



**MGM's College of Engineering and Technology
Kamothe, Navi Mumbai-410209**

Department of Computer Engineering

Laboratory Manual

Skill Base Lab Course - Python Programming

Course Code: CSL405

Second Year

Semester- IV

Academic Year 2021-22

CONTENTS

S. No.	TOPIC	PAGE NO.
1	<i>Vision and Mission of College and Department</i>	3
2	<i>Teaching Scheme, Examination Scheme and University Syllabus</i>	4
3	<i>Lab Objectives and Lab Outcomes (COs)</i>	5
4	<i>Program Specific Outcomes (PSOs) & Lab Outcomes (POs)</i>	6
5	<i>CO & PO Mapping</i>	8
6	<i>List of Experiments</i>	9
7	<i>Experiment Plan</i>	10
8	<i>Experiment 01: Introduction to Python & basic programming concepts in Python. (Downloading and Installation of Python)</i>	11
9	<i>Experiment 02: Exploring basics of python like data types (strings, list, array, dictionaries, set, tuples) and control statements</i>	19
10	<i>Experiment 03: Creating functions, classes and objects using python. Demonstrate exception handling and inheritance.</i>	25
11	<i>Experiment No. 4: Exploring Files and directories 4.a. Python program to append data to existing file and then display the entire file 4.b. Python program to count number of lines, words and characters in a file 4.c. Python program to display file available in current directory</i>	30
12	<i>Experiment No 5: Creating GUI with python containing widgets such as labels, textbox, radio, checkboxes and custom dialog boxes</i>	34
13	<i>Experiment No 6: Menu driven program for data structure using built in function for link list, stack and queue.</i>	38
14	<i>Experiment No 7: Program to demonstrate CRUD (create, read, update and delete) operations on database (SQLite/ MySQL) using python.</i>	41
15	<i>Experiment No 8: Creation of simple socket for basic information exchange between server and client.</i>	49
16	<i>Experiment No 9: Creating web application using Django web framework to demonstrate functionality of user login and registration (also validating user detail using regular expression).</i>	53
17	<i>Experiment No 10: Programs on Threading using python.</i>	62
18	<i>Experiment No 11 Exploring basics of NumPy Methods.</i>	67
19	<i>Experiment No 12: Program to send email and read content of URL.</i>	77



MGM's College of Engineering and Technology

Vision:

To become one of the outstanding Engineering Institutes in India by providing a conducive and vibrant environment to achieve excellence in the field of Technology.

Mission:

To empower the aspiring professional students to be prudent enough to explore the world of technology and mould them to be proficient to reach the pinnacle of success in the competitive global economy.

Department of Computer Engineering

Vision:

To motivate and empower the students of Computer Engineering to become globally competent citizens with ethics to serve and lead the society.

To provide a stimulating educational environment for computer engineering graduates to face tomorrow's challenges and to inculcate social responsibility in them.

Mission:

- 1. To provide excellent academic environment by adopting an innovative teaching techniques through well developed curriculum.*
- 2. To foster a self learning atmosphere for students to provide ethical solutions for societal challenges.*
- 3. To establish Center of Excellence in various domains of Computer Engineering and promote active research and development.*
- 4. To enhance the competency of the faculty in the latest technology through continuous development programs.*
- 5. To foster networking with alumni and industries.*

Study and Evaluation Scheme

<i>Course Code</i>	<i>Course Name</i>	<i>Teaching Scheme</i>			<i>Credits Assigned</i>			
<i>CSL405</i>	<i>Skill Base Lab Course: Python Programming</i>	<i>Theory</i>	<i>Practical</i>	<i>Tutorial</i>	<i>Theory</i>	<i>Practical</i>	<i>Tutorial</i>	<i>Total</i>
		2	02	--	--	2	--	2

<i>Course Code</i>	<i>Course Name</i>	<i>Examination Scheme</i>		
<i>CSL405</i>	<i>Skill Base Lab Course: Python Programming</i>	<i>Term Work</i>	<i>Oral</i>	<i>Total</i>
		25	--	25

Term Work	
1	<i>Term work should consist of 12 experiments.</i>
2	<i>Journal must include at least 2 assignments</i>
3	<i>Mini Project based on the content of the syllabus (Group of 2-3 students)</i>
4	<i>The final certification and acceptance of term work ensures that satisfactory performance of laboratory work and minimum passing marks in term work.</i>
5	<i>Total 25 Marks (Journal: 10-marks, Attendance: 05-marks, and Mini Project: 10-marks)</i>

Lab Objectives & Lab Outcome

Course Objectives:

1.	<i>Basics of Python programming</i>
2.	<i>Decision Making, Data structure and Functions in Python</i>
3.	<i>Object Oriented Programming using Python</i>
4.	<i>Web framework for developing</i>

Course Outcomes:

CO1	<i>To understand basic concepts in python.</i>
CO2	<i>To explore contents of files, directories and text processing with python</i>
CO3	<i>To develop program for data structure using built in functions in python.</i>
CO4	<i>To explore Django web framework for developing python-based web application.</i>
CO5	<i>To understand Multi-threading concepts using python.</i>

Program Specific Outcomes (PSOs)

The Computer Engineering graduates will be able to

PSO 1	<i>Acquire skills to design, analyses and develop algorithms and implement them using high-level programming languages</i>
PSO 2	<i>Contribute their engineering skills in computing and information engineering domains like network design and administration, database design and knowledge engineering.</i>
PSO 3	<i>Develop strong skills in systematic planning, developing, testing implementing and providing IT solutions for different domains which helps in the betterment of life.</i>

Program Outcomes (POs)

Engineering Graduates will be able to:

PO1) Engineering knowledge: *Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.*

PO2) Problem analysis: *Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.*

PO3) Design/development of solutions: *Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.*

PO4) Conduct investigations of complex problems: *Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.*

PO5) Modern tool usage: *Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.*

PO6) The engineer and society: *Apply reasoning informed by the contextual knowledge to assess societal ,health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.*

PO7) Environment and sustainability: *Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.*

PO8) Ethics: *Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.*

PO9) Individual and team work: *Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.*

PO10) Communication: *Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.*

PO11) Project management and finance: *Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.*

PO12) Life-long learning: *Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.*

MGMCEET

Mapping Lab Outcomes (CO) - Program Outcomes (PO)

Subject Weight	Course Outcomes		Contribution to Program outcomes											
			P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12
PRATICAL	CO1	To understand basic concepts in python.	1	1	2					1	2	1	1	1
	CO2	To explore contents of files, directories and text processing with python		3	1		1				1	1	2	1
	CO3	To develop program for data structure using built in functions in python.		1	1		1				1	1	2	3
	CO4	To explore Django web framework for developing python-based web application.		1			1			1	1	1	2	3
	CO5	To understand Multi-threading concepts using python.		1	1		1			1	1	1	2	1

List of Experiments

Sr. No.	Experiments Name
1	<i>Experiment 01: Introduction to Python & basic programming concepts in Python. (Downloading and Installation of Python)</i>
2	<i>Experiment 02: Exploring basics of python like data types (strings, list, array, dictionaries, set, tuples) and control statements</i>
3	<i>Experiment 03: Creating functions, classes and objects using python. Demonstrate exception handling and inheritance.</i>
4	<i>Experiment No. 4: Exploring Files and directories 4.a. Python program to append data to existing file and then display the entire file 4.b. Python program to count number of lines, words and characters in a file 4.c. Python program to display file available in current directory</i>
5	<i>Experiment No 5: Creating GUI with python containing widgets such as labels, textbox, radio, checkboxes and custom dialog boxes</i>
6	<i>Experiment No 6: Menu driven program for data structure using built in function for link list, stack and queue.</i>
7	<i>Experiment No 7: Program to demonstrate CRUD (create, read, update and delete) operations on database (SQLite/ MySQL) using python.</i>
8	<i>Experiment No 8: Creation of simple socket for basic information exchange between server and client.</i>
9	<i>Experiment No 9: Creating web application using Django web framework to demonstrate functionality of user login and registration (also validating user detail using regular expression).</i>
10	<i>Experiment No 10: Programs on Threading using python.</i>
11	<i>Experiment No 11 Exploring basics of NumPy Methods.</i>
12	<i>Experiment No 12: Program to send email and read content of URL.</i>

Experiment Plan

Module No.	Week No.	Experiments Name	Course Outcome
1	W1	Experiment 01: Introduction to Python & basic programming concepts in Python. (Downloading and Installation of Python)	
2	W2	Experiment 02: Exploring basics of python like data types (strings, list, array, dictionaries, set, tuples) and control statements	CO1
3	W3	Experiment 03: Creating functions, classes and objects using python. Demonstrate exception handling and inheritance.	CO5
4	W4	Experiment No. 4: Exploring Files and directories 4.a. Python program to append data to existing file and then display the entire file 4.b. Python program to count number of lines, words and characters in a file 4.c. Python program to display file available in current directory	CO5
5	W5	Experiment No 5: Creating GUI with python containing widgets such as labels, textbox, radio, checkboxes and custom dialog boxes	CO5
6	W6	Experiment No 6: Menu driven program for data structure using built in function for link list, stack and queue.	CO5
7	W7	Experiment No 7: Program to demonstrate CRUD (create, read, update and delete) operations on database (SQLite/MySQL) using python.	CO5
8	W8	Experiment No 8: Creation of simple socket for basic information exchange between server and client.	CO5
9	W9	Experiment No 9: Creating web application using Django web framework to demonstrate functionality of user login and registration (also validating user detail using regular expression).	CO3
10	W10	Experiment No 10: Programs on Threading using python.	CO3
11	W11	Experiment No 11 Exploring basics of NumPy Methods.	CO4
12	W12	Experiment No 12: Program to send email and read content of URL.	CO4

Experiment No. 1

Aim: *Introduction to Python & basic programming concepts in Python.(Downloading and Installation of Python)*

Software Required: *Python3.9 / Online Editor*

Theory:

❖ **Fundamentals of Python**

- *Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language.*
- *It was created by Guido van Rossum during 1985- 1990. It's Named After Monty Python*
- *Despite all the reptile icons in the Python world, the truth is that Python creator Guido van Rossum named it after the BBC comedy series Monty Python's Flying Circus. He is a big fan of Monty Python.*
- **Python** *is a high-level, interpreted, interactive and object-oriented scripting language.*
- *Python is designed to be highly readable.*
- *It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.*
- **Python** *is a MUST for students and working professionals to become a great Software Engineer specially when they are working in Web Development Domain.*

❖ **Characteristics of Python**

- *It supports functional and structured programming methods as well as OOP.*
- *It can be used as a scripting language or can be compiled.*
- *It provides very high-level dynamic data types and supports dynamic type checking.*
- *It supports automatic garbage collection.*
- *It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.*

❖ **Python used in the field of**

- *Systems Programming*
- *GUIs*
- *Internet Scripting*
- *Component Integration*
- *Database Programming*
- *Numeric and Scientific Programming*
- *Gaming, Images, Serial Ports, XML, Robots, and More*

❖ **Technical strength of Python**

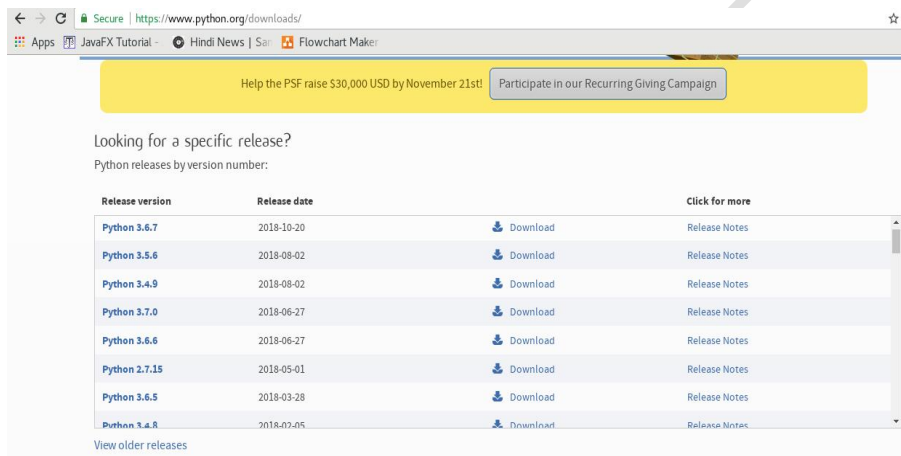
- *It's Object-Oriented*

- *It's Free*
- *It's Portable*
- *It's Powerful*
- *It's Mixable*
- *It's Easy to Use*
- *It's Easy to Learn*

Python Installation

❖ How to Install Python on Windows (Environment Set-up)

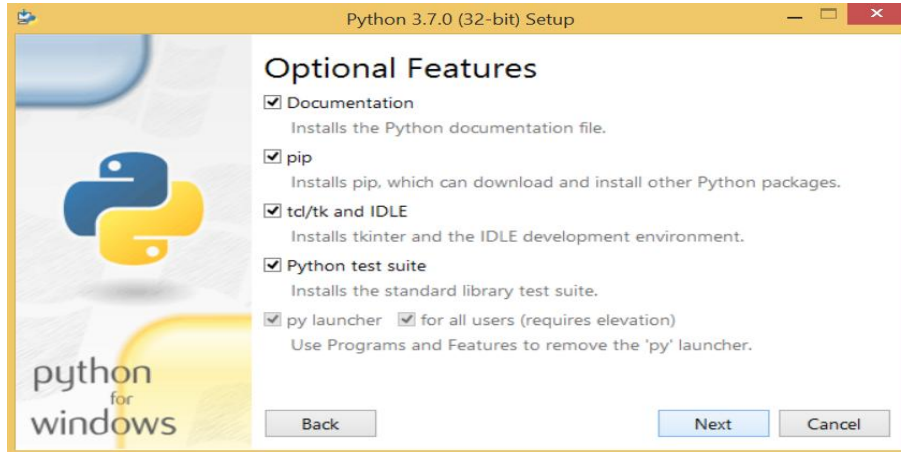
1. Visit the link <https://www.python.org/downloads/> to download the latest release of Python.



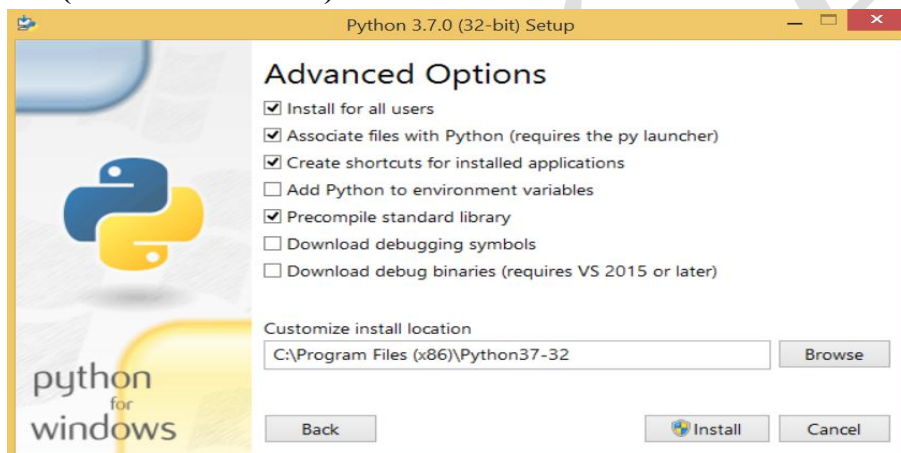
2. Double-click the executable file, which is downloaded; the following window will open. Select **Customize installation** and proceed.



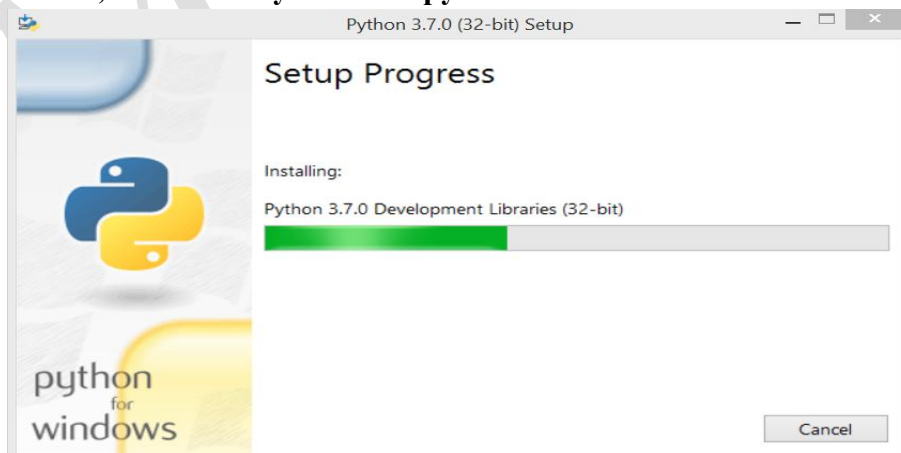
3. The following window shows all the optional features. All the features need to be installed and are checked by default; we need to click next to continue.



