Diploma in

IT, Networking and Cloud

Module 4
Web Designing

Lab Manual

Learning outcome – Python Programming

After achieving this learning outcome, a student will be able to make websites, web servers, game frameworks, desktop and CLI applications, and IDE using Python. In order to achieve this learning outcome, a student has to complete the following:

Activities:

1. Install Python software in the system and print a string using print statement (4Hrs)
2. Print given string using indentation (space between characters) (1Hrs)
3. Define Integer Variables, floating variables and string variables (1Hr)
4. Write a program to add numbers and strings to the correct list using the append list method (2Hrs)
5. Write a python program to add, subtract, multiply and divide given two numbers by using arithmetic operators(2Hrs)
6. Write a python program multiplying strings to form string with repeating sequence (2Hrs)
7. Write a Python program to get the largest number from a list by using max and mini commands (1Hr)
8. Write a Python program to find whether a given number (accept from the user) is even or odd by using if else command (2Hrs)
9. Write a Python program to create a histogram from a given list of integers by using for while loop (1Hrs)
10. Write a Python program to compute the greatest common divisor (GCD) of two positive integers by using loops (2Hrs)
11. Write a Python program to get the least common multiple (LCM) of two positive integers using if else and while commands(2Hrs)
12. Write a Python program to sort (ascending and descending) a dictionary by value (2Hrs)
13. Write a Python program to create a tuple. (2Hrs)
14. Write a Python program to create a tuple with different data types (1Hrs)
15. Write a Python program to create a dictionary and perform functional operations (2Hrs)
16. Write a Python program to find maximum and the minimum value in a set. (1 Hrs)
17. Write a python program to copy content of a text file to another file (3Hrs)
18. Write a python program to copy content of an image/video file to another file (3Hrs)
19. Write program to demonstrate exception handling while creating file, reading file, writing file and deleting a file(4 Hrs)
20. Write a program to demonstrate file handling in connecting to database and performing operations (2 Hrs)
21. Create student database in mysql and connect using python (3 Hrs)
22. Perform CRUD operations on mysql database using python (4 Hrs)
23. Perform CRUD operations on MongoDB database using python (3 Hrs)
24. Create a student class with related data elements and methods. Write functionalities for basic operations of student in a college (3Hrs)
25. With student class already created, perform database crud operations including validation of data while writing to DB and when reading from DB (4 Hrs)
26. Create a simple python Flask app with at least 3 basic routes (4 Hrs)

Learning outcome – Web Application with Django Framework

After achieving this learning outcome, a student will be able to make websites, web servers, game frameworks, desktop and CLI applications, and IDE using Python. In order to achieve this learning outcome, a student has to complete the following:

Activities:

Skills related to working with django framework (120 hr)

1. Install django framework libraries in python and test for installation
2. Setup first django application
3. Creating django application with MVT architecture
4. Create django application that handles file uploading
5. Create django application that reads data from CSV and display on page
6. Create django application that reads data from JSON and display on page
7. Creating django application which implement CRUD operations over database
8. Create django application validates user credentials on login page
9. Create admin panel using django, using AJAX
10. Perform crud operations in django app using AJAX at single page
11. Create django application which sends email to any recipient

Activity 1

Aim: Install Python and write a Python program to print a string using the print statement

Learning outcome: Able to make websites, web servers, game frameworks, desktop and CLI applications, and IDE using Python.

Duration: 4 hours

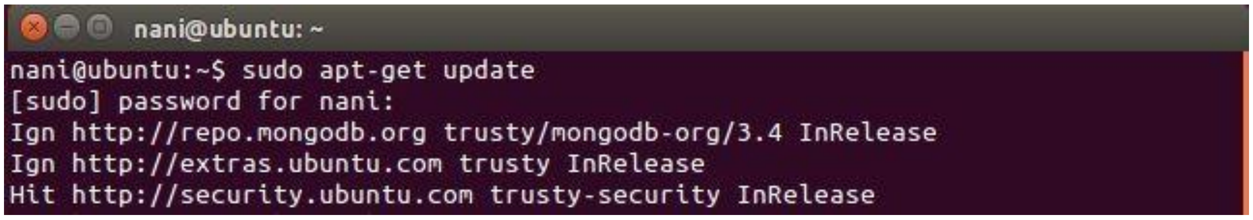
List of Hardware/Software requirements:

1. Laptop/Computer with Windows/Linux OS - Ubuntu 18.04 LTS
2. Python Software

Program:

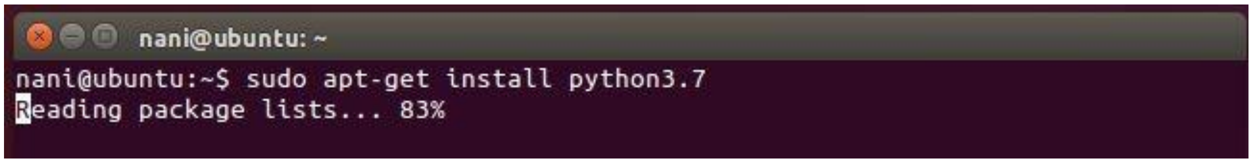
Installation steps for Linux:

- Ctrl+Alt+T Open Terminal
- Type Command : sudo apt-get update



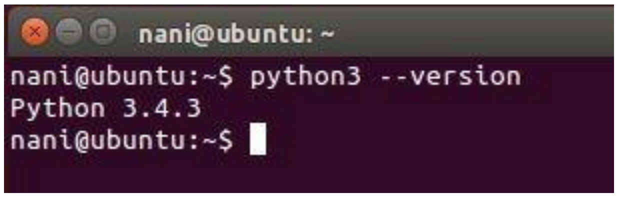
```
nani@ubuntu: ~
nani@ubuntu:~$ sudo apt-get update
[sudo] password for nani:
Ign http://repo.mongodb.org trusty/mongodb-org/3.4 InRelease
Ign http://extras.ubuntu.com trusty InRelease
Hit http://security.ubuntu.com trusty-security InRelease
```

- Type command : sudo apt-get install python3.7



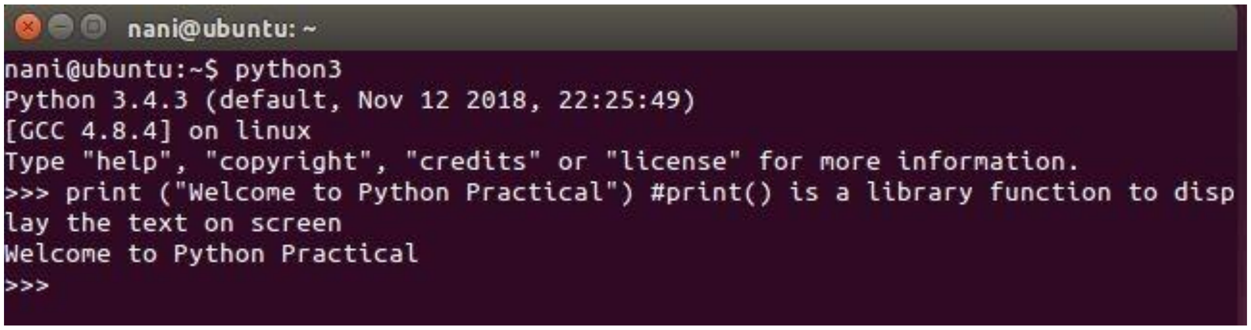
```
nani@ubuntu: ~
nani@ubuntu:~$ sudo apt-get install python3.7
Reading package lists... 83%
```

- After installation check python version



```
nani@ubuntu: ~
nani@ubuntu:~$ python3 --version
Python 3.4.3
nani@ubuntu:~$
```

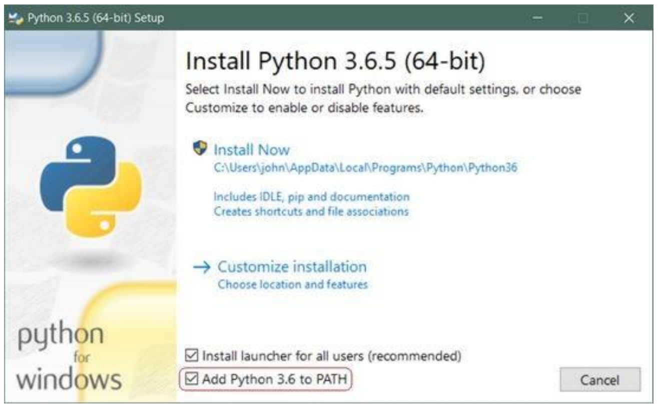
- Start programming



```
nani@ubuntu: ~
nani@ubuntu:~$ python3
Python 3.4.3 (default, Nov 12 2018, 22:25:49)
[GCC 4.8.4] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print ("Welcome to Python Practical") #print() is a library function to disp
lay the text on screen
Welcome to Python Practical
>>>
```

Installation Steps for Windows:

- Install Python software in the system.
- Open the browser and type the python.org/downloads.
- Click on download .
- Choice either Windows x86-64 executable installer for 64-bit or Windows x86 executable installer for 32-bit.
- After downloading a file the below page will appear.



Python 3.6.5 (64-bit) Setup

Install Python 3.6.5 (64-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

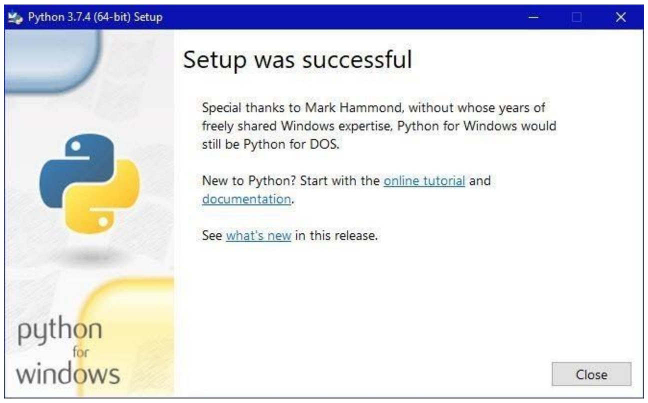
Install Now
C:\Users\john\AppData\Local\Programs\Python\Python36

Includes IDLE, pip and documentation
Creates shortcuts and file associations

Customize installation
Choose location and features

☑ Install launcher for all users (recommended)
☑ Add Python 3.6 to PATH

Figure 1: Installing



Setup was successful

Special thanks to Mark Hammond, without whose years of freely shared Windows expertise, Python for Windows would still be Python for DOS.

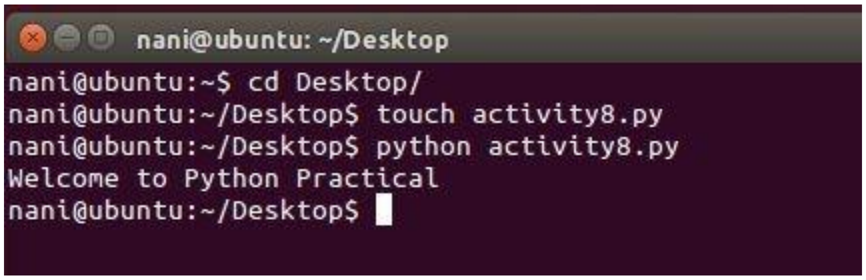
New to Python? Start with the online tutorial and documentation.

See what's new in this release.

Figure 2: Installed successfully

print ("Welcome to Python Practical") #print() is a library function to display the text on screen

Output/Results snippet:



```
nani@ubuntu:~$ cd Desktop/
nani@ubuntu:~/Desktop$ touch activity8.py
nani@ubuntu:~/Desktop$ python activity8.py
Welcome to Python Practical
nani@ubuntu:~/Desktop$
```

References:

1. https://www.ics.uci.edu/~pattis/common/handouts/pythoneclipsejava/python.html

Activity 2

Aim: Write a python program to print given string using indentation (space between characters)

Learning outcome: Able to make websites, web servers, game frameworks, desktop and CLI applications, and IDE using Python.

Duration: 1 hour

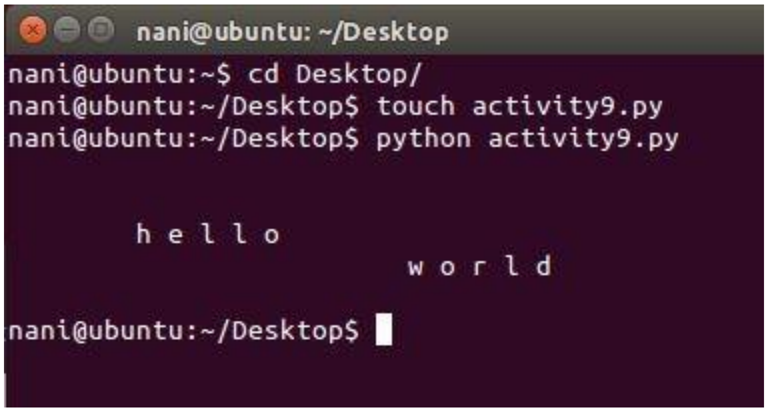
List of Hardware/Software requirements:

1. Laptop/Computer with Linux OS - Ubuntu 18.04 LTS
2. Python3 Software

Program:

```
print("\n\n\th e l l o \n \t \t \t w o r l d \n") # tab \t, new line \n and space is used
```

Output/Results snippet:



```
nani@ubuntu:~$ cd Desktop/
nani@ubuntu:~/Desktop$ touch activity9.py
nani@ubuntu:~/Desktop$ python activity9.py


        h e l l o
                         w o r l d

nani@ubuntu:~/Desktop$ 
```

References:

1. https://stackoverflow.com/questions/18756510/printing-with-indentation-in-python

Activity 3

Aim: Define Integer Variables, floating variables and string variables.

Learning outcome: Able to make websites, web servers, game frameworks, desktop and CLI applications, and IDE using Python.

Duration: 1 hour

List of Hardware/Software requirements:

1. Laptop/Computer with Linux OS - Ubuntu 18.04 LTS
2. Python3 Software

Program:

```
myint = 7                        # declare integer variable
print(myint)                     # print the value of the variable
myfloat = 7.0                            # declare floating variable
print(myfloat)
myfloat = float(7)
print(myfloat)
mystring = 'hello'               # declare string variable
print(mystring)
mystring = "hello"
print(mystring)
```

Output:

```
7

7.0

7.0

hello

hello
```

References:

- https://www.tutorialspoint.com/python/python_overview.htm
- https://www.w3schools.com/python/
- https://www.javatpoint.com/python-tutorial

Activity 4

Aim: Write a program to add numbers and strings to the correct list using the append list method.

Learning outcome: Able to make websites, web servers, game frameworks, desktop and CLI applications, and IDE using Python.

Duration: 2 hours

List of Hardware/Software requirements:

1. Laptop/Computer with Linux OS - Ubuntu 18.04 LTS
2. Python3 Software

Program:

```
numbers = [1,2,3]
strings = ["Hello","World"]                    # declare and initialize list
names = ["John", "Eric", "Jessica"]
second_name = names[1]
print(numbers)                                 # print the values of list
print(strings)
print("The second name on the names list is %s" % second_name)
```

Output:

```
[1, 2, 3]

['Hello', 'World']

The second name on the names list is ['Eric']
```

References:

- https://www.tutorialspoint.com/python/python_overview.htm
- https://www.w3schools.com/python/
- https://www.javatpoint.com/python-tutorial

Activity 5

Aim: Write a python program to add, subtract, multiply and divide given two numbers by using arithmetic operators.

Learning outcome: Able to make websites, web servers, game frameworks, desktop and CLI applications, and IDE using Python.

Duration: 2 hours

List of Hardware/Software requirements:

1. Laptop/Computer with Linux OS - Ubuntu 18.04 LTS
2. Python3 Software

Program:

```
print("1. Addition");
print("2. Subtraction");
print("3. Multiplication");
print("4. Division");
print("5. Exit");
choice = int(input("Enter your choice: "));          # take input from console
if (choice>=1 and choice<=4):                        # taking input from 1 to 4
    print("Enter two numbers: ");
    num1 = int(input());
    num2 = int(input());
if choice == 1:                                              # perform addition if 1
    res = num1 + num2;
    print("Result = ", res);
elif choice == 2:                                    # perform subtraction if 2
    res = num1 - num2;
    print("Result = ", res);
elif choice == 3:                                    # perform multiplication if 3
    res = num1 * num2;
    print("Result = ", res);
elif choice == 4:                                    # perform division if 4
    res = num1 / num2;
    print("Result = ", res);
elif choice == 5:
    exit();                                          # exit if 5
else:
    print("Wrong input..!!");                        # print message for any other input
```

Output:

```
1. Addition

2. Subtraction

3. Multiplication

4. Division

5. Exit

Enter your choice: 1

Enter two numbers:

2

4

('Result = ', 6)
```

References:

- https://www.tutorialspoint.com/python/python_overview.htm
- https://www.w3schools.com/python/
- https://www.javatpoint.com/python-tutorial

Activity 6

Aim: Write a python program multiplying strings to form a string with repeating sequence.

Learning outcome: Able to make websites, web servers, game frameworks, desktop and CLI applications, and IDE using Python.

Duration: 2 hours

List of Hardware/Software requirements:

1. Laptop/Computer with Linux OS - Ubuntu 18.04 LTS
2. Python3 Software

Program:

```
x = 'Welcome '
y = 'python '
print ((x+y)*5)                         # printing the concatenated strings 5 times
```

Output:

```
Welcome python Welcome python Welcome python Welcome python Welcome python
```

References:

- https://www.tutorialspoint.com/python/python_overview.htm
- https://www.w3schools.com/python/
- https://www.javatpoint.com/python-tutorial

Activity 7

Aim: Write a Python program to get the largest number from a list by using max and min commands.

Learning outcome: Able to make websites, web servers, game frameworks, desktop and CLI applications, and IDE using Python.

Duration: 1 hour

List of Hardware/Software requirements:

1. Laptop/Computer with Linux OS - Ubuntu 18.04 LTS
2. Python3 Software

Program:

```
list = [1, 2, 3]
print (max(list))              # max() finds the maximum among values
print (min(list))              # min() finds the minimum among values
```

Output:

```
3

1
```

References:

- https://www.tutorialspoint.com/python/python_overview.htm
- https://www.w3schools.com/python/
- https://www.javatpoint.com/python-tutorial

Activity 8

Aim: Write a Python program to find whether a given number (accept from the user) is even or odd by using if else command.

Learning outcome: Able to make websites, web servers, game frameworks, desktop and CLI applications, and IDE using Python.

Duration: 2 hours

List of Hardware/Software requirements:

1. Laptop/Computer with Linux OS - Ubuntu 18.04 LTS
2. Python3 Software

Program:

```
num = int(input("Enter a number: "))                # taking input from console
mod = num % 2                                    # extracting the remainder value of input
if mod > 0:
    print("This is an odd number.")
else:
    print("This is an even number.")
```

Output:

```
Enter a number:  5

This is an odd number.

Enter a number: 2

This is an even number.
```

References:

- https://www.tutorialspoint.com/python/python_overview.htm
- https://www.w3schools.com/python/
- https://www.javatpoint.com/python-tutorial

Activity 9

Aim: Write a Python program to create a histogram from a given list of integers by using for while loop.

Learning outcome: Able to make websites, web servers, game frameworks, desktop and CLI applications, and IDE using Python.

Duration: 1 hour

List of Hardware/Software requirements:

1. Laptop/Computer with Linux OS - Ubuntu 18.04 LTS
2. Python3 Software

Program:

```
def histogram( items ):                        # create function
    for n in items:                            # for loop through values of n
            output = ''
            times = n
            while( times > 0 ):                # check condition of counter
                    output += '*'
                    times = times–1    # decrement the counter
            print(output)                      # print the output
histogram([2, 3, 6, 5])
```

Output:

```
**

***

******

*****
```

References:

- https://www.tutorialspoint.com/python/python_overview.htm
- https://www.w3schools.com/python/
- https://www.javatpoint.com/python-tutorial

Activity 10

Aim: Write a Python program to compute the greatest common divisor (GCD) of two positive integers by using loop

Learning outcome: Able to make websites, web servers, game frameworks, desktop and CLI applications, and IDE using Python.

Duration: 2 hours

List of Hardware/Software requirements:

1. Laptop/Computer with Linux OS - Ubuntu 18.04 LTS
2. Python3 Software

Program:

```
def gcd(x, y):                                              # create function gcd()
    gcd = 1
    if x % y == 0:
        return y
    for k in range(int(y / 2), 0, -1):          # for loop from y/2 to 0 increment -1
        if x % k == 0 and y % k == 0:
            gcd = k
            break
    return gcd
print(gcd(12, 17))                                   # print the gcd value
print(gcd(4, 6))
```

Output:

```
1
2
```

References:

- https://www.tutorialspoint.com/python/python_overview.htm
- https://www.w3schools.com/python/
- https://www.javatpoint.com/python-tutorial

Activity 11

Aim: Write a Python program to get the least common multiple (LCM) of two positive integers using if else and while commands.

Learning outcome: Able to make websites, web servers, game frameworks, desktop and CLI applications, and IDE using Python.

Duration: 2 hours

List of Hardware/Software requirements:

1. Laptop/Computer with Linux OS - Ubuntu 18.04 LTS
2. Python3 Software

Program:

```
def lcm(x, y):                                  # define function
    if x > y:
        z = x
    else:
        z = y
    while(True):
        if((z % x == 0) and (z % y == 0)):      # check if both conditions are true
            lcm = z
            break
        z += 1
    return lcm
print(lcm(4, 6))                                # print the lcm value
print(lcm(15, 17))
```

Output:

```
12
255
```

References:

- https://www.tutorialspoint.com/python/python_overview.htm
- https://www.w3schools.com/python/
- https://www.javatpoint.com/python-tutorial

Activity 12

Aim: Write a Python program to sort (ascending and descending) a dictionary by value

Learning outcome: Able to make websites, web servers, game frameworks, desktop and CLI applications, and IDE using Python.

Duration: 2 hour

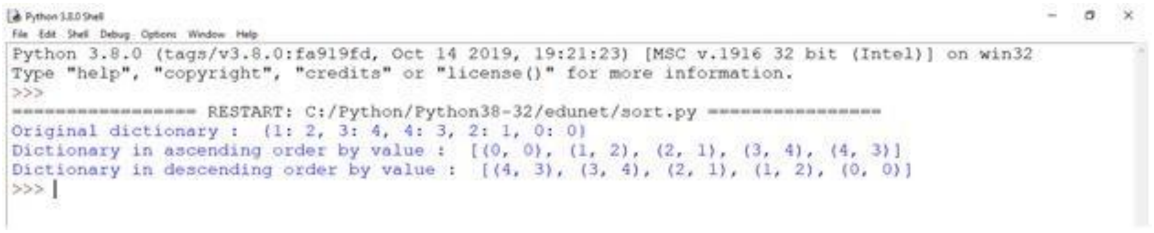
List of Hardware/Software requirements:

1. Laptop/Computer with Linux OS - Ubuntu 18.04 LTS
2. Python 3 software

Program:

```
import operator
d = {1: 2, 3: 4, 4: 3, 2: 1, 0: 0}
print('Original dictionary : ',d)
sorted_d = sorted(d.items(), key=operator.itemgetter(0))
print('Dictionary in ascending order by value : ',sorted_d)
sorted_d = sorted(d.items(), key=operator.itemgetter(0),reverse=True)
print('Dictionary in descending order by value : ',sorted_d)
```

Output/Results snippet:



```
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================= RESTART: C:/Python/Python38-32/edunet/sort.py ================
Original dictionary :  {1: 2, 3: 4, 4: 3, 2: 1, 0: 0}
Dictionary in ascending order by value :  [(0, 0), (1, 2), (2, 1), (3, 4), (4, 3)]
Dictionary in descending order by value :  [(4, 3), (3, 4), (2, 1), (1, 2), (0, 0)]
>>> |
```

References:

- https://www.w3resource.com/python-exercises/dictionary/python-data-type-dictionary-exercise-1.php
- https://stackoverflow.com/questions/20577840/python-dictionary-sorting-in-descending-order-based-on-values/41866830
- https://www.youtube.com/watch?v=trWU2GDqXS4
- https://www.youtube.com/watch?v=-UGJvJNZ7i4

Activity 13

Aim: Write a Python program to create a tuple.

Learning outcome: Able to make websites, web servers, game frameworks, desktop and CLI applications, and IDE using Python.

Duration: 2 hour

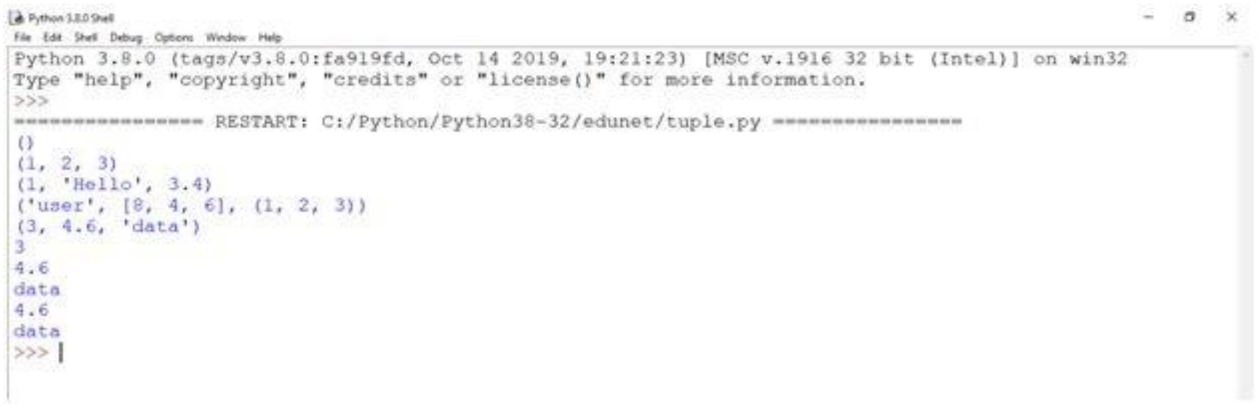
List of Hardware/Software requirements:

1. Laptop/Computer with Linux OS - Ubuntu 18.04 LTS
2. Python 3 software

Program:

```
# empty tuple
# Output: ()
my_tuple = ()
print(my_tuple)
# tuple having integers
# Output: (1, 2, 3)
my_tuple = (1, 2, 3)
print(my_tuple)
# tuple with mixed data types
# Output: (1, "Hello", 3.4)
my_tuple = (1, "Hello", 3.4)
print(my_tuple)
# nested tuple
# Output: ("user", [8, 4, 6], (1, 2, 3))
my_tuple = ("user", [8, 4, 6], (1, 2, 3))
print(my_tuple)
# tuple can be created without parentheses
# also called tuple packing
# Output: 3, 4.6, "data"
my_tuple = 3, 4.6, "data"
print(my_tuple)
# tuple unpacking is also possible
```

Output/Results snippet:



```
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================ RESTART: C:/Python/Python38-32/edunet/tuple.py ================
()
(1, 2, 3)
(1, 'Hello', 3.4)
('user', [8, 4, 6], (1, 2, 3))
(3, 4.6, 'data')
3
4.6
data
4.6
data
>>> 
```

References:

- https://www.w3schools.com/python/python_tuples.asp
- https://www.w3resource.com/python-exercises/tuple/python-tuple-exercise-3.php
- https://www.programiz.com/python-programming/tuple
- https://www.geeksforgeeks.org/tuples-in-python/

Activity 14

Aim: Write a Python program to create a tuple with different data types

Learning outcome: Able to make websites, web servers, game frameworks, desktop and CLI applications, and IDE using Python.

