django.db.backends.mysql',
 'OPTIONS': {' read_default_file': '/path/to/my.cnf',},
}
}
```

My.cnf file is as follows:

```
# my.cnf
[client]
database = NAME

user = USER

password = PASSWORD
default-character-set = utf8
```

Step2: Modify the Projects urls.py as given below from

django.conf.urls import include, url

Uncomment the next two lines to enable the admin:

from django.contrib import admin admin.autodiscover()

```
urlpatterns = [
    # Examples:
    # url(r'^$', MyBlog.views.archive, name='archive'),
    #url(r'^MyBlog/', include('MyBlog.MyBlog.urls')),
    # Uncomment the admin/doc line below to enable admin documentation:
    # url(r'^admin/doc/', include('django.contrib.admindocs.urls')),

    # Uncomment the next line to enable the admin:
 # (r'^$', 'django.views.generic.simple.redirect_to',
 # {'url': '/blog/'}),    url(r'^blog/',
```

include('blog.urls')),   url(r'^admin/',

include(admin.site.urls)),

]

Step3: Under blog app create a file named urls.py and type the following

from django.conf.urls import include,url from blog.views import

archive,create_blogpost

urlpatterns = [url(r'^$', archive, name='archive'),

url(r'^create/', create_blogpost, name='create_blogpost'),

    ]

Step4: Under models.py type the following from

django.db import models

Create your models here. from

django import forms class

BlogPost(models.Model):

title

models.CharField(max_length=

150)   body =

models.TextField()

timestamp =

models.DateTimeField()

```
   class Meta:
      ordering = ('-timestamp',)
class BlogPostForm(forms.ModelForm):
   class Meta:
      model = BlogPost
```

exclude = ('timestamp',)

It contains the details of table and Model form uses the model already created to create the form. This approach avoids duplication of code and goes with python philosophy Do not Repeat Yourself.

Step5: Registering your app in the admin: To register your app type the following into

admin.py from django.contrib import admin import site from blog.models import

BlogPost

Register your models here. from blog

import models class

BlogPostAdmin(admin.ModelAdmin):

   list_display = ('title', 'timestamp')

admin.site.register(models.BlogPost,BlogPostAdmin)

Step6: Type the following into views.py from

django.shortcuts import render

Create your views here. from datetime import

datetime from django.http import

HttpResponseRedirect from django.shortcuts

import render from blog.models import BlogPost,

BlogPostForm

def archive(request):

   posts = BlogPost.objects.all()[:10]

return render(request, 'archive.html',

                  {'posts': posts, 'form': BlogPostForm()})

def create_blogpost(request):

if request.method == 'POST':

     form = BlogPostForm(request.POST)

     if form.is_valid():

```
        post = form.save(commit=False)

post.timestamp=datetime.now()

        post.save()

    return HttpResponseRedirect('/blog/')
```

It is displaying the 10 most recent blogs posted by users/admin.

Step7: Type the following lines into archive.html