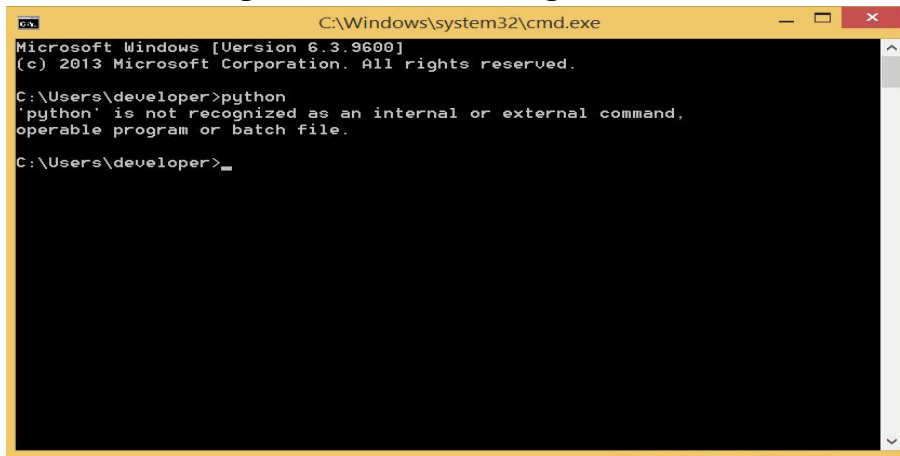
4. The following window shows a list of advanced options. Check all the options which you want to install and click next. Here, we must notice that the first check-box (install for all users) must be checked.



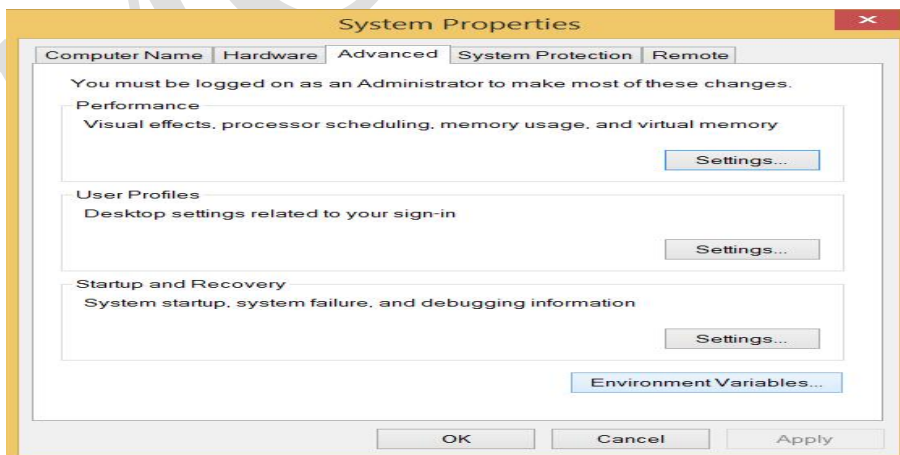
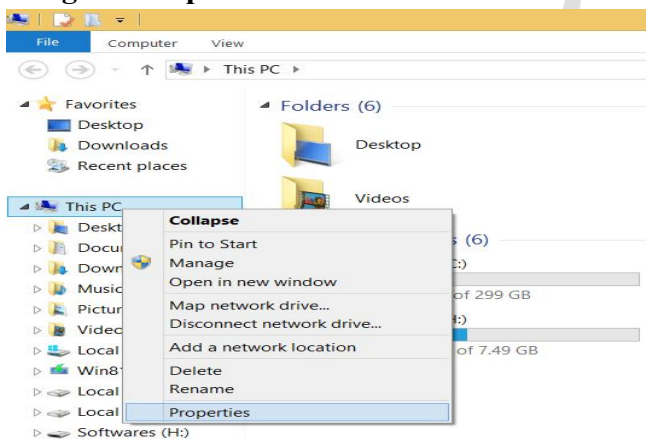
5. Now, we are ready to install python-3.6.7. Let's install it.



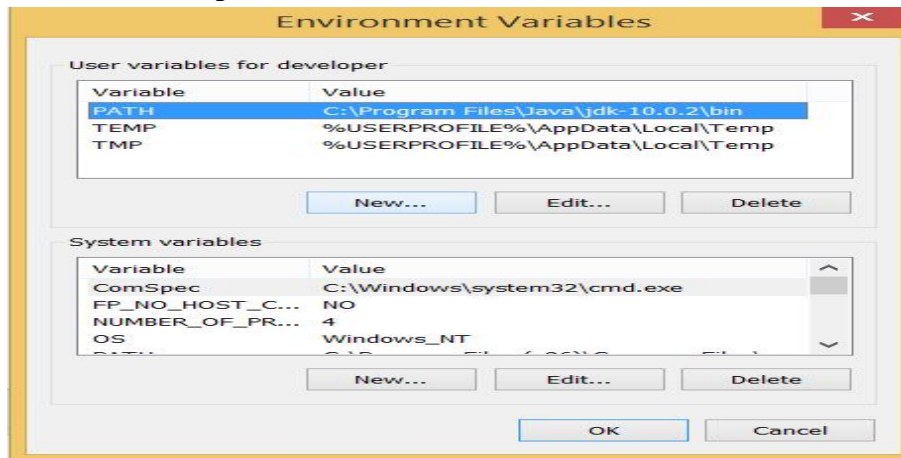
6. Now, try to run python on the command prompt. (press window button + R)
Type the command python in case of python2 or python3 in case of python3. It will show an error as given in the below image. It is because we haven't set the path.



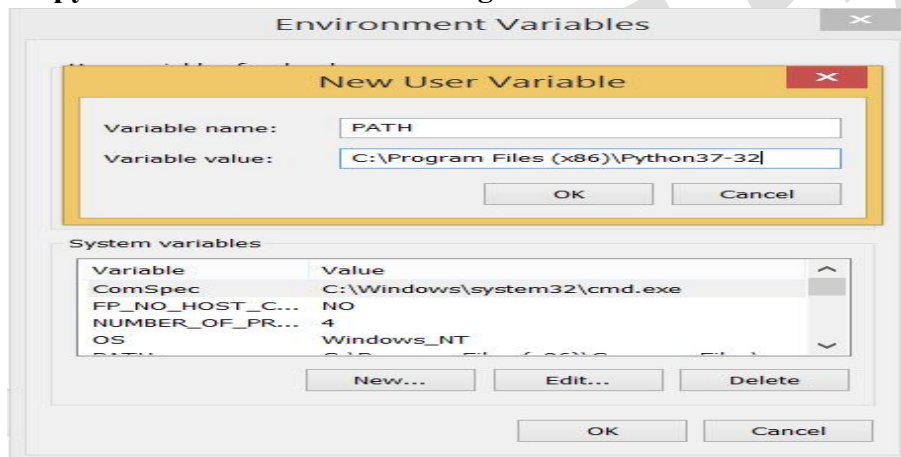
7. To set the path of python, we need to right click on "my computer / This PC" and go to Properties → Advanced → Environment Variables.



8. Add the new path variable in the user variable section.



9. Type PATH as the variable name and set the path to the installation directory of the python shown in the below image.



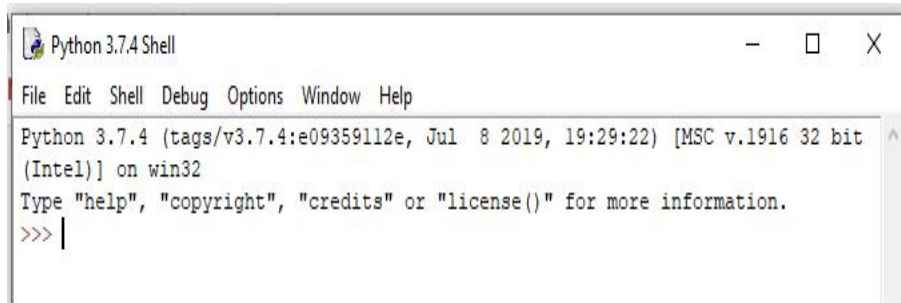
Now, the path is set; we are ready to run python on our local system. Restart CMD, and type python again. It will open the python interpreter shell where we can execute the python statements.

❖ Executing Python program

- Python provides us the two ways to run a program:

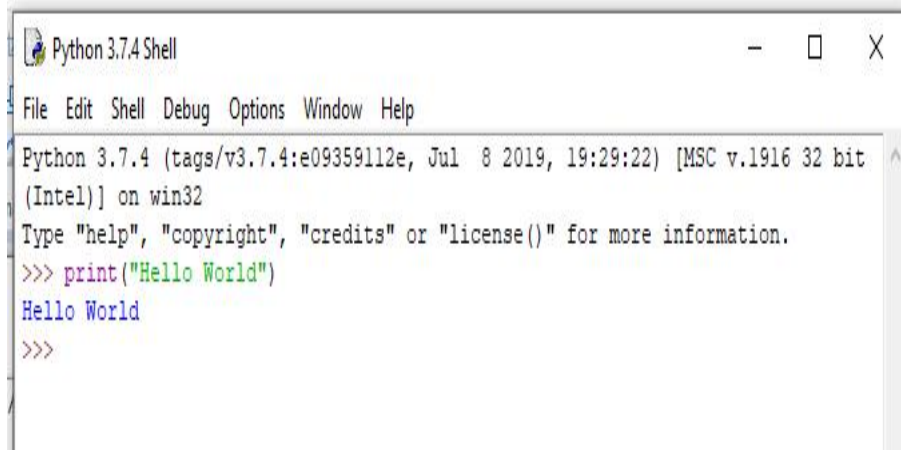
1. Using Interactive interpreter prompt

- Python provides us the feature to execute the Python statement one by one at the interactive prompt. It is preferable in the case where we are concerned about the output of each line of our Python program.
- To open the interactive mode, open the terminal (or command prompt) and type python (python3 in case if you have Python2 and Python3 both installed on your system).
- It will open the following prompt where we can execute the Python statement and check their impact on the console.



```
Python 3.7.4 Shell
File Edit Shell Debug Options Window Help
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul 8 2019, 19:29:22) [MSC v.1916 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> |
```

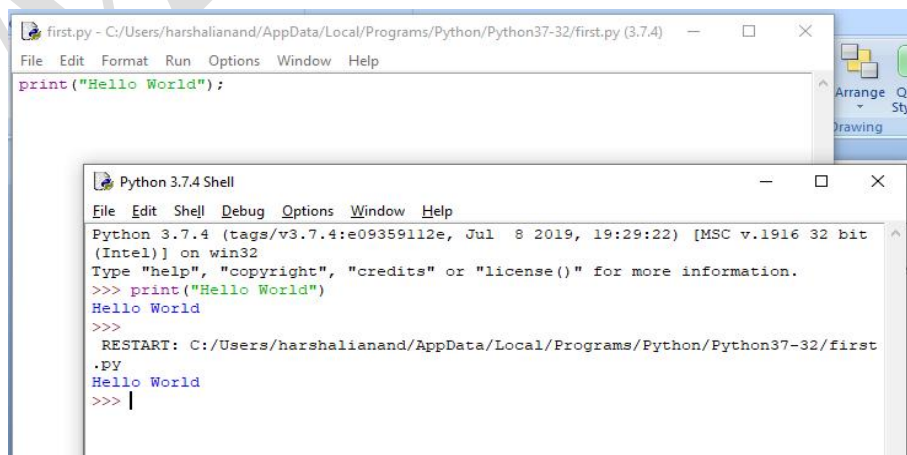
Here, we get the message **"Hello World !"** printed on the console.



```
Python 3.7.4 Shell
File Edit Shell Debug Options Window Help
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul 8 2019, 19:29:22) [MSC v.1916 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> print("Hello World")
Hello World
>>>
```

2. Using a script file

- Interpreter prompt is good to run the individual statements of the code. However, we cannot write the code every-time on the terminal.
- We need to write our code into a file which can be executed later. For this purpose, open an editor like notepad or using new file in python editor, create a file named `first.py` (Python used `.py` extension) and write the following code in it.
- `print ("hello world");`
- here, we have used `print()` function to print the message on the console.
- To run this file named as `first.py`, we need to run the following command on the terminal.



```
first.py - C:/Users/harshalianand/AppData/Local/Programs/Python/Python37-32/first.py (3.7.4)
File Edit Format Run Options Window Help
print("Hello World");

Python 3.7.4 Shell
File Edit Shell Debug Options Window Help
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul 8 2019, 19:29:22) [MSC v.1916 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> print("Hello World")
Hello World
>>>
RESTART: C:/Users/harshalianand/AppData/Local/Programs/Python/Python37-32/first
.py
Hello World
>>> |
```


❖ *Execute the following task using the basic knowledge of Python Language*

1. *Python Program to Print Hello MGM CET*
2. *Python Program to Check If number is Even or Odd*
3. *Python program to check whether the input character is an alphabet*
4. *Python Program to Check if a Number is Positive Negative or Zero*
5. *Python Program to Check Whether a String is Palindrome or Not*

Conclusion:

Result:

Industrial Application:

- *Systems Programming*
- *GUIs*
- *Internet Scripting*
- *Component Integration*
- *Database Programming*
- *Numeric and Scientific Programming*
- *Gaming, Images, Serial Ports, XML, Robots, and More*

Questions:

1. *How to comment a statement in Python?*

2. *Enlist the applications of Python.*

3. *What is IDE?*

4. *State the shortcut to execute the Python code while using script file.*

5. *Write control statements in Python.*

6. *What is variable?*

7. *Enlist editors or framework where we can write Python programs and also can create Python project.*

8. *Shortcut to get help in Python.*

9. *What is the extension of Python file?*

10. *Define constant*

Experiment No. 2

Aim: Exploring basics of python like data types (strings, list, array, dictionaries, set, tuples) and control statements

Software Required: Python3.9 / Online Editor

Theory:

(i) Numbers

```
a = 5
print(a, "is of type", type(a))
```

Output:

```
a = 2.0
print(a, "is of type", type(a))
```

Output:

```
a = 1+2j
print(a, "is complex number?", isinstance(1+2j,complex))
```

Output:

(ii) Python List

```
a = [5,10,15,20,25,30,35,40]
```

```
# a[2] = 15
print("a[2] = ", a[2])
```

Output:

```
# a[0:3] = [5, 10, 15]
print("a[0:3] = ", a[0:3])
```

Output:

```
# a[5:] = [30, 35, 40]
print("a[5:] = ", a[5:])
```

Output:

(iii) Python Tuple

```
t = (5,'program', 1+3j)
```

```
# t[1] = 'program'
```

```
print("t[1] = ", t[1])
```

Output:

```
# t[0:3] = (5, 'program', (1+3j))
print("t[0:3] = ", t[0:3])
```

Output:

```
# Generates error
# Tuples are immutable
t[0] = 10
```

(iv) Python Strings

```
s = "This is a string"
print(s)
```

Output:

```
s = """A multiline
string"""
print(s)
```

Output:

```
s = 'Hello world!'

# s[4] = 'o'
print("s[4] = ", s[4])
```

Output:

```
# s[6:11] = 'world'
print("s[6:11] = ", s[6:11])
```

Output:

```
# Generates error
# Strings are immutable in Python
s[5] = 'd'
```

(v) Python Set

```
a = {5,2,3,1,4}
```

```
# printing set variable  
print("a = ", a)
```

Output:

```
# data type of variable a  
print(type(a))
```

Output:

(vi) Python Dictionary

```
d = {1:'value','key':2}  
print(type(d))
```

Output:

```
print("d[1] = ", d[1]);
```

Output:

```
print("d['key'] = ", d['key']);
```

Output:

```
# Generates error  
print("d[2] = ", d[2]);
```

Output:

(vii) Conversion between data types

We can convert between different data types by using different type conversion functions like int(), float(), str(), etc.

```
>>> float(5)
```

Output:

Conversion from float to int will truncate the value (make it closer to zero).

```
>>> int(10.6)
```

Output:

```
>>> int(-10.6)
```

Output:

Conversion to and from string must contain compatible values.

```
>>> float('2.5')
```

Output:

```
>>> str(25)
```

Output:

```
>>> int('1p')
```

Output:

```
>>> set([1,2,3])
```

Output:

```
>>> tuple({5,6,7})
```

Output:

```
>>> list('hello')
```

Output:

To convert to dictionary, each element must be a pair:

```
>>> dict([[1,2],[3,4]])
```

Output:

```
>>> dict([(3,26),(4,44)])
```

Output:

Conditional Statements in Python

Syntax

```
if condition1:  
    statements
```

```
elif condition2:  
    statements
```

```
else:  
    statements
```

Consider the example below:

```
X = 10
```

```
Y = 12
```

```
if X < Y: print('X is less than Y') elif X > Y:
    print('X is greater than Y')
else:
    print('X and Y are equal')
```

Output:

While Loop

Here, first the condition is checked and if it's true, control will move inside the loop and execute the statements inside the loop until the condition becomes false. We use this loop when we are not sure how many times we need to execute a group of statements or you can say that when we are unsure about the number of iterations.

Consider the example:

Syntax and Usage:

```
count = 0
while (count < 10):
    print (count)
    count = count + 1
```

```
print ("Good bye!")
```

Output =

For Loop

Like the While loop, the For loop also allows a code block to be repeated certain number of times. The difference is, in For loop we know the amount of iterations required unlike While loop, where iterations depends on the condition. You will get a better idea about the difference between the two by looking at the syntax:

Syntax:

for variable in Sequence:

statements

Notice here, we have specified the range, that means we know the number of times the code block will be executed.