Duration: 1 hour

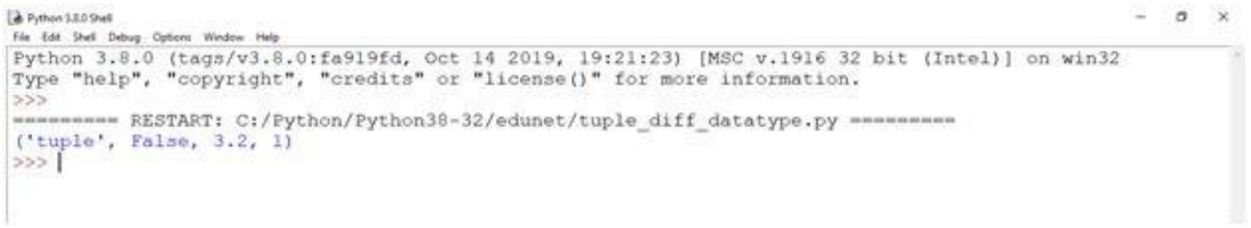
List of Hardware/Software requirements:

1. Laptop/Computer with Linux OS - Ubuntu 18.04 LTS
2. Python 3 software

Program:

```
#Create a tuple with different data types
tuplex = ("tuple", False, 3.2, 1)
print(tuplex)
```

Output/Results snippet:



```
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========= RESTART: C:/Python/Python38-32/edunet/tuple_diff_datatype.py =========
('tuple', False, 3.2, 1)
>>> |
```

References:

- https://www.w3resource.com/python-exercises/tuple/python-tuple-exercise-2.php
- https://www.youtube.com/watch?v=eXnZfHwzSiI
- http://python.mykvs.in/Programs/class%20xi/cs/tuple%20p.pdf

Activity 15

Aim: Write a Python program to create a set

Learning outcome: Able to make websites, web servers, game frameworks, desktop and CLI applications, and IDE using Python.

Duration: 2 hour

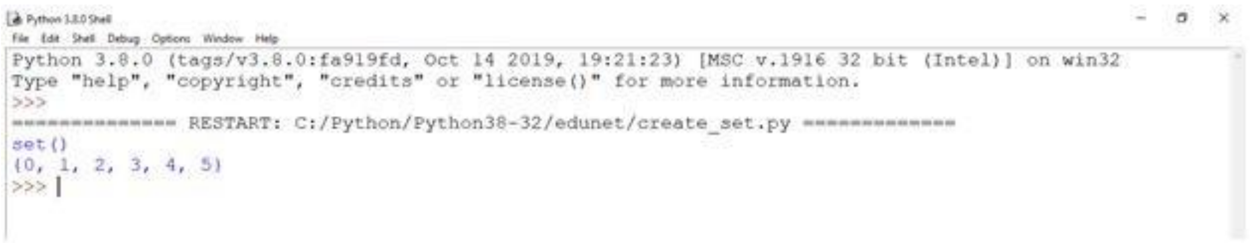
List of Hardware/Software requirements:

1. Laptop/Computer with Linux OS - Ubuntu 18.04 LTS
2. Python 3 software

Program:

```
#A new empty set
setx = set()
print(setx)
set()
#A non empty set
n = set([0, 1, 2, 3, 4, 5])
print(n)
```

Output/Results snippet:



```
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: C:/Python/Python38-32/edunet/create_set.py =============
set()
{0, 1, 2, 3, 4, 5}
>>> 
```

References:

- https://www.w3schools.com/python/python_sets.asp
- https://www.geeksforgeeks.org/python-sets/
- https://www.youtube.com/watch?v=MEPlLAjPvXY

Activity 16

Aim: Write a Python program to add member(s) in a set

Learning outcome: Able to make websites, web servers, game frameworks, desktop and CLI applications, and IDE using Python.

Duration: 1 hour

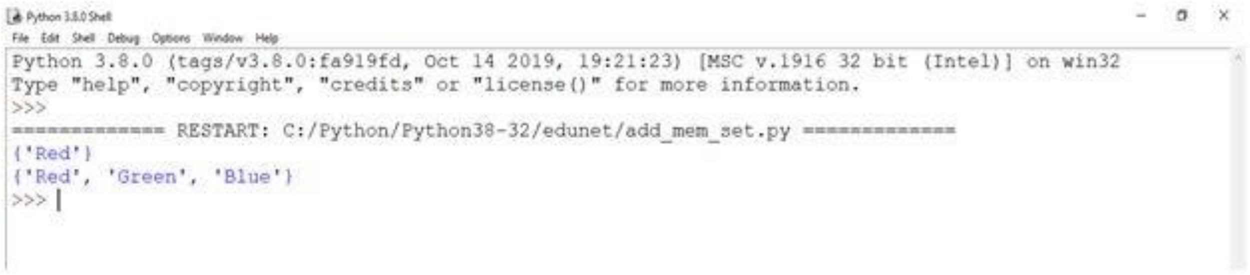
List of Hardware/Software requirements:

1. Laptop/Computer with Linux OS - Ubuntu 18.04 LTS
2. Python 3 software

Program:

```
#A new empty set
color_set = set()
#Add a single member
color_set.add("Red")
print(color_set)
#Add multiple items
color_set.update(["Blue", "Green"])
print(color_set)
```

Output/Results snippet:



```
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: C:/Python/Python38-32/edunet/add_mem_set.py =============
{'Red'}
{'Red', 'Green', 'Blue'}
>>> 
```

References:

- https://www.w3resource.com/python-exercises/sets/python-sets-exercise-3.php
- https://www.geeksforgeeks.org/set-add-python/

Activity 17

Aim: Write a Python program to find maximum and the minimum value in a set.

Learning outcome: Able to make websites, web servers, game frameworks, desktop and CLI applications, and IDE using Python.

Duration: 1 hour

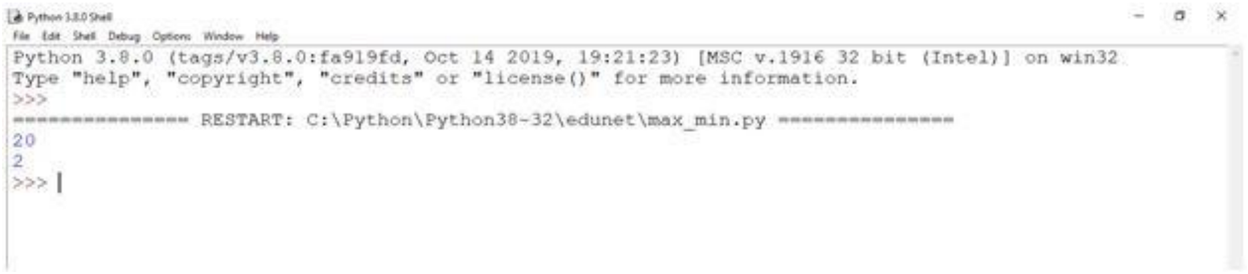
List of Hardware/Software requirements:

1. Laptop/Computer with Linux OS - Ubuntu 18.04 LTS
2. Python 3 software

Program:

```
#Create a set
seta = set([5, 10, 3, 15, 2, 20])
#Find maximum value
print(max(seta))
#Find minimum value
print(min(seta))
```

Output/Results snippet:



```
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=============== RESTART: C:\Python\Python38-32\edunet\max_min.py ===============
20
2
>>> |
```

References:

- https://www.w3resource.com/python-exercises/sets/python-sets-exercise-14.php
- https://www.geeksforgeeks.org/python-maximum-minimum-set/

Activity 18

Aim: Write a Python program to find the length of a set (1 Hrs)

Learning outcome: Able to make websites, web servers, game frameworks, desktop and CLI applications, and IDE using Python.

Duration: 1 hour

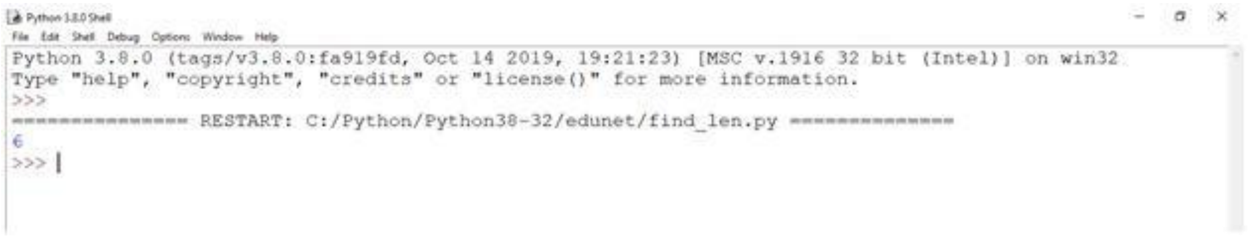
List of Hardware/Software requirements:

1. Laptop/Computer with Linux OS - Ubuntu 18.04 LTS
2. Python 3 software

Program:

```
#Create a set
seta = set([5, 10, 3, 15, 2, 20])
#Find the length use len()
print(len(seta))
```

Output/Results snippet:



```
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=============== RESTART: C:/Python/Python38-32/edunet/find_len.py ===============
6
>>> |
```

References:

- https://www.w3resource.com/python-exercises/sets/python-sets-exercise-15.php
- https://www.edureka.co/blog/python-list-length/
- https://www.youtube.com/watch?v=hbVekSSSzVM

Activity 19

Aim: Write a Python program to convert temperatures to and from Centigrade to Fahrenheit

Learning outcome: Able to make websites, web servers, game frameworks, desktop and CLI applications, and IDE using Python.

Duration: 2 hour

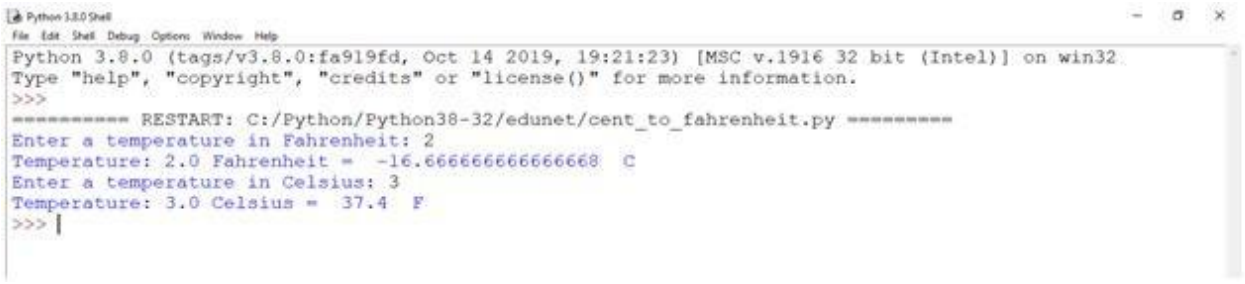
List of Hardware/Software requirements:

1. Laptop/Computer with Linux OS - Ubuntu 18.04 LTS
2. Python 3 software

Program:

```
Fahrenheit =  float(input("Enter a temperature in Fahrenheit: "))
Celsius = (Fahrenheit - 32) * 5.0/9.0
print ("Temperature:", Fahrenheit, "Fahrenheit = ", Celsius, " C")
Celsius =  float(input("Enter a temperature in Celsius: "))
Fahrenheit = 9.0/5.0 * Celsius + 32
print ("Temperature:", Celsius, "Celsius = ", Fahrenheit, " F")
```

Output/Results snippet:



```
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:/Python/Python38-32/edunet/cent_to_fahrenheit.py =========
Enter a temperature in Fahrenheit: 2
Temperature: 2.0 Fahrenheit =  -16.666666666666668  C
Enter a temperature in Celsius: 3
Temperature: 3.0 Celsius =  37.4  F
>>> |
```

References:

- https://beginnersbook.com/2019/05/python-Program-to-convert-celsius-to-fahrenheit-and-vice-versa/
- https://www.youtube.com/watch?v=_fxLlOO0Pts
- https://www.Programming-techniques.com/2019/03/python-Program-to-convert-celsius-to-fahrenheit-and-vice-versa.html

Activity 20

Aim: Write a python program to find Fibonacci series

Learning outcome: Able to make websites, web servers, game frameworks, desktop and CLI applications, and IDE using Python.

Duration: 2 hour

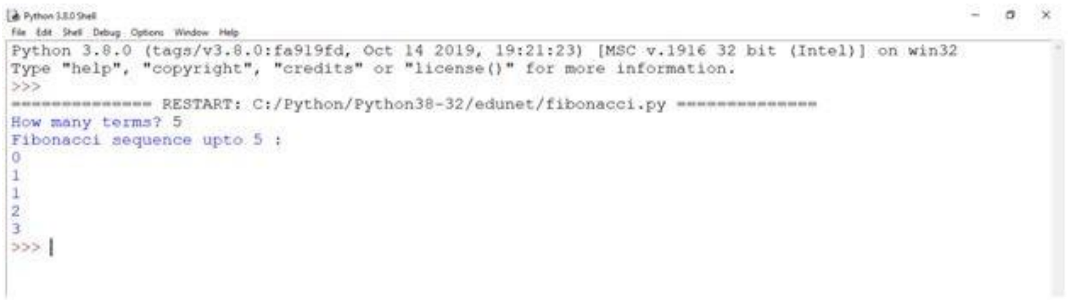
List of Hardware/Software requirements:

1. Laptop/Computer with Linux OS - Ubuntu 18.04 LTS
2. Python 3 software

Program:

```
pterms = int(input("How many terms? "))
n1 = 0
n2 = 1
count = 0
if pterms <= 0:
 print("Please enter a positive integer")
elif pterms == 1:
 print("Fibonacci sequence upto",pterms,":")
 print(n1)
else:
 print("Fibonacci sequence upto",pterms,":")
 while count < pterms:
     print(n1,end='\n')
      nth = n1 + n2
      n1 = n2
      n2 = nth
      count += 1
```

Output/Results snippet:



```
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: C:/Python/Python38-32/edunet/fibonacci.py ==============
How many terms? 5
Fibonacci sequence upto 5 :
0
1
1
2
3
>>> |
```

Activity 21

Aim: Write a python program to find factorial using function in Python IDLE.

Learning outcome: Able to make websites, web servers, game frameworks, desktop and CLI applications, and IDE using Python.

Duration: 2 hour

List of Hardware/Software requirements:

1. Laptop/Computer with Linux OS - Ubuntu 18.04 LTS
2. Python 3 software

Program:

```
# Function name using recursion_function(n)
def recursion_factorial(n):
if n == 1:
return n
else:
return n*recursion_factorial(n-1)
# read the value of input for factorial number
num = int(input('\n Enter your factorial number '))
# check if the number is negative
if num < 0:
print("Sorry, factorial does not exist its negative numbers")
elif num == 0:
print("The factorial of 0 is 1")
else:
print("The factorial of", num, "is", recursion_factorial(num))
```

Output/Results snippet

Activity 22

Aim: Write a python program to find whether the given string is palindrome or not by using function.

Learning outcome: Able to make websites, web servers, game frameworks, desktop and CLI applications, and IDE using Python.

Duration: 2 hour

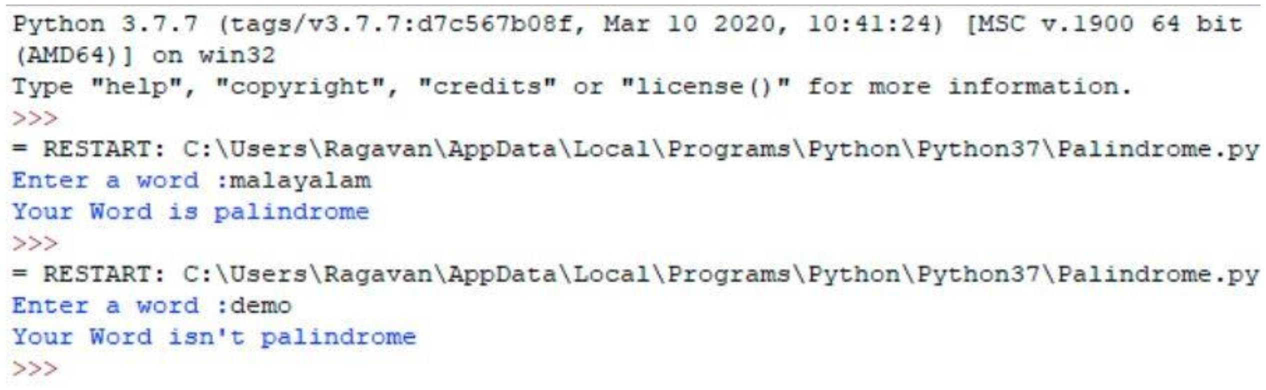
List of Hardware/Software requirements:

1. Laptop/Computer with Linux OS - Ubuntu 18.04 LTS
2. Python 3 software

Program:

```
#enter input of the word
Word = str(input("Enter a word :"))
#check the word whether palindrome or not
if(Word==Word[::-1]):
print("Your Word is palindrome")
else:
print("Your Word isn't palindrome")
```

Output/Results Snippet



```
Python 3.7.7 (tags/v3.7.7:d7c567b08f, Mar 10 2020, 10:41:24) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\Ragavan\AppData\Local\Programs\Python\Python37\Palindrome.py
Enter a word :malayalam
Your Word is palindrome
>>>
= RESTART: C:\Users\Ragavan\AppData\Local\Programs\Python\Python37\Palindrome.py
Enter a word :demo
Your Word isn't palindrome
>>>
```

References

- https://docs.python.org/3/tutorial/
- https://www.tutorialsteacher.com/python
- https://realpython.com/

Activity 23

Aim: Write a python class to reverse a string word by word.

Learning outcome: Able to make websites, web servers, game frameworks, desktop and CLI applications, and IDE using Python.

Duration: 2 hour

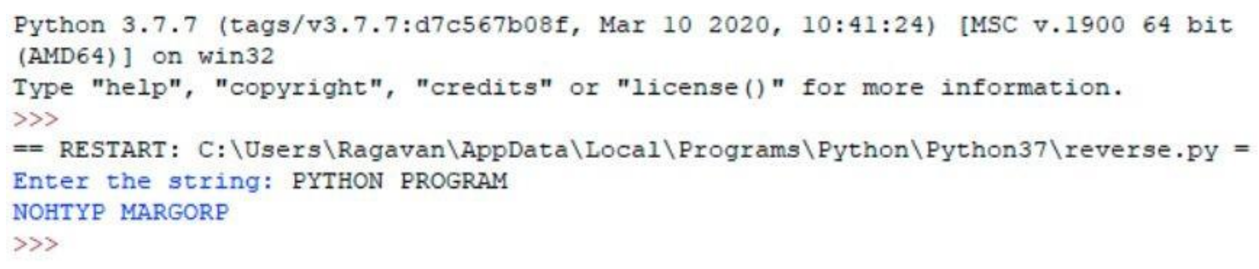
List of Hardware/Software requirements:

1. Laptop/Computer with Linux OS - Ubuntu 18.04 LTS
2. Python 3 software

Program:

```
# Python program to Reverse each word of a string
# function definition
def reverseword(input_word):
w = input_word.split(" ")
# Splitting the string into a list of words
# reversing each word and creating a new list of words
nw = [i[::-1] for i in w]
# Joining the new list of words to form a new string
ns = " ".join(nw)
return ns
# main() method
input_word = input("Enter the string: ")
print(reverseword(input_word))
```

Output/Result Snippet



```
Python 3.7.7 (tags/v3.7.7:d7c567b08f, Mar 10 2020, 10:41:24) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
== RESTART: C:\Users\Ragavan\AppData\Local\Programs\Python\Python37\reverse.py =
Enter the string: PYTHON PROGRAM
NOHTYP MARGORP
>>>
```

References

- https://docs.python.org/3/tutorial/
- https://www.tutorialsteacher.com/python
- https://realpython.com/

Activity 24

Aim: Write a python class named as circle by a radius and two methods of computer area and perimeter of a circle.

Learning outcome: Able to make websites, web servers, game frameworks, desktop and CLI applications, and IDE using Python.

Duration: 2 hour

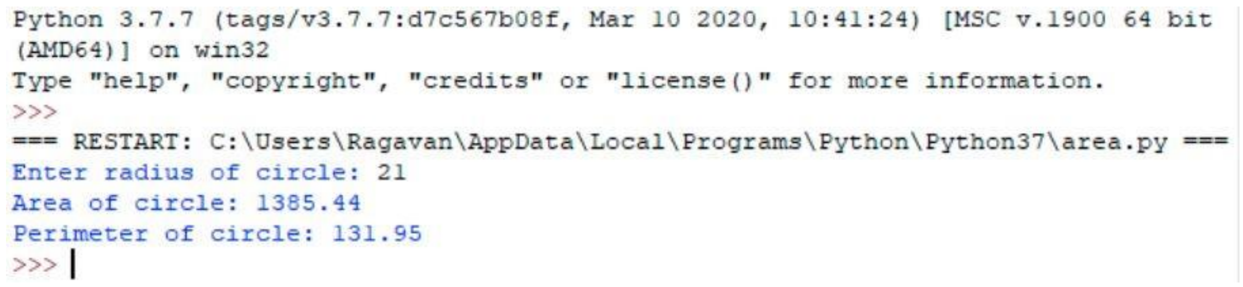
List of Hardware/Software requirements:

1. Laptop/Computer with Linux OS - Ubuntu 18.04 LTS
2. Python 3 software

Program:

```
#using math package
import math
#using math function calculate the value
class circle():
def___init_(self,radius):
self.radius=radius
def area(self):
return math.pi*(self.radius**2)
def perimeter(self):
return 2*math.pi*self.radius
#read value of area input from user
value_of_circle=int(input("Enter radius of circle: "))
#Object for Class
obj=circle(value_of_circle)
print("Area of circle:",round(obj.area(),2))
print("Perimeter of circle:",round(obj.perimeter(),2))
```

Output/Result Snippet



```
Python 3.7.7 (tags/v3.7.7:d7c567b08f, Mar 10 2020, 10:41:24) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=== RESTART: C:\Users\Ragavan\AppData\Local\Programs\Python\Python37\area.py ===
Enter radius of circle: 21
Area of circle: 1385.44
Perimeter of circle: 131.95
>>> |
```

Activity 25

Aim: Write a python program to sort a list of elements using bubble sort algorithm.

Learning outcome: Able to make websites, web servers, game frameworks, desktop and CLI applications, and IDE using Python.

Duration: 2 hour

List of Hardware/Software requirements:

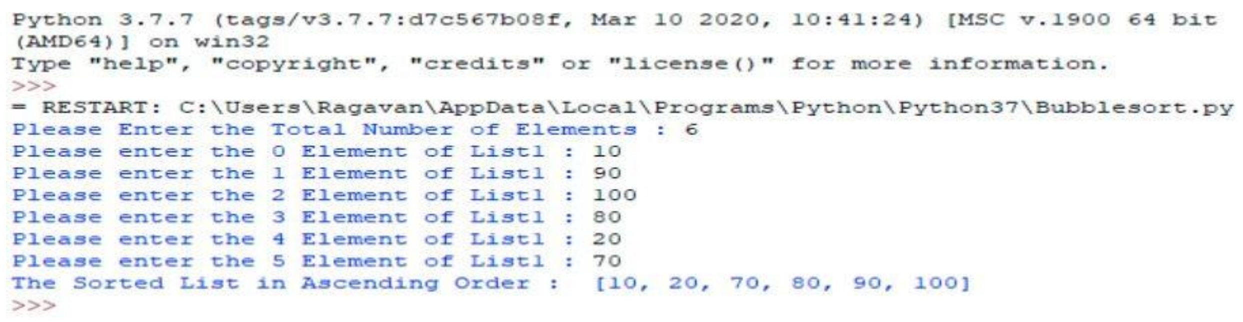
1. Laptop/Computer with Linux OS - Ubuntu 18.04 LTS
2. Python 3 software

Program:

```
# Python Program for Bubble Sort
def bubblesort(a, number):
for i in range(number -1):
for j in range(number - i - 1):
if(a[j] > a[j + 1]):
temp =  a[j]
a[j] = a[j + 1]
a[j + 1] = temp
a = []
number = int(input("Please Enter the Total Number of Elements : "))
for i in range(number):
value = int(input("Please enter the %d Element of List1 : " %i))
a.append(value)
bubblesort(a, number)
print("The Sorted List in Ascending Order : ", a)
```

Note: Here, we are using Nested for Loop to iterate each element in a given List. Inside the loop, we are using the If statement to sort items in an ascending order using Bubble Sort

Output Snippet



```
Python 3.7.7 (tags/v3.7.7:d7c567b08f, Mar 10 2020, 10:41:24) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\Ragavan\AppData\Local\Programs\Python\Python37\Bubblesort.py
Please Enter the Total Number of Elements : 6
Please enter the 0 Element of List1 : 10
Please enter the 1 Element of List1 : 90
Please enter the 2 Element of List1 : 100
Please enter the 3 Element of List1 : 80
Please enter the 4 Element of List1 : 20
Please enter the 5 Element of List1 : 70
The Sorted List in Ascending Order :  [10, 20, 70, 80, 90, 100]
>>>
```

Activity 26

Aim: Write a python program to copy content of a file to another file (3Hrs)

Learning outcome: Able to make websites, web servers, game frameworks, desktop and CLI applications, and IDE using Python.