```html
<!DOCTYPE html>


<html lang="en" xmlns="http://www.w3.org/1999/xhtml">

<head>

    <meta charset="utf-8" />

    <title></title>

</head>

<body>

<form action="/blog/create/" method=post>{% csrf_token %}

    <table>{{ form }}</table><br>

    <input type=submit>

</form>

<hr>

{% for post in posts %}

    <h2>{{ post.title }}</h2>

    <p>{{ post.timestamp }}</p>

    <p>{{ post.body }}</p>
```

   `<hr>`

`{% endfor %}`

`</body> </html>`

This is the template which displays the blog posts that are separated by horizontal rule.

Once you have typed all of the above

i.　　　Go to Projects→Django Check

If it succeeds then

ii.　　　Goto Projects→Django Make Migrations

If it succeeds then

iii.　　　Goto Projects→Django Migrate

If it Succeeds then
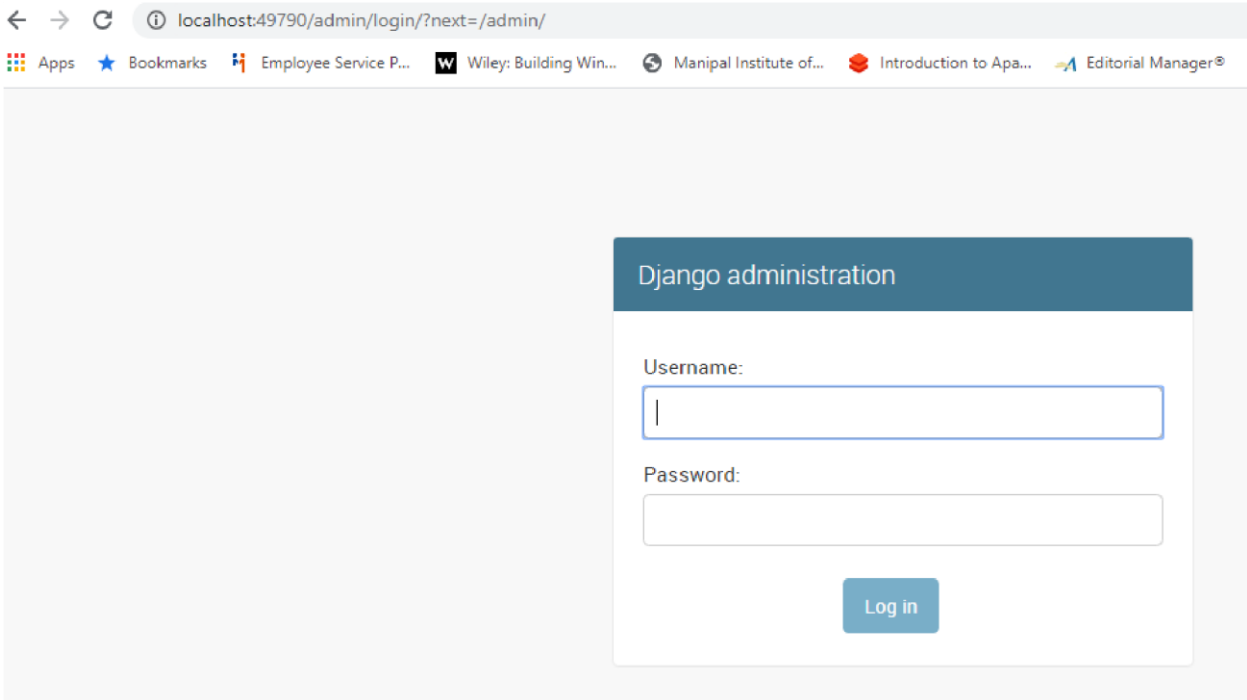
iv.　　　Goto Projects→Django Create Superuser

You have to repeat the above four steps whenever you modify the model or use different database.

Once you have created the superuser you can open the website. It looks as follows
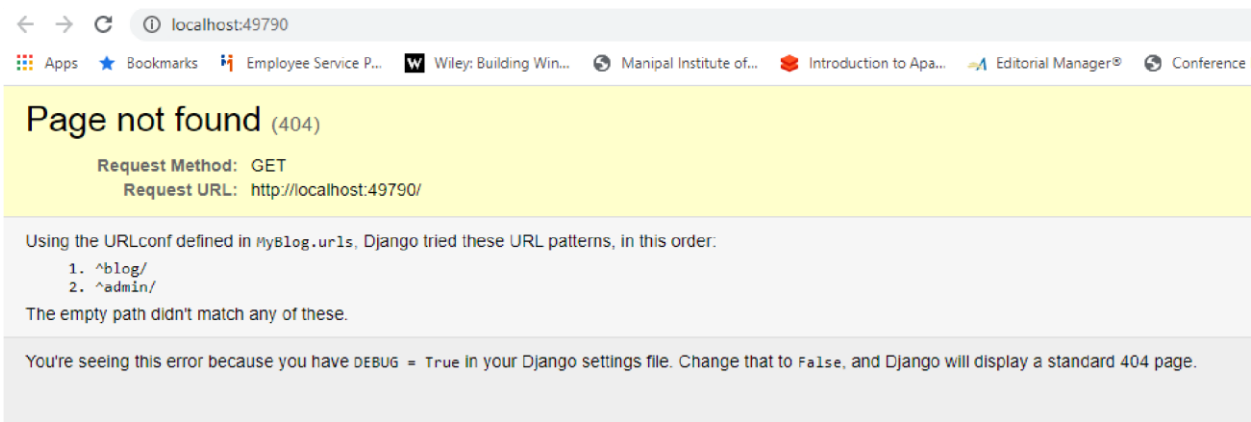
append /admin to the host name you will get the following screen

Type the superuser name and password you will be taken to following admin page. In the admin page you can see entry for Blog Posts as you have registered it.

Add a blog post You will be taken to following screen you can observe only title and timestamp are visible as per our code

Now you append the blog to the address you will get the following output.

As per our instructions users cannot edit the timestamp and current date and time will be taken for user entry. The blog entered by admin is also displayed. Once you enter the blog post details it will display it as under.

Title: [_____]

Body:

[   ]

Submit

Internet Technology Lab

May 10, 2020, 4:29 p.m.

Do the lab excercises

Internet Technology Lab

May 10, 2020, 10:50 a.m.

Welcome to the Lab

96

LAB ASSIGNMENTS:

1. Design a web site using Django, which is a website directory – A site containing links to other websites. A web page has different categories.
   - A category table has a name, number of visits, and number of likes.
   - A page table refers to a category, has a title, URL, and many views.

   Design a form that populates the above database and displays it.

2. Consider the following tables:
   WORKS(person-name,Company-name,Salary)
   LIVES(Person_name, Street, City)