Consider the example:

```
fruits = ['Banana', 'Apple', 'Grapes']
```

```
for index in range(len(fruits)):
    print (fruits[index])
```

Output:

Nested Loops

It basically means a loop inside a loop. It can be a For loop inside a While loop and vice-versa. Even a For loop can be inside a For loop or a While loop inside a While loop.

Consider the example:

```
count = 1
for i in range(10):
    print (str(i) * i)

    for j in range(0, i):
        count = count + 1
```

Output

Conclusion:

Result:

Experiment No. 3

Aim: *Creating functions, classes and objects using python. Demonstrate exception handling and inheritance.*

Software Required: *Python3.9 / Online Editor*

Theory:

Python Function

Functions are the most important aspect of an application. A function can be defined as the organized block of reusable code, which can be called whenever required.

Python allows us to divide a large program into the basic building blocks known as a function. The function contains the set of programming statements enclosed by {}. A function can be called multiple times to provide reusability and modularity to the Python program.

The Function helps to programmer to break the program into the smaller part. It organizes the code very effectively and avoids the repetition of the code. As the program grows, function makes the program more organized.

*Python provide us various inbuilt functions like **range()** or **print()**. Although, the user can create its functions, which can be called user-defined functions.*

There are mainly two types of functions.

- **User-define functions** - *The user-defined functions are those define by the **user** to perform the specific task.*
- **Built-in functions** - *The built-in functions are those functions that are **pre-defined** in Python.*

There are the following advantages of Python functions.

- *Using functions, we can avoid rewriting the same logic/code again and again in a program.*
- *We can call Python functions multiple times in a program and anywhere in a program.*
- *We can track a large Python program easily when it is divided into multiple functions.*
- *Reusability is the main achievement of Python functions.*
- *However, Function calling is always overhead in a Python program.*

Creating a Function

Python provides the **def** keyword to define the function. The syntax of the define function is given below.

Syntax:

```
def my_function(parameters):  
    function_block  
return expression
```

Let's understand the syntax of functions definition.

- The **def** keyword, along with the function name is used to define the function.
- The identifier rule must follow the function name.
- A function accepts the parameter (argument), and they can be optional.
- The function block is started with the colon (:), and block statements must be at the same indentation.
- The **return** statement is used to return the value. A function can have only one **return**

Function Calling

In Python, after the function is created, we can call it from another function. A function must be defined before the function call; otherwise, the Python interpreter gives an error. To call the function, use the function name followed by the parentheses.

Consider the following example of a simple example that prints the message "Hello World".

```
#function definition  
def hello_world():  
    print("hello world")  
#function calling  
hello_world()
```

Output:

hello world

Arguments in function

The arguments are types of information which can be passed into the function. The arguments are specified in the parentheses. We can pass any number of arguments, but they must be separate them with a comma.

Types of arguments

There may be several types of arguments which can be passed at the time of function call.

1. *Required arguments*
2. *Keyword arguments*
3. *Default arguments*
4. *Variable-length arguments*

1. Required Arguments

Till now, we have learned about function calling in Python. However, we can provide the arguments at the time of the function call. As far as the required arguments are concerned, these are the arguments which are required to be passed at the time of function calling with the exact match of their positions in the function call and function definition. If either of the arguments is not provided in the function call, or the position of the arguments is changed, the Python interpreter will show the error.

2. Default Arguments

Python allows us to initialize the arguments at the function definition. If the value of any of the arguments is not provided at the time of function call, then that argument can be initialized with the value given in the definition even if the argument is not specified at the function call.

3. Variable-length Arguments (*args)

In large projects, sometimes we may not know the number of arguments to be passed in advance. In such cases, Python provides us the flexibility to offer the comma-separated values which are internally treated as tuples at the function call. By using the variable-length arguments, we can pass any number of arguments.

*However, at the function definition, we define the variable-length argument using the ***args** (star) as ***<variable - name >**.*

4. Keyword arguments (**kwargs)

Python allows us to call the function with the keyword arguments. This kind of function call will enable us to pass the arguments in the random order.

The name of the arguments is treated as the keywords and matched in the function calling and definition. If the same match is found, the values of the arguments are copied in the function definition.

Questions:

1. What are the various Python language editor available for writing Python code?

2. Enlist advantages of Python language.

3. Enlist various applications of Python language.

4. Can we execute scilab code in Matlab? Explain.

5. What is IDE?

6. Can we execute numerical methods programmatically in Python? Explain.

7. Write in short about Python language.

8. Enlist various real life applications of Python language.

9. What is PyCharm? Write in short about PyCharm?

10. Write in details about Python list.

Experiment No. 4

Aim: *Exploring Files and directories*

4.a. Python program to append data to existing file and then display the entire file

4.b. Python program to count number of lines, words and characters in a file

4.c. Python program to display file available in current directory .

Software Required: Python3.9 / Online Editor

Theory:

Files:

Files are named locations on disk to store related information. They are used to permanently store data in a non-volatile memory (e.g. hard disk). Since Random Access Memory (RAM) is volatile (which loses its data when the computer is turned off), we use files for future use of the data by permanently storing them.

When we want to read from or write to a file, we need to open it first. When we are done, it needs to be closed so that the resources that are tied with the file are freed.

Hence, in Python, a file operation takes place in the following order:

1. Open a file
2. Read or write (perform operation)
3. Close the file

Python File Methods

There are various methods available with the file object. Some of them have been used in the above examples.

Here is the complete list of methods in text mode with a brief description:

Method	Description
<code>close()</code>	Closes an opened file. It has no effect if the file is already closed.
<code>detach()</code>	Separates the underlying binary buffer from the <code>TextIOBase</code> and returns it.
<code>fileno()</code>	Returns an integer number (file descriptor) of the file.
<code>flush()</code>	Flushes the write buffer of the file stream.
<code>isatty()</code>	Returns <code>True</code> if the file stream is interactive.

<code>read(n)</code>	Reads at most <code>n</code> characters from the file. Reads till end of file if it is negative or <code>None</code> .
<code>readable()</code>	Returns <code>True</code> if the file stream can be read from.
<code>readline(n=-1)</code>	Reads and returns one line from the file. Reads in at most <code>n</code> bytes if specified.
<code>readlines(n=-1)</code>	Reads and returns a list of lines from the file. Reads in at most <code>n</code> bytes/characters if specified.
<code>seek(offset,from=SEEK_SET)</code>	Changes the file position to <code>offset</code> bytes, in reference to <code>from</code> (start, current, end).
<code>seekable()</code>	Returns <code>True</code> if the file stream supports random access.
<code>tell()</code>	Returns an integer that represents the current position of the file's object.
<code>truncate(size=None)</code>	Resizes the file stream to <code>size</code> bytes. If <code>size</code> is not specified, resizes to current location.
<code>writable()</code>	Returns <code>True</code> if the file stream can be written to.
<code>write(s)</code>	Writes the string <code>s</code> to the file and returns the number of characters written.
<code>writelines(lines)</code>	Writes a list of <code>lines</code> to the file.

Python Directory:

If there are a large number of files to handle in our Python program, we can arrange our code within different directories to make things more manageable.

A directory or folder is a collection of files and subdirectories. Python has the `os` module that provides us with many useful methods to work with directories (and files as well).

All files are contained within various directories, and Python has no problem handling these too. The os module has several methods that help you create, remove, and change directories.

1. The mkdir() Method

You can use the mkdir() method of the os module to create directories in the current directory. You need to supply an argument to this method which contains the name of the directory to be created.

Syntax

os.mkdir("newdir")

Example

Following is the example to create a directory test in the current directory –

```
#!/usr/bin/python
import os
# Create a directory "test"
os.mkdir("test")
```

2. The chdir() Method

You can use the chdir() method to change the current directory. The chdir() method takes an argument, which is the name of the directory that you want to make the current directory.

Syntax

os.chdir("newdir")

Example

Following is the example to go into "/home/newdir" directory –

```
#!/usr/bin/python
import os
# Changing a directory to "/home/newdir"
os.chdir("/home/newdir")
```

3. The getcwd() Method

The getcwd() method displays the current working directory.

Syntax

os.getcwd()

Example

Following is the example to give current directory –

```
#!/usr/bin/python
import os
# This would give location of the current directory
os.getcwd()
```


4. The rmdir() Method

The rmdir() method deletes the directory, which is passed as an argument in the method.

Before removing a directory, all the contents in it should be removed.

Syntax

os.rmdir('dirname')

Example

Following is the example to remove "/tmp/test" directory. It is required to give fully qualified name of the directory, otherwise it would search for that directory in the current directory.

```
#!/usr/bin/python
```

```
import os
```

```
# This would remove "/tmp/test" directory.
```

```
os.rmdir( "/tmp/test" )
```

Conclusion:

Result:

Experiment No. 5

Aim: *Creating GUI with python containing widgets such as labels, textbox, radio, checkboxes and custom dialog boxes*

Software Required: *Python3.9 / Online Editor*

Theory:

Python provides various options for developing graphical user interfaces (GUIs). Most important are listed below.

- 1. Tkinter – Tkinter is the Python interface to the Tk GUI toolkit shipped with Python.*
- 2. wxPython – This is an open-source Python interface for wxWindows <http://wxpython.org>.*
- 3. JPython – JPython is a Python port for Java which gives Python scripts seamless access to Java class libraries on the local machine <http://www.jython.org>.*

Tkinter Programming

Tkinter is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit.

Creating a GUI application using Tkinter is an easy task. All you need to do is perform the following steps –

- *Import the Tkinter module.*
- *Create the GUI application main window.*
- *Add one or more of the above-mentioned widgets to the GUI application.*
- *Enter the main event loop to take action against each event triggered by the user.*

Example

```
#!/usr/bin/python  
import Tkinter  
top = Tkinter.Tk()  
# Code to add widgets will go here...  
top.mainloop()
```

This would create a following window –



Tkinter Widgets

Tkinter provides various controls, such as buttons, labels and text boxes used in a GUI application. These controls are commonly called widgets.

There are currently 15 types of widgets in Tkinter. –

Sr.No.	Operator & Description
1	Button <i>The Button widget is used to display buttons in your application.</i>
2	Canvas <i>The Canvas widget is used to draw shapes, such as lines, ovals, polygons and rectangles, in your application.</i>
3	Checkbutton <i>The Checkbutton widget is used to display a number of options as checkboxes. The user can select multiple options at a time.</i>
4	Entry <i>The Entry widget is used to display a single-line text field for accepting values from a user.</i>
5	Frame <i>The Frame widget is used as a container widget to organize other widgets.</i>
6	Label <i>The Label widget is used to provide a single-line caption for other widgets. It can also contain images.</i>
7	Listbox <i>The Listbox widget is used to provide a list of options to a user.</i>
8	Menubutton

	<i>The Menubutton widget is used to display menus in your application.</i>
9	<i>Menu</i> <i>The Menu widget is used to provide various commands to a user. These commands are contained inside Menubutton.</i>
10	<i>Message</i> <i>The Message widget is used to display multiline text fields for accepting values from a user.</i>
11	<i>Radiobutton</i> <i>The Radiobutton widget is used to display a number of options as radio buttons. The user can select only one option at a time.</i>
12	<i>Scale</i> <i>The Scale widget is used to provide a slider widget.</i>
13	<i>Scrollbar</i> <i>The Scrollbar widget is used to add scrolling capability to various widgets, such as list boxes.</i>
14	<i>Text</i> <i>The Text widget is used to display text in multiple lines.</i>
15	<i>Toplevel</i> <i>The Toplevel widget is used to provide a separate window container.</i>
16	<i>Spinbox</i> <i>The Spinbox widget is a variant of the standard Tkinter Entry widget, which can be used to select from a fixed number of values.</i>
17	<i>PanedWindow</i> <i>A PanedWindow is a container widget that may contain any number of panes, arranged horizontally or vertically.</i>
18	<i>LabelFrame</i> <i>A labelframe is a simple container widget. Its primary purpose is to act as a spacer or container for complex window layouts.</i>
19	<i>tkMessageBox - This module is used to display message boxes in your applications.</i>

Standard attributes

- *Dimensions*
- *Colors*
- *Fonts*
- *Anchors*
- *Relief styles*
- *Bitmaps*
- *Cursors*

Geometry Management

All Tkinter widgets have access to specific geometry management methods, which have the purpose of organizing widgets throughout the parent widget area. Tkinter exposes the following geometry manager classes: pack, grid, and place.

- *The pack() Method – This geometry manager organizes widgets in blocks before placing them in the parent widget.*
- *The grid() Method – This geometry manager organizes widgets in a table-like structure in the parent widget.*
- *The place() Method – This geometry manager organizes widgets by placing them in a specific position in the parent widget.*

Conclusion:

Result:

Experiment No. 6

Aim: Menu driven program for data structure using built in function for link list, stack and queue

Software Required: Python3.9 / Online Editor

Theory:

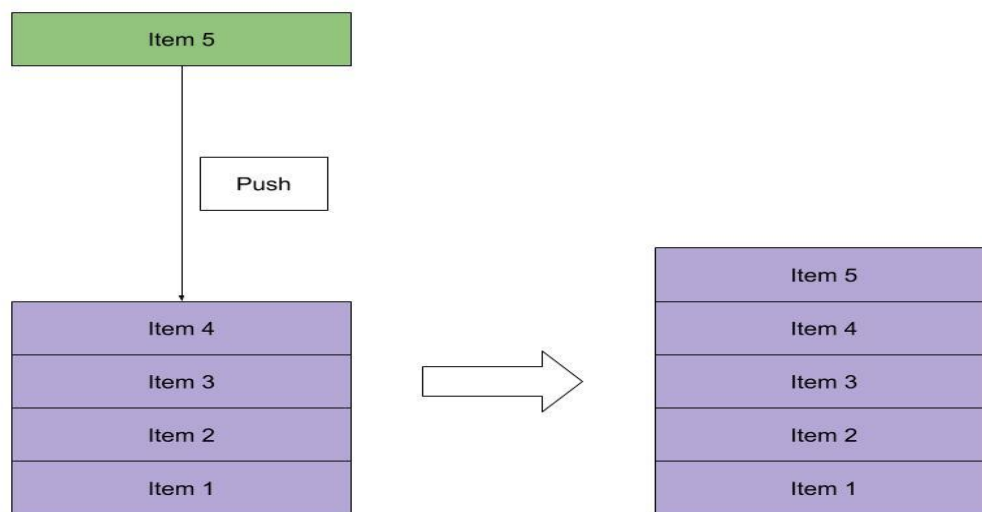
How do they Work?

Stack

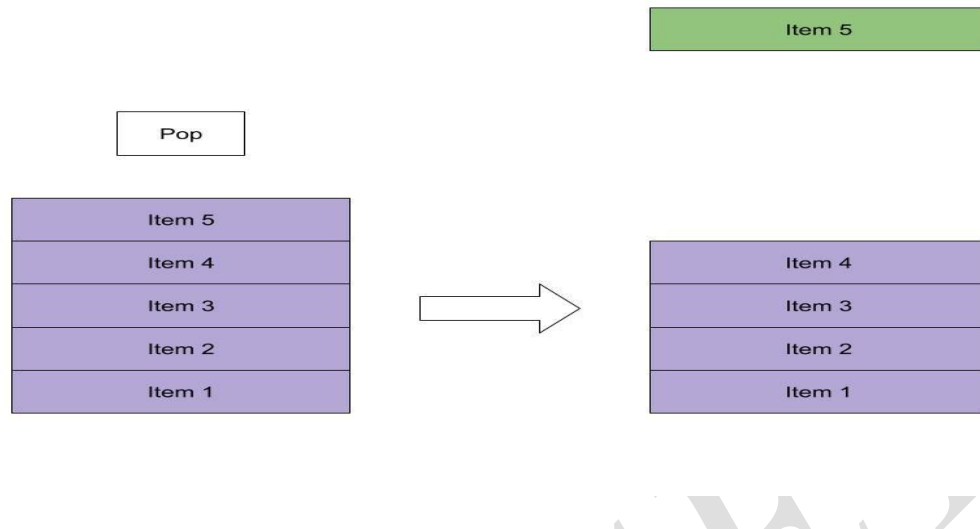
Stacks, like the name suggests, follow the **Last-in-First-Out (LIFO)** principle. As if stacking coins one on top of the other, the last coin we put on the top is the one that is the first to be removed from the stack later.