Duration: 2 hour

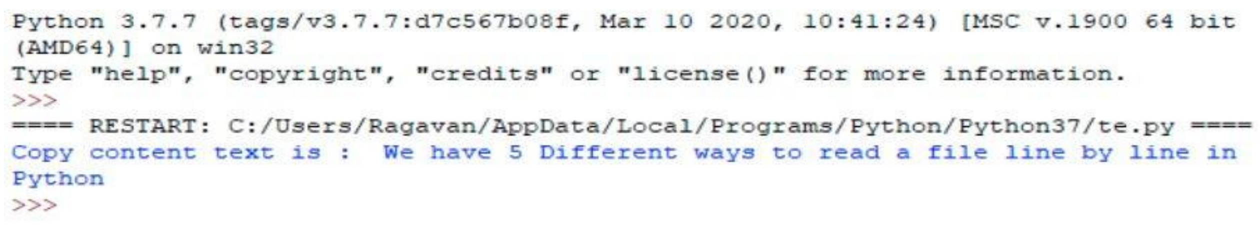
List of Hardware/Software requirements:

1. Laptop/Computer with Linux OS - Ubuntu 18.04 LTS
2. Python 3 software0

Program:

```
#Input files for Read data
#Output file to copy content from input file
with open("input.txt" ,"r") as f:
with open("output.txt", "w") as f1:
#writes content from existing to new file
for line in f:
f1.write(line)
#display content of copying text
print("Copy content text is : ",line)
#close files
f.close()
f1.close()
```

Output Snippet



```
Python 3.7.7 (tags/v3.7.7:d7c567b08f, Mar 10 2020, 10:41:24) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==== RESTART: C:/Users/Ragavan/AppData/Local/Programs/Python/Python37/te.py ====
Copy content text is :  We have 5 Different ways to read a file line by line in
Python
>>>
```

References

- https://docs.python.org/3/tutorial/
- https://www.tutorialsteacher.com/python
- https://realpython.com/

Activity 27

Aim: Write a python program to find the frequency of words in a file.

Learning outcome: Able to make websites, web servers, game frameworks, desktop and CLI applications, and IDE using Python.

Duration: 2 hour

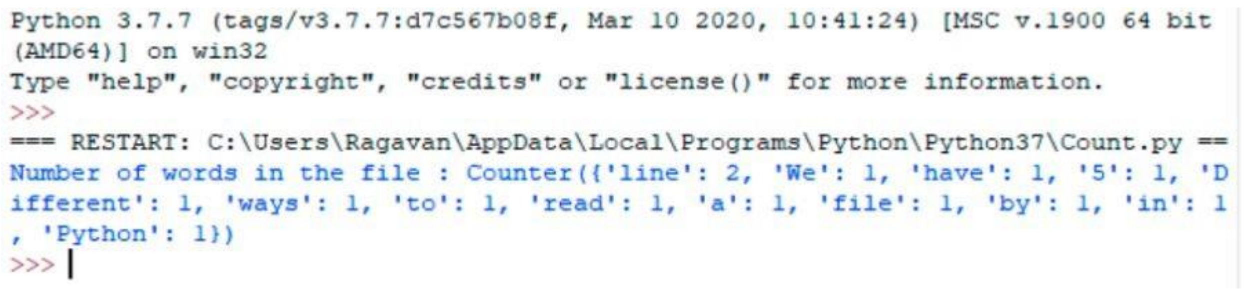
List of Hardware/Software requirements:

1. Laptop/Computer with Linux OS - Ubuntu 18.04 LTS
2. Python 3 software

Program:

```
#import counter for read data
from collections import Counter
def word_count(fname):
#Open a file and read text
with open(fname) as f:
#Separating a text word by word using split()
return Counter(f.read().split())
print("Number of words in the file :",word_count("output.txt"))
```

Output Snippet



```
Python 3.7.7 (tags/v3.7.7:d7c567b08f, Mar 10 2020, 10:41:24) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=== RESTART: C:\Users\Ragavan\AppData\Local\Programs\Python\Python37\Count.py ==
Number of words in the file : Counter({'line': 2, 'We': 1, 'have': 1, '5': 1, 'D
ifferent': 1, 'ways': 1, 'to': 1, 'read': 1, 'a': 1, 'file': 1, 'by': 1, 'in': 1
, 'Python': 1})
>>> |
```

References

- https://docs.python.org/3/tutorial/
- https://www.tutorialsteacher.com/python
- https://realpython.com/

Activity 28

Aim: Write a python program to illustrate exception handling

Learning outcome: Able to make websites, web servers, game frameworks, desktop and CLI applications, and IDE using Python.

Duration: 4 hours

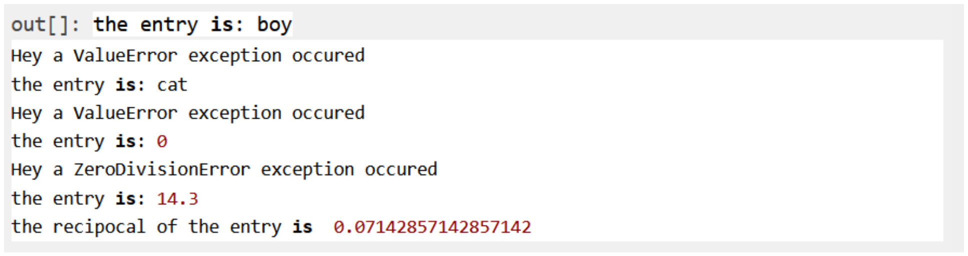
List of Hardware/Software requirements:

3. Laptop/Computer with Linux OS - Ubuntu 18.04 LTS
4. Python 3 software

Program:

```
import sys
list=['boy', 'cat',0 ,14.3]
for entry in list:

    try:
        print("the entry is:" , entry)
        r=1/int(entry)
    except(ValueError):
        print("Hey a ValueError exception occured")
    except(ZeroDivisionError):
        print("Hey a ZeroDivisionError exception occured")
    except:
        print("some error occur")
print("the recipocal of the entry is ",r)
```

Output Snippet



```
out[]: the entry is: boy
Hey a ValueError exception occured
the entry is: cat
Hey a ValueError exception occured
the entry is: 0
Hey a ZeroDivisionError exception occured
the entry is: 14.3
the recipocal of the entry is  0.07142857142857142
```

References

- https://blog.ineuron.ai/File-Handling-and-Exception-Handling-in-PYTHON-Yx6AqWHtOF

Activity 29

Aim: Develop a python code to read a text file, copy the contents to another file after removing the blank lines.

Learning outcome: Able to make websites, web servers, game frameworks, desktop and CLI applications, and IDE using Python.

Duration: 2 hours

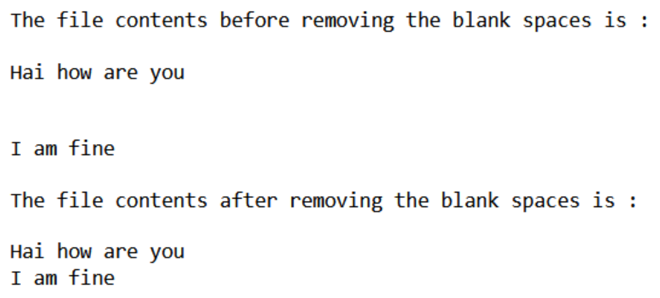
List of Hardware/Software requirements:

- Laptop/Computer with Linux OS - Ubuntu 18.04 LTS
- Python 3 software

Program:

```
fh = open('Blank.txt','r')
h = open('BlankRemove.txt','w')
b = fh.readlines()
print('The file contents before removing the blank spaces is :\n')
for line in b :
    print(line,end = '')
for line in b :
    if(len(line)>0) :
        line = line.lstrip()
        h.write(line)
fh.close()
h.close()
h = open('BlankRemove.txt','r')
y = h.read()
print('\n\nThe file contents after removing the blank spaces is :\n')
print(y)
h.close()
```

Output Snippet



```
The file contents before removing the blank spaces is :

Hai how are you


I am fine

The file contents after removing the blank spaces is :

Hai how are you
I am fine
```

References
https://www.geeksforgeeks.org/python-file-handling/

Activity 30

Aim: Create a database in mysql and connect using python

Learning outcome: Able to make websites, web servers, game frameworks, desktop and CLI applications, and IDE using Python.

Duration: 3 hours

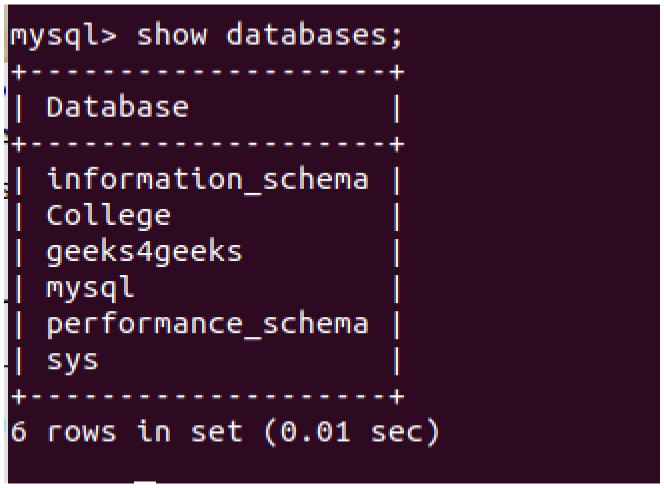
List of Hardware/Software requirements:

5. Laptop/Computer with Linux OS - Ubuntu 18.04 LTS
6. Python 3 software

Program:

```
# importing required libraries
import mysql.connector

dataBase = mysql.connector.connect(
  host ="localhost",
  user ="user",
  passwd ="gfg"
)

# preparing a cursor object
cursorObject = dataBase.cursor()

# creating database
cursorObject.execute("CREATE DATABASE geeks4geeks")
```

Output Snippet



```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| College            |
| geeks4geeks        |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
6 rows in set (0.01 sec)
```

References
https://www.geeksforgeeks.org/python-mysql-create-database/

Activity 31

Aim: Perform CRUD operations on mysql database using python

Learning outcome: Able to make websites, web servers, game frameworks, desktop and CLI applications, and IDE using Python.

Duration: 4 hours

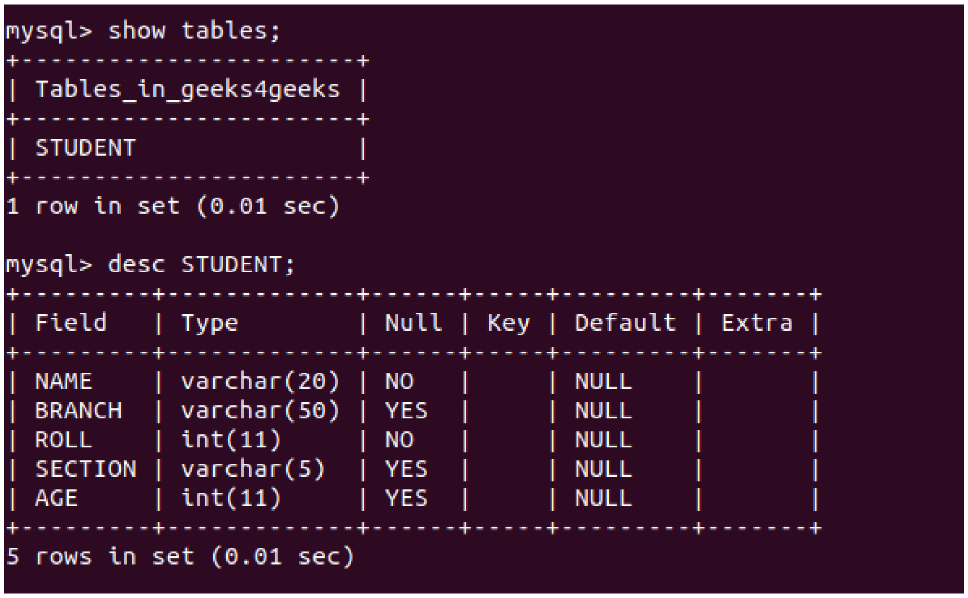
List of Hardware/Software requirements:

7. Laptop/Computer with Linux OS - Ubuntu 18.04 LTS
8. Python 3 software

Program:

```
# importing required library
import mysql.connector

# connecting to the database
dataBase = mysql.connector.connect(
host = "localhost",
user = "user",
passwd = "gfg",
database = "geeks4geeks" )

# preparing a cursor object
cursorObject = dataBase.cursor()

# creating table
studentRecord = """CREATE TABLE STUDENT (
NAME VARCHAR(20) NOT NULL,
BRANCH VARCHAR(50),
ROLL INT NOT NULL,
SECTION VARCHAR(5),
AGE INT
)"""

# table created
cursorObject.execute(studentRecord)

# disconnecting from server
dataBase.close()
```

Output Snippet



```
mysql> show tables;
+-----------------------+
| Tables_in_geeks4geeks |
+-----------------------+
| STUDENT               |
+-----------------------+
1 row in set (0.01 sec)

mysql> desc STUDENT;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| NAME    | varchar(20) | NO   |     | NULL    |       |
| BRANCH  | varchar(50) | YES  |     | NULL    |       |
| ROLL    | int(11)     | NO   |     | NULL    |       |
| SECTION | varchar(5)  | YES  |     | NULL    |       |
| AGE     | int(11)     | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
5 rows in set (0.01 sec)
```

References
https://www.geeksforgeeks.org/python-mysql-create-database/

Activity 32

Aim: Perform CRUD operations on MongoDB database using python

Learning outcome: Able to make websites, web servers, game frameworks, desktop and CLI applications, and IDE using Python.

Duration: 3 hours

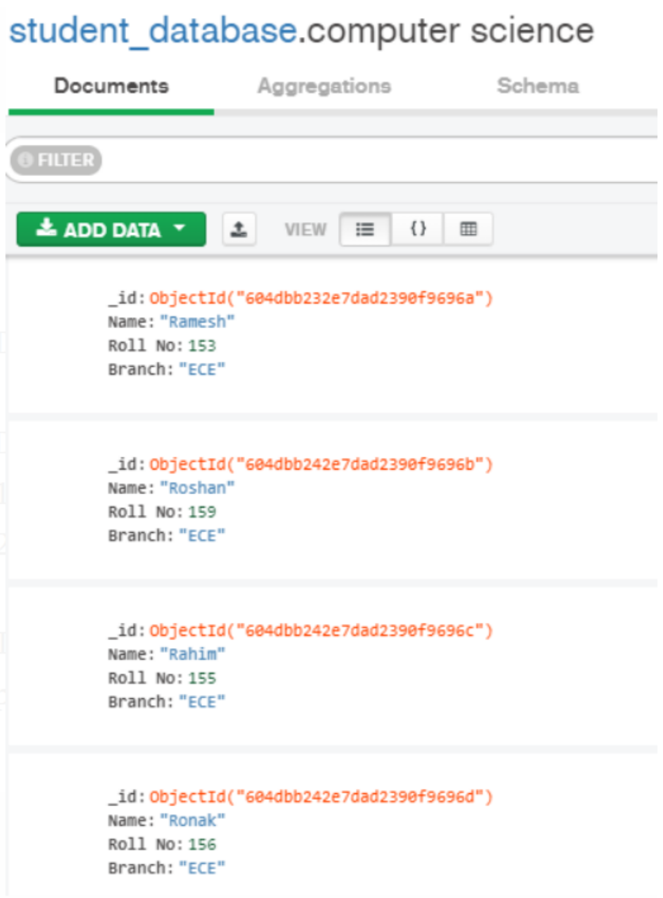
List of Hardware/Software requirements:

9. Laptop/Computer with Linux OS - Ubuntu 18.04 LTS
10. Python 3 software

Program:
```
# importing required library
import pymongo
connection_url="mongodb://localhost:27017/"
client.list_database_names()

# connecting to the database
database_name="student_database"
student_db=client[database_name]

# creating collection
collection_name="computer science"
collection=student_db[collection_name]
student_db.list_collection_names()
#Inserting document
```

```
document={"Name":"Raj",
"Roll No ":  153,
"Branch ": "CSE"}
collection.insert_one(document)
#Reading a document
query={"Name":"Raj"}
print(collection.find_one(query))
#updating
query={"Roll No":{"$eq":153}}
present_data=collection.find_one(query)
new_data={'$set':{"Name":'Ramesh'}}
collection.update_one(present_data,new_data)
# deleting
query={"Roll No":153}
collection.delete_one(query)
```

Output Snippet



```
_id: ObjectId("604dbb232e7dad2390f9696a")
Name: "Ramesh"
Roll No: 153
Branch: "ECE"

_id: ObjectId("604dbb242e7dad2390f9696b")
Name: "Roshan"
Roll No: 159
Branch: "ECE"

_id: ObjectId("604dbb242e7dad2390f9696c")
Name: "Rahim"
Roll No: 155
Branch: "ECE"

_id: ObjectId("604dbb242e7dad2390f9696d")
Name: "Ronak"
Roll No: 156
Branch: "ECE"
```

References

https://medium.com/analytics-vidhya/crud-operations-in-mongodb-using-python-49b7850d627e

Activity 33

Aim: Create a class with related data elements and methods. Write functionalities for basic operations

Learning outcome: Able to make websites, web servers, game frameworks, desktop and CLI applications, and IDE using Python.

Duration: 3 hours

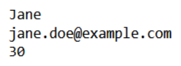
List of Hardware/Software requirements:

11. Laptop/Computer with Linux OS - Ubuntu 18.04 LTS
12. Python 3 software

Program:

```
import datetime # we will use this for date objects

class Person:

    def __init__(self, name, surname, birthdate, address, telephone, email):
        self.name = name
        self.surname = surname
        self.birthdate = birthdate

        self.address = address
        self.telephone = telephone
        self.email = email

    def age(self):
        today = datetime.date.today()
        age = today.year - self.birthdate.year

        if today < datetime.date(today.year, self.birthdate.month, self.birthdate.day):
            age -= 1

        return age

person = Person(
    "Jane",
    "Doe",
    datetime.date(1992, 3, 12), # year, month, day
    "No. 12 Short Street, Greenville",
    "555 456 0987",
    "jane.doe@example.com"
)

print(person.name)
print(person.email)
print(person.age())
```

Output Snippet



```
Jane
jane.doe@example.com
30
```

References
https://python-textbok.readthedocs.io/en/1.0/Classes.html

Activity 34

Aim: Python program using a flag to validate if the input given by the user is an integer.#Datatype check.

Learning outcome: Able to make websites, web servers, game frameworks, desktop and CLI applications, and IDE using Python.

Duration: 4 hours

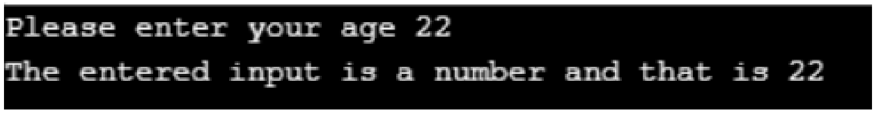
List of Hardware/Software requirements:

13. Laptop/Computer with Linux OS - Ubuntu 18.04 LTS
14. Python 3 software

Program:

```
#Declare a variable validInt which is also considered as flag and set it to false
validInt = False
#Consider the while condition to be true and prompt the user to enter the input
while not validInt:
            #The user is prompted to enter the input
age1 = input('Please enter your age ')
             #The input entered by the user is checked to see if it’s a digit or a number
if age1.isdigit():
                         #The flag is set to true if the if condition is true
validInt = True
             else:
                          print('The input is not a valid number')
 #This statement is printed if the input entered by the user is a number
 print('The entered input is a number and that is ' + str(age1))
```

Output Snippet



```
Please enter your age 22
The entered input is a number and that is 22
```

References
https://www.educba.com/python-validation/

Activity 35

Aim: Create a simple python Flask app with at least 3 basic routes

Learning outcome: Able to make websites, web servers, game frameworks, desktop and CLI applications, and IDE using Python.

Duration: 4 hours

List of Hardware/Software requirements:

15. Laptop/Computer with Linux OS - Ubuntu 18.04 LTS
16. Python 3 software

Program:

```
# Installing virtual environment and flask
  pip install virtualenv

  pip install Flask

# Importing flask module in the project is mandatory

# An object of Flask class is our WSGI application.

from flask import Flask

 # Flask constructor takes the name of

# current module (__name__) as argument.

app = Flask(__name__)

 # The route() function of the Flask class is a decorator,

# which tells the application which URL should call

# the associated function.

@app.route('/')

# '/' URL is bound with hello_world() function.

def hello_world():

return 'Hello World'


# main driver function

if __name__ == '__main__':
```

# run() method of Flask class runs the application

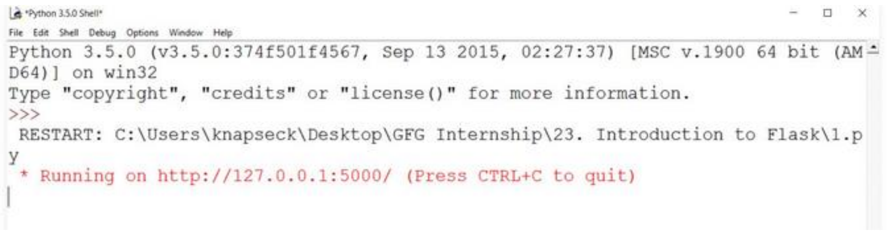
# on the local development server.

app.run()

#Save it in a file and then run the script we will be getting an output

like this.

Output Snippet



```
Python 3.5.0 (v3.5.0:374f501f4567, Sep 13 2015, 02:27:37) [MSC v.1900 64 bit (AM
D64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:\Users\knapseck\Desktop\GFG Internship\23. Introduction to Flask\1.p
y
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

References

https://www.geeksforgeeks.org/flask-creating-first-simple-application/

Activity 1

Aim: Install Django framework libraries in python and test for installation

Learning outcome: Able to install Django.

Duration: 3 hour

List of Hardware/Software requirements:

1. VSCode
2. Python

Code/Program/Procedure (with comments):

Step 1) Creating environment for Django project

A) Install latest version of python

1. Download and install latest version of python from the url https://www.python.org/downloads/

B) Check for installed version of python

1. Press Window + R to open command prompt
2. Type cmd in open box and press ok button
3. Command prompt will open
4. On command prompt type following command, it will display the current python version installed on your laptop

```
> python --version
```

C) Install pipenev

```
>   pip3 install pipenv
```

D) Install visual studio code editor

Visit URI and download https://code.visualstudio.com/

Step 2) Creating the first Project with django

A) Switch to Desktop

```
> cd Desktop
```

B) Create Project folder

```
> mkdir <<projectname_folder>>
> mkdir learndjango
```

Move to learndjango directory

```
>cd learndjango
```

C) Install django

```
>pipenv install django
```

Successfully created the virtual environment

- Run django-admin to start new project.
- django admin is utility comes along with django

```
 >django-admin
> django-admin startproject <<project_name>>
> django-admin startproject learndjango
```

E) Go Back to the terminal and execute the command

```
> django-admin startproject learndjango .
```

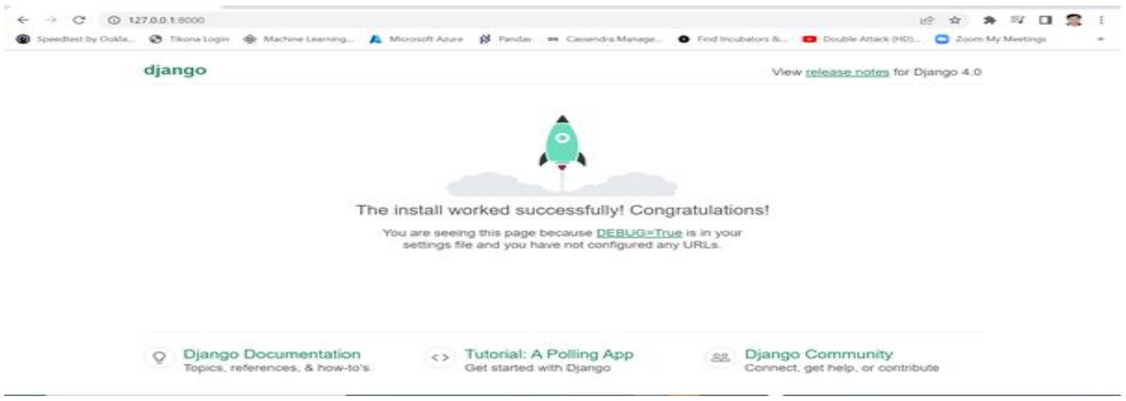
- Start webserver - Run the command

```
>python manage.py runserver
```

It will start server at http://127.0.0.1:8000/

- Now open browser and type address http://127.0.0.1:8000/ to open home page of our Django project learndjango

Output/Results snippet:



View release notes for Django 4.0

The install worked successfully! Congratulations!

You are seeing this page because DEBUG=True is in your settings file and you have not configured any URLs.

Activity 2

Aim: Setup first Django application

Learning outcome: Able to display Hello World in the web browser using Django.

Duration: 3 hour

List of Hardware/Software requirements:

- VSCode
- Python

Code/Program/Procedure (with comments):

1. In the VS Code Terminal with your virtual environment activated, run the administrative utility's `startapp` command in your project folder (where `manage.py` resides):

```
python manage.py startapp hello
```

The command creates a folder called `hello` that contains a number of code files and one subfolder. Of these, you frequently work with `views.py` (that contains the functions that define pages in your web app) and models.py (that contains classes defining your data objects). The migrations folder is used by Django's administrative utility to manage database versions as discussed later in this tutorial. There are also the files apps.py (app configuration), admin.py (for creating an administrative interface), and tests.py (for creating tests), which are not covered here.

2. Modify hello/views.py to match the following code, which creates a single view for the app's home page:

```
from django.http import HttpResponse


def home(request):

    return HttpResponse("Hello, Django!")
```

3. Create a file, hello/urls.py, with the contents below. The urls.py file is where you specify patterns to route different URLs to their appropriate views. The code below contains one route to map root URL of the app ("") to the views.home function that you just added to hello/views.py:

```
from django.urls import path

from hello import views


urlpatterns = [

    path("", views.home, name="home"),

]
```

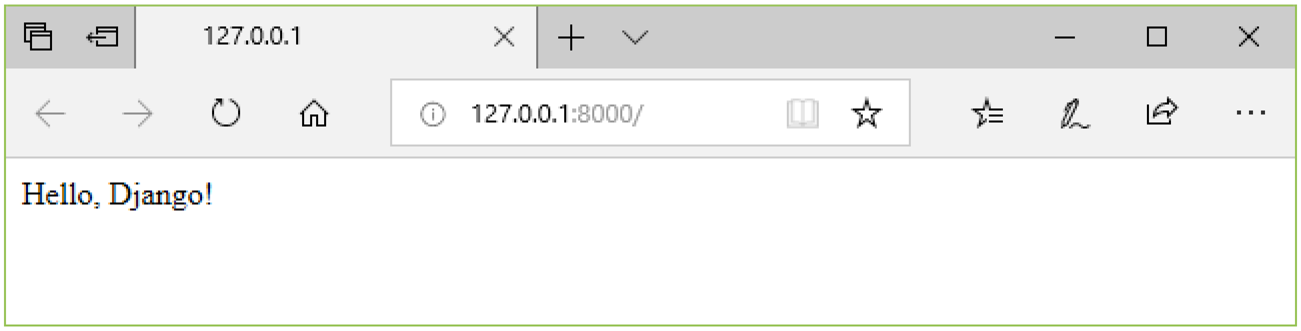
4. The web_project folder also contains a urls.py file, which is where URL routing is actually handled. Open web_project/urls.py and modify it to match the following code (you can retain the instructive comments if you like). This code pulls in the app's hello/urls.py using django.urls.include, which keeps the app's routes contained within the app. This separation is helpful when a project contains multiple apps.