   Assume Table data suitably. Design a Django webpage and include an option to insert data into WORKS table by accepting data from the user using TextBoxes. Also, include an option to retrieve the names of people who work for a particular company along with the cities they live in (particular company name must be accepted from the user).

HOME ASSIGNMENTS:

Assume a table "Institutes" with institute_id, name, and no_of_courses are the fields. Create a web page that retrieves all the data from "Institutes" table displays only Institute names in the list box.

Lab No: 10

Databases - Part II

LAB ASSIGNMENTS:

1. There are three tables in the database an author table has a first name, a last name and an email address. A publisher table has a name, a street address, a city, a state/province, a country, and a Web site. A book table has a title and a publication date. It also has one or more authors (a many-to-many relationship with authors) and a single publisher (a one-to-many relationship - aka foreign key - to publishers). Design a form which populates and retrieves the information from the above database using Django.

2. Create a Django Page for entry of a Product information (title, price and description) and save it into the db. Create the index page where you would view the product entries in an unordered list.

3. Create a web page with DropDownList, Textboxes and Buttons. Assume the table 'Human' with First name, Last name, Phone, Address and City as fields.
   When the page is loaded, only first names will be displayed in the drop-down list. On selecting the name, other details will be displayed in the respective TextBoxes. On clicking the update button, the table will be updated with new entries made in the text box. On clicking the delete button, the selected record will be deleted from the table, and the DropDownList is refreshed.

HOME ASSIGNMENTS:

Create a web page that receives the following information from a set of students: Student Id, Student Name, Course Name and Date of Birth. The application should also display the information of all the students once the data is Entered.

Lab No: 11

END SEMESTER EXAMINATION

Lab No: 12

MINI PROJECT EVALUATION

References:

1.   Mark Lutz, Learning Python, 5th Edition, O'Reilly, 2013

1.   Nigel George, Mastering Django, Packt Publishing, 2016.

2.   Leif Azzopardi and David Maxwell, Tango with Django 2, Apress, 2019