To implement a stack, therefore, we need two simple operations:

- *push* - adds an element to the top of the stack:



- *pop* - removes the element at the top of the stack:

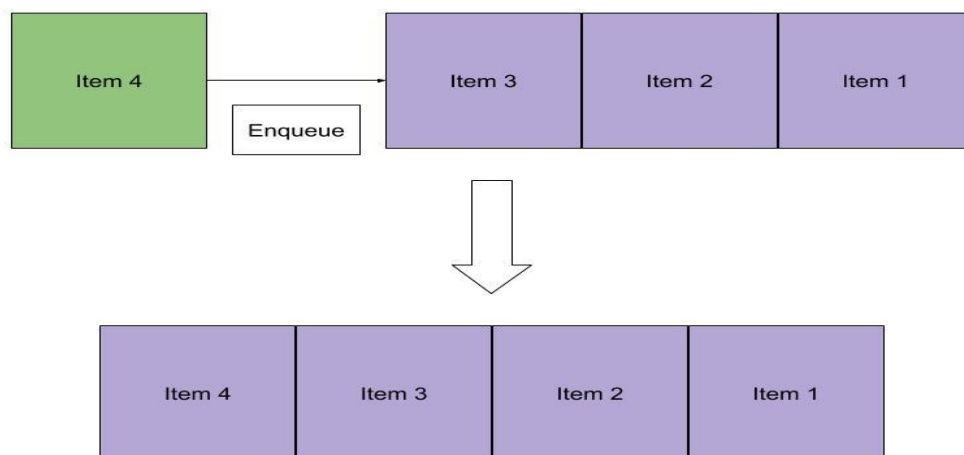


Queue

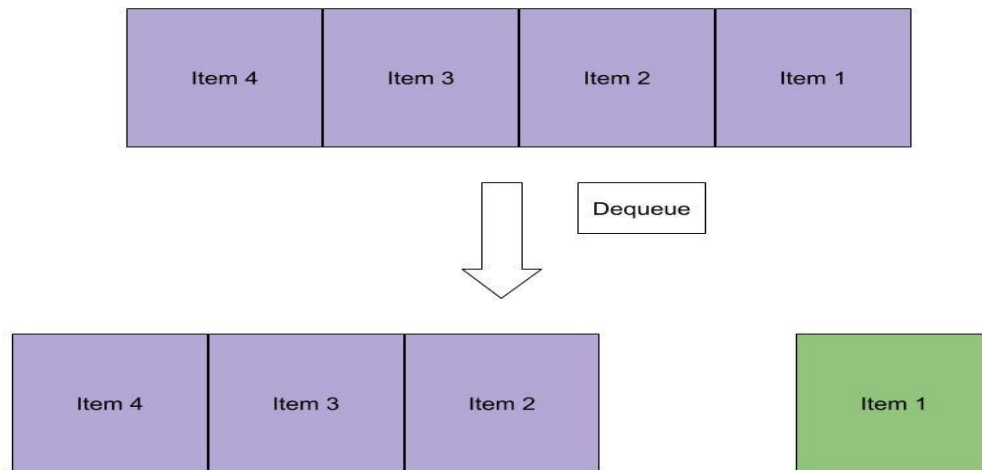
Queues, like the name suggests, follow the **First-in-First-Out (FIFO)** principle. As if waiting in a queue for the movie tickets, the first one to stand in line is the first one to buy a ticket and enjoy the movie.

To implement a queue, therefore, we need two simple operations:

- **enqueue** - adds an element to the end of the queue:



- *dequeue* - removes the element at the beginning of the queue:



Conclusion:

Result:

Experiment No. 7

Aim: Program to demonstrate CRUD (create, read, update and delete) operations on database (SQLite/ MySQL) using python.

Software Required: Python3.9 / Online Editor

Theory:

Database CRUD Operations in Python

In this we will learn how to perform CRUD operations in Python with the SQLite database. Python has built-in support for SQLite in the form of the `sqlite3` module. This module contains functions for performing persistent CRUD operations on SQLite database.

Sqlite Database

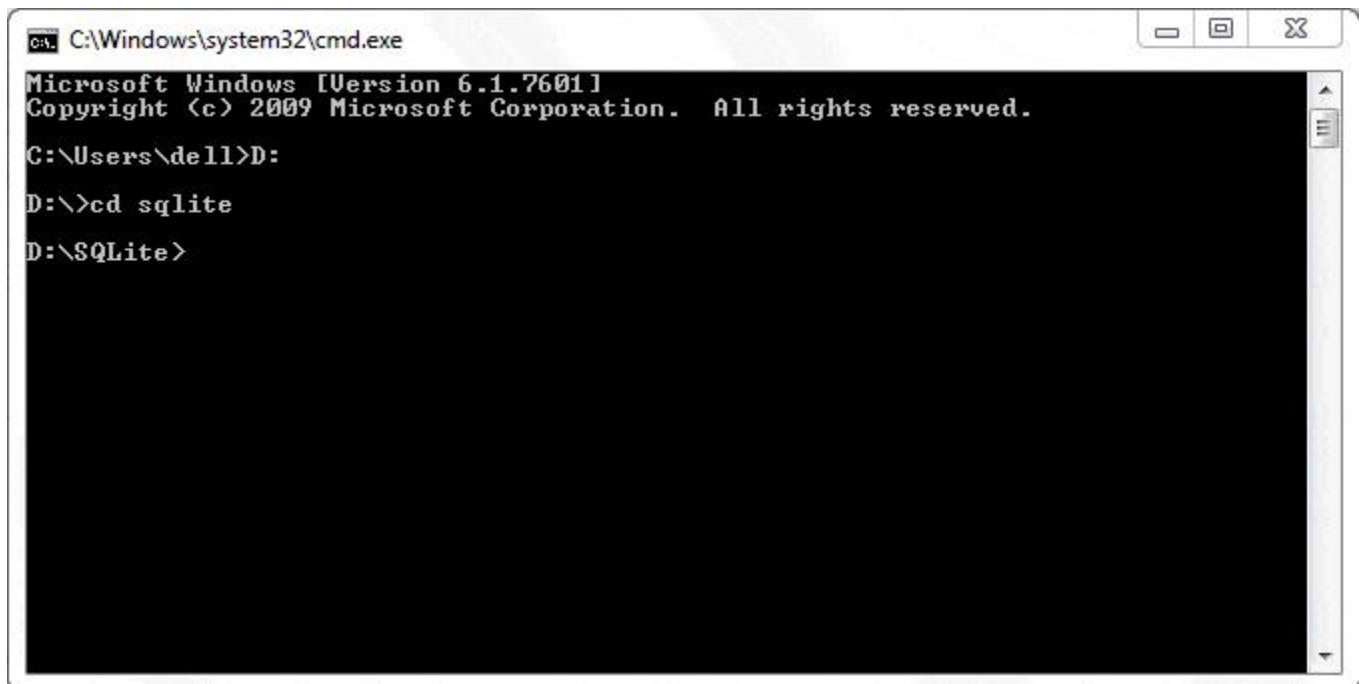
SQLite is a self-contained transactional relational database engine that doesn't require a server configuration, as in the case of Oracle, MySQL, etc. It is an open source and in-process library developed by D. Richard Hipp in August 2000. The entire SQLite database is contained in a single file, which can be put anywhere in the computer's file system.

SQLite is widely used as an embedded database in mobile devices, web browsers and other stand-alone applications. In spite of being small in size, it is a fully ACID compliant database conforming to ANSI SQL standards.

SQLite is freely downloadable from the official web site <https://www.sqlite.org/download.html>. This page contains pre-compiled binaries for all major operating systems. A bundle of command-line tools contain command-line shell and other utilities to manage SQLite database files.

We shall download the latest version of SQLite (version 3.25.1) along with command-line tools and extract the archive.

To create a new SQLite database, navigate from the command prompt to the folder where you have unzipped the archive and enter the following command:



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\dell>D:
D:\>cd sqlite
D:\SQLite>
```

Sqlite3 Command

It is now possible to execute any SQL query. The following statement creates a new table. (Ensure that the statement ends with a semicolon)

```
sqlite> create table student(name text, age int, marks real);
```

Add a record in the above table.

```
sqlite> insert into student values('Ramesh', 21, 55.50);
```

To retrieve the record, use the SELECT query as below:

```
sqlite> select * from student;
```

```
Ramesh|21|55.5
```

Python DB-API

Python Database API is a set of standards recommended by a Special Interest Group for database module standardization. Python modules that provide database interfacing functionality with all major database products are required to adhere to this standard. DB-API standards were further modified to DB-API 2.0 by another Python Enhancement proposal (PEP-249).

Standard Python distribution has in-built support for SQLite database connectivity. It contains sqlite3 module which adheres to DB-API 2.0 and is written by Gerhard Haring. Other RDBMS products also have DB-API compliant modules:

- *MySQL: PyMySQL module*
- *Oracle: Cx-Oracle module*
- *SQL Server: PyMsSql module*
- *PostgreSQL: psycopg2 module*
- *ODBC: pyodbc module*

As per the prescribed standards, the first step in the process is to obtain the connection to the object representing the database. In order to establish a connection with a SQLite database, sqlite3 module needs to be imported and the connect() function needs to be executed.

```
>>> import sqlite3
```

```
>>> db=sqlite3.connect('test.db')
```

The connect() function returns a connection object referring to the existing database or a new database if it doesn't exist.

The following methods are defined in the connection class:

Method	Description
<i>cursor()</i>	<i>Returns a Cursor object which uses this Connection.</i>
<i>commit()</i>	<i>Explicitly commits any pending transactions to the database. The method should be a no-op if the underlying db does not support transactions.</i>
<i>rollback()</i>	<i>This optional method causes a transaction to be rolled back to the starting point. It may not be implemented everywhere.</i>
<i>close()</i>	<i>Closes the connection to the database permanently. Attempts to use the connection after calling this method will raise a DB-API Error.</i>

A cursor is a Python object that enables you to work with the database. It acts as a handle for a given SQL query; it allows the retrieval of one or more rows of the result. Hence, a cursor object is obtained from the connection to execute SQL queries using the following statement:

```
>>> cur=db.cursor()
```

The following methods of the cursor object are useful.

Method	Description
<i>execute()</i>	<i>Executes the SQL query in a string parameter</i>
<i>executemany()</i>	<i>Executes the SQL query using a set of parameters in the list of tuples</i>
<i>fetchone()</i>	<i>Fetches the next row from the query result set.</i>
<i>fetchall()</i>	<i>Fetches all remaining rows from the query result set.</i>
<i>callproc()</i>	<i>Calls a stored procedure.</i>
<i>close()</i>	<i>Closes the cursor object.</i>

The commit() and rollback() methods of the connection class ensure transaction control. The execute() method of the cursor receives a string containing the SQL query. A string having an incorrect SQL query raises an exception, which should be properly handled. That's why the execute() method is placed within the try block and the effect of the SQL query is persistently saved using the commit() method. If however, the SQL query fails, the resulting exception is processed by the except block and the pending transaction is undone using the rollback() method.

Typical use of the execute() method is as follows:

Example:

Try:

```
cur=db.cursor()
cur.execute("Query")
db.commit()
print ("success message")
```

except:

```
print ("error")
db.rollback()
db.close()
```

Create a New Table

A string enclosing the CREATE TABLE query is passed as parameter to the execute() method of the cursor object. The following code creates the student table in the test.db database.

Example: Create a New Table in Sqlite

```
import sqlite3
db=sqlite3.connect('test.db')try:
    cur =db.cursor()
    cur.execute("""CREATE TABLE student (
    StudentID INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT (20) NOT NULL,
    age INTEGER,
    marks REAL);""")
    print ('table created successfully')
except:
    print ('error in operation')
    db.rollback()
db.close()
```

This can be verified using the .tables command in sqlite shell.

E:\SQLite>sqlite3 test.db

SQLite version 3.25.1 2018-09-18 20:20:44

Enter ".help" for usage hints.

```
sqlite> .tables  
student
```

Insert a Record

Once again, the `execute()` method of the cursor object should be called with a string argument representing the `INSERT` query syntax. We have created a student table having three fields: name, age and marks. The string holding the `INSERT` query is defined as:

```
qry="INSERT INTO student (name, age, marks) VALUES ('Rajeev',20,50);"
```

We have to use it as a parameter to the `execute()` method. To account for possible exceptions, the `execute()` statement is placed in the try block as explained earlier. The complete code for the inset operation is as follows:

Example: Insert a Record in Sqlite

```
import sqlite3  
db=sqlite3.connect('test.db')  
qry="insert into student (name, age, marks) values('Rajeev', 20, 50);"try:  
    cur=db.cursor()  
    cur.execute(qry)  
    db.commit()  
    print ("one record added successfully")  
except:  
    print ("error in operation")  
    db.rollback()  
db.close()
```

You can check the result by using the `SELECT` query in Sqlite shell.

```
sqlite> select * from student;  
1|Rajeev|20|50.0
```

Using Parameters in a Query

Often, the values of Python variables need to be used in SQL operations. One way is to use Python's `string format()` function to put Python data in a string. However, this may lead to SQL injection attacks to your program. Instead, use parameter substitution as recommended in Python DB-API. The `?` character is used as a placeholder in the query string and provides the values in the form of a tuple in the `execute()` method. The following example inserts a record using the parameter substitution method:

Example:

```
import sqlite3  
db=sqlite3.connect('test.db')  
qry="insert into student (name, age, marks) values(?,?,?);"try:  
    cur=db.cursor()  
    cur.execute(qry, ('Vijaya', 16,75))  
    db.commit()
```

```

    print ("one record added successfully")except:
    print("error in operation")
    db.rollback()
db.close()

```

The `executemany()` method is used to add multiple records at once. Data to be added should be given in a list of tuples, with each tuple containing one record. The list object (containing tuples) is the parameter of the `executemany()` method, along with the query string.

Example:

```

import sqlite3
db=sqlite3.connect('test.db')
qry="insert into student (name, age, marks) values(?,?,?);"
students=[('Amar', 18, 70), ('Deepak', 25, 87)]try:
    cur=db.cursor()
    cur.executemany(qry, students)
    db.commit()
    print ("records added successfully")except:
    print ("error in operation")
    db.rollback()
db.close()

```

Retrieve Records

When the query string holds a `SELECT` query, the `execute()` method forms a result set object containing the records returned. Python DB-API defines two methods to fetch the records:

- `fetchone()`: Fetches the next available record from the result set. It is a tuple consisting of values of each column of the fetched record.
- `fetchall()`: Fetches all remaining records in the form of a list of tuples. Each tuple corresponds to one record and contains values of each column in the table.

When using the `fetchone()` method, use a loop to iterate through the result set, as below:

Example: Fetch Records

```

import sqlite3
db=sqlite3.connect('test.db')
sql="SELECT * from student;"
cur=db.cursor()
cur.execute(sql)while True:
    record=cur.fetchone()
    if record==None:
        break
    print (record)
db.close()

```

When executed, the following output is displayed in the Python shell:

Result:

```
(1, 'Rajeev', 20, 50.0)
(2, 'Vijaya', 16, 75.0)
(3, 'Amar', 18, 70.0)
(4, 'Deepak', 25, 87.0)
```

The fetchall() method returns a list of tuples, each being one record.

Example:

```
students=cur.fetchall()
for rec in students:
    print (rec)
```

Update a Record

The query string in the execute() method should contain an UPDATE query syntax. To update the value of 'age' to 17 for 'Amar', define the string as below:

```
qry="update student set age=17 where name='Amar';"
```

You can also use the substitution technique to pass the parameter to the UPDATE query.

Example: Update Record

```
import sqlite3
db=sqlite3.connect('test.db')
qry="update student set age=? where name=?;"
try:
    cur=db.cursor()
    cur.execute(qry, (19,'Deepak'))
    db.commit()
    print("record updated successfully")
except:
    print("error in operation")
    db.rollback()
db.close()
```

Delete a Record

The query string should contain the DELETE query syntax. For example, the below code is used to delete 'Bill' from the student table.

```
qry="DELETE from student where name='Bill';"
```

You can use the ? character for parameter substitution.

Example: Delete Record

```
import sqlite3
db=sqlite3.connect('test.db')
qry="DELETE from student where name=?;"
try:
    cur=db.cursor()
    cur.execute(qry, ('Bill',))
    db.commit()
```

```
print("record deleted successfully")except:  
    print("error in operation")  
    db.rollback()  
db.close()
```

Conclusion:

Result:

Experiment No. 8

Aim: *Creation of simple socket for basic information exchange between server and client.*

Software Required: *Python3.9 / Online Editor*

Theory:

Python Socket Programming

To understand python socket programming, we need to know about three interesting topics –

- 1. Socket Server*
- 2. Socket Client*
- 3. Socket.*

So, what is a server? Well, a server is a software that waits for client requests and serves or processes them accordingly.

On the other hand, a client is requester of this service. A client program request for some resources to the server and server responds to that request.

Socket is the endpoint of a bidirectional communications channel between server and client. Sockets may communicate within a process, between processes on the same machine, or between processes on different machines. For any communication with a remote program, we have to connect through a socket port.

The main objective of this socket programming tutorial is to get introduce you how socket server and client communicate with each other. You will also learn how to write python socket server program.

Python Socket Example

We have said earlier that a socket client requests for some resources to the socket server and the server responds to that request.

So we will design both server and client model so that each can communicate with them. The steps can be considered like this.

- 1. Python socket server program executes at first and wait for any request*
- 2. Python socket client program will initiate the conversation at first.*
- 3. Then server program will response accordingly to client requests.*
- 4. Client program will terminate if user enters “bye” message. Server program will also terminate when client program terminates, this is optional and we can keep server program running indefinitely or terminate with some specific command in client request.*

Python Socket Server

We will save python socket server program as socket_server.py. To use python socket connection, we need to import socket module. Then, sequentially we need to perform some task to establish connection between server and client.

We can obtain host address by using `socket.gethostname()` function. It is recommended to use port address above 1024 because port number lesser than 1024 are reserved for standard internet protocol.

See the below python socket server example code, the comments will help you to understand the code.