```
from django.contrib import admin

from django.urls import include, path


urlpatterns = [

    path("", include("hello.urls")),
    path('admin/', admin.site.urls)
```

```
]
```

5. Save all modified files.
6. In the VS Code Terminal, again with the virtual environment activated, run the development server with `python manage.py runserver` and open a browser to `http://127.0.0.1:8000/` to see a page that renders "Hello, Django".

Output/Results snippet:



Hello, Django!

Activity 3

Aim: Creating Django application with MVT architecture
Learning outcome: Able to understand creating Django using MVT.
Duration: 5 hour
List of Hardware/Software requirements:

1. Django
2. Python
3. VS Code

Code/Program/Procedure (with comments):

Step: Create Virtual Environment

It is suggested to have a dedicated virtual environment for each Django project, and one way to manage a virtual environment is venv, which is included in Python.
With venv, you can create a virtual environment by typing this in the command prompt, remember to navigate to where you want to create your project:
Windows:

```
py -m venv myproject
```

Unix/MacOS:

```
python -m venv myproject
```

This will set up a virtual environment, and create a folder named "myproject" with subfolders and files, like this:

```
myproject
  Include
  Lib
  Scripts
  pyvenv.cfg
```

Then you have to activate the environment, by typing this command:

Windows:

```
myproject\Scripts\activate.bat
```

Unix/MacOS:

```
source myproject/bin/activate
```

Once the environment is activated, you will see this result in the command prompt:

Windows:

```
(myproject) C:\Users\Your Name>
```

Unix/MacOS:

```
(myproject) ... $
```

Note: You must activate the virtual environment every time you open the command prompt to work on your project.

Step - Install Django

Finally, we can install Django.

Remember to install Django while you are in the virtual environment!

Django is installed using pip, with this command:

Windows:

```
(myproject) C:\Users\Your Name>py -m pip install Django
```

Unix/MacOS:

```
(myproject) ... $ python -m pip install Django
```

Which will give a result that looks like this (at least on my Windows machine):

```
Collecting Django
  Downloading Django-4.0.3-py3-none-any.whl (8.0 MB)
      |                                                          | 8.0 MB 2.2 MB/s
Collecting sqlparse>=0.2.2
  Using cached sqlparse-0.4.2-py3-none-any.whl (42 kB)
Collecting asgiref<4,>=3.4.1
  Downloading asgiref-3.5.0-py3-none-any.whl (22 kB)
Collecting tzdata; sys_platform == "win32"
  Downloading tzdata-2021.5-py2.py3-none-any.whl (339 kB)
      |                                                          | 339 kB 6.4 MB/s
Installing collected packages: sqlparse, asgiref, tzdata, Django
Successfully installed Django-4.0.3 asgiref-3.5.0 sqlparse-0.4.2 tzdata-2021.5
WARNING: You are using pip version 20.2.3; however, version 22.0.4 is available.
You should consider upgrading via the 'C:\Users\Your Name\myproject\Scripts\python.exe -m pip install --upgrade pip'
command.
```

That's it" Now you have installed Django in your new project, running in a virtual environment!

Windows, Mac, or Unix?

You can run this project on either one. There are some small differences, like when writing commands in the command prompt, Windows uses py as the first word in the command line, while Unix and MacOS use python:

Windows:

```
py --version
```

Unix/MacOS:

```
python --version
```

In the rest of this tutorial, we will be using the Windows command.

Step - Check Django Version

You can check if Django is installed by asking for its version number like this:

```
(myproject) C:\Users\Your Name>django-admin --version
```

If Django is installed, you will get a result with the version number:

```
4.0.3
```

Step - Django Create Project

My First Project

Once you have come up with a suitable name for your Django project, like mine: myworld, navigate to where in the file system you want to store the code (in the virtual environment), and run this command in the command prompt:

```
django-admin startproject myworld
```

Django creates a myworld folder on my computer, with this content:

```
myworld
    manage.py
    myworld/
        __init__.py
        asgi.py
        settings.py
        urls.py
        wsgi.py
```

These are all files and folders with a specific meaning, you will learn about some of them later in this tutorial, but for now, it is more important to know that this is the location of your project, and that you can start building applications in it.

Run the Django Project

Now that you have a Django project, you can run it, and see what it looks like in a browser.

Navigate to the /myworld folder and execute this command in the command prompt:

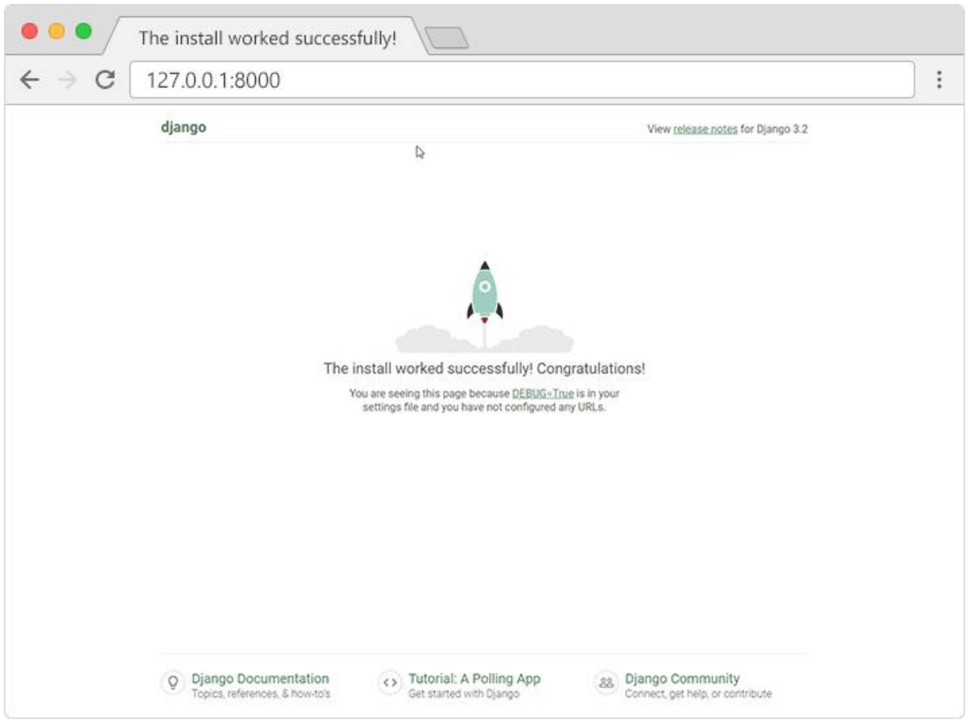
```
py manage.py runserver
```

Which will produce this result:

```
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s):
admin, auth, contenttypes, sessions.
```

```
Run 'python manage.py migrate' to apply them.
December 02, 2021 - 13:14:51
Django version 3.2.9, using settings 'myworld.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Open a new browser window and type 127.0.0.1:8000 in the address bar.
The result:


The install worked successfully!
127.0.0.1:8000
django
View release notes for Django 3.2
Django Documentation
Topics, references, & how-to's
Tutorial: A Polling App
Get started with Django
Django Community
Connect, get help, or contribute

The install worked successfully! Congratulations!

You are seeing this page because DEBUG=True is in your settings file and you have not configured any URLs.

Step -Django Create App

Create App

I will name my app members.
Start by navigating to the selected location where you want to store the app, and run the command below.
If the server is still running, and you are not able to write commands, press [CTRL] [BREAK] to stop the server and you should be back in the virtual environment.

```
py manage.py startapp members
```

Django creates a folder named members in my project, with this content:

```
myworld
    manage.py
    myworld/
    members/
        migrations/
            __init__.py
        __init__.py
        admin.py
        apps.py
        models.py
        tests.py
        views.py
```

These are all files and folders with a specific meaning. You will learn about most of them later in this tutorial.
First, take a look at the file called views.py.

Step - Django Views

Views

Django views are Python functions that takes http requests and returns http response, like HTML documents.
A web page that uses Django is full of views with different tasks and missions.
Views are usually put in a file called views.py located on your app's folder.
There is a views.py in your members folder that looks like this:
members/views.py:

```
from django.shortcuts import render

# Create your views here.
```

Find it and open it, and replace the content with this:
members/views.py:

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

def index(request):
    return HttpResponse("Hello world!")
```

This is a simple example on how to send a response back to the browser.
But how can we execute the view? Well, we must call the view via a URL.

URLs

Create a file named urls.py in the same folder as the views.py file, and type this code in it:
members/urls.py:

```
from django.urls import path
from . import views

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

The urls.py file you just created is specific for the members application. We have to do some routing in the root directory myworld as well. This may seem complicated, but for now, just follow the instructions below.
There is a file called urls.py on the myworld folder, open that file and add the include module in the import statement, and also add a path() function in the urlpatterns[] list, with arguments that will route users that comes in via 127.0.0.1:8000/members/.
Then your file will look like this:
myworld/urls.py:

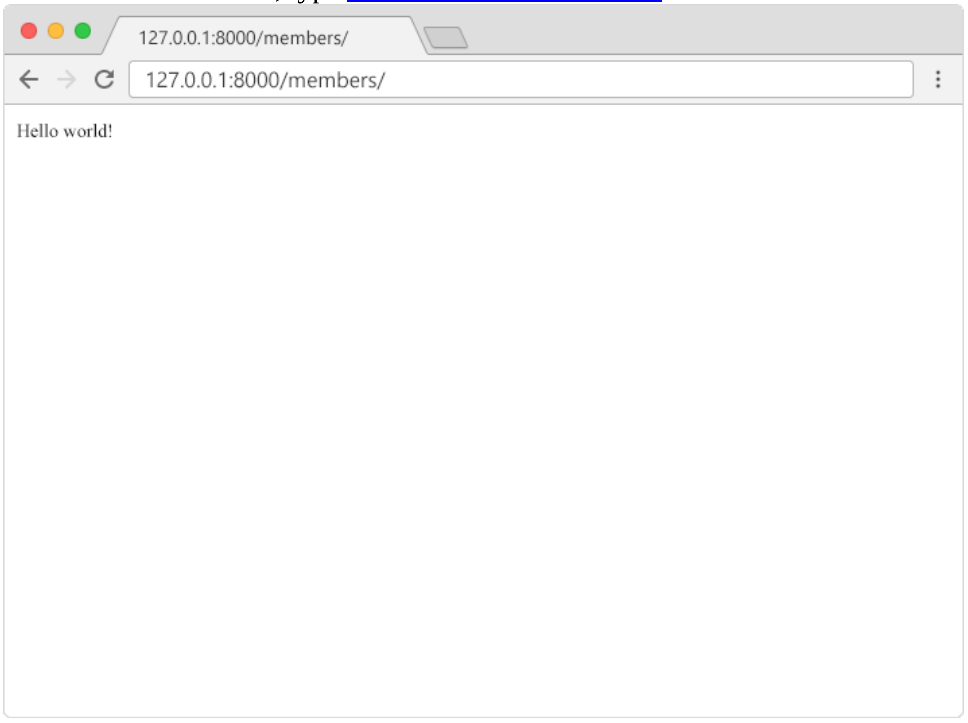
```
from django.contrib import admin
from django.urls import include, path
```

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

If the server is not running, navigate to the /myworld folder and execute this command in the command prompt:

```
py manage.py runserver
```

In the browser window, type 127.0.0.1:8000/members/ in the address bar.



Hello world!

Step-Django Templates

Templates

In the Django Intro page, we learned that the result should be in HTML, and it should be created in a template, so let's do that.

Create a templates folder inside the members folder, and create a HTML file named myfirst.html.

The file structure should be something like this:

```
myworld
    manage.py
    myworld/
    members/
        templates/
            myfirst.html
```

Open the HTML file and insert the following:

members/templates/myfirst.html:

```
<!DOCTYPE html>
<html>
<body>
```

```
<h1>Hello World!</h1>
<p>Welcome to my first Django project!</p>

</body>
</html>
```

Modify the View

Open the views.py file and replace the index view with this:

members/views.py:

```
from django.http import HttpResponse
from django.template import loader

def index(request):
  template = loader.get_template('myfirst.html')
  return HttpResponse(template.render())
```

Change Settings

To be able to work with more complicated stuff than "Hello World!", We have to tell Django that a new app is created.

This is done in the settings.py file in the myworld folder.

Look up the INSTALLED_APPS[] list and add the members app like this:

myworld/settings.py:

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

Then run this command:

```
py manage.py migrate
```

Which will produce this output:

```
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
```

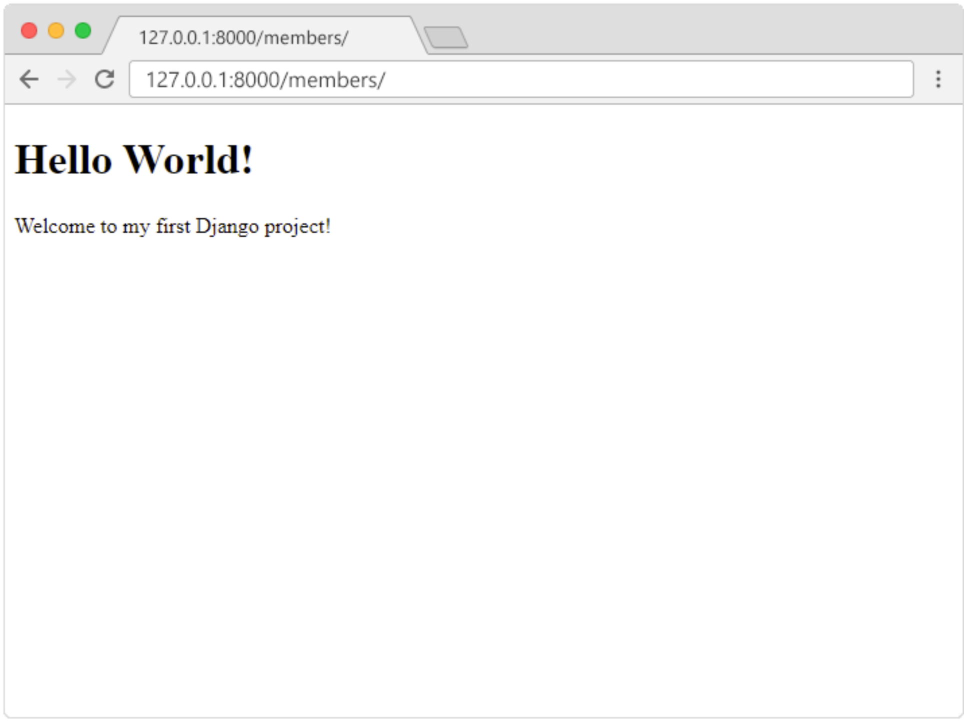
```
 Applying auth.0006_require_contenttypes_0002... OK
 Applying auth.0007_alter_validators_add_error_messages... OK
 Applying auth.0008_alter_user_username_max_length... OK
 Applying auth.0009_alter_user_last_name_max_length... OK
 Applying auth.0010_alter_group_name_max_length... OK
 Applying auth.0011_update_proxy_permissions... OK
 Applying auth.0012_alter_user_first_name_max_length... OK
 Applying sessions.0001_initial... OK
```

```
(myproject)C:\Users\Your Name\myproject\myworld>
```

Start the server by navigating to the /myworld folder and execute this command:

```
py manage.py runserver
```

In the browser window, type 127.0.0.1:8000/members/ in the address bar.
The result should look like this:



Hello World!

Welcome to my first Django project!

References - https://www.w3schools.com/django/django_create_project.php
https://www.w3schools.com/django/django_create_app.php
https://www.w3schools.com/django/django_views.php
https://www.w3schools.com/django/django_templates.php

Activity 4

Aim: Create Django application that handles file uploading

Learning outcome: Able to understand file handling and how to upload file .

Duration: 4 hour

List of Hardware/Software requirements:

1. Django
2. Python
3. VS Code

Code/Program/Procedure (with comments):

Django provides built-in library and methods that help to upload a file to the server.

The **forms.FileField()** method is used to create a file input and submit the file to the server. While working with files, make sure the HTML form tag contains **enctype="multipart/form-data"** property.

Template (index.html)

It will create an HTML form which contains a file input component.

```
<body>
<form method="POST" class="post-form" enctype="multipart/form-data">
        {% csrf_token %}
        {{ form.as_p }}
        <button type="submit" class="save btn btn-default">Save</button>
</form>
</body>
```

Form (forms.py)

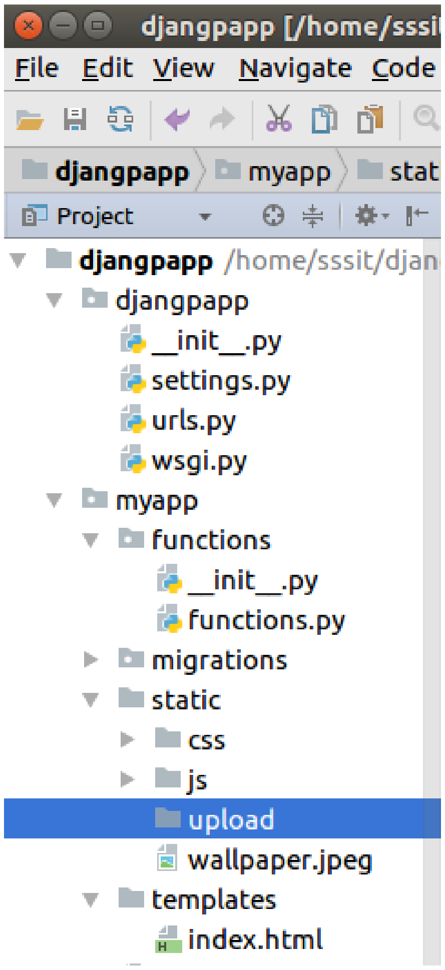
```
from django import forms
class StudentForm(forms.Form):
    firstname = forms.CharField(label="Enter first name",max_length=50)
    lastname  = forms.CharField(label="Enter last name", max_length = 10)
    email     = forms.EmailField(label="Enter Email")
    file      = forms.FileField() # for creating file input
```

View (views.py)

Here, one extra parameter request.FILES is required in the constructor. This argument contains the uploaded file instance.

```
from django.shortcuts import render
from django.http import HttpResponse
from myapp.functions.functions import handle_uploaded_file
from myapp.form import StudentForm
def index(request):
    if request.method == 'POST':
        student = StudentForm(request.POST, request.FILES)
        if student.is_valid():
            handle_uploaded_file(request.FILES['file'])
            return HttpResponse("File uploaded successfuly")
    else:
        student = StudentForm()
        return render(request,"index.html",{'form':student})
```
Specify URL (urls.py)
```
from django.contrib import admin
from django.urls import path
from myapp import views
urlpatterns = [
    path('admin/', admin.site.urls),
    path('index/', views.index),
]
```
Upload Script (functions.py)

This function is used to read the uploaded file and store at provided location. Put this code into the functions.py file. But first create this file into the project.
```
def handle_uploaded_file(f):
    with open('myapp/static/upload/'+f.name, 'wb+') as destination:
        for chunk in f.chunks():
            destination.write(chunk)
```
Now, create a directory upload to store the uploaded file. Our project structure looks like below.



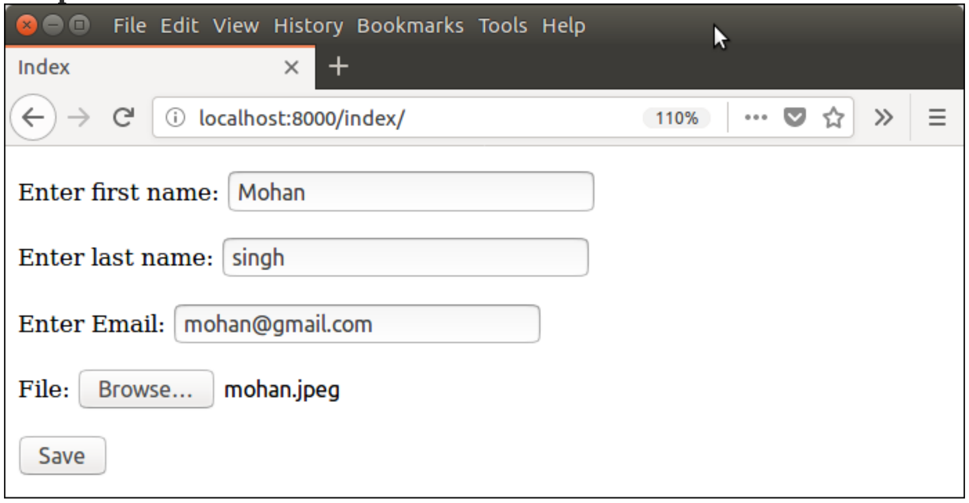
```
djangpapp /home/sssit/djan
  djangpapp
    __init__.py
    settings.py
    urls.py
    wsgi.py
  myapp
    functions
      __init__.py
      functions.py
    migrations
    static
      css
      js
      upload
      wallpaper.jpeg
    templates
      index.html
```

Initially, this directory is empty. so, let's upload a file to it and later on it will contain the uploaded file.

Start Server

```
python manage.py runserver
```

Output



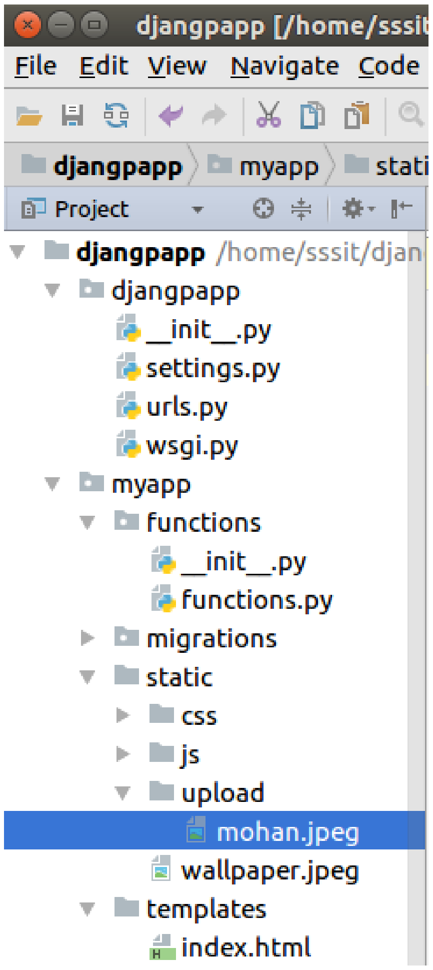
Enter first name: Mohan

Enter last name: singh

Enter Email: mohan@gmail.com

File: mohan.jpeg

Submit this form and see the **upload** folder. Now, it contains the uploaded file.



```
djangpapp /home/sssit/djan
    djangpapp
        __init__.py
        settings.py
        urls.py
        wsgi.py
    myapp
        functions
            __init__.py
            functions.py
        migrations
        static
            css
            js
            upload
                mohan.jpeg
            wallpaper.jpeg
        templates
            index.html
```

References - https://www.javatpoint.com/django-file-upload

Activity 5

Aim: Create Django application that reads data from CSV and display on page

Learning outcome: Able to understand basic computer network technology.
Duration: 5 hour
List of Hardware/Software requirements:

1. Django
2. Python
3. VSCode

Code/Program/Procedure (with comments):

How to create CSV output¶

This document explains how to output CSV (Comma Separated Values) dynamically using Django views. To do this, you can either use the Python CSV library or the Django template system.

Using the Python CSV library¶

Python comes with a CSV library, **csv**. The key to using it with Django is that the **csv** module's CSV-creation capability acts on file-like objects, and Django's **HttpResponse** objects are file-like objects.

Here's an example:

```
import csv
from django.http import HttpResponse

def some_view(request):
    # Create the HttpResponse object with the appropriate CSV header.
    response = HttpResponse(
        content_type='text/csv',
        headers={'Content-Disposition': 'attachment; filename="somefilename.csv"'},
    )

    writer = csv.writer(response)
    writer.writerow(['First row', 'Foo', 'Bar', 'Baz'])
    writer.writerow(['Second row', 'A', 'B', 'C', '"Testing"', "Here's a quote"])

    return response
```

The code and comments should be self-explanatory, but a few things deserve a mention:

- The response gets a special MIME type, *text/csv*. This tells browsers that the document is a CSV file, rather than an HTML file. If you leave this off, browsers will probably interpret the output as HTML, which will result in ugly, scary gobbledygook in the browser window.
- The response gets an additional **Content-Disposition** header, which contains the name of the CSV file. This filename is arbitrary; call it whatever you want. It'll be used by browsers in the "Save as…" dialog, etc.
- You can hook into the CSV-generation API by passing **response** as the first argument to **csv.writer**. The **csv.writer** function expects a file-like object, and **HttpResponse** objects fit the bill.
- For each row in your CSV file, call **writer.writerow**, passing it an iterable.
- The CSV module takes care of quoting for you, so you don't have to worry about escaping strings with quotes or commas in them. Pass **writerow()** your raw strings, and it'll do the right thing.