```
import socket
def server_program():
    # get the hostname
    host = socket.gethostname()
    port = 5000 # initiate port no above 1024

    server_socket = socket.socket() # get instance
    # look closely. The bind() function takes tuple as argument
    server_socket.bind((host, port)) # bind host address and port together

    # configure how many client the server can listen simultaneously
    server_socket.listen(2)
    conn, address = server_socket.accept() # accept new connection
    print("Connection from: " + str(address))
    while True:
        # receive data stream. it won't accept data packet greater than 1024 bytes
        data = conn.recv(1024).decode()
        if not data:
            # if data is not received break
            break
        print("from connected user: " + str(data))
        data = input(' -> ')
        conn.send(data.encode()) # send data to the client

    conn.close() # close the connection

if __name__ == '__main__':
    server_program()
```

So our python socket server is running on port 5000 and it will wait for client request. If you want server to not quit when client connection is closed, just remove the if condition and break statement. Python while loop is used to run the server program indefinitely and keep waiting for client request.

Python Socket Client

We will save python socket client program as `socket_client.py`. This program is similar to the server program, except binding.

The main difference between server and client program is, in server program, it needs to bind host address and port address together.

See the below python socket client example code, the comment will help you to understand the code.

```
import socket
def client_program():
    host = socket.gethostname() # as both code is running on same pc
    port = 5000 # socket server port number

    client_socket = socket.socket() # instantiate
    client_socket.connect((host, port)) # connect to the server

    message = input("-> ") # take input

    while message.lower().strip() != 'bye':
        client_socket.send(message.encode()) # send message
        data = client_socket.recv(1024).decode() # receive response

        print('Received from server: ' + data) # show in terminal

        message = input("-> ") # again take input

    client_socket.close() # close the connection

if __name__ == '__main__':
    client_program()
```

Python Socket Programming Output

To see the output, first run the socket server program. Then run the socket client program. After that, write something from client program. Then again write reply from server program. At last, write bye from client program to terminate both program. Below short video will show how it worked on my test run of socket server and client example programs.

```
pankaj$ python3.6 socket_server.py
Connection from: ('127.0.0.1', 57822)
from connected user: Hi -> Hello
from connected user: How are you? -> Good
from connected user: Awesome! -> Ok then, bye!
pankaj$
```

*pankaj\$ python3.6 socket_client.py -> Hi
Received from server: Hello -> How are you?
Received from server: Good -> Awesome!
Received from server: Ok then, bye! -> Bye
pankaj\$*
*Notice that socket server is running on port 5000 but client also requires a socket port to connect to the server. This port is assigned randomly by client connect call. In this case, it's 57822.
So, that's all for Python socket programming, python socket server and socket client example programs.*

Conclusion:

Result:

Experiment No. 9

Aim: *Creating web application using Django web framework to demonstrate functionality of user login and registration (also validating user detail using regular expression).*

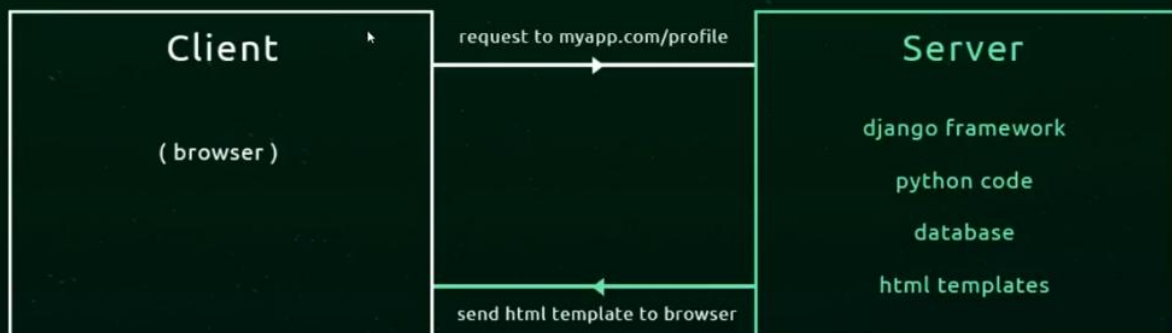
Software Required: *Python3.9 / Online Editor*

Theory: What is Django?

What Is django ?

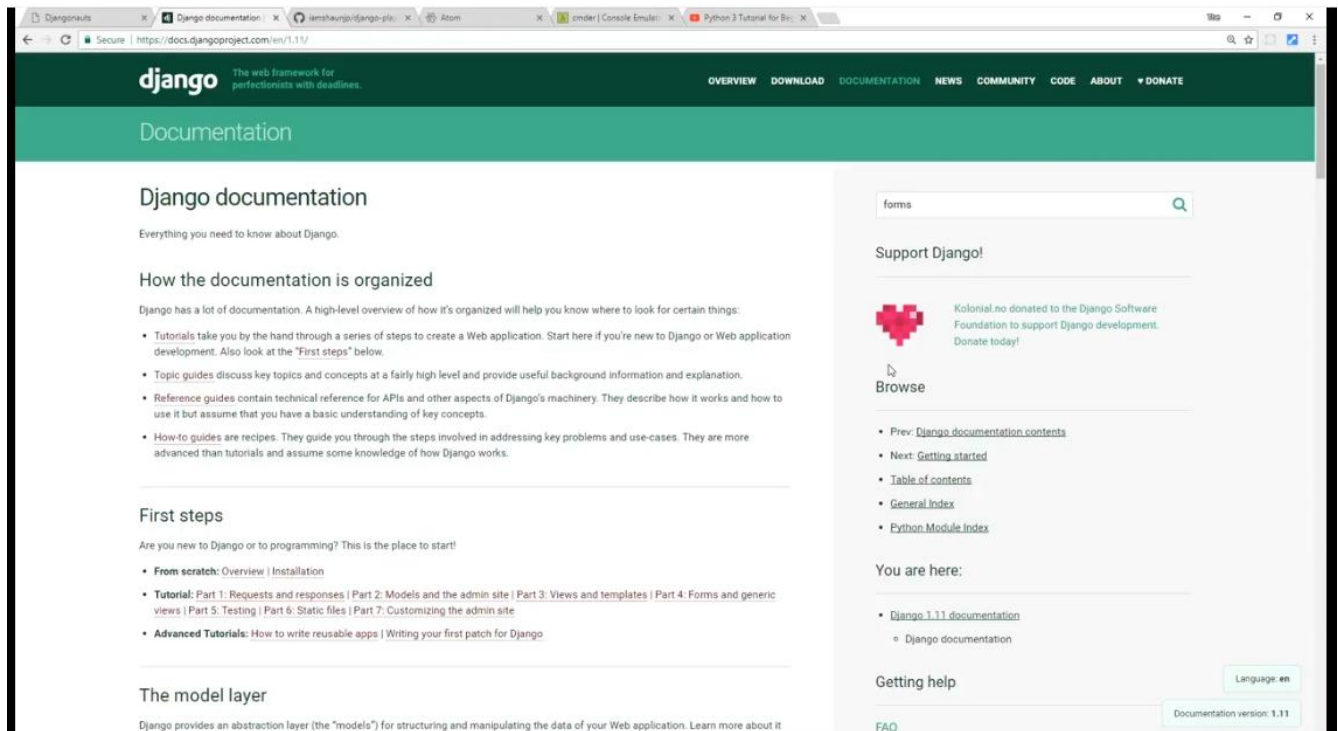
- A web application framework built with Python
- Comes baked with many cool features to help with...
 - User authentication
 - Templating language
 - Routing
 - ...And much more
- Allows us to easily create dynamic web apps using Python

What Is django ?



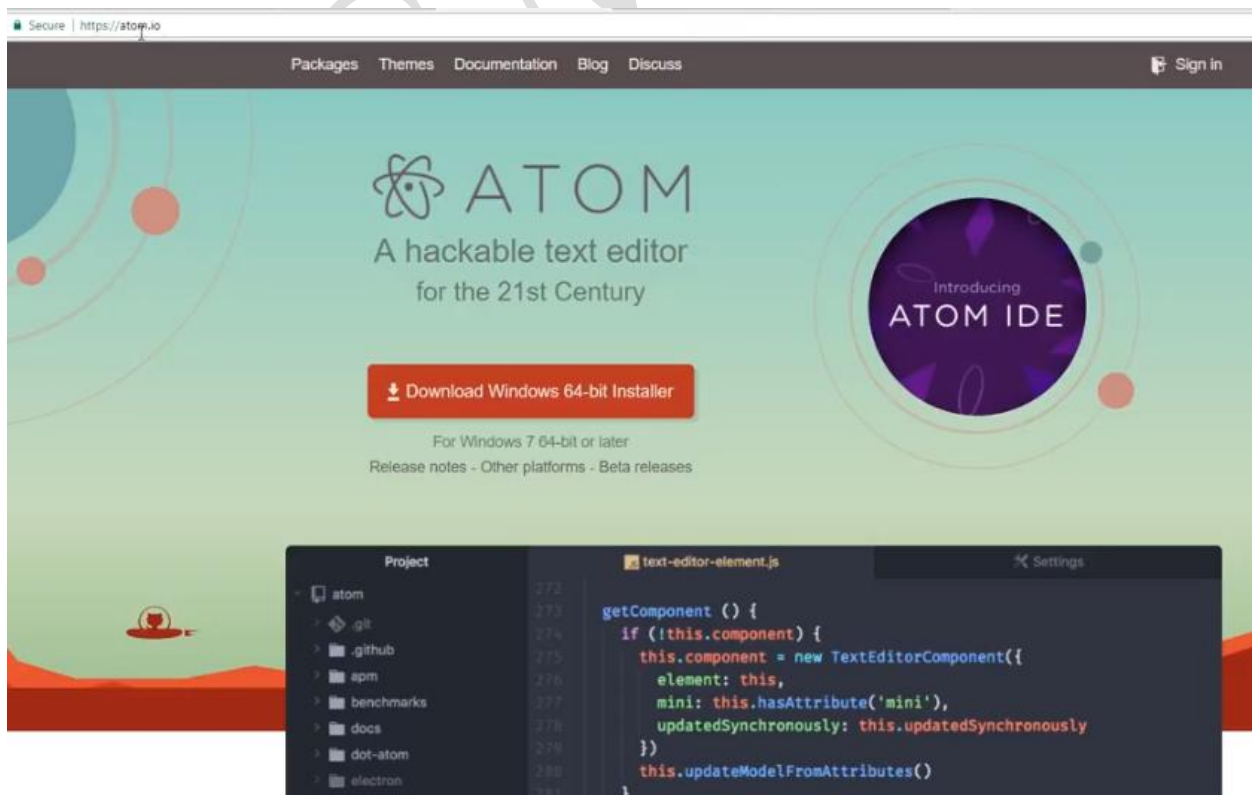
Django documentation is available at

<https://docs.djangoproject.com/en/4.0/>



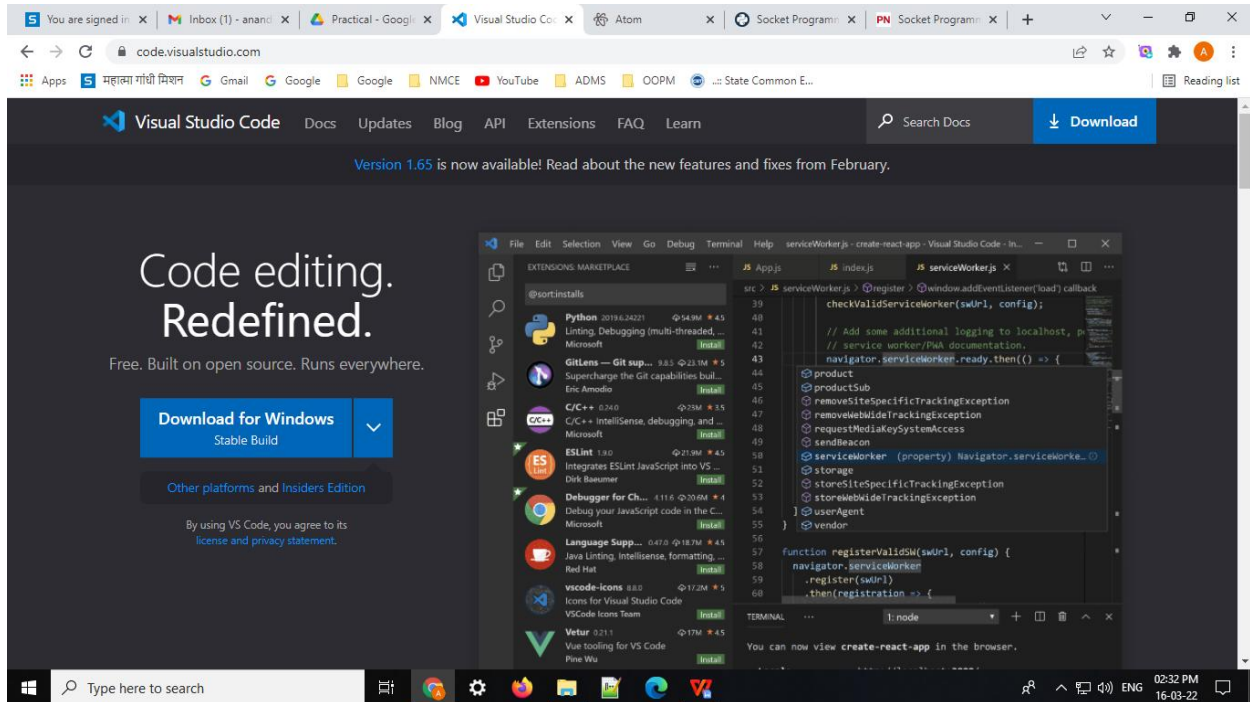
Download editor to write Django code

You can download Atom editor from <https://atom.io/>

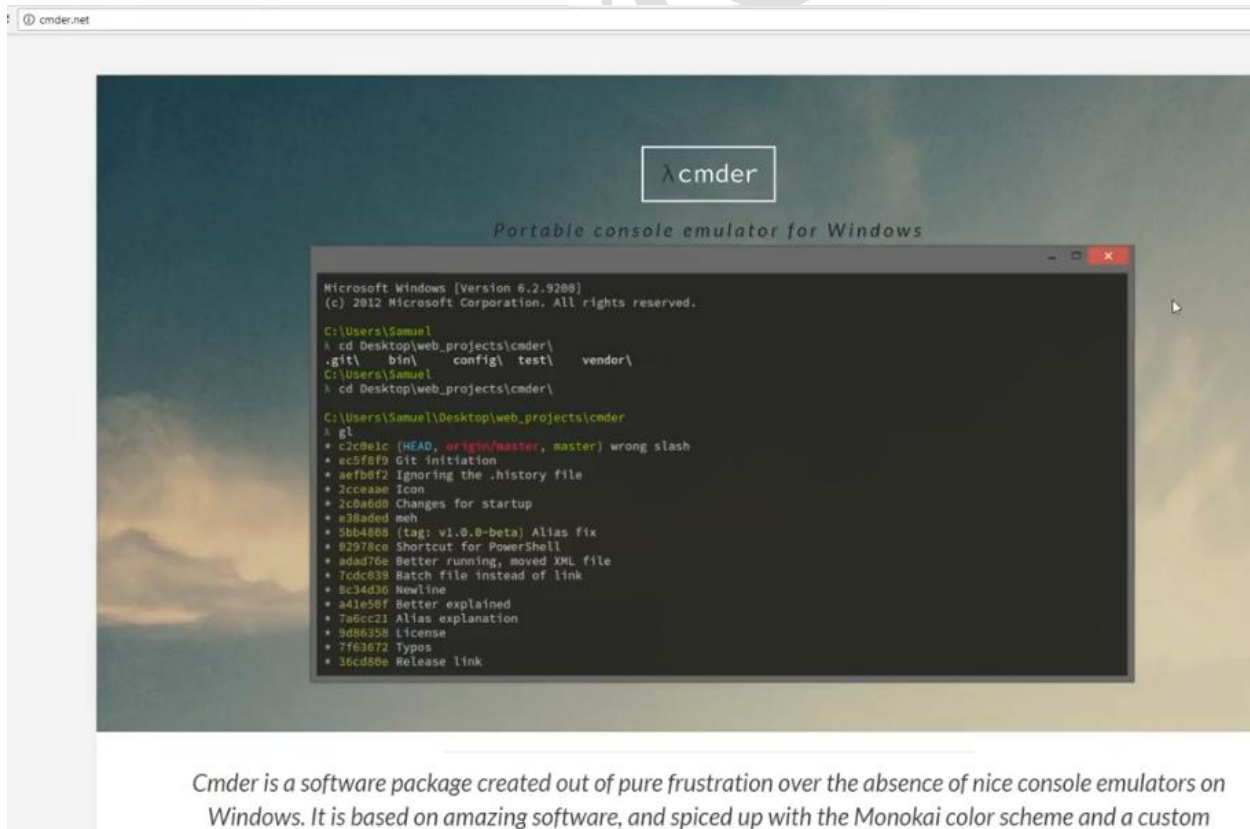


Prepared by: Prof. Anand A. Ingle

Or you can download Visual Studio Code from <https://code.visualstudio.com/>



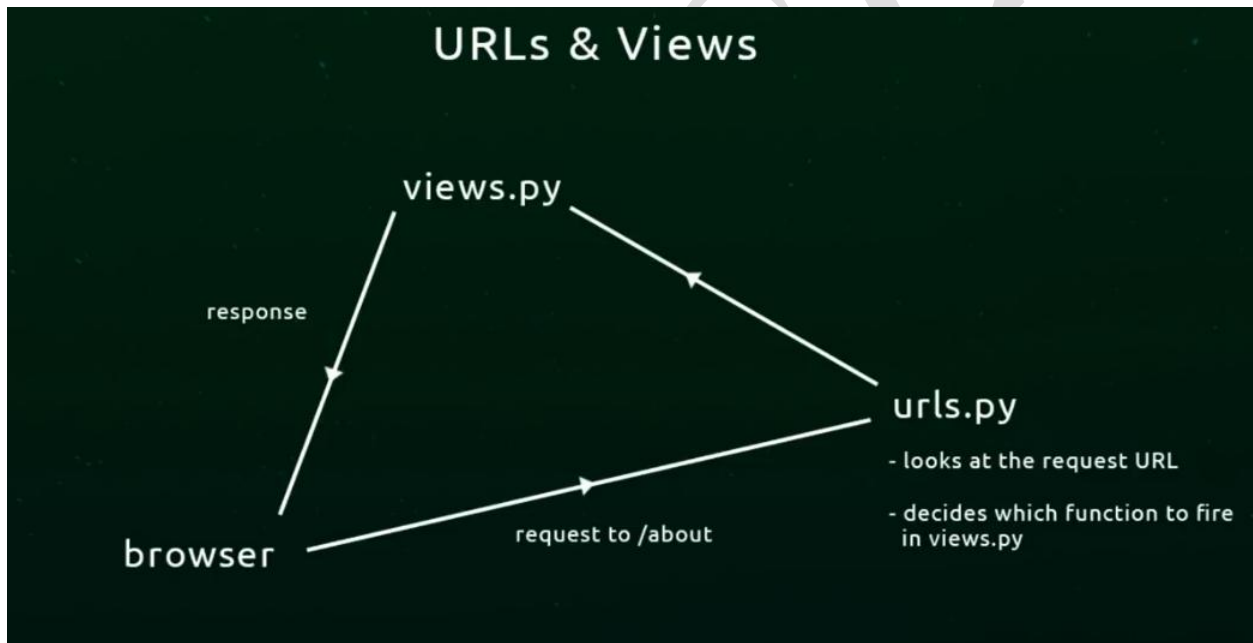
We can use our command prompt or else we can use cmdr from <https://cmdr.net/>



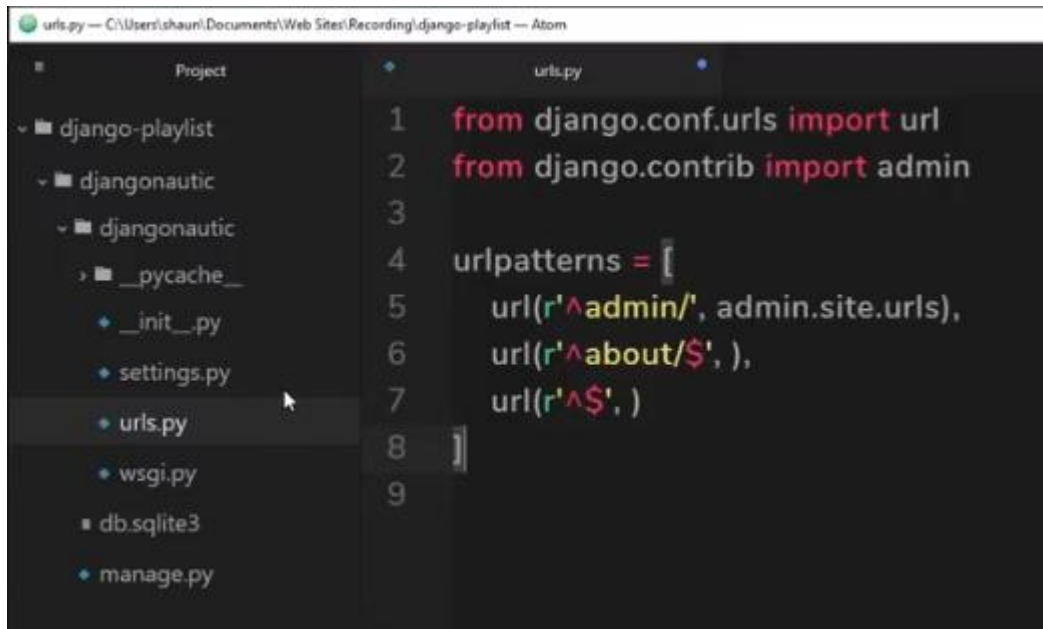
Cmdr is a software package created out of pure frustration over the absence of nice console emulators on Windows. It is based on amazing software, and spiced up with the Monokai color scheme and a custom

Now follow the steps:

- Install python
- Then install django using pip
 - pip install django
- Create a folder (directory) , where we want to create a django project
- Move to that folder in command prompt
- Now create django project in that folder using command
 - django-admin startproject project_name
 - * In that folder setting file have setting of our project
 - * manage.py is very important file, we can start a local server using this manage.py
 - python manage.py runserver
 - (1. we must be in our app folder while executing this command
 - 2. when we run this command dbsqlite3 file will be created automatically in that folder)



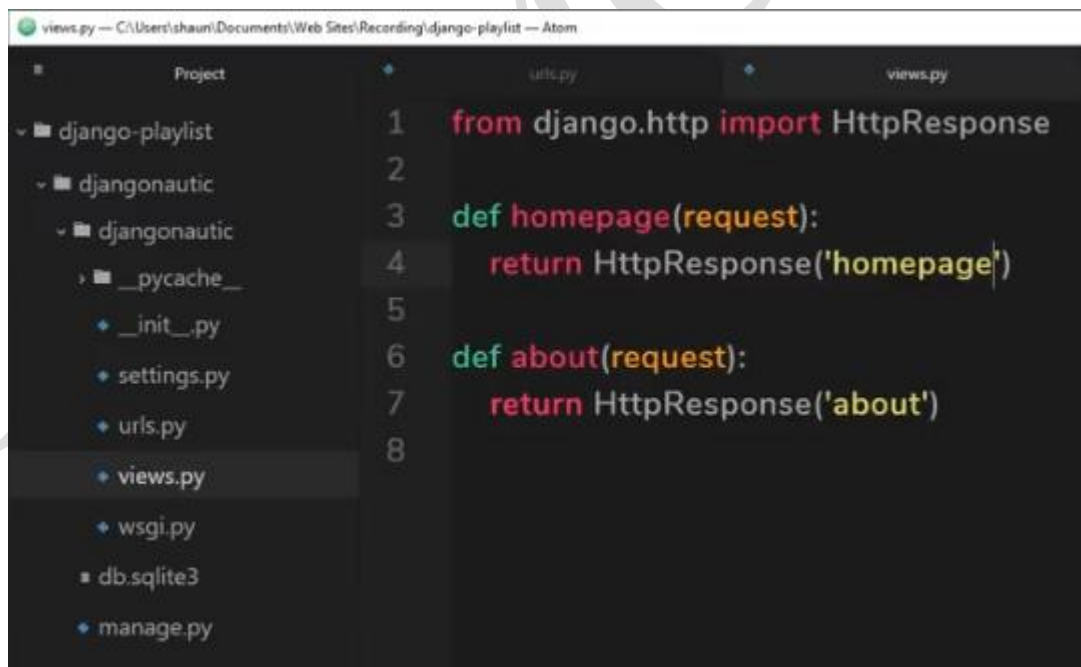
Now write following code in url.py



```
1 from django.conf.urls import url
2 from django.contrib import admin
3
4 urlpatterns = [
5     url(r'^admin/', admin.site.urls),
6     url(r'^about/$', ),
7     url(r'^$', )
8 ]
9
```

Now create `views.py` file in our app folder (right click on app folder and select new file and name it as `views.py`) and write below code

Our `views.py` will look like

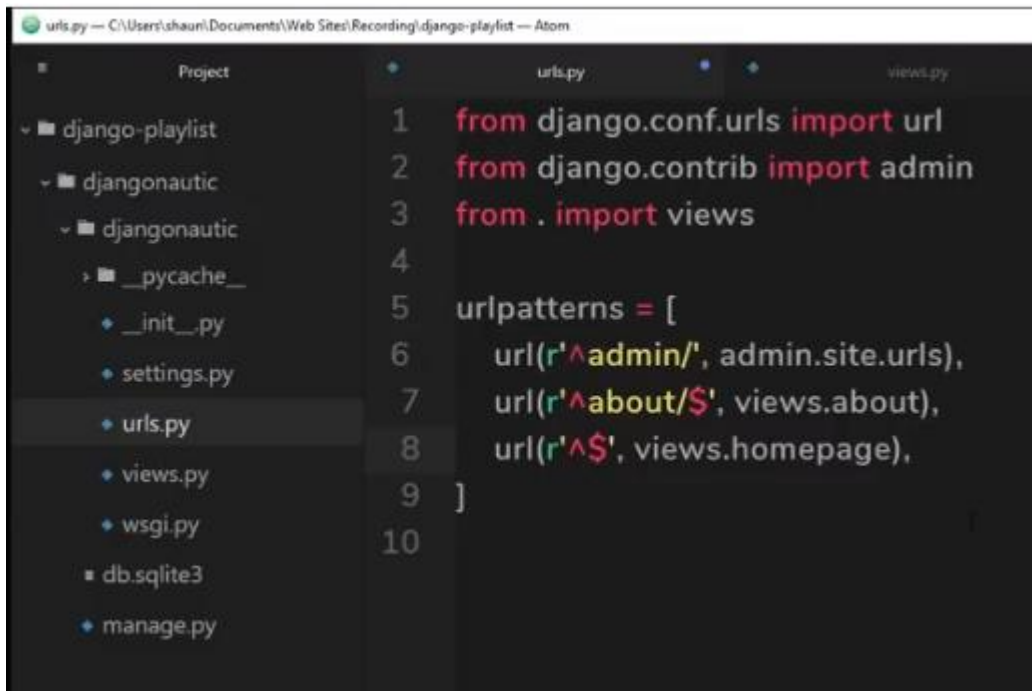


```
1 from django.http import HttpResponse
2
3 def homepage(request):
4     return HttpResponse('homepage')
5
6 def about(request):
7     return HttpResponse('about')
8
```

Then import `views` file in `urls.py` as below

→ `from . import views`

Our `urls.py` file will look like this



```
1 from django.conf.urls import url
2 from django.contrib import admin
3 from . import views
4
5 urlpatterns = [
6     url(r'^admin/', admin.site.urls),
7     url(r'^about/$', views.about),
8     url(r'^$', views.homepage),
9 ]
10
```

Now go to command prompt and run python local server using command

→ `python manage.py runserver`

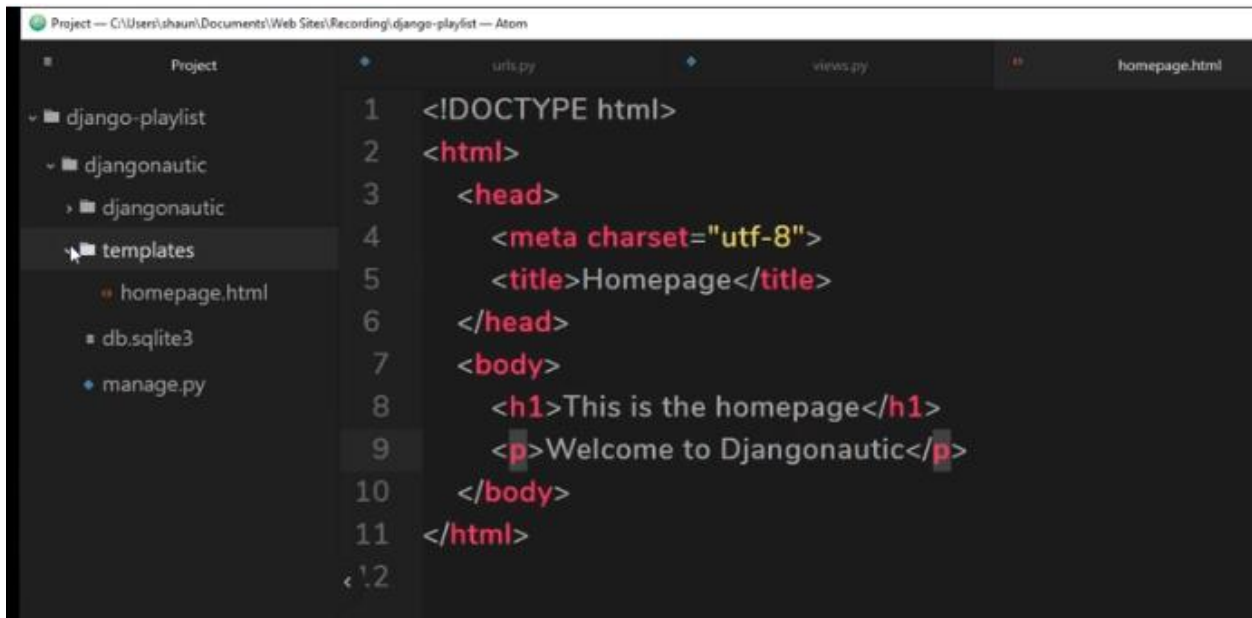
After that, to check the output open internet browser and type the address display in command prompt after starting local server, as below

→ <http://127.0.0.1:8000> or we can type localhost:8000 to get the output, type / after 8000 and check result also type /about and check result

Now we will add HTML templates in this

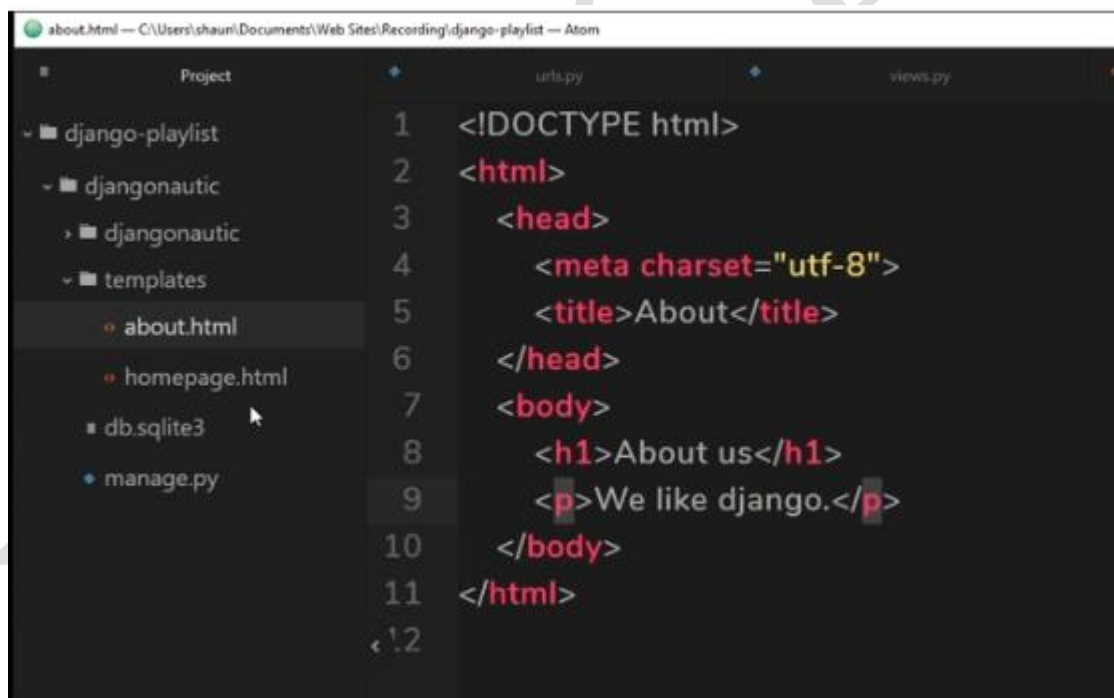
- For that create a templates directory (folder) in the root app folder
- Now go to that folder and create two templates for about page as about.html and for homepage as homepage.html

Our homepage.html will look like this



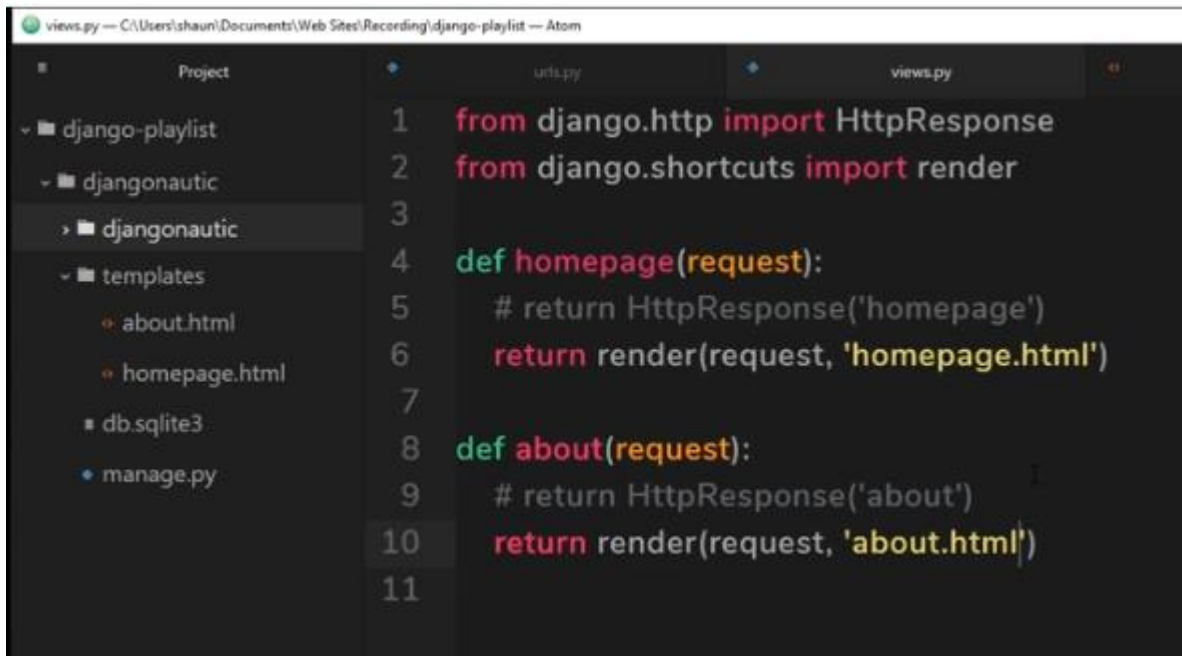
```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="utf-8">
5     <title>Homepage</title>
6   </head>
7   <body>
8     <h1>This is the homepage</h1>
9     <p>Welcome to Djangonautic</p>
10  </body>
11 </html>
```

Now our about.html will look like this