Streaming large CSV files¶

When dealing with views that generate very large responses, you might want to consider using Django's **StreamingHttpResponse** instead. For example, by streaming a file that takes a long time to generate you can avoid a load balancer dropping a connection that might have otherwise timed out while the server was generating the response.

In this example, we make full use of Python generators to efficiently handle the assembly and transmission of a large CSV file:

```
import csv

from django.http import StreamingHttpResponse

class Echo:
    """An object that implements just the write method of the file-like
    interface.
    """
    def write(self, value):
        """Write the value by returning it, instead of storing in a buffer."""
        return value
```

```
def some_streaming_csv_view(request):
    """A view that streams a large CSV file."""
    # Generate a sequence of rows. The range is based on the maximum number of
    # rows that can be handled by a single sheet in most spreadsheet
    # applications.
    rows = (["Row {}".format(idx), str(idx)] for idx in range(65536))
    pseudo_buffer = Echo()
    writer = csv.writer(pseudo_buffer)
    return StreamingHttpResponse(
        (writer.writerow(row) for row in rows),
        content_type="text/csv",
        headers={'Content-Disposition': 'attachment; filename="somefilename.csv"'},
    )
```

Using the template system¶

Alternatively, you can use the Django template system to generate CSV. This is lower-level than using the convenient Python **csv** module, but the solution is presented here for completeness.

The idea here is to pass a list of items to your template, and have the template output the commas in a **for** loop.

Here's an example, which generates the same CSV file as above:

```
from django.http import HttpResponse
from django.template import loader

def some_view(request):
    # Create the HttpResponse object with the appropriate CSV header.
    response = HttpResponse(
        content_type='text/csv',
        headers={'Content-Disposition': 'attachment; filename="somefilename.csv"'},
    )

    # The data is hard-coded here, but you could load it from a database or
    # some other source.
    csv_data = (
        ('First row', 'Foo', 'Bar', 'Baz'),
        ('Second row', 'A', 'B', 'C', '"Testing"', "Here's a quote"),
    )

    t = loader.get_template('my_template_name.txt')
    c = {'data': csv_data}
    response.write(t.render(c))
    return response
```

The only difference between this example and the previous example is that this one uses template loading instead of the CSV module. The rest of the code – such as the **content_type='text/csv'** – is the same.

Then, create the template **my_template_name.txt**, with this template code:

```
{% for row in data %}"{{ row.0|addslashes }}", "{{ row.1|addslashes }}", "{{ row.2|addslashes }}", "{{ row.3|addslashes }}", "{{ row.4|addslashes }}"
{% endfor %}
```

This short template iterates over the given data and displays a line of CSV for each row. It uses the **addslashes** template filter to ensure there aren't any problems with quotes.

Another Code for the same activity -

Quick Django Application Setup

Type the bellow commands in terminal

```
mkdir django_csv
cd django_csv
virtualenv env
pip3 install django
django-admin startproject django_csv .
django-admin startapp csv_app
```

Open your settings.py file and add app name, which we have created with the name csv_app

```
INSTALLED_APPS = [
    ...,

    'csv_app', # <- this
]
```

Create a View to display all html code

```
from django.views.generic.base import TemplateView

class CSVPageView(TemplateView):
    template_name = "csv_home.html"
```

csv_app/templates/csv_home.html

```
<!DOCTYPE html>
<html>
    <head><title>CSV Examples</title></head>
    <body>
        <h3>CSV Example - Read Write Examples</h3>
        <ul>
            <li>Write Operation
                <ul>
                    <li>
                        <a href="{% url 'csv_simple_write' %}">Simple CSV Write Operation</a>
                    </li>
                    <li>
                        <a href="{% url 'csv_dictionary_write' %}">Writing CSV File From a Dictionary</a>
                    </li>
                    <li>
                        <a href="{% url 'csv_database_write' %}">Database Data CSV Write Operation</a>
                    </li>
                </ul>
            </li>
            <br>
            <li>Read Operation
                <ul>
                    <li>
                        <a href="{% url 'csv_simple_read' %}">Simple CSV Read Operation</a>
                    </li>
                </ul>
            </li>
        </ul>
        {{csv_data}}
        {% if csv_data %}
        sad
            {{csv_data}}
        {% endif %}
    </body>
</html>
```

Note: if you run the above file it will give an error because we have not created URLs. We are going to create those URLs below in Read/Write operation code.

Create a file named urls.py in your csv_app folder and the code. Note the URLs of these apps can be created here.

```
from django.urls import path
from csv_app import views

urlpatterns = [
  path('', views.CSVPageView.as_view(), name='csv_home_page'),
]
```

The last thing, we need to import csv_app/urls.py in the main folder django_csv/urls.py. Edit django_csv/urls.py and django_csv/urls.py should look like below:

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

from csv_app import urls as csv_app_urls

urlpatterns = [
    path('admin/', admin.site.urls),

    path('', include(csv_app_urls))
]
```

Python CSV Library Code Explanation

- HttpResponse(content_type='text/csv') – This tells browsers that the document is a CSV file, instead of HTML file.
- response['Content-Disposition'] = 'attachment; filename="csv_simple_write.csv"' – This contains CSV filename and downloads files with that name.
- Hooking into the CSV-generation API is easy: simply pass response because of the initial argument to csv.writer. The csv.writer perform expects a file-like object, and HttpResponse objects match the bill.
- writer.writerow(['first_name', 'last_name', 'phone_number', 'country']) – Will add/write a row in your csv file.

Simple CSV Export/Write Operation

```
# csv_app/views.py

import csv
from django.http import HttpResponse

# Simple CSV Write Operation
def csv_simple_write(request):
    # Create the HttpResponse object with the appropriate CSV header.
    response = HttpResponse(content_type='text/csv')
    response['Content-Disposition'] = 'attachment; filename="csv_simple_write.csv"'

    writer = csv.writer(response)
    writer.writerow(['first_name', 'last_name', 'phone_number', 'country'])
    writer.writerow(['Huzaif', 'Sayyed', '+919954465169', 'India'])
    writer.writerow(['Adil', 'Shaikh', '+91545454169', 'India'])
    writer.writerow(['Ahtesham', 'Shah', '+917554554169', 'India'])

    return response# csv_app/urls.py
...
urlpatterns = [
    ...,
    path('export/csv-simple-write/', views.csv_simple_write, name='csv_simple_write'),
]
```

The above code will download a CSV file with name csv_simple_write.csv with data inside it.

Writing CSV File From a Dictionary

```
# csv_app/views.py

import csv
from django.http import HttpResponse
# Writing CSV File From a Dictionary
def csv_dictionary_write(request):
    # Create the HttpResponse object with the appropriate CSV header.
    response = HttpResponse(content_type='text/csv')
    response['Content-Disposition'] = 'attachment; filename="csv_dictionary_write.csv"'

    fieldnames = ['first_name', 'last_name', 'phone_number', 'country']
    writer = csv.DictWriter(response, fieldnames=fieldnames)

    writer.writeheader()
    writer.writerow({'first_name':'Huzaif', 'last_name':'Sayyed', 'phone_number':'+919954465169', 'country':'India'})
    writer.writerow({'first_name':'Adil', 'last_name':'Shaikh', 'phone_number':'+91545454169', 'country':'India'})
    writer.writerow({'first_name':'Ahtesham', 'last_name':'Shah', 'phone_number':'+917554554169', 'country':'India'})

    return response# csv_app/urls.py
...
urlpatterns = [
    ...,
    path('export/csv-dictionary-write/', views.csv_dictionary_write, name='csv_dictionary_write'),
]
```
The above code will create a CSV file with data from the dictionary.

Dynamic Database Data Export to CSV for Django Export CSV

```
# csv_app/views.py

import csv
from django.http import HttpResponse
from .models import UserDetail

def csv_database_write(request):

    # Get all data from UserDetail Databse Table
    users = UserDetail.objects.all()

    # Create the HttpResponse object with the appropriate CSV header.
    response = HttpResponse(content_type='text/csv')
    response['Content-Disposition'] = 'attachment; filename="csv_database_write.csv"'

    writer = csv.writer(response)
    writer.writerow(['first_name', 'last_name', 'phone_number', 'country'])

    for user in users:
        writer.writerow([user.first_name, user.last_name, user.phone_number, user.country])

    return response# csv_app/urls.py
...
urlpatterns = [
    ...,
    path('export/csv-database-write/', views.csv_database_write, name='csv_database_write'),
]
```
The above code will create a CSV file with data from the database. The data will be of the User model.

Simple CSV Read Operation

```
# csv_app/views.py

import csv
from django.http import HttpResponse
from .models import UserDetail
import os

def csv_simple_read(request):

    path = os.path.dirname(__file__)
    file = os.path.join(path, 'csv_readfile.csv')

    with open(file) as csv_file:
        csv_reader = csv.reader(csv_file, delimiter=',')
        line_count = 0
        # See your console/terminal
        for row in csv_reader:
            if line_count == 0:
                print('\n\nColumn names are {}, {}, {}'.format(row[0], row[1], row[3], row[2]))
                line_count += 1
            else:
                print('\t{} {} lives in {}, and his phone number is {}.'.format(row[0], row[1], row[3], row[2]))
                line_count += 1
        print('Processed {} lines.\n\n'.format(line_count))

        return redirect('/') # Redirect to home# csv_app/urls.py
...
urlpatterns = [
    ...,
    path('export/csv-simple-read/', views.csv_simple_read, name='csv_simple_read'),
]
```

The above code will read a CSV file and display it in Console and then redirect to the home page.

References- https://docs.djangoproject.com/en/4.0/howto/outputting-csv/
https://studygyaan.com/django/how-to-export-csv-file-with-django
https://github.com/studygyaan/How-to-Create-CSV-File-With-Django

Activity 6

Aim: Create Django application that reads data from JSON and display on page.
Learning outcome: Able to understand how to reads data from JSON and display on page.
Duration: 3 hour
List of Hardware/Software requirements:

1. Django
2. Python
3. VS Code

Code/Program/Procedure (with comments):

JSON

JSON (JavaScript Object Notation) is a lightweight data-interchange format. The official Internet media type for JSON is application/json. The JSON filename extension is .json. It is easy for humans to read and write and for machines to parse and generate.

Django JsonResponse

JsonResponse is an HttpResponse subclass that helps to create a JSON-encoded response. Its default Content-Type header is set to application/json. The first parameter, data, should be a dict instance. If the safe parameter is set to False, any object can be passed for serialization; otherwise only dict instances are allowed.

Django JsonResponse example

In the following example, we create a Django application that sends a file to the client. The file is a JPEG image, which is located in the images directory in the project root directory.

```
$ mkdir jsonresponse
$ cd jsonresponse
$ mkdir src
$ cd src
```

We create the project and the and src directories. Then we locate to the src directory.

```
$ django-admin startproject jsonresponse .
```

We create a new Django project in the src directory.
Note: If the optional destination is provided, Django will use that existing directory as the project directory. If it is omitted, Django creates a new directory based on the project name. We use the dot (.) to create a project inside the current working directory.

```
$ cd ..
$ pwd
/c/Users/Jano/Documents/pyprogs/django/jsonresponse
```

We locate to the project directory.

```
$ tree /f
src
│   manage.py
│
└───jsonresponse
        settings.py
        urls.py
        views.py
        wsgi.py
        __init__.py
```

These are the contents of the project directory.
Note: The Django way is to put functionality into apps, which are created with django-admin startapp. In this tutorial, we do not use an app to make the example simpler. We focus on demonstrating how to send JSON response.

src/jsonresponse/urls.py

```
from django.contrib import admin
from django.urls import path
from .views import send_json

urlpatterns = [
    path('admin/', admin.site.urls),
    path('sendjson/', send_json, name='send_json'),
]
```

We add a new route page; it calls the send_json() function from the views.py module.

src/jsonresponse/views.py

```
from django.http import JsonResponse

def send_json(request):

    data = [{'name': 'Peter', 'email': 'peter@example.org'},
            {'name': 'Julia', 'email': 'julia@example.org'}]

    return JsonResponse(data, safe=False)
```

Inside send_json(), we define a list of dictionaries. Since it is a list, we set safe to False. If we did not set this parameter, we would get a TypeError with the following message:

```
In order to allow non-dict objects to be serialized set the safe parameter to False.
```

By default, JsonResponse accepts only Python dictionaries.

```
$ python manage.py runserver
```

We run the server and navigate to http://127.0.0.1:8000/sendjson/.

```
$ curl localhost:8000/sendjson/
[{"name": "Peter", "email": "peter@example.org"},
{"name": "Julia", "email": "julia@example.org"}]
```

We use the curl tool to make the GET request.
In this tutorial, we have demonstrated how to send JSON data in Django.

References : https://zetcode.com/django/jsonresponse/

Activity 7

Aim: Creating Django application which implement CRUD operations over database
Learning outcome: Able to create model and perform CRUD operations.
Duration: 6 hour
List of Hardware/Software requirements:

1. Django
2. Python
3. VS Code

Code/Program/Procedure (with comments):

Django Models

SQLite Database

When we created the Django project, we got an empty SQLite database. It was created in the myworld root folder. We will use this database in this tutorial.

Create Table (Model)

To create a new table, we must create a new model.
In the /members/ folder, open the models.py file. It is almost empty by default, with only an import statement and a comment:
members/models.py:

```
from django.db import models

# Create your models here.
```

To add a Members table in our database, start by creating a Members class, and describe the table fields in it:
members/models.py:

```
from django.db import models

class Members(models.Model):
  firstname = models.CharField(max_length=255)
  lastname = models.CharField(max_length=255)
```

The first field, "firstname" is a Text field, and will contain the first name of the members.
The second field, "lastname" is also a Text field, with the members' last name.
Both "firstname" and "lastname" is set up to have a maximum of 255 characters.
Then navigate to the /myworld/ folder and run this command:

```
py manage.py makemigrations members
```

Which will result in this output:

```
Migrations for 'members':
  members\migrations\0001_initial.py
  - Create model Members

(myproject) C:\Users\Your Name\myproject\myworld>
```
Django creates a file with any new changes and stores the file in the /migrations/ folder.
Next time you run py manage.py migrate Django will create and execute an SQL statement, based on the content of the new file in the migrations folder.
Run the migrate command:
```
py manage.py migrate
```
Which will result in this output:
```
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, members, sessions
Running migrations:
  Applying members.0001_initial... OK

(myproject) C:\Users\Your Name\myproject\myworld>
```
The SQL statement created from the model is:
```
CREATE TABLE "members_members" (
"id" INT NOT NULL PRIMARY KEY AUTOINCREMENT,
"firstname" varchar(255) NOT NULL,
"lastname" varchar(255) NOT NULL);
```

Now you have a Members table in you database!

Django Add Members

Add Records

The Members table is empty, we should add some members to it.
In the next chapters you will learn how to make a user interface that will take care of CRUD operations (Create, Read, Update, Delete), but for now, let's write Python code directly in the Python interpreter (Python shell) and add some members in our database, without the user interface.
To open a Python shell, type this command:
```
py manage.py shell
```
Now we are in the shell, the result should be something like this:
```
Python 3.9.2 (tags/v3.9.2:1a79785, Feb 19 2021, 13:44:55) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>>
```
At the bottom, after the three >>> write the following:
```
>>> from members.models import Members
```
Hit [enter] and write this to look at the empty Members table:
```
>>> Members.objects.all()
```
This should give you an empty QuerySet object, like this:
```
<QuerySet []>
```
A QuerySet is a collection of data from a database.
Read more about QuerySets in the Django QuerySet chapter.
Add a record to the table, by executing these two lines:
```
>>> member = Members(firstname='Emil', lastname='Refsnes')
>>> member.save()
```
Execute this command to see if the Members table got a member:

```
>>> Members.objects.all().values()
```

Hopefully, the result will look like this:

```
<QuerySet [{'id': 1, 'firstname': 'Emil', 'lastname': 'Refsnes'}]>
```

Add Multiple Records

You can add multiple records by making a list of Members obejcts, and execute .save() on each entry:

```
>>> member1 = Members(firstname='Tobias', lastname='Refsnes')
>>> member2 = Members(firstname='Linus', lastname='Refsnes')
>>> member3 = Members(firstname='Lene', lastname='Refsnes')
>>> member4 = Members(firstname='Stale', lastname='Refsnes')
>>> members_list = [member1, member2, member3, member4]
>>> for x in members_list:
>>>   x.save()
```

Now there are 5 members in the Members table:

```
>>> Members.objects.all().values()
<QuerySet [{'id': 1, 'firstname': 'Emil', 'lastname': 'Refsnes'},
{'id': 2, 'firstname': 'Tobias', 'lastname': 'Refsnes'}, {'id': 3, 'firstname': 'Linus', 'lastname': 'Refsnes'}, {'id': 4, 'firstname':
'Lene', 'lastname': 'Refsnes'}, {'id': 5, 'firstname': 'Stale', 'lastname': 'Refsnes'}]>
```

View in Browser

We want to see the result in a web page, not in a Python shell environment.
To see the result in a web page, we can create a *view* for this particular task.
In the members app, open the views.py file, if you have followed the previous chapters of this tutorial, it should look like this:

members/views.py:

```
from django.http import HttpResponse
from django.template import loader

def index(request):
  template = loader.get_template('myfirst.html')
  HttpResponse(template.render())
```

Change the content in the views.py file to look like this instead:

members/views.py:

```
from django.http import HttpResponse
from django.template import loader
from .models import Members

def index(request):
  mymembers = Members.objects.all().values()
  output = ""
  for x in mymembers:
    output += x["firstname"]
  return HttpResponse(output)
```

As you can see in line 3, the Members model is imported, and the index view does the following:

- makes a mymembers object with all the values of the Members model.
- Loops through all items in the mymembers object to build a string with all the firstname values.

- Returns the string as output to the browser.

See the result in your browser. If you are still in the Python shell, write this command to exit the shell:

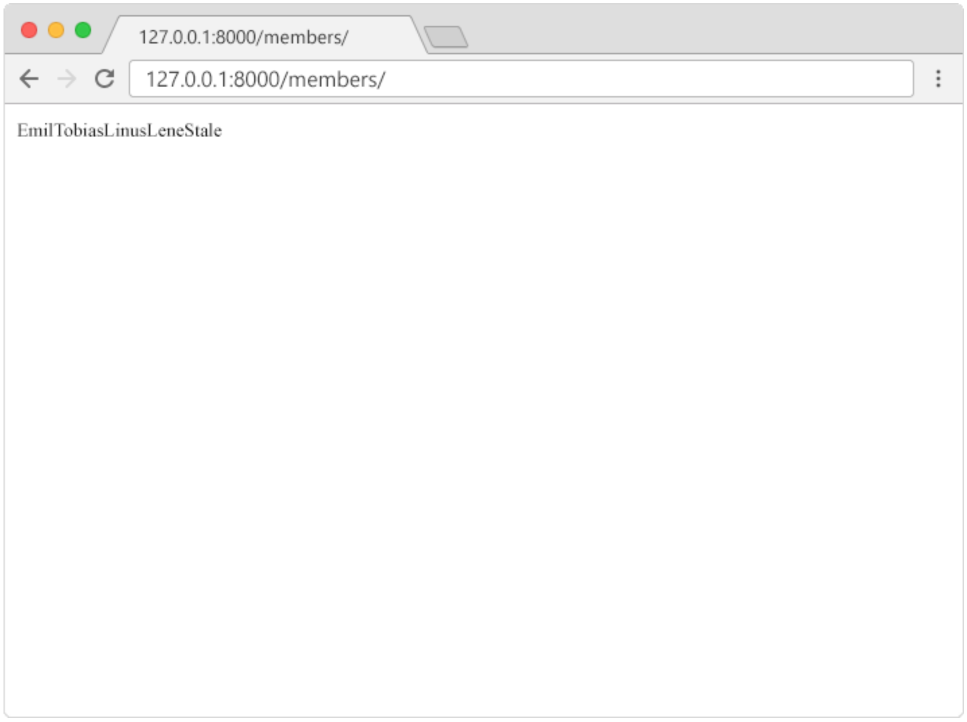
```
>>> quit()
```

Navigate to the /myworld/ folder and type this to start the server:

```
py manage.py runserver
```

In the browser window, type 127.0.0.1:8000/members/ in the address bar.

The result:



```
EmilTobiasLinusLeneStale
```

Adding Template

To add some HTML around the values, we will create a template for the application.

All templates must be located in the templates folder off your app, if you have not already created a templates folder, do it now.

In the templates folder, create a file named index.html, with the following content:

members/templates/index.html:

```
<h1>Members</h1>

<table border="1">
{% for x in mymembers %}
<tr>
<td>{{ x.id }}</td>
<td>{{ x.firstname }}</td>
<td>{{ x.lastname }}</td>
</tr>
{% endfor %}
</table>
```

Did you notice the {% %} and {{ }} parts? They are called template tags.

Template tags allows you to perform logic and render variables in your templates, you will learn more about template tags later.

Modify the View

Change the index view to include the template:

members/views.py:

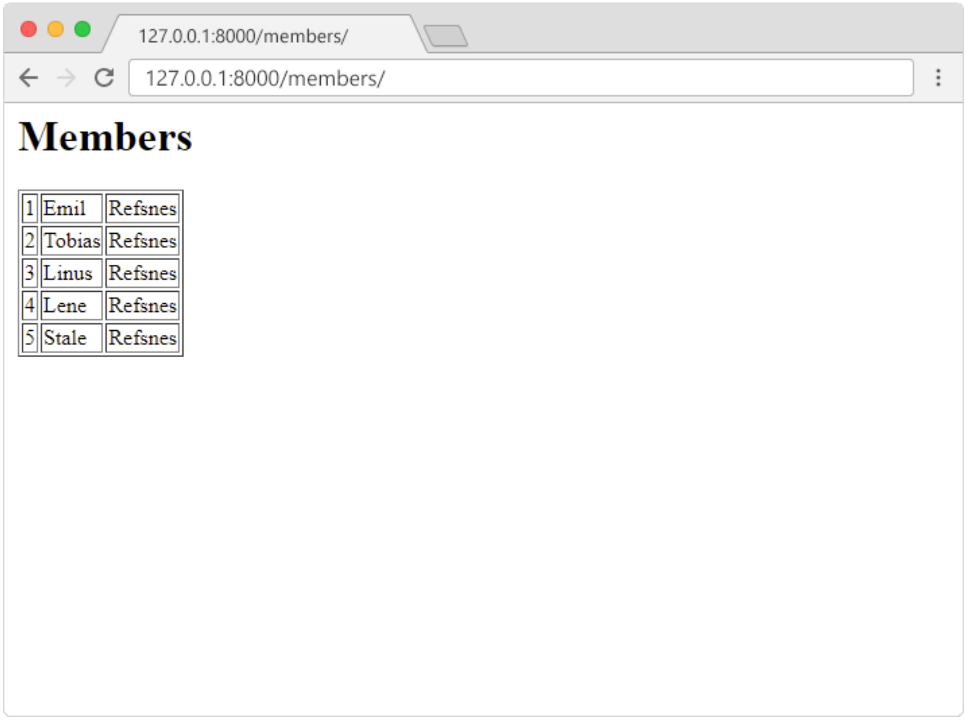
```
from django.http import HttpResponse
from django.template import loader
from .models import Members

def index(request):
  mymembers = Members.objects.all().values()
  template = loader.get_template('index.html')
  context = {
    'mymembers': mymembers,
  }
  return HttpResponse(template.render(context, request))
```

The index view does the following:

- Creates a mymembers object with all the values of the Members model.
- Loads a the index.html template.
- Creates an object containing the mymember object.
- Sends the object to the template.
- Outputs the HTML that is rendered by the template.

In the browser window, type 127.0.0.1:8000/members/ in the address bar.

The result:



Members

1	Emil	Refsnes
2	Tobias	Refsnes
3	Linus	Refsnes
4	Lene	Refsnes
5	Stale	Refsnes

Django Add Record

Adding Records

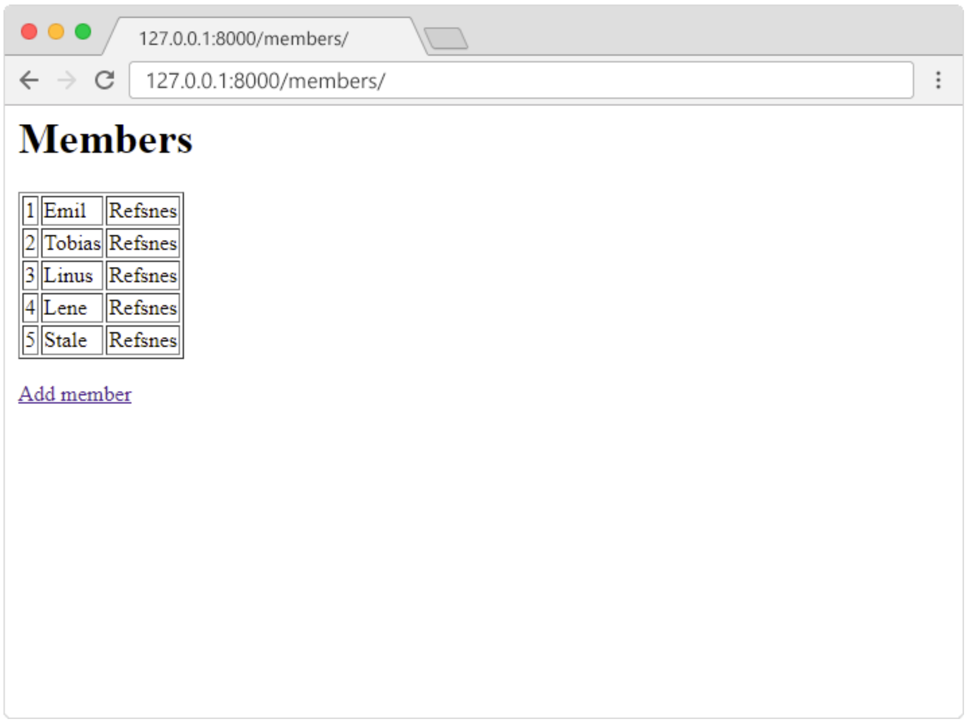
So far we have created a Members table in our database, and we have inserted five records by writing code in the Python shell.
We have also made a template that allows us to display the content of the table in a web page.
Now we want to be able to create new members from a web page.

Template

Start by adding a link in the members template:
members/templates/index.html:

```
<h1>Members</h1>

<table border="1">
{% for x in mymembers %}
<tr>
<td>{{ x.id }}</td>
<td>{{ x.firstname }}</td>
<td>{{ x.lastname }}</td>
</tr>
{% endfor %}
</table>

<p>
<a href="add/">Add member</a>
</p>
```

The result will look like this:


127.0.0.1:8000/members/

Members

1	Emil	Refsnes
2	Tobias	Refsnes
3	Linus	Refsnes
4	Lene	Refsnes
5	Stale	Refsnes

Add member

New Template

Add a new template in the templates folder, named add.html:

members/templates/add.html:

```
<h1>Add member</h1>

<form action="addrecord/" method="post">
{% csrf_token %}
First Name:<br>
<input name="first">
<br><br>
Last Name:<br>
<input name="last">
<br><br>
<input type="submit" value="Submit">
</form>
```

The template contains an empty HTML form with two input fields and a submit button.
Note: Django requires this line in the form:
{% csrf_token %}
to handle Cross Site Request Forgeries in forms where the method is POST.

View

Next, add a view in the members/views.py file, name the new view add:

members/views.py:

```
from django.http import HttpResponse
from django.template import loader
from .models import Members
```

```
def index(request):
  mymembers = Members.objects.all().values()
  template = loader.get_template('index.html')
  context = {
    'mymembers': mymembers,
  }
  return HttpResponse(template.render(context, request))

def add(request):
  template = loader.get_template('add.html')
  return HttpResponse(template.render({}, request))
```

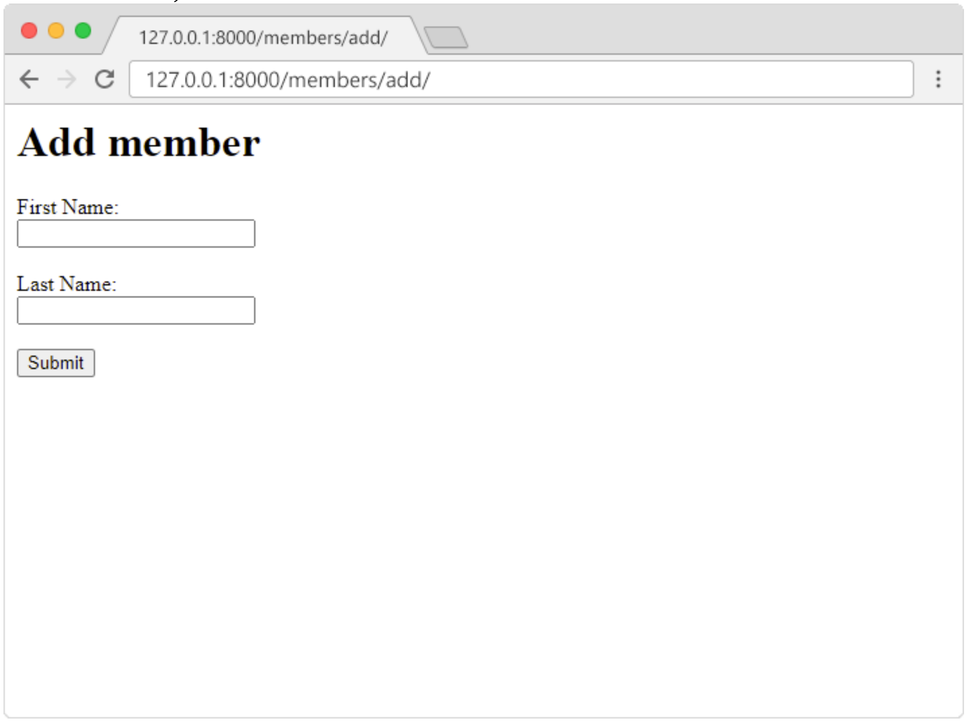
URLs

Add a path() function in the members/urls.py file, that points the url 127.0.0.1:8000/members/add/ to the right location:

members/urls.py:

```
from django.urls import path
from . import views

urlpatterns = [
  path('', views.index, name='index'),
  path('add/', views.add, name='add'),
]
```

In the browser, click the "Add member" link and the result should look like this:



127.0.0.1:8000/members/add/

127.0.0.1:8000/members/add/

Add member

First Name:

Last Name:

More URLs

Did you notice the action attribute in the HTML form? The action attribute specifies where to send the form data, in this case the form data will be sent to addrecord/, so we must add a path() function in the members/urls.py file that points to the right view:

members/urls.py:

```
from django.urls import path
from . import views

urlpatterns = [
  path('', views.index, name='index'),
  path('add/', views.add, name='add'),
  path('add/addrecord/', views.addrecord, name='addrecord'),
]
```

Code for Adding Records

So far we have made the user interface, and we point the URL to the view called addrecord, but we have not made the view yet.

Make sure you add the addrecord view in the in the members/views.py file:

members/views.py:

```
from django.http import HttpResponse, HttpResponseRedirect
from django.template import loader
from django.urls import reverse
from .models import Members

def index(request):
  mymembers = Members.objects.all().values()
  template = loader.get_template('index.html')
  context = {
    'mymembers': mymembers,
  }
  return HttpResponse(template.render(context, request))

def add(request):
  template = loader.get_template('add.html')
  return HttpResponse(template.render({}, request))

def addrecord(request):
  x = request.POST['first']
  y = request.POST['last']
  member = Members(firstname=x, lastname=y)
  member.save()
  return HttpResponseRedirect(reverse('index'))
```

Changes that are made in the views.py file:

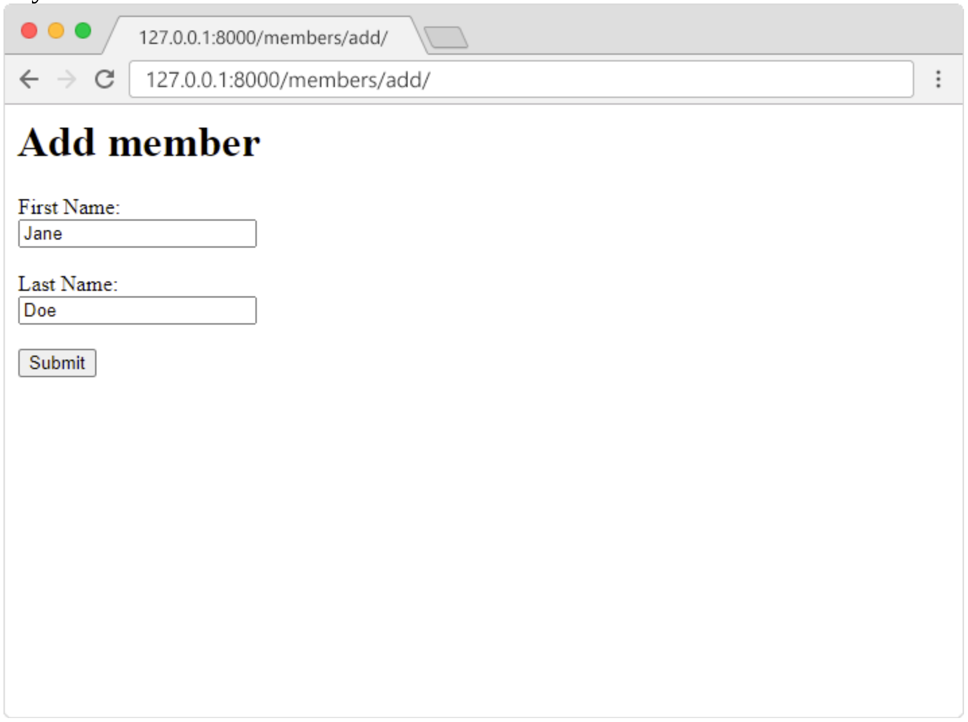
Line 1: import HttpResponseRedirect

Line 3: import reverse

The addrecord view does the following:

- Gets the first name and last name with the request.POST statement.
- Adds a new record in the members table.
- Redirects the user back to the index view.

Try to add a new record and see how it works:



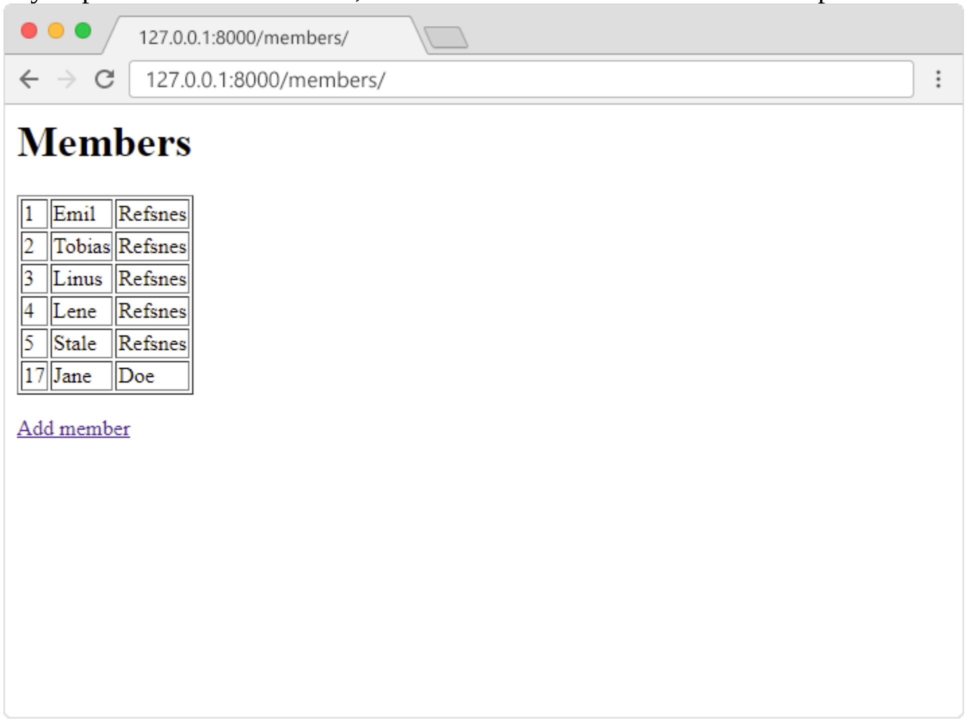
Add member

First Name:
Jane

Last Name:
Doe

Submit

If you press the submit button, the members table should have been updated:



Members

1	Emil	Refsnes
2	Tobias	Refsnes
3	Linus	Refsnes
4	Lene	Refsnes
5	Stale	Refsnes
17	Jane	Doe

Add member

Django Delete Record

Deleting Records

To delete a record we do not need a new template, but we need to make some changes to the members template.

Of course, you can chose how you want to add a delete button, but in this example, we will add a "delete" link for each record in a new table column.
The "delete" link will also contain the ID of each record.

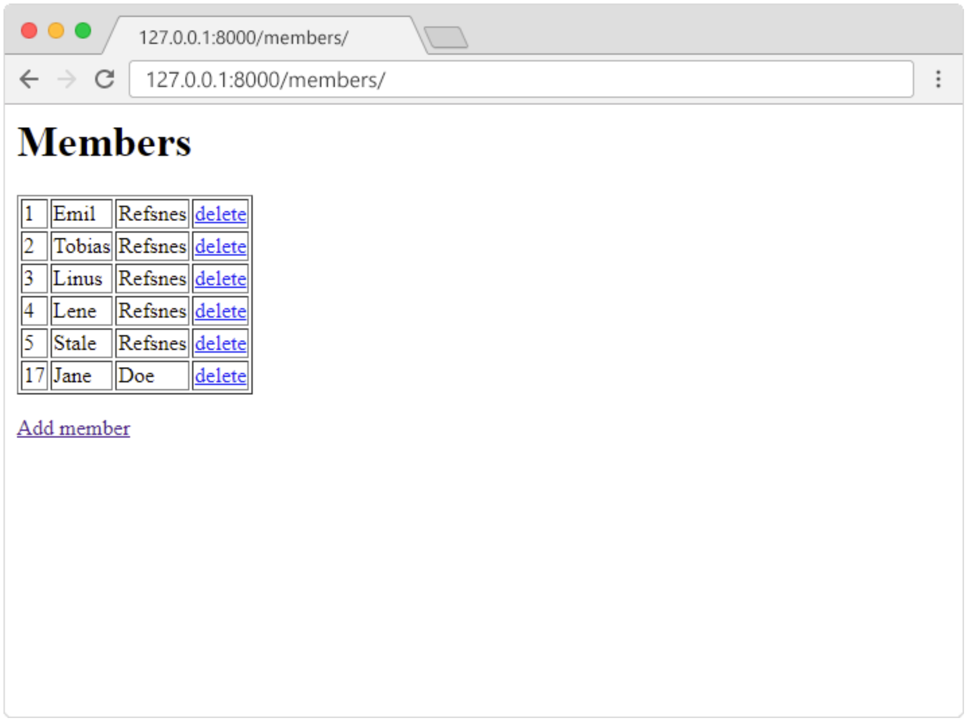
Modify Template

Add a "delete" column in the members template:

members/templates/index.html:

```
<h1>Members</h1>

<table border="1">
{% for x in mymembers %}
  <tr>
  <td>{{ x.id }}</td>
  <td>{{ x.firstname }}</td>
  <td>{{ x.lastname }}</td>
  <td><a href="delete/{{ x.id }}">delete</a></td>
  </tr>
{% endfor %}
</table>

<p>
<a href="add/">Add member</a>
</p>
```

The result will look like this:



127.0.0.1:8000/members/

Members

1	Emil	Refsnes	delete
2	Tobias	Refsnes	delete
3	Linus	Refsnes	delete
4	Lene	Refsnes	delete
5	Stale	Refsnes	delete
17	Jane	Doe	delete

Add member

URLs

The "delete" link in the HTML table points to 127.0.0.1:8000/members/delete/ so we will add a path() function in the members/urls.py file, that points the url to the right location, with the ID as a parameter:
members/urls.py:
```
from django.urls import path
from . import views

urlpatterns = [
  path('', views.index, name='index'),
  path('add/', views.add, name='add'),
  path('add/addrecord/', views.addrecord, name='addrecord'),
  path('delete/<int:id>', views.delete, name='delete'),
]
```

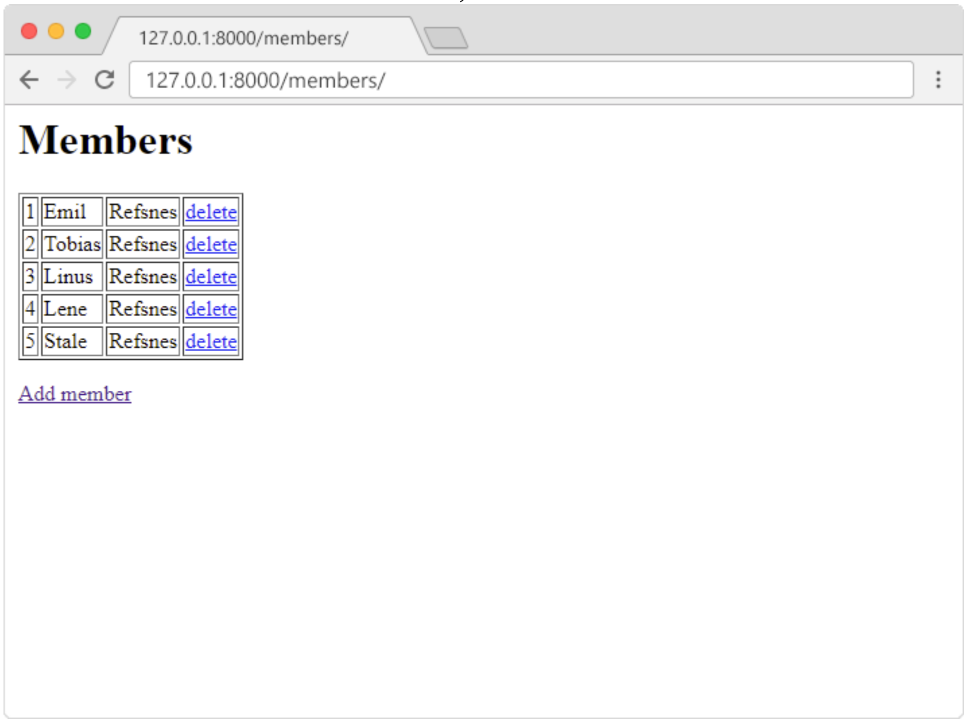
Code for Deleting Records

Now we need to add a new view called delete in the members/views.py file:
members/views.py:
```
from django.http import HttpResponse, HttpResponseRedirect
from django.template import loader
from django.urls import reverse

from .models import Members

def index(request):
  mymembers = Members.objects.all().values()
  template = loader.get_template('index.html')
  context = {
    'mymembers': mymembers,
  }
  return HttpResponse(template.render(context, request))

def add(request):
  template = loader.get_template('add.html')
  return HttpResponse(template.render({}, request))

def addrecord(request):
  x = request.POST['first']
  y = request.POST['last']
  member = Members(firstname=x, lastname=y)
  member.save()
  return HttpResponseRedirect(reverse('index'))

def delete(request, id):
  member = Members.objects.get(id=id)
  member.delete()
  return HttpResponseRedirect(reverse('index'))
```
The delete view does the following:

- Gets the id as an argument.
- Uses the id to locate the correct record in the Members table.

- Deletes that record.
- Redirects the user back to the index view.

Click on the "delete" link for Jane Doe, and see the result:



127.0.0.1:8000/members/

Members

1	Emil	Refsnes	delete
2	Tobias	Refsnes	delete
3	Linus	Refsnes	delete
4	Lene	Refsnes	delete
5	Stale	Refsnes	delete

Add member

Django Update Record

Updating Records

To update a record, we need the ID of the record, and we need a template with an interface that let us change the values.

First we need to make some changes in the index.html template.

Modify Template

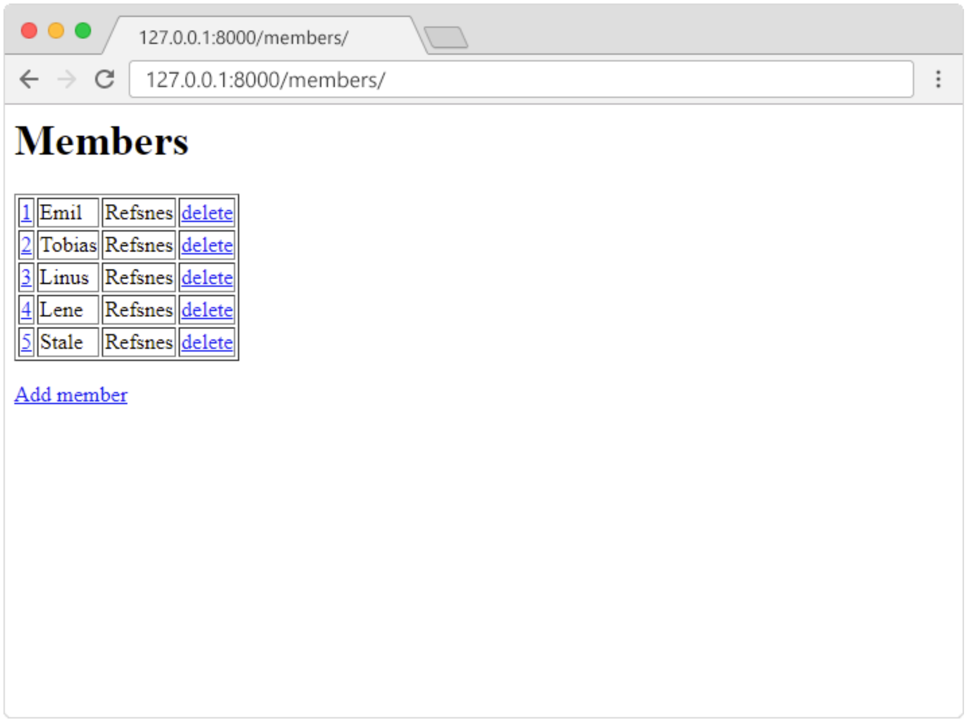
Start by adding a link for each member in the table:

members/templates/index.html:

```
<h1>Members</h1>

<table border="1">
{% for x in mymembers %}
<tr>
<td><a href="update/{{ x.id }}">{{ x.id }}</a></td>
<td>{{ x.firstname }}</td>
<td>{{ x.lastname }}</td>
<td><a href="delete/{{ x.id }}">delete</a>
</tr>
{% endfor %}
</table>
```

```
<p>
<a href="add/">Add member</a>
</p>
```

The link goes to a view called update with the ID of the current member.
The result will look like this:



Members

1	Emil	Refsnes	delete
2	Tobias	Refsnes	delete
3	Linus	Refsnes	delete
4	Lene	Refsnes	delete
5	Stale	Refsnes	delete

Add member

View

Next, add the update view in the members/views.py file:

members/views.py:

```
from django.http import HttpResponse, HttpResponseRedirect
from django.template import loader
from django.urls import reverse
from .models import Members

def index(request):
  mymembers = Members.objects.all().values()
  template = loader.get_template('index.html')
  context = {
    'mymembers': mymembers
  }
  return HttpResponse(template.render(context, request))

def add(request):
  template = loader.get_template('add.html')
  return HttpResponse(template.render({}, request))

 def addrecord(request):
  first = request.POST['first']
  last = request.POST['last']
  member = Members(firstname=first, lastname=last)
  member.save()

return HttpResponseRedirect(reverse('index'))

def delete(request, id):
  member = Members.objects.get(id=id)
  member.delete()
  return HttpResponseRedirect(reverse('index'))

def update(request, id):
  mymember = Members.objects.get(id=id)
  template = loader.get_template('update.html')
  context = {
    'mymember': mymember,
  }
  return HttpResponse(template.render(context, request))
```

The update view does the following:

- Gets the id as an argument.
- Uses the id to locate the correct record in the Members table.
- loads a template called update.html.
- Creates an object containing the member.
- Sends the object to the template.
- Outputs the HTML that is rendered by the template.

New Template

Add a new template in the templates folder, named update.html:
members/templates/update.html:

```
<h1>Update member</h1>

<form action="updaterecord/{{ mymember.id }}" method="post">
{% csrf_token %}
First Name:<br>
<input name="first" value="{{ mymember.firstname }}">
<br><br>
Last Name:<br>
<input name="last" value="{{ mymember.lastname }}">
<br><br>
<input type="submit" value="Submit">
</form>
```

The template contains an HTML form with the values from the selected member.
Note: Django requires this line in the form:
{% csrf_token %}
to handle Cross Site Request Forgeries in forms where the method is POST.

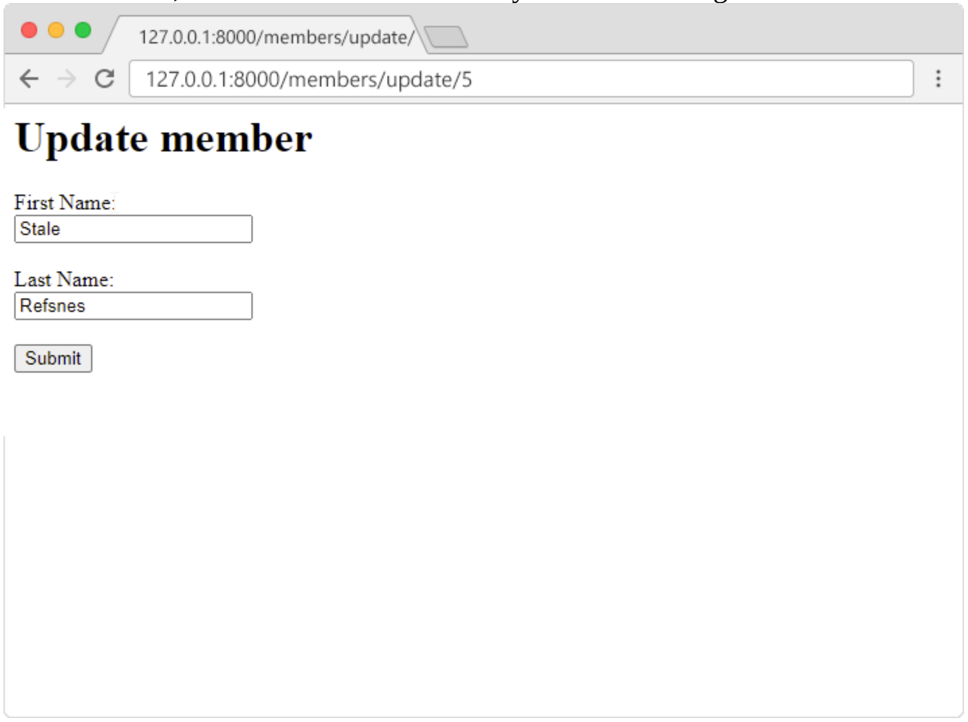
URLs

Add a path() function in the members/urls.py file, that points the url 127.0.0.1:8000/members/update/ to the right location, with the ID as a parameter:

members/urls.py:

```
from django.urls import path

from . import views

urlpatterns = [
  path('', views.index, name='index'),
  path('add/', views.add, name='add'),
  path('add/addrecord/', views.addrecord, name='addrecord'),
  path('delete/<int:id>', views.delete, name='delete'),
  path('update/<int:id>', views.update, name='update'),
]
```

In the browser, click the ID of the member you want to change and the result should look like this:


127.0.0.1:8000/members/update/

127.0.0.1:8000/members/update/5

Update member

First Name: Stale

Last Name: Refsnes

What Happens on Submit?

Did you notice the action attribute in the HTML form? The action attribute specifies where to send the form data, in this case the form data will be sent to:

updaterecord/{{ mymember.id }}, so we must add a path() function in the members/urls.py file that points to the right view:

members/urls.py:

```
from django.urls import path

from . import views

urlpatterns = [
  path('', views.index, name='index'),
  path('add/', views.add, name='add'),
  path('add/addrecord/', views.addrecord, name='addrecord'),
  path('delete/<int:id>', views.delete, name='delete'),
  path('update/<int:id>', views.update, name='update'),
  path('update/updaterecord/<int:id>', views.updaterecord, name='updaterecord'),
]
```

Code for Updating Records

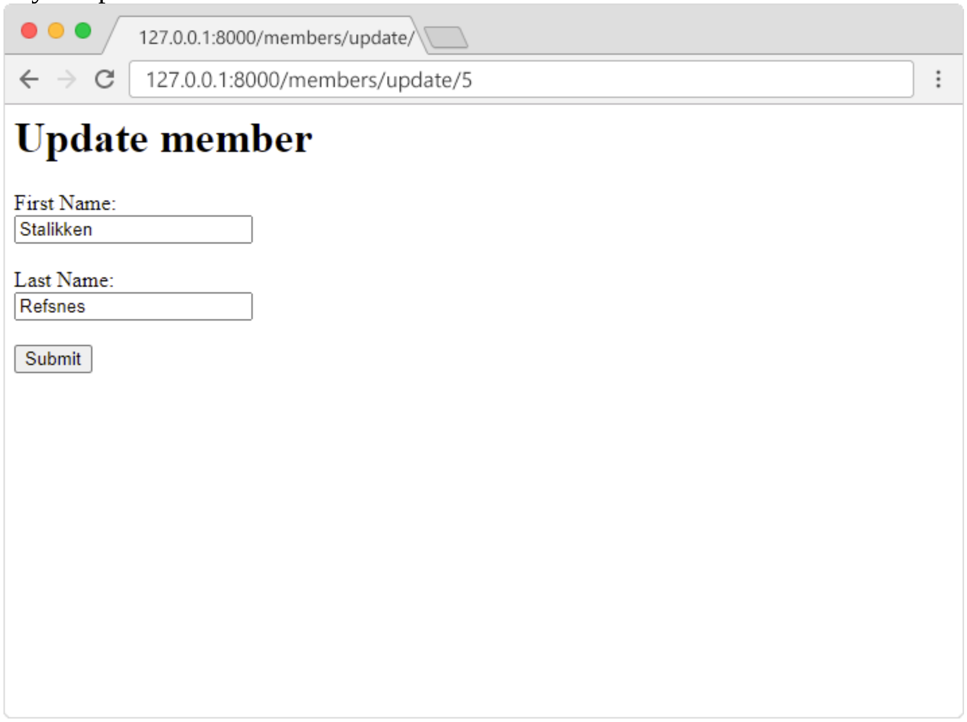
So far we have made the user interface, and we point the URL to the view called updaterecord, but we have not made the view yet.

Make sure you add the updaterecord view in the in the members/views.py file:

members/views.py:

```
from django.http import HttpResponse, HttpResponseRedirect
from django.template import loader
from django.urls import reverse
from .models import Members

def index(request):
  mymembers = Members.objects.all().values()
  template = loader.get_template('index.html')
  context = {
    'mymembers': mymembers,
  }
  return HttpResponse(template.render(context, request))

def add(request):
  template = loader.get_template('add.html')
  return HttpResponse(template.render({}, request))

def addrecord(request):
  x = request.POST['first']
  y = request.POST['last']
  member = Members(firstname=x, lastname=y)
  member.save()
  return HttpResponseRedirect(reverse('index'))

def delete(request, id):
  member = Members.objects.get(id=id)
  member.delete()
  return HttpResponseRedirect(reverse('index'))

def update(request, id):
  mymember = Members.objects.get(id=id)
  template = loader.get_template('update.html')
  context = {
    'mymember': mymember,
  }
  return HttpResponse(template.render(context, request))

def updaterecord(request, id):
  first = request.POST['first']
  last = request.POST['last']
  member = Members.objects.get(id=id)
  member.firstname = first
  member.lastname = last
  member.save()
  return HttpResponseRedirect(reverse('index'))
```

The updaterecord function will update the record in the members table with the selected ID.
Try to update a record and see how it works:

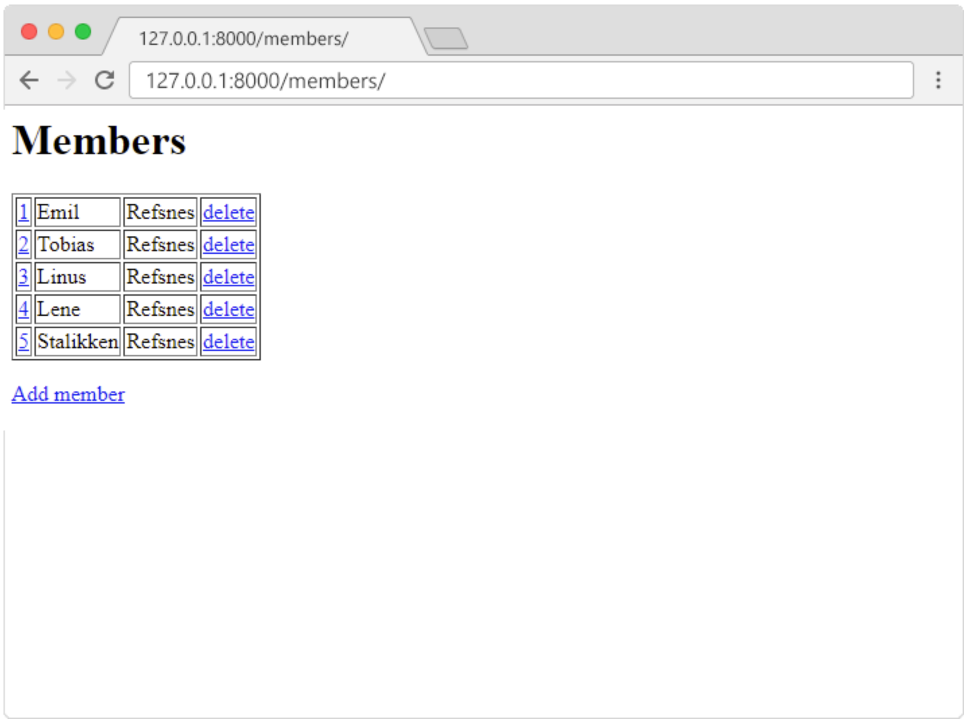

127.0.0.1:8000/members/update/
127.0.0.1:8000/members/update/5

Update member

First Name:
Stalikken

Last Name:
Refsnes

If you press the submit button, the members table should have been updated:


127.0.0.1:8000/members/

Members

1	Emil	Refsnes	delete
2	Tobias	Refsnes	delete
3	Linus	Refsnes	delete
4	Lene	Refsnes	delete
5	Stalikken	Refsnes	delete

Add member

Refernces - https://www.w3schools.com/django/django_models.php
https://www.w3schools.com/django/django_add_members.php
https://www.w3schools.com/django/django_add_template.php
https://www.w3schools.com/django/django_add_record.php
https://www.w3schools.com/django/django_delete_record.php
https://www.w3schools.com/django/django_update_record.php

Activity 8

Aim: Create Django application validates user credentials on login pages
Learning outcome: Able to understand basic user authentication and authorization .
Duration: 4 hour
List of Hardware/Software requirements:

1. Django
2. Python

Code/Program/Procedure (with comments):
For following code URL: https://docs.djangoproject.com/en/4.0/topics/auth/

User authentication in Django¶

Django comes with a user authentication system. It handles user accounts, groups, permissions and cookie-based user sessions. This section of the documentation explains how the default implementation works out of the box, as well as how to extend and customize it to suit your project's needs.

Overview¶

The Django authentication system handles both authentication and authorization. Briefly, authentication verifies a user is who they claim to be, and authorization determines what an authenticated user is allowed to do. Here the term authentication is used to refer to both tasks.
The auth system consists of:

- Users
- Permissions: Binary (yes/no) flags designating whether a user may perform a certain task.
- Groups: A generic way of applying labels and permissions to more than one user.
- A configurable password hashing system
- Forms and view tools for logging in users, or restricting content
- A pluggable backend system

The authentication system in Django aims to be very generic and doesn't provide some features commonly found in web authentication systems. Solutions for some of these common problems have been implemented in third-party packages:

- Password strength checking
- Throttling of login attempts
- Authentication against third-parties (OAuth, for example)
- Object-level permissions

Installation¶

Authentication support is bundled as a Django contrib module in **django.contrib.auth**. By default, the required configuration is already included in the **settings.py** generated by **django-admin startproject**, these consist of two items listed in your **INSTALLED_APPS** setting:

17. **'django.contrib.auth'** contains the core of the authentication framework, and its default models.
18. **'django.contrib.contenttypes'** is the Django content type system, which allows permissions to be associated with models you create.

and these items in your **MIDDLEWARE** setting:

19. **SessionMiddleware** manages sessions across requests.
20. **AuthenticationMiddleware** associates users with requests using sessions.

With these settings in place, running the command **manage.py migrate** creates the necessary database tables for auth related models and permissions for any models defined in your installed apps

=====

For following code refer url- https://docs.djangoproject.com/en/4.0/topics/auth/default/

Using the Django authentication system¶

This document explains the usage of Django's authentication system in its default configuration. This configuration has evolved to serve the most common project needs, handling a reasonably wide range of tasks, and has a careful implementation of passwords and permissions. For projects where authentication needs differ from the default, Django supports extensive extension and customization of authentication.
Django authentication provides both authentication and authorization together and is generally referred to as the authentication system, as these features are somewhat coupled.

User objects¶

User objects are the core of the authentication system. They typically represent the people interacting with your site and are used to enable things like restricting access, registering user profiles, associating content with creators etc. Only one class of user exists in Django's authentication framework, i.e., **'superusers'** or admin **'staff'** users are just user objects with special attributes set, not different classes of user objects.
The primary attributes of the default user are:

- **username**
- **password**
- **email**
- **first_name**

- **last_name**

See the **full API documentation** for full reference, the documentation that follows is more task oriented.

Creating users¶

The most direct way to create users is to use the included **create_user()** helper function:

```
>>> from django.contrib.auth.models import User
>>> user = User.objects.create_user('john', 'lennon@thebeatles.com', 'johnpassword')

# At this point, user is a User object that has already been saved
# to the database. You can continue to change its attributes
# if you want to change other fields.
>>> user.last_name = 'Lennon'
>>> user.save()
```

If you have the Django admin installed, you can also create users interactively.

Creating superusers¶

Create superusers using the **createsuperuser** command:

```
$ python manage.py createsuperuser --username=joe --email=joe@example.com
```

You will be prompted for a password. After you enter one, the user will be created immediately. If you leave off the **--username** or **--email** options, it will prompt you for those values.

Changing passwords¶

Django does not store raw (clear text) passwords on the user model, but only a hash (see documentation of how passwords are managed for full details). Because of this, do not attempt to manipulate the password attribute of the user directly. This is why a helper function is used when creating a user.

To change a user's password, you have several options:

manage.py changepassword *username* offers a method of changing a user's password from the command line. It prompts you to change the password of a given user which you must enter twice. If they both match, the new password will be changed immediately. If you do not supply a user, the command will attempt to change the password whose username matches the current system user.

You can also change a password programmatically, using **set_password()**:

```
>>> from django.contrib.auth.models import User
>>> u = User.objects.get(username='john')
>>> u.set_password('new password')
>>> u.save()
```

If you have the Django admin installed, you can also change user's passwords on the authentication system's admin pages.

Django also provides views and forms that may be used to allow users to change their own passwords.

Changing a user's password will log out all their sessions. See Session invalidation on password change for details.

Authenticating users¶

authenticate(*request=None*, ***credentials*)¶

Use **authenticate()** to verify a set of credentials. It takes credentials as keyword arguments, **username** and **password** for the default case, checks them against each authentication backend, and returns a **User** object if the credentials are valid for a backend. If the credentials aren't valid for any backend or if a backend raises **PermissionDenied**, it returns **None**. For example:

```
from django.contrib.auth import authenticate
user = authenticate(username='john', password='secret')
if user is not None:
    # A backend authenticated the credentials
else:
    # No backend authenticated the credentials
```

request is an optional **HttpRequest** which is passed on the **authenticate()** method of the authentication backends.

Note
This is a low level way to authenticate a set of credentials; for example, it's used by the **RemoteUserMiddleware**. Unless you are writing your own authentication system, you probably won't use this. Rather if you're looking for a way to login a user, use the **LoginView**.

Permissions and Authorization¶

Django comes with a built-in permissions system. It provides a way to assign permissions to specific users and groups of users.
It's used by the Django admin site, but you're welcome to use it in your own code.
The Django admin site uses permissions as follows:

- Access to view objects is limited to users with the "view" or "change" permission for that type of object.
- Access to view the "add" form and add an object is limited to users with the "add" permission for that type of object.
- Access to view the change list, view the "change" form and change an object is limited to users with the "change" permission for that type of object.
- Access to delete an object is limited to users with the "delete" permission for that type of object.

Permissions can be set not only per type of object, but also per specific object instance. By using the **has_view_permission()**, **has_add_permission()**, **has_change_permission()** and **has_delete_permission()** methods provided by the **ModelAdmin** class, it is possible to customize permissions for different object instances of the same type.
User objects have two many-to-many fields: **groups** and **user_permissions**. **User** objects can access their related objects in the same way as any other Django model:

```
myuser.groups.set([group_list])
myuser.groups.add(group, group, ...)
myuser.groups.remove(group, group, ...)
myuser.groups.clear()
myuser.user_permissions.set([permission_list])
myuser.user_permissions.add(permission, permission, ...)
myuser.user_permissions.remove(permission, permission, ...)
myuser.user_permissions.clear()
```

Default permissions¶

When **django.contrib.auth** is listed in your **INSTALLED_APPS** setting, it will ensure that four default permissions – add, change, delete, and view – are created for each Django model defined in one of your installed applications.
These permissions will be created when you run **manage.py migrate**; the first time you run **migrate** after adding **django.contrib.auth** to **INSTALLED_APPS**, the default permissions will be created for all previously-installed models, as well as for any new models being installed at that time. Afterward, it will create default permissions for new models each time you run **manage.py migrate** (the function that creates permissions is connected to the **post_migrate** signal).
Assuming you have an application with an **app_label** **foo** and a model named **Bar**, to test for basic permissions you should use:

- add: **user.has_perm('foo.add_bar')**
- change: **user.has_perm('foo.change_bar')**
- delete: **user.has_perm('foo.delete_bar')**
- view: **user.has_perm('foo.view_bar')**

The **Permission** model is rarely accessed directly.

Groups¶

django.contrib.auth.models.Group models are a generic way of categorizing users so you can apply permissions, or some other label, to those users. A user can belong to any number of groups.
A user in a group automatically has the permissions granted to that group. For example, if the group **Site editors** has the permission **can_edit_home_page**, any user in that group will have that permission.
Beyond permissions, groups are a convenient way to categorize users to give them some label, or extended functionality. For example, you could create a group **'Special users'**, and you could write code that could, say, give them access to a members-only portion of your site, or send them members-only email messages.

Programmatically creating permissions¶

While custom permissions can be defined within a model's **Meta** class, you can also create permissions directly. For example, you can create the **can_publish** permission for a **BlogPost** model in **myapp**:

```
from myapp.models import BlogPost
from django.contrib.auth.models import Permission
from django.contrib.contenttypes.models import ContentType

content_type = ContentType.objects.get_for_model(BlogPost)
permission = Permission.objects.create(
    codename='can_publish',
    name='Can Publish Posts',
    content_type=content_type,
)
```

The permission can then be assigned to a **User** via its **user_permissions** attribute or to a **Group** via its **permissions** attribute.

Proxy models need their own content type

If you want to create permissions for a proxy model, pass **for_concrete_model=False** to **ContentTypeManager.get_for_model()** to get the appropriate **ContentType**:

```
content_type = ContentType.objects.get_for_model(BlogPostProxy, for_concrete_model=False)
```

Permission caching¶

The **ModelBackend** caches permissions on the user object after the first time they need to be fetched for a permissions check. This is typically fine for the request-response cycle since permissions aren't typically checked immediately after they are added (in the admin, for example). If you are adding permissions and checking them immediately afterward, in a test or view for example, the easiest solution is to re-fetch the user from the database. For example:

```
from django.contrib.auth.models import Permission, User
from django.contrib.contenttypes.models import ContentType
from django.shortcuts import get_object_or_404

from myapp.models import BlogPost

def user_gains_perms(request, user_id):
    user = get_object_or_404(User, pk=user_id)
    # any permission check will cache the current set of permissions
    user.has_perm('myapp.change_blogpost')

    content_type = ContentType.objects.get_for_model(BlogPost)
    permission = Permission.objects.get(
        codename='change_blogpost',
        content_type=content_type,
    )
    user.user_permissions.add(permission)

    # Checking the cached permission set
    user.has_perm('myapp.change_blogpost')  # False

    # Request new instance of User
    # Be aware that user.refresh_from_db() won't clear the cache.
    user = get_object_or_404(User, pk=user_id)

    # Permission cache is repopulated from the database
    user.has_perm('myapp.change_blogpost')  # True

    ...
```

Proxy models¶

Proxy models work exactly the same way as concrete models. Permissions are created using the own content type of the proxy model. Proxy models don't inherit the permissions of the concrete model they subclass:

```
class Person(models.Model):
    class Meta:
        permissions = [('can_eat_pizzas', 'Can eat pizzas')]

class Student(Person):
    class Meta:
        proxy = True
        permissions = [('can_deliver_pizzas', 'Can deliver pizzas')]

>>> # Fetch the content type for the proxy model.
>>> content_type = ContentType.objects.get_for_model(Student, for_concrete_model=False)
>>> student_permissions = Permission.objects.filter(content_type=content_type)
>>> [p.codename for p in student_permissions]
['add_student', 'change_student', 'delete_student', 'view_student',
'can_deliver_pizzas']
>>> for permission in student_permissions:
...     user.user_permissions.add(permission)
>>> user.has_perm('app.add_person')
False
>>> user.has_perm('app.can_eat_pizzas')
False
>>> user.has_perms(('app.add_student', 'app.can_deliver_pizzas'))
True
```

Authentication in web requests¶

Django uses sessions and middleware to hook the authentication system into **request objects**.

These provide a **request.user** attribute on every request which represents the current user. If the current user has not logged in, this attribute will be set to an instance of **AnonymousUser**, otherwise it will be an instance of **User**.

You can tell them apart with **is_authenticated**, like so:

```
if request.user.is_authenticated:
    # Do something for authenticated users.
    ...
else:
    # Do something for anonymous users.
    ...
```

How to log a user in¶

If you have an authenticated user you want to attach to the current session - this is done with a **login()** function.

login(*request, user, backend=None*)¶

To log a user in, from a view, use **login()**. It takes an **HttpRequest** object and a **User** object. **login()** saves the user's ID in the session, using Django's session framework.

Note that any data set during the anonymous session is retained in the session after a user logs in.

This example shows how you might use both **authenticate()** and **login()**:

```
from django.contrib.auth import authenticate, login
```

```
def my_view(request):
    username = request.POST['username']
    password = request.POST['password']
    user = authenticate(request, username=username, password=password)
    if user is not None:
        login(request, user)
        # Redirect to a success page.
        ...
    else:
        # Return an 'invalid login' error message.
        ...
```

Selecting the authentication backend¶

When a user logs in, the user's ID and the backend that was used for authentication are saved in the user's session. This allows the same authentication backend to fetch the user's details on a future request. The authentication backend to save in the session is selected as follows:

21. Use the value of the optional **backend** argument, if provided.
22. Use the value of the **user.backend** attribute, if present. This allows pairing **authenticate()** and **login()**: **authenticate()** sets the **user.backend** attribute on the user object it returns.
23. Use the **backend** in **AUTHENTICATION_BACKENDS**, if there is only one.
24. Otherwise, raise an exception.

In cases 1 and 2, the value of the **backend** argument or the **user.backend** attribute should be a dotted import path string (like that found in **AUTHENTICATION_BACKENDS**), not the actual backend class.

How to log a user out¶

logout(*request*)¶

To log out a user who has been logged in via **django.contrib.auth.login()**, use **django.contrib.auth.logout()** within your view. It takes an **HttpRequest** object and has no return value. Example:

```
from django.contrib.auth import logout

def logout_view(request):
    logout(request)
    # Redirect to a success page.
```

Note that **logout()** doesn't throw any errors if the user wasn't logged in.

When you call **logout()**, the session data for the current request is completely cleaned out. All existing data is removed. This is to prevent another person from using the same web browser to log in and have access to the previous user's session data. If you want to put anything into the session that will be available to the user immediately after logging out, do that *after* calling **django.contrib.auth.logout()**.

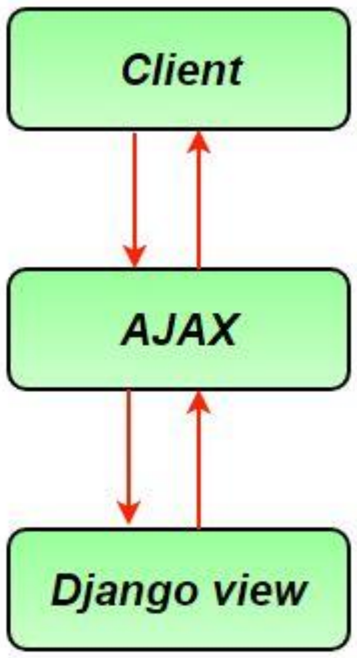
References: https://docs.djangoproject.com/en/4.0/topics/auth/
https://docs.djangoproject.com/en/4.0/topics/auth/default/
https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Authentication

Activity 9

Aim: Create admin panel using Django using AJAX.
Learning outcome: Able to understand basic how to implement AJAX .
Duration: 4 hour
List of Hardware/Software requirements:

1. Django
2. Python

Code/Program/Procedure (with comments):
Handling Ajax request in Django



Introduction

This tutorial explains how to carry out an ajax request in the Django web framework. We will create a simple post-liking app as a part of the example.

Glossary

- Project Initialization
- Create models
- Create views
- Write URLs
- Carry out a request with Jquery Ajax.
- Register models to admin and add some posts.

Implementation:

1. Initiate the Django Project – Here I am assuming that you are done with Django Installation.

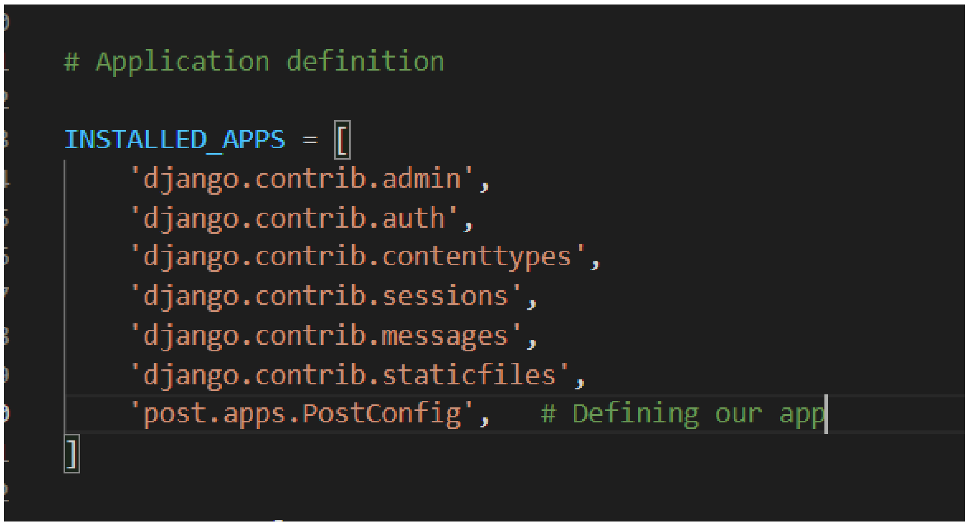
- To Create a Django Project execute:

```
$ django-admin startproject django_example
```

- After creating a project we need to create a Django app. To create an app say "post" execute the following:

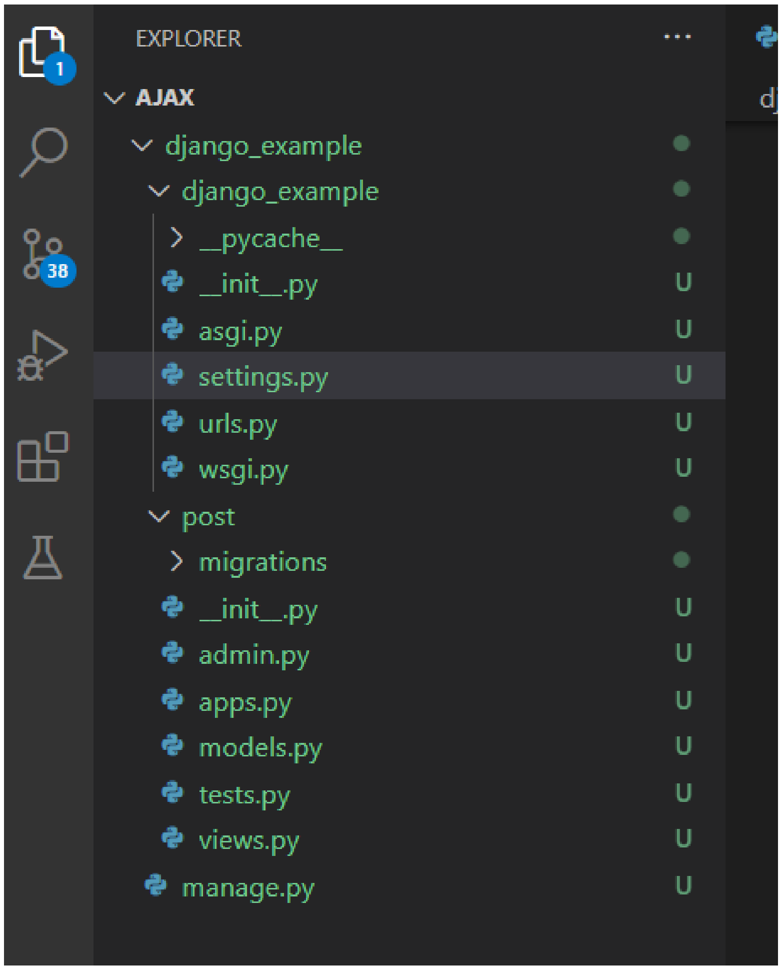
```
$ cd django_example
$ python manage.py startapp post
```

- Go to django_example/settings.py add the post-app



```
# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'post.apps.PostConfig',    # Defining our app
]
```

- Now you will have files something like this:



```
EXPLORER
AJAX
  django_example
    django_example
      __pycache__
      __init__.py        U
      asgi.py            U
      settings.py        U
      urls.py            U
      wsgi.py            U
    post
      migrations
      __init__.py        U
      admin.py           U
      apps.py            U
      models.py          U
      tests.py           U
      views.py           U
    manage.py            U
```

2. Create models: To create models, go to the post directory and open models.py.

- In models.py. First, create a post table. To create a post table you'll need to write:

```
class Post(models.Model):
    post_heading = models.CharField(max_length=200)
    post_text = models.TextField()
        def __unicode__(self):      # If python2 use __str__ if python3
            return unicode(self.post_heading)
```

- Then In models.py, create like a table. To create like a table you'll need to write:

```
class Like(models.Model):
    post = models.ForeignKey(Post, on_delete = models.CASCADE)
```

- Make Migration and migrate step:

```
$ python manage.py makemigrations
$ python manage.py migrate
```

After completing these steps, we have our database tables ready to use.

3. Create Views:

To create views, we need to go to the post directory and open views.py

- First, import Previously created Models and HTTP response

```
from .models import Post, Like
from django.http import HttpResponse
```

- Create an index view to render all the posts. Code sample:

```
def index(request):
    posts = Post.objects.all()  # Getting all the posts from database
    return render(request, 'post/index.html', { 'posts': posts })
```

- Create like posts view to like a post. This view will be called when we will hit a "like button". Code sample:

```
def likePost(request):
    if request.method == 'GET':
        post_id = request.GET['post_id']
        likedpost = Post.objects.get(pk=post_id) #getting the liked posts
        m = Like(post=likedpost) # Creating Like Object
        m.save()  # saving it to store in database
        return HttpResponse("Success!") # Sending an success response
    else:
        return HttpResponse("Request method is not a GET")
```

Once our view gets created we will move to write a template and jQuery to perform an ajax request.

4. Create URLs:

To create URLs, open django_example/urls.py. Your django_example/urls.py should look something like this:

```
from django.conf.urls import include, url
from django.contrib import admin
urlpatterns = [
     url(r'^admin/', admin.site.urls),
     url(r'^', include('post.urls')),   # To make post app available at /
   ]
```

To create URLs, create file post/urls.py. Your post/urls.py should look something like this:

```
from django.conf.urls import url
from . import views
urlpatterns = [
        url(r'^$', views.index, name='index'),  # index view at /
        url(r'^likepost/$', views.likePost, name='likepost'),   # likepost view at /likepost
   ]
```

5. Making templates and carrying out ajax requests:

- Create a file post/templates/post/index.html. Code sample:

```
<!DOCTYPE html>
<html>
<head>
    <title>Like Post App</title>
</head>
<body>
    <p id="message"></p>
    {% for post in posts %}
    <h3>{{ forloop.counter }}) {{ post.post_heading }}</h3>
    <p>{{ post.post_text }} </p>
    <a class="likebutton" id="like{{post.id}}" href="#" data-catid="{{ post.id }}">Like</a>
    {% endfor %}
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.0/jquery.min.js"></script>
    <script type="text/javascript">
    $('.likebutton').click(function(){
    var catid;
    catid = $(this).attr("data-catid");
    $.ajax(
    {
        type:"GET",
        url: "/likepost",
        data:{
                 post_id: catid
        },
        success: function( data )
        {
            $( '#like'+ catid ).remove();
            $( '#message' ).text(data);
        }
     })
});
</script>
</body>
</html>
```

Basically, what we are doing here is - we are making an ajax get request -> /likepost?post_id=<id_of_liked_post>

6. To Register models to admin and add some posts:

- Open post/admin.py.
- Import Models to admin.py.