```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="utf-8">
5     <title>About</title>
6   </head>
7   <body>
8     <h1>About us</h1>
9     <p>We like django.</p>
10  </body>
11 </html>
```

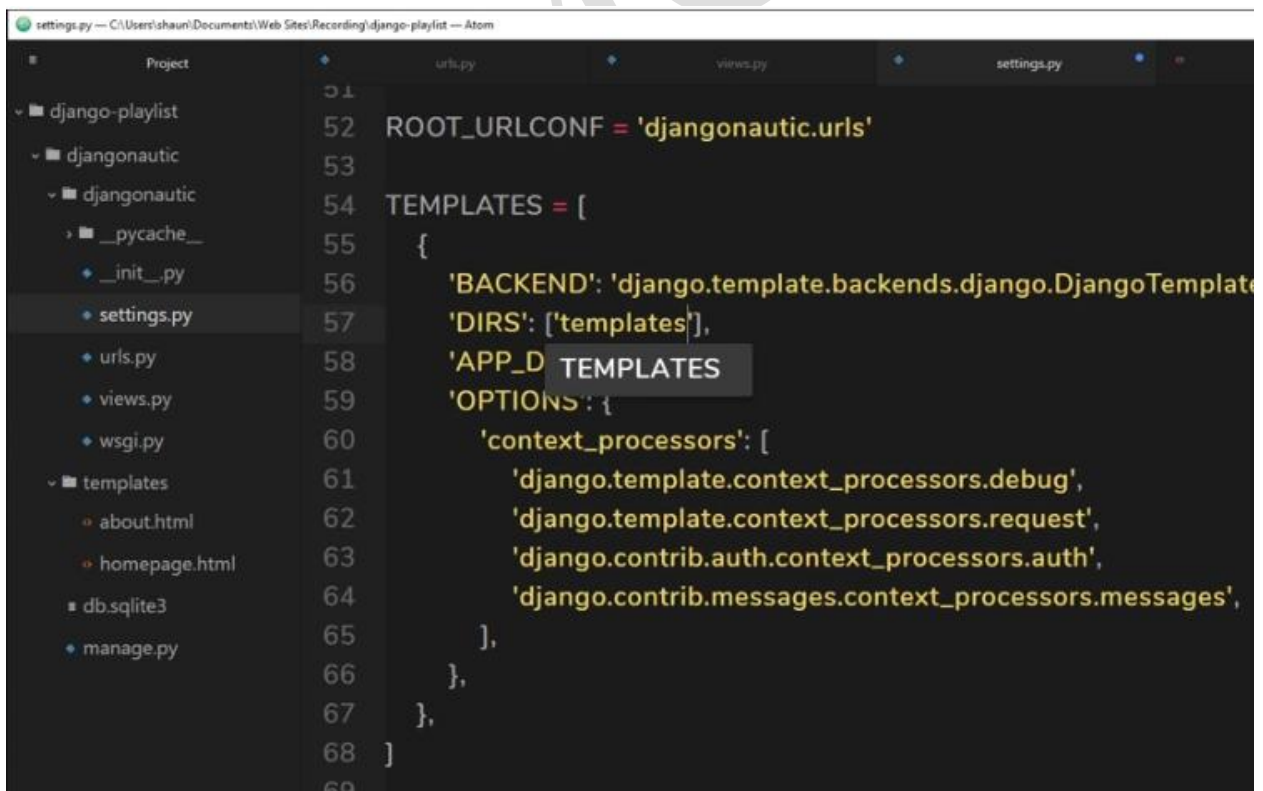
Now update our views.py file as

A screenshot of the Atom text editor. The left sidebar shows a project tree with folders 'django-playlist', 'djangonautic', and 'templates'. The 'djangonautic' folder is expanded, showing 'about.html' and 'homepage.html'. The 'views.py' file is open in the editor, showing Python code for Django views. The code includes imports for 'HttpResponse' and 'render', and two view functions: 'homepage' and 'about'.

```
1 from django.http import HttpResponse
2 from django.shortcuts import render
3
4 def homepage(request):
5     # return HttpResponse('homepage')
6     return render(request, 'homepage.html')
7
8 def about(request):
9     # return HttpResponse('about')
10    return render(request, 'about.html')
11
```

Now when we run this file we will get error, template does not exist

For that we have to go to setting file- scroll down and check for templates and now update our templates folder name in that DIR - empty square bracket as follows

A screenshot of the Atom text editor. The left sidebar shows the project tree with 'settings.py' selected. The 'settings.py' file is open in the editor, showing Django settings. The 'TEMPLATES' setting is highlighted, and the 'DIRS' list is being edited to include an empty string for the current directory.

```
52 ROOT_URLCONF = 'djangonautic.urls'
53
54 TEMPLATES = [
55     {
56         'BACKEND': 'django.template.backends.django.DjangoTemplateBackend',
57         'DIRS': ['templates'],
58         'APP_DIRS': True,
59         'OPTIONS': {
60             'context_processors': [
61                 'django.template.context_processors.debug',
62                 'django.template.context_processors.request',
63                 'django.contrib.auth.context_processors.auth',
64                 'django.contrib.messages.context_processors.messages',
65             ],
66         },
67     ],
68 ]
```

Now run our app now.

Steps for creating apps in Django

- Create app using the command as
→ `python manage.py startapp (app_name)`
- Create an `urls.py` file in that app folder, as we don't have it over there

Steps for using database in Django

- Create super user as
→ `python manage.py createsuperuser`
After executing this command follow the instructions
 - * it will ask for username, by default it will take some value. If we want to change the username we need to type that name here
 - * then it will ask for email address – its optional
 - * then it will ask for password – password should be strong
 - * need to enter same password againAnd now our user is created
Here our username is admin and password is (admin1234)

Conclusion:

Result:

Experiment No. 10

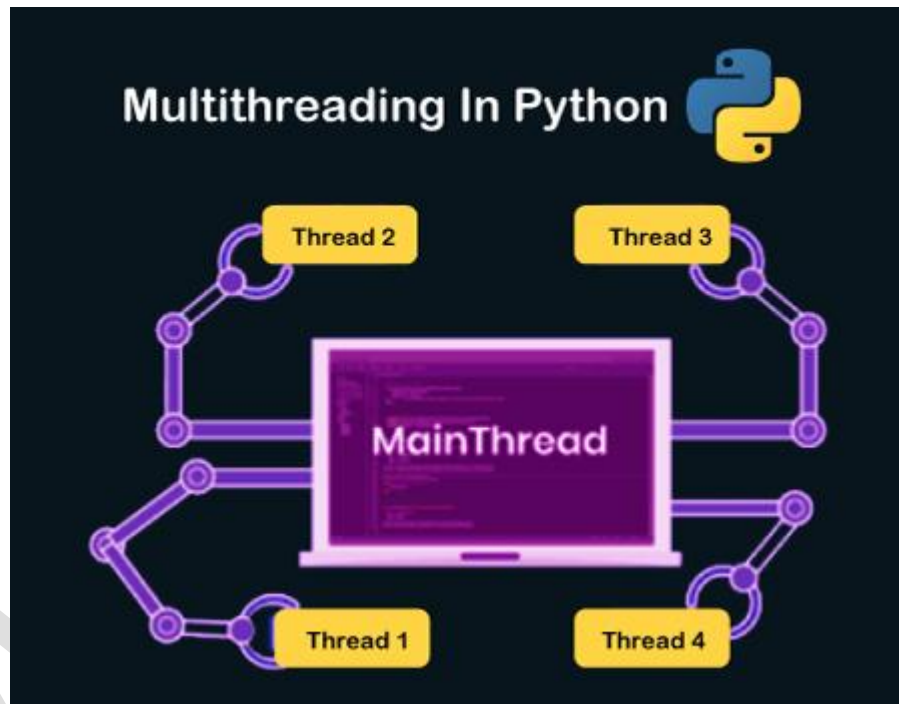
Aim: Programs on Threading using python.

Software Required: Python3.9 / Online Editor

Theory:

Multithreading in Python 3

A thread is the smallest unit of a program or process executed independently or scheduled by the Operating System. In the computer system, an Operating System achieves multitasking by dividing the process into threads. A thread is a lightweight process that ensures the execution of the process separately on the system. In Python 3, when multiple processors are running on a program, each processor runs simultaneously to execute its tasks separately.



Python Multithreading

Multithreading is a threading technique in Python programming to run multiple threads concurrently by rapidly switching between threads with a CPU help (called context switching). Besides, it allows sharing of its data space with the main threads inside a process that share information and communication with other threads easier than individual processes. Multithreading aims to perform multiple tasks simultaneously, which increases performance, speed and improves the rendering of the application.

Note: The Python Global Interpreter Lock (GIL) allows running a single thread at a time, even the machine has multiple processors.

Benefits of Multithreading in Python

Following are the benefits to create a multithreaded application in Python, as follows:

- 1. It ensures effective utilization of computer system resources.*
- 2. Multithreaded applications are more responsive.*
- 3. It shares resources and its state with sub-threads (child) which makes it more economical.*
- 4. It makes the multiprocessor architecture more effective due to similarity.*
- 5. It saves time by executing multiple threads at the same time.*
- 6. The system does not require too much memory to store multiple threads.*

When to use Multithreading in Python?

*It is a very useful technique for time-saving and improving the performance of an application. Multithreading allows the programmer to divide application **tasks** into sub-tasks and simultaneously run them in a program. It allows threads to communicate and share resources such as files, data, and memory to the same processor. Furthermore, it increases the user's responsiveness to continue running a program even if a part of the application is the length or blocked*

How to achieve multithreading in Python?

There are two main modules of multithreading used to handle threads in Python

- 1. The thread module*
- 2. The threading module*

Thread modules

*It is started with Python 3, designated as obsolete, and can only be accessed with **_thread** that supports backward compatibility.*

Syntax:

`thread.start_new_thread (function_name, args[, kwargs])`

*To implement the thread module in Python, we need to import a **thread** module and then define a function that performs some action by setting the target with a variable.*

Threading Modules

The threading module is a high-level implementation of multithreading used to deploy an application in Python

To use multithreading, we need to import the threading module in Python Program.

Thread Class Methods

Methods	Description
start()	<i>A start() method is used to initiate the activity of a thread. And it calls only once for each thread so that the execution of the thread can begin.</i>
run()	<i>A run() method is used to define a thread's activity and can be overridden by a class that extends the threads class.</i>
join()	<i>A join() method is used to block the execution of another code until the thread terminates.</i>

Follow the given below steps to implement the threading module in Python Multithreading:

1. Import the threading module

*Create a new thread by importing the **threading** module, as shown.*

Syntax:

```
import threading
```

*A **threading** module is made up of a **Thread** class, which is instantiated to create a Python thread.*

2. Declaration of the thread parameters: *It contains the target function, argument, and kwargs as the parameter in the **Thread()** class.*

- **Target:** *It defines the function name that is executed by the thread.*
- **Args:** *It defines the arguments that are passed to the target function name.*

For example:

```
import threading
```

```
def print_hello(n):
```



```
print("Hello, how old are you ", n)
```

```
t1 = threading.Thread( target = print_hello, args =(18, ))
```

In the above code, we invoked the **print_hello()** function as the target parameter. The **print_hello()** contains one parameter **n**, which passed to the **args** parameter.

3. Start a new thread: To start a thread in Python multithreading, call the thread class's object. The **start()** method can be called once for each thread object; otherwise, it throws an exception error.

Syntax:

```
t1.start()
```

```
t2.start()
```

4. Join method: It is a **join()** method used in the thread class to halt the main thread's execution and waits till the complete execution of the thread object. When the thread object is completed, it starts the execution of the main thread in Python.

Joinmethod.py

```
1. import threading
2. def print_hello(n):
3.     Print("Hello, how old are you? ", n)
4.     T1 = threading.Thread( target = print_hello, args = (20, ))
5.     T1.start()
6.     T1.join()
7.     Print("Thank you")
```

Output:

```
Hello, how old are you? 20
```

```
Thank you
```

When the above program is executed, the **join()** method halts the execution of the main thread and waits until the thread **t1** is completely executed. Once the **t1** is successfully executed, the main thread starts its execution.

Note: If we do not use the **join()** method, the interpreter can execute any print statement inside the Python program. Generally, it executes the first print statement because the interpreter executes the lines of codes from the program's start.

5. Synchronizing Threads in Python

It is a thread synchronization mechanism that ensures no two threads can simultaneously execute a particular segment inside the program to access the shared resources. The situation may be termed as critical sections. We use a race condition to avoid the critical section condition, in which two threads do not access resources at the same time.

Conclusion:

Result:

Experiment No. 11

Aim: Exploring basics of NumPy Methods.

Software Required: Python3.9 / Online Editor

Theory:

What are NumPy Arrays?

NumPy is a Python package that stands for 'Numerical Python'. It is the core library for scientific computing, which contains a powerful n-dimensional array object.

Where is NumPy used?

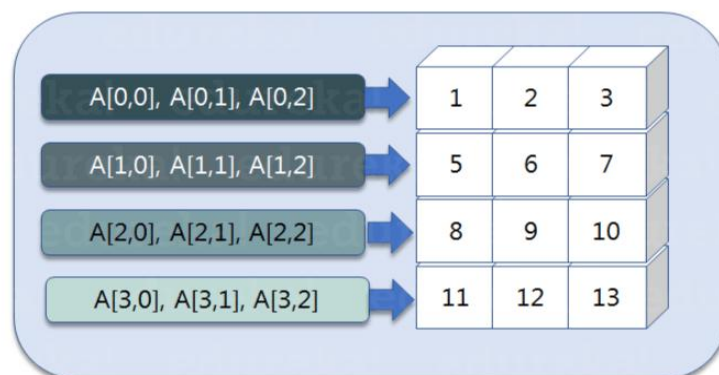
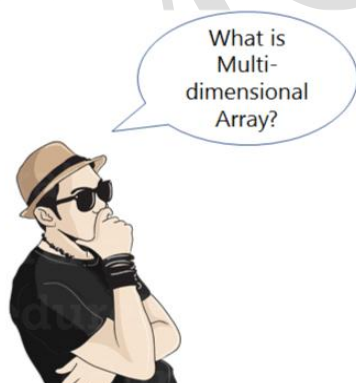
Python NumPy arrays provide tools for integrating C, C++, etc. It is also useful in linear algebra, random number capability etc. NumPy array can also be used as an efficient multi-dimensional container for generic data. Now, let me tell you what exactly is a Python NumPy array.

Python NumPy Array: Numpy array is a powerful N-dimensional array object which is in the form of rows and columns. We can initialize NumPy arrays from nested Python lists and access its elements. In order to perform these NumPy operations, the next question which will come in your mind is:

How do I install NumPy?

To install Python NumPy, go to your command prompt and type "pip install numpy". Once the installation is completed, go to your IDE (For example: PyCharm) and simply import it by typing: "import numpy as np"

let us understand what exactly is a multi-dimensional numPy array.



Here, I have different elements that are stored in their respective memory locations. It is said to be two dimensional because it has rows as well as columns. In the above image, we have 3 columns and 4 rows available.

How do I start NumPy?

Single-dimensional Numpy Array:

```
1import numpy as np
2a=np.array([1,2,3])
3print(a)
```

Output – []

Multi-dimensional Array:

```
1a=np.array([(1,2,3),(4,5,6)])
2print(a)
```

O/P – [[]
[]]

Python NumPy Array v/s List

Why NumPy is used in Python?

We use python NumPy array instead of a list because of the below three reasons:

1. *Less Memory*
2. *Fast*
3. *Convenient*

The very first reason to choose python NumPy array is that it occupies less memory as compared to list. Then, it is pretty fast in terms of execution and at the same time, it is very convenient to work with NumPy. So these are the major advantages that Python NumPy array has over list.

```
1import numpy as np
2
3import time
4import sys
5S= range(1000)
6print(sys.getsizeof(5)*len(S))
7
8D= np.arange(1000)
9print(D.size*D.itemsize)
```

O/P –

The above output shows that the memory allocated by list (denoted by S) is 14000 whereas the memory allocated by the NumPy array is just 4000. From this, you can conclude that there is a major difference between the two and this makes Python NumPy array as the preferred choice over list.

Next, let's see how python NumPy array is faster and more convenient when compared to list.