```
from .models import Post, Like
```

- Register your models:

```
admin.site.register(Post)
admin.site.register(Like)
```

Now add some posts using the Django default admin portal. Visit http://localhost:8000/ to view.

Output/Results snippet:

Reference : https://www.geeksforgeeks.org/handling-ajax-request-in-django/
https://github.com/saganshul/django_tutorials/tree/master/django_ajax/django_example

Activity 10

Aim: Perform crud operations in Django app using AJAX at single page
Learning outcome: Able to understand basic computer network technology.
Duration: 5 hour
List of Hardware/Software requirements:

1. Django
2. JQuery
3. Python

Code/Program/Procedure (with comments):

Let's see the example of Python Django Ajax CRUD Operations. Ajax is a way of making web development more dynamic. Using Ajax we can load data from the server without reloading the web pages. AJAX stands for Asynchronous Javascript and XML.
In this tutorial, you'll learn how to do CRUD operations using Django, Ajax Form Submission and JSON. We will be using the Class Based Views Functions. CRUD operation is the most basic operation that we perform on databases. CRUD stands for Create, Read, Update, Delete. We'll take an example of a User Management System where we will add, update and delete users and its detail.

Basic Configuration
In this tutorial, we are going to use JQuery for implementing Ajax requests.

base.html

```
{% load static %}
<!doctype html>
```

```
<html>
  <head>
    <meta charset="utf-8">
    <title>{% block title %}Title{% endblock title %}</title>
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u"
crossorigin="anonymous">
    {% block stylesheet %}{% endblock stylesheet %}
  </head>
  <body>
    <main>
    {% block content %}
    {% endblock content %}
    </main>
    <script src="https://code.jquery.com/jquery-3.1.0.min.js"></script>
    <script src='https://code.jquery.com/jquery-3.2.1.min.js'></script>
    <script src='https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js'></script>
    {% block javascript %}
    {% endblock javascript%}
  </body>
</html>
```

Example – Django Ajax CRUD Operations
I will be taking a simple example of User Detail Add Application. I will be working in an app called crud_ajax. For CRUD operation we will be using CrudUser Model.

# models.py

```
from django.db import models

class CrudUser(models.Model):
    name = models.CharField(max_length=30, blank=True)
    address = models.CharField(max_length=100, blank=True)
    age = models.IntegerField(blank=True, null=True)
```

Listing User Details Operation
Let's get started by initially loading all user details. For that, we are going to use ListView.

# Views.py

```
from django.views.generic import ListView
from .models import CrudUser

class CrudView(ListView):
    model = CrudUser
    template_name = 'crud_ajax/crud.html'
    context_object_name = 'users'


# urls.py

from django.urls import path
from crud_ajax import  views

urlpatterns = [
    path('crud/',  views.CrudView.as_view(), name='crud_ajax'),
]
```

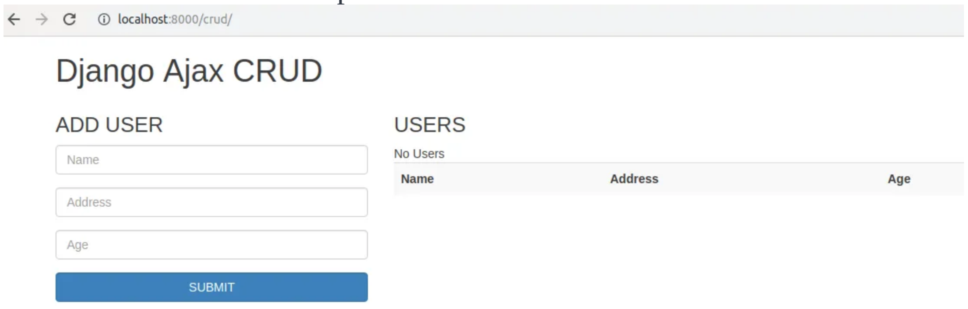
Crud.html

```
{% extends 'base.html' %}
{% load static %}

{% block title %}Django Ajax CRUD{% endblock %}

{% block content %}
<div class="container">
    <h1>Django Ajax CRUD</h1>
    <div class="row">
      <div class="col-md-4 ">
        <h3>ADD USER</h3>
        <form id="addUser" action="">
          <div class="form-group">
            <input class="form-control" type="text" name="name" placeholder="Name" required>
          </div>
          <div class="form-group">
            <input class="form-control" type="text" name="address" placeholder="Address" required>
          </div>
          <div class="form-group">
            <input class="form-control" type="number" name="age" min="10" max="100" placeholder="Age"
required>
```

```
          </div>
          <button class="btn btn-primary form-control" type="submit">SUBMIT</button>
        </form>
      </div>
      <div class="col-md-8">
        <h3>USERS</h3>
        <table id="userTable" class="table table-striped">
          <tr>
            <th>Name</th>
            <th>Address</th>
            <th colspan="3">Age</th>
          </tr>
          {% if users %}
          {% for user in users %}
          <tr id="user-{{user.id}}">
              <td class="userName userData" name="name">{{user.name}}</td>
              <td class="userAddress userData" name="address">{{user.address}}</td>
              <td class="userAge userData" name="age">{{user.age}}</td>
              <td align="center">
                  <button class="btn btn-success form-control" onClick="editUser({{user.id}})" data-
toggle="modal" data-target="#myModal")">EDIT</button>
              </td>
              <td align="center">
                  <button class="btn btn-danger form-control"
onClick="deleteUser({{user.id}})">DELETE</button>
              </td>
          </tr>
          {% endfor %}
          {% else %}
            No Users
          {% endif %}
        </table>
      </div>
    </div>
  </div>
  <!-- Modal -->
  <div class="modal fade" id="myModal" tabindex="-1" role="dialog" aria-labelledby="myModalLabel">
    <div class="modal-dialog" role="document">
      <div class="modal-content">
        <div class="modal-header">
          <button type="button" class="close" data-dismiss="modal" aria-label="Close"><span aria-
hidden="true">×</span></button>
          <h4 class="modal-title" id="myModalLabel">Update User</h4>
        </div>
        <form id="updateUser" action="">
        <div class="modal-body">
```

```
            <input class="form-control" id="form-id" type="hidden" name="formId"/>
            <label for="name">Name</label>
            <input class="form-control" id="form-name" type="text" name="formName"/>
            <label for="address">Address</label>
            <input class="form-control" id="form-address" type="text" name="formAddress"/>
            <label for="age">Age</label>
            <input class="form-control" id="form-age" type="number" name="formAge" min=10 max=100/>
          </div>
          <div class="modal-footer">
            <button type="submit" class="btn btn-primary" >Save changes</button>
            <button type="button" class="btn btn-default" data-dismiss="modal">Close</button>
          </div>
          </form>
        </div>
      </div>
    </div>
{% endblock %}
{% block javascript %}
{% endblock javascript %}
```

In our HTML code, you will find the Add User form and modal for update user details, we will be going to use it later. So far our template should look like this:



Now it’s time to go with Ajax Calls.

Create and Read User Django Ajax
We will not be using any forms.py instead, we have hardcoded HTML form directly in the template. As you can see in our crud.html file.

Let’s create a view to handle the Ajax request.

```
# Views.py

from .models import CrudUser
from django.views.generic import View
from django.http import JsonResponse
```

```
class CreateCrudUser(View):
    def  get(self, request):
        name1 = request.GET.get('name', None)
        address1 = request.GET.get('address', None)
        age1 = request.GET.get('age', None)

        obj = CrudUser.objects.create(
            name = name1,
            address = address1,
            age = age1
        )

        user = {'id':obj.id,'name':obj.name,'address':obj.address,'age':obj.age}

        data = {
            'user': user
        }
        return JsonResponse(data)
```

Note– We are not rendering the template, we are just returning JsonResponse and we are also using Ajax Get Method.

We are getting the form data using request.GET.get(address, None), address is the form’s input name attribute. Then we are creating a user and sending back the user which we have created so to display it on the web page.

```
# urls.py
from django.urls import path
from crud_ajax import views

urlpatterns = [
    path('crud/',  views.CrudView.as_view(), name='crud_ajax'),
    path('ajax/crud/create/',  views.CreateCrudUser.as_view(), name='crud_ajax_create'),
]
```

Form in Crud.html for adding Users.

```
<form id="addUser" action="">
          <div class="form-group">
```

```
      <input class="form-control" type="text" name="name" placeholder="Name" required>
    </div>
    <div class="form-group">
      <input class="form-control" type="text" name="address" placeholder="Address" required>
    </div>
    <div class="form-group">
      <input class="form-control" type="number" name="age" min="10" max="100" placeholder="Age"
required>
    </div>
    <button class="btn btn-primary form-control" type="submit">SUBMIT</button>
</form>
```

Now let again add javascript and jquery ajax request in Crud.html which we have created while listing the users

Crud.html

```
...

{% block javascript %}
<script>
// Create Django Ajax Call
$("form#addUser").submit(function() {
    var nameInput = $('input[name="name"]').val().trim();
    var addressInput = $('input[name="address"]').val().trim();
    var ageInput = $('input[name="age"]').val().trim();
    if (nameInput && addressInput && ageInput) {
        // Create Ajax Call
        $.ajax({
            url: '{% url "crud_ajax_create" %}',
            data: {
                'name': nameInput,
                'address': addressInput,
                'age': ageInput
            },
            dataType: 'json',
            success: function (data) {
                if (data.user) {
                  appendToUsrTable(data.user);
                }
            }
        });
      } else {
        alert("All fields must have a valid value.");
    }
    $('form#addUser').trigger("reset");
    return false;
});
function appendToUsrTable(user) {
  $("#userTable > tbody:last-child").append(`
        <tr id="user-${user.id}">
            <td class="userName" name="name">${user.name}</td>
            '<td class="userAddress" name="address">${user.address}</td>
            '<td class="userAge" name="age">${user.age}</td>
            '<td align="center">
                <button class="btn btn-success form-control" onClick="editUser(${user.id})" data-
toggle="modal" data-target="#myModal")">EDIT</button>
            </td>
            <td align="center">
                <button class="btn btn-danger form-control"
onClick="deleteUser(${user.id})">DELETE</button>
            </td>
        </tr>
    `);
}
</script>
{% endblock javascript %}
```

The working of the above JQuery is like that when you submit the form id='addUser', It will take the values of the input fields and ajax call will be gone to the server.

```
$.ajax({
            url: '{% url "crud_ajax_create" %}',
            data: {
                'name': nameInput,
                'address': addressInput,
                'age': ageInput
            },
            dataType: 'json',
            success: function (data) {
                if (data.user) {
                  appendToUsrTable(data.user);
                }
            }
});
```

Let us understand what the above ajax request commanding to the browser.

URL – where resource is located:
url: '/ajax/crud/create/',
Data Type – Type of Data we want in return:

```
dataType: 'json',
```

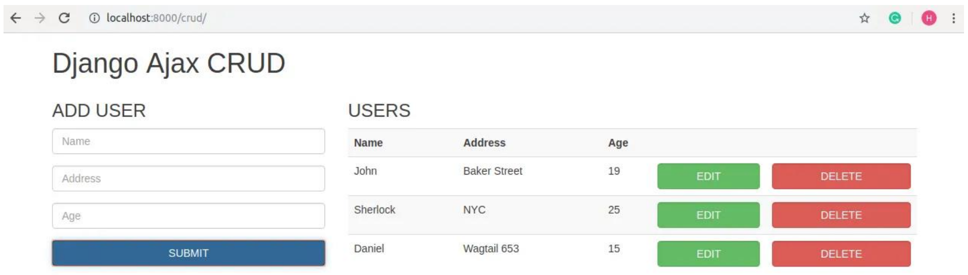
Return Data – [Dictionary]() of data what we want to send:

```
data: {
        'name': nameInput,
        'address': addressInput,
        'age': ageInput
   },
```

Success Response – Once we receive response from ajax, execute the code:

```
success: function (data) {
        if (data.user) {
          appendToUsrTable(data.user);
        }
      }
```

When the user fills the form and submits it. Submitted data goes to the server and after validation into the database and the function appendToUsrTable is called. This appendToUsrTable [function]() is just a jQuery where it appends the user which we have added using User Add Form. When a user is added it will look like this:


localhost:8000/crud/

Django Ajax CRUD

ADD USER

Name
Address
Age
SUBMIT

USERS

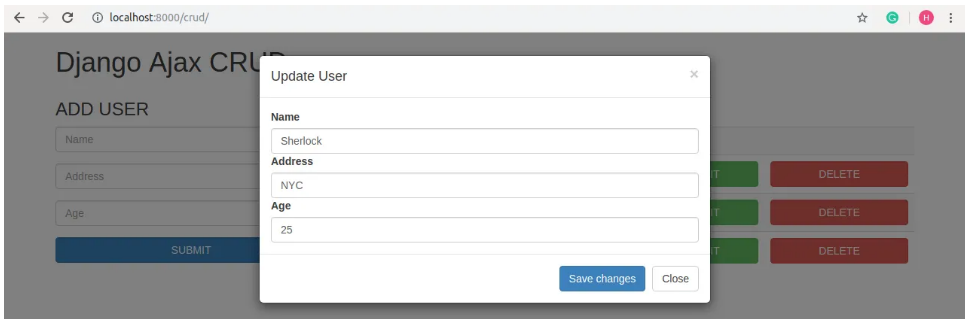
Name	Address	Age		
John	Baker Street	19	EDIT	DELETE
Sherlock	NYC	25	EDIT	DELETE
Daniel	Wagtail 653	15	EDIT	DELETE

Update User Django Ajax

```
# Views.py

class UpdateCrudUser(View):
    def  get(self, request):
        id1 = request.GET.get('id', None)
        name1 = request.GET.get('name', None)
        address1 = request.GET.get('address', None)
        age1 = request.GET.get('age', None)

        obj = CrudUser.objects.get(id=id1)
        obj.name = name1
        obj.address = address1
        obj.age = age1
        obj.save()

        user = {'id':obj.id,'name':obj.name,'address':obj.address,'age':obj.age}

        data = {
            'user': user
        }
        return JsonResponse(data)# urls.py

from django.urls import path
from crud_ajaximport  views

urlpatterns = [
    path('crud/',  views.CrudView.as_view(), name='crud_ajax'),
    path('ajax/crud/create/',  views.CreateCrudUser.as_view(), name='crud_ajax_create'),
    path('ajax/crud/update/',  views.UpdateCrudUser.as_view(), name='crud_ajax_update'),
]
```

crud.html

```
...
// Create Django Ajax Call
$("form#updateUser").submit(function() {
    var idInput = $('input[name="formId"]').val().trim();
    var nameInput = $('input[name="formName"]').val().trim();
    var addressInput = $('input[name="formAddress"]').val().trim();
    var ageInput = $('input[name="formAge"]').val().trim();
    if (nameInput && addressInput && ageInput) {
        // Create Ajax Call
        $.ajax({
            url: '{% url "crud_ajax_update" %}',
            data: {
                'id': idInput,
                'name': nameInput,
                'address': addressInput,
                'age': ageInput
            },
            dataType: 'json',
            success: function (data) {
                if (data.user) {
                  updateToUserTabel(data.user);
                }
            }
        });
        } else {
          alert("All fields must have a valid value.");
    }
    $('form#updateUser').trigger("reset");
    $('#myModal').modal('hide');
    return false;
});

// Update Django Ajax Call
function editUser(id) {
  if (id) {
    tr_id = "#user-" + id;
    name = $(tr_id).find(".userName").text();
    address = $(tr_id).find(".userAddress").text();
    age = $(tr_id).find(".userAge").text();
    $('#form-id').val(id);
    $('#form-name').val(name);
    $('#form-address').val(address);
    $('#form-age').val(age);
  }
}
function updateToUserTabel(user){
    $("#userTable #user-" + user.id).children(".userData").each(function() {
        var attr = $(this).attr("name");
        if (attr == "name") {
          $(this).text(user.name);
        } else if (attr == "address") {
          $(this).text(user.address);
        } else {
          $(this).text(user.age);
        }
      });
}
```
Update user is similar to Create User. In views.py instead of create query, we are using an update query. In Ajax Request call we are sending one more parameter id to identify the row in the database.
We are also using two jQuery function – editUser and updateToUserTabel
editUser – is used for updating the form id="updateUser" which is inside the modal. And updateToUserTabel is used in the success of Ajax request for updating the user detail which we have updated.



Name
Sherlock
Address
NYC
Age
25

Delete User Django Ajax

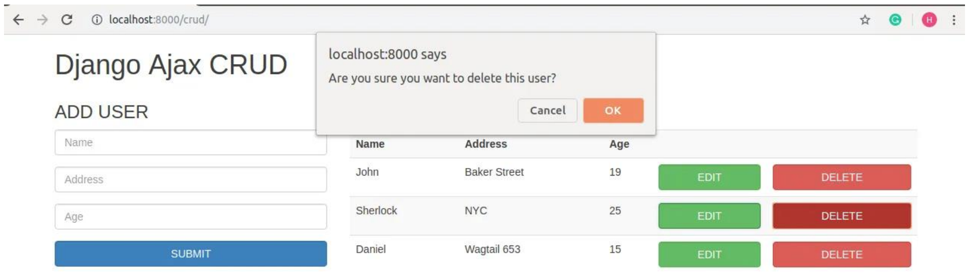
```
# Views.py

class DeleteCrudUser(View):
    def  get(self, request):
        id1 = request.GET.get('id', None)
        CrudUser.objects.get(id=id1).delete()
        data = {
            'deleted': True
        }
        return JsonResponse(data)# urls.py

from django.urls import path
from crud_ajax import views

urlpatterns = [
    path('crud/',  views.CrudView.as_view(), name='crud_ajax'),
    path('ajax/crud/create/',  views.CreateCrudUser.as_view(), name='crud_ajax_create'),
    path('ajax/crud/update/',  views.UpdateCrudUser.as_view(), name='crud_ajax_update'),
    path('ajax/crud/delete/',  views.DeleteCrudUser.as_view(), name='crud_ajax_delete'),
]
```

Crud.html

```
// Delete Django Ajax Call
function deleteUser(id) {
  var action = confirm("Are you sure you want to delete this user?");
  if (action != false) {
    $.ajax({
        url: '{% url "crud_ajax_delete" %}',
        data: {
            'id': id,
        },
        dataType: 'json',
        success: function (data) {
            if (data.deleted) {
              $("#userTable #user-" + id).remove();
            }
        }
    });
  }
```

}
Deleting User using Ajax Django is one of the easiest ways. In ajax request, we are sending an id of the object which we want to delete to view. In View, delete query is executed and deleted flag is sent in Response. In success, we directly write the jQuery function to remove the section which we want to remove from the table using id dynamically.

Output/Results snippet:



localhost:8000 says
Are you sure you want to delete this user?

Name	Address	Age		
John	Baker Street	19	EDIT	DELETE
Sherlock	NYC	25	EDIT	DELETE
Daniel	Wagtail 653	15	EDIT	DELETE

Reference: https://studygyaan.com/django/how-to-execute-crud-using-django-ajax-and-json
Github link for Django AJAX CRUD- https://github.com/studygyaan/How-To-Execute-CRUD-Using-Django-Ajax-and-JSON

Activity 11

Aim: Create Django application which sends email to any recipient.
Learning outcome: Learn how to send email using Django.
Duration: 3 hour
List of Hardware/Software requirements:

1. Python
2. Django
3. VS Code

Code/Program/Procedure (with comments):

Step 1: Create a simple Django App

Step 2: In your Django app’s settings.py file, enter the following,

```
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_USE_TLS = True
EMAIL_PORT = 587
EMAIL_HOST_USER = #sender's email-id
EMAIL_HOST_PASSWORD = #password associated with above email-id
```

#In the above code, EMAIL_HOST_USER = ‘xabc6457@gmail.com’ and EMAIL_HOST_PASSWORD = ‘xyz123abc@’ are the lines where you need to add the sender’s mail id and password. xabc6457@gmail.com and xyz123abc@ are just examples.

Step 3: Now to use this in our application, move to views.py and add these lines at the top section as below.

```
from django.conf import settings
from django.core.mail import send_mail
```

Step 4: Generally, emails are sent to the users who signup right? So, in the signup view function, add these lines.

```
subject = 'welcome to Hello world'
message = f'Hi {user.username}, thank you for registering in geeksforgeeks.'
email_from = settings.EMAIL_HOST_USER
recipient_list = [user.email, ]
send_mail( subject, message, email_from, recipient_list )
```

In above code ,
#subject refers to the email subject.
#message refers to the email message, the body of the email.
#email_from refers to the sender’s details.This takes the EMAIL_HOST_USER from settings.py file, where you added those lines of code earlier.
#recipient_list is the list of recipients to whom the mail has to be sent that is, whoever registers to your application they receive the email.
#send_mail is an inbuilt Django function that takes subject, message, email_from, and recipient’s list as arguments, this is responsible to send emails.

Output/Results snippet:

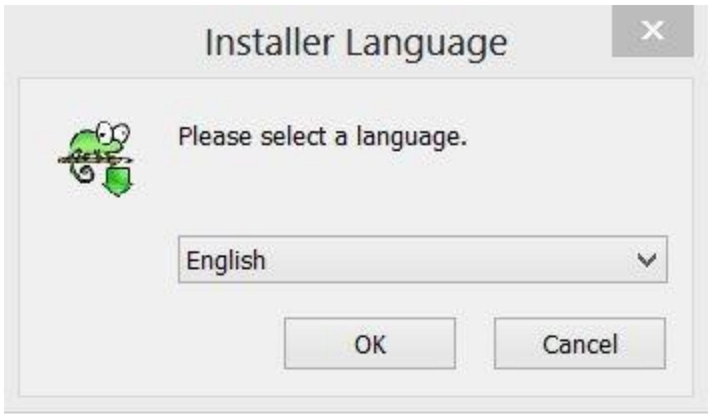
Now, register any user to your application, and they will receive mail from the email account you had mentioned.

Reference : https://www.geeksforgeeks.org/setup-sending-email-in-django-project/

Annexure

1. Installing Notepad++

a. Download Notepad++. You can download Notepad++ by clicking here. Click on the "Download" button, and the program will begin downloading to your computer.
   Click here Download 32-bit x86
   Click here Download 64-bit x64
b. Double-click on the downloaded installer to start the installation. It is a wizard-driven installation.
c. Let's start……………
   Select language.
d. Click next and follow instructions


Installer Language
Please select a language.
English
OK
Cancel