```
1import time
```

```

2import sys
3
4SIZE = 1000000
5
6L1= range(SIZE)
7L2= range(SIZE)
8A1= np.arange(SIZE)
9A2=np.arange(SIZE)
10
11start= time.time()
12result=[(x,y) for x,y in zip(L1,L2)]
13print((time.time()-start)*1000)
14
15start=time.time()
16result= A1+A2
17print((time.time()-start)*1000)

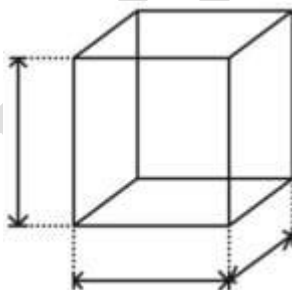
```

O/P –

In the above code, we have defined two lists and two numpy arrays. Then, we have compared the time taken in order to find the sum of lists and sum of numpy arrays both. If you see the output of the above program, there is a significant change in the two values. List took 380ms whereas the numpy array took almost 49ms. Hence, numpy array is faster than list. Now, if you noticed we had run a 'for' loop for a list which returns the concatenation of both the lists whereas for numpy arrays, we have just added the two array by simply printing $A1+A2$. That's why working with numpy is much easier and convenient when compared to the lists.

Therefore, the above examples proves the point as to why you should go for python numpy array

Python NumPy Operations



➤ **ndim:**

You can find the dimension of the array, whether it is a two-dimensional array or a single dimensional array. So, let us see this practically how we can find the dimensions. In the below code, with the help of 'ndim' function, I can find whether the array is of single dimension or multi dimension.

```

1import numpy as np
2a = np.array([(1,2,3),(4,5,6)])
3print(a.ndim)

```

Output –

Since the output is 2 it is a two-dimensional array (multi dimension).

➤ **itemsize:**

You can calculate the byte size of each element. In the below



code, I have defined a single dimensional array and with the help of 'itemsize' function, we can find the size of each element.

```
1import numpy as np
2a = np.array([(1,2,3)])
3print(a.itemsize)
```

Output –

So every element occupies _____ byte in the above numpy array.

- **dtype:**

You can find the data type of the elements that are stored in an array. So, if you want to know the data type of a particular element, you can use 'dtype' function which will print the datatype along with the size. In the below code, I have defined an array where I have used the same function.

```
1import numpy as np
2a = np.array([(1,2,3)])
3print(a.dtype)
```

Output –

As you can see, the data type of the array is _____. Similarly, you can find the size and shape of the array using 'size' and 'shape' function respectively.

```
1import numpy as np
2a = np.array([(1,2,3,4,5,6)])
3print(a.size)
4print(a.shape)
Output –
```

Next, let us move forward and see what are the other operations that you can perform with python numpy module. We can also perform reshape as well as slicing operation using python numpy operation. But, what exactly is reshape and slicing?

reshape:

Reshape is when you change the number of rows and columns which gives a new view to an object. Now, let us take an example to reshape the below array:



As you can see in the above image, we have 3 columns and 2 rows which has converted into 2 columns and 3 rows. Let me show you practically how it's done.

edureka!

```

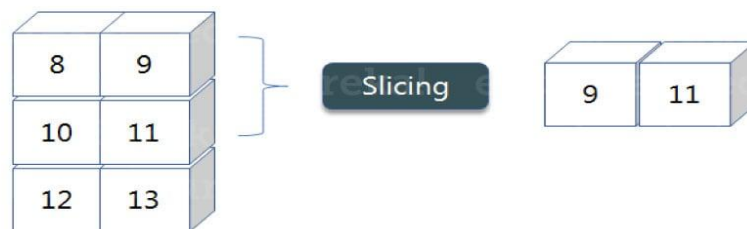
1import numpy as np
2a = np.array([(8,9,10),(11,12,13)])
3print(a)
4a=a.reshape(3,2)
5print(a)

```

Output – `[[8 9] [10 11] [12 13]]`

slicing:

As you can see the 'reshape' function has showed its magic. Now, let's take another operation i.e Slicing. Slicing is basically extracting particular set of elements from an array. This slicing operation is pretty much similar to the one which is there in the list as well. Consider the following example:



Before getting into the above example, let's see a simple one. We have an array and we need a particular element (say 3) out of a given array. Let's consider the below example:

```

1import numpy as np
2a=np.array([(1,2,3,4),(3,4,5,6)])
3print(a[0,2])

```

Output –

Here, the array(1,2,3,4) is your index 0 and (3,4,5,6) is index 1 of the python numpy array. Therefore, we have printed the second element from the zeroth index.

Taking one step forward, let's say we need the 2nd element from the zeroth and first index of the array. Let's see how you can perform this operation:

```

1import numpy as np
2a=np.array([(1,2,3,4),(3,4,5,6)])
3print(a[0:,2])

```

Output – `[3 5]`

Here colon represents all the rows, including zero. Now to get the 2nd element, we'll call index 2 from both of the rows which gives us the value 3 and 5 respectively.


```
1a= np.array([(1,2,3),(3,4,5)])
2print(a.sum(axis=0))
```

Output – [4 6 8]

Therefore, the sum of all the columns are added where $1+3=4$, $2+4=6$ and $3+5=8$. Similarly, if you replace the axis by 1, then it will print [6 12] where all the rows get added.

Square Root & Standard Deviation

There are various mathematical functions that can be performed using python numpy. You can find the square root, standard deviation of the array. So, let's implement these operations:

```
1import numpy as np
2a=np.array([(1,2,3),(3,4,5)])
3print(np.sqrt(a))
4print(np.std(a))
```

Output – [[1.41421356 1.73205081 1.73205081]
 [1.73205081 2.0 2.23606798]]

As you can see the output above, the square root of all the elements are printed. Also, the standard deviation is printed for the above array i.e how much each element varies from the mean value of the python numpy array.

Addition Operation

You can perform more operations on numpy array i.e addition, subtraction, multiplication and division of the two matrices.

```
1import numpy as np
2x= np.array([(1,2,3),(3,4,5)])
3y= np.array([(1,2,3),(3,4,5)])
4print(x+y)
```

Output – [[2 4 6]
 [6 8 10]]

This is extremely simple! Right? Similarly, we can perform other operations such as subtraction, multiplication and division. Consider the below example:

```
1import numpy as np
2x= np.array([(1,2,3),(3,4,5)])
3y= np.array([(1,2,3),(3,4,5)])
4print(x-y)
5print(x*y)
6print(x/y)
```

Output – `[[[[]]]]`

Vertical & Horizontal Stacking

Next, if you want to concatenate two arrays and not just add them, you can perform it using two ways – vertical stacking and horizontal stacking.

```
1import numpy as np
2x=np.array([(1,2,3),(3,4,5)])
3y= np.array([(1,2,3),(3,4,5)])
4print(np.vstack((x,y)))
5print(np.hstack((x,y)))
```

Output – `[[[[]]]]`

ravel

There is one more operation where you can convert one numpy array into a single column i.e ravel.

```
1import numpy as np
2x=np.array([(1,2,3),(3,4,5)])
3print(x.ravel())
```

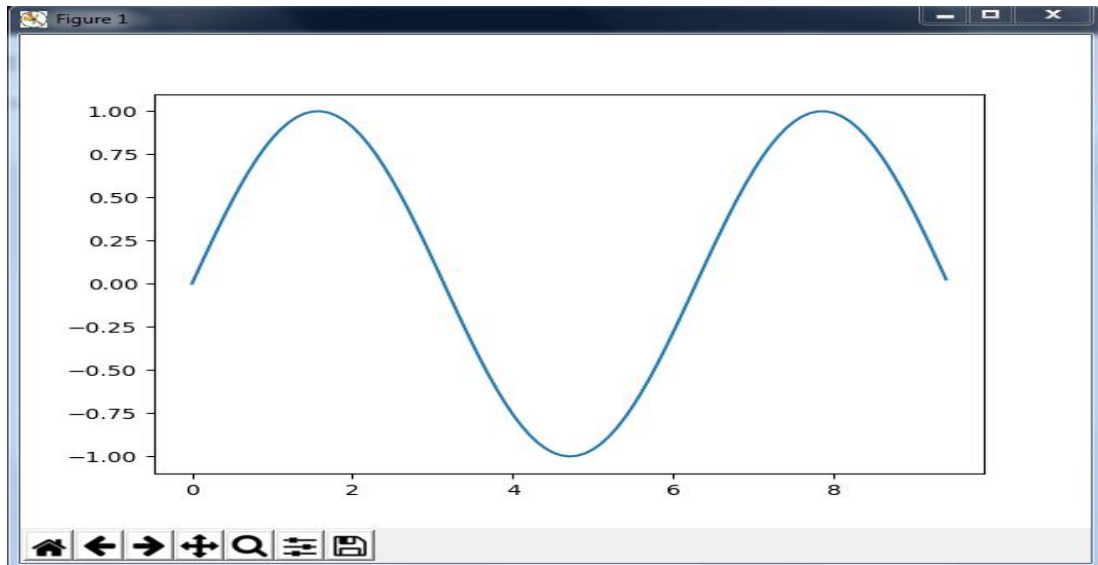
Output – `[]`

Python Numpy Special Functions

There are various special functions available in numpy such as sine, cosine, tan, log etc. First, let's begin with sine function where we will learn to plot its graph. For that, we need to import a module called matplotlib. let's see how these graphs are plotted.

```
1import numpy as np
2import matplotlib.pyplot as plt
3x= np.arange(0,3*np.pi,0.1)
4y=np.sin(x)
5plt.plot(x,y)
6plt.show()
```

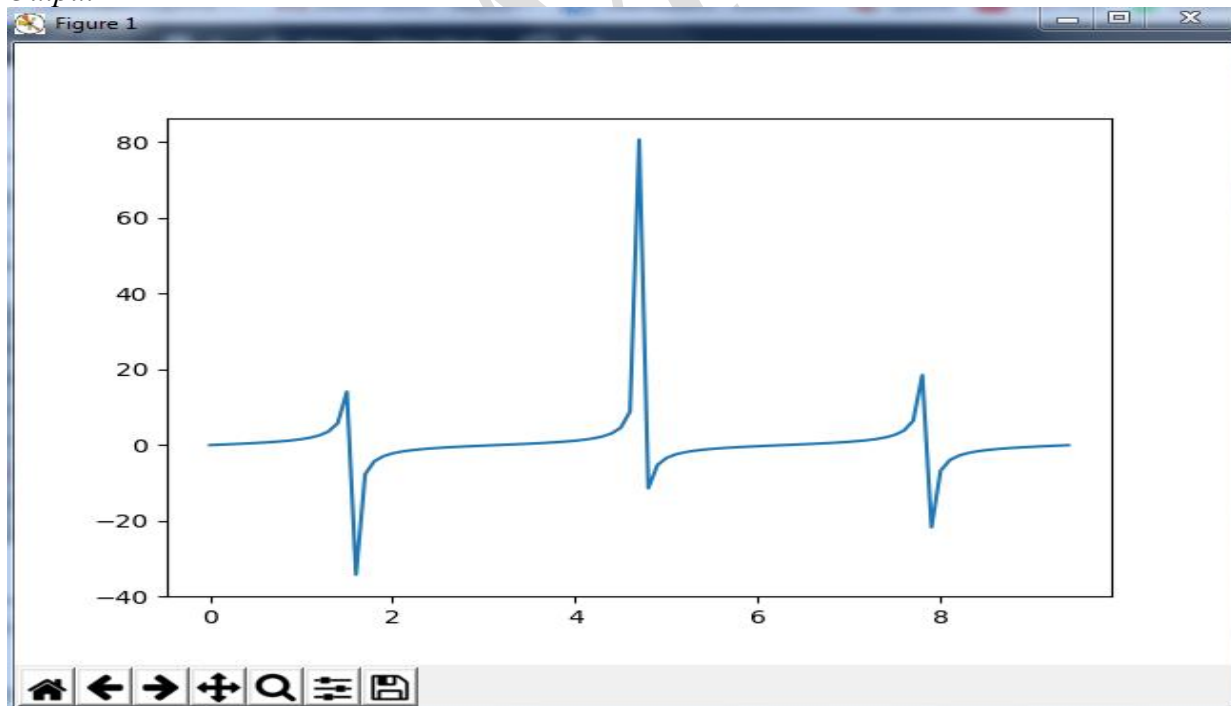
Output –



Similarly, you can plot a graph for any trigonometric function such as cos, tan etc. let's say **tan**.

```
1import numpy as np
2import matplotlib.pyplot as plt
3x= np.arange(0,3*np.pi,0.1)
4y=np.tan(x)
5plt.plot(x,y)
6plt.show()
```

Output –



let's see some other special functionality in numpy array such as exponential and logarithmic function. Now in exponential, the e value is somewhere equal to 2.7 and in log, it is actually log

base 10. When we talk about natural log i.e log base e, it is referred as Ln. So let's see how it is implemented practically:

```
1a= np.array([1,2,3])
2print(np.exp(a))
```

Output – []

As you can see the above output, the exponential values are printed i.e e raise to the power 1 is e, which gives the result as 2.718... Similarly, e raise to the power of 2 gives the value somewhere near 7.38 and so on. Next, in order to calculate log, let's see how you can implement it:

```
1import numpy as np
2import matplotlib.pyplot as plt
3a= np.array([1,2,3])
4print(np.log(a))
```

Output – []

Here, we have calculated natural log which gives the value as displayed above. Now, if we want log base 10 instead of Ln or natural log, you can follow the below code:

```
1import numpy as np
2import matplotlib.pyplot as plt
3a= np.array([1,2,3])
4print(np.log10(a))
```

Output – []

Conclusion:

Result:

Experiment No. 12

Aim: Program to send email and read content of URL.

Software Required: Python3.9 / Online Editor

Theory:

Here, we are going to learn how to send a simple basic mail using Python code. Python, being a powerful language don't need any external library to import and offers a native library to send emails- "SMTP lib".

"smtplib" creates a Simple Mail Transfer Protocol client session object which is used to send emails to any valid email id on the internet. Different websites use different port numbers. Here, we are using a Gmail account to send a mail. Port number used here is '587'. And if you want to send mail using a website other than Gmail, you need to get the corresponding information.

Steps to send mail from Gmail account:

First of all, "smtplib" library needs to be imported.

After that, to create a session, we will be using its instance SMTP to encapsulate an SMTP connection.

```
s = smtplib.SMTP('smtp.gmail.com', 587)
```

In this, you need to pass the first parameter of the server location and the second parameter of the port to use. For Gmail, we use port number 587.

For security reasons, now put the SMTP connection in the TLS mode. TLS (Transport Layer Security) encrypts all the SMTP commands. After that, for security and authentication, you need to pass your Gmail account credentials in the login instance. The compiler will show an authentication error if you enter invalid email id or password.

Store the message you need to send in a variable say, message. Using the sendmail() instance, send your message. sendmail() uses three parameters: sender_email_id, receiver_email_id and message_to_be_sent. The parameters need to be in the same sequence.

This will send the email from your account. After you have completed your task, terminate the SMTP session by using quit().

```
# Python code to illustrate Sending mail from  
# your Gmail account  
import smtplib
```

```
# creates SMTP session  
s = smtplib.SMTP('smtp.gmail.com', 587)
```

```

# start TLS for security
s.starttls()

# Authentication
s.login("sender_email_id", "sender_email_id_password")

# message to be sent
message = "Message_you_need_to_send"

# sending the mail
s.sendmail("sender_email_id", "receiver_email_id", message)

# terminating the session
s.quit()
Sending same message to multiple people

```

If you need to send the same message to different people. You can use for loop for that. For example, you have a list of email ids to which you need to send the same mail. To do so, insert a “for” loop between the initialization and termination of the SMTP session. Loop will initialize turn by turn and after sending the email, SMTP session will be terminated.

```

# Python code to illustrate Sending mail
# to multiple users
# from your Gmail account
import smtplib

# list of email_id to send the mail
li = ["xxxxx@gmail.com", "yyyyy@gmail.com"]

for dest in li:
    s = smtplib.SMTP('smtp.gmail.com', 587)
    s.starttls()
    s.login("sender_email_id", "sender_email_id_password")
    message = "Message_you_need_to_send"
    s.sendmail("sender_email_id", dest, message)
    s.quit()

```

Important Points:

➤ *This code can send simple mail which doesn't have any attachment or any subject.*

- *One of the most amazing things about this code is that we can send any number of emails using this and Gmail mostly put your mail in the primary section. Sent mails would not be detected as Spam generally.*
- *File handling can also be used to fetch email id from a file and further used for sending the emails.*

Conclusion:

Result:

References:

1. <https://www.w3schools.com/python/>
2. <https://www.journaldev.com/15906/python-socket-programming-server-client>
3. <https://www.tutorialsteacher.com/python/database-crud-operation-in-python>
4. <https://www.pythontutorial.net/advanced-python/python-threading/>
5. <https://www.geeksforgeeks.org/multithreading-python-set-1/>
6. <https://www.javatpoint.com/multithreading-in-python-3>
7. <https://www.geeksforgeeks.org/send-mail-gmail-account-using-python/>