



Saraswati College Of Engineering, Kharghar Computer Department

**Saraswati College of Engineering,  
Kharghar**

**Name: Shaikh Adnan Shaukat Ali**

**Roll no: 55**

**Subject: Python**

**Class/Sem: SE/IV**

---

Saraswati College Of Engineering, Kharghar

Department of Computer Engineering

S.E.

# Open Source Tech Lab (OSTL)

## -:Lab Manual:-

SCOE's

Saraswati College Of Engineering, Kharghar  
Computer Department



## Saraswati College of Engineering, Kharghar

### Vision:

To become center of excellence in Engineering education and research.

### Mission:

To educate students to become quality technocrats for taking up challenges in all facets of life.

## Department of Computer Engineering

### Vision:

To imprint knowledge to our students to excel in engineering culture and research and nurture them to become ethically rich professionals.

### Mission:

To provide simulating learning environment with a technological orientation to maximize individual potential.



SARASWATI EDUCATION SOCIETY'S  
SARASWATI COLLEGE OF ENGINEERING,  
NAVI MUMBAI, KHARGHAR.

COMPUTER ENGINEERING DEPARTMENT

**Programme Outcomes (PO)**

At the end of the program, a student will be able to:

1. Apply knowledge of mathematics, science and engineering.
2. Utilize the computer engineering knowledge in all domains, viz., health care, banking and Finance, other professions such as medical, law, etc.
3. Design and conduct experiments as well as to analyze and interpret data.
4. Analyze the problem, subdivide it into smaller tasks with well-defined interface for interaction among components, and complete the task within the specified time frame and financial constraints,
5. Design a system, component or process to meet the desired needs within realistic constraints such as economic, environmental, social, political and Ethical ability,
6. Design, implement, and evaluate secure hardware and/or software systems with assured quality and efficiency,
7. Communicate effectively the engineering solution to customers/users or peers,
8. Understand professional and ethical responsibilities,
9. Understand contemporary issues and to get engaged in lifelong learning by independently and continually expanding knowledge and abilities,
10. Function in multidisciplinary teams,
11. Identify, formulate and solve engineering problems.



SARASWATI EDUCATION SOCIETY'S  
SARASWATI COLLEGE OF ENGINEERING,  
NAVI MUMBAI, KHARGHAR.

COMPUTER ENGINEERING DEPARTMENT

Program Educational Objectives (PEO)

1. To prepare the candidate for a successful career in the industry and make him acquainted with the latest software and hardware,
2. To enable student to work productively as computer engineers, including supportive teamwork and leadership roles on multidisciplinary teams,
3. Graduates are prepared to be responsible computing professionals in their own area of interest,
4. To provide the candidate with a sound foundation in mathematics, software technologies, database technologies, networking, hardware and to prepare them for post graduate studies and research programs,
5. To promote the awareness of lifelong learning among students and to introduce them to professional ethics and codes of professional practice,
6. To demonstrate effective communication skills in oral, written and electronic media.

---

Lab Code	Lab Name	Credit
<b>CSL405</b>	<b>Open Source Technology Lab</b>	<b>2</b>

**Course Outcomes:**

1. To understand basic concepts in python and perl.
2. To explore contents of files, directories and text processing with python
3. To develop program for data structure using built in functions in python.
4. To explore django web framework for developing python based web application.
5. To understand file handling and database handling using perl.
6. To explore basics of two way communication between client and server using python and perl

**Prerequisites:** Knowledge of some programming language like C, Java

**Content:**

Sr. No	Module Name	Detailed Content
1	Python basics	Data types in python ,Operators in python, Input and Output, Control statement, Arrays in python, String and Character in python, Functions, List and Tuples, Dictionaries Exception, Introduction to OOP, Classes , Objects , Interfaces, Inheritance
2	Advanced Python	Files in Python, Directories, Building Modules, Packages, Text Processing, Regular expression in python.
3	Data Structure in Python	Link List, Stack, Queues, Dequeues
4	Python Integration Primer	Graphical User interface ,Networking in Python , Python database connectivity, Introduction to Django
5	Basics of Perl	Perl Overview, Variables, Control Statements, Subroutines, Objects, Packages and Modules
6	Perl advanced	Working with Files, Data manipulation, Database Systems, Networking

**Text Books**

1. Core Python Programming, Dr. R. Nageswara Rao, Dreamtech Press
2. Beginning Python: Using Python 2.6 and Python 3.1. James Payne, Wrox publication
3. Perl: The Complete Reference. Second Edition. Martin C. Brown, McGraw-Hill
4. Introduction to computing and problem solving using python , E Balagurusamy, McGraw Hill Education

---

### Reference Book

1. Perl Black Book, 2nd Edition: Steven Holzner, Dreamtech Press
2. Learn Python the Hard Way: (3rd Edition) (Zed Shaw's Hard Way Series)
3. Python Projects , Laura Cassell, Alan Gauld, wrox publication

### Digital Material:

1. "The Python Tutorial", <http://docs.python.org/release/3.0.1/tutorial/>
2. Beginning Perl, <https://www.perl.org/books/beginning-perl/>
3. <http://spoken-tutorial.org>
4. [www.staredusolutions.org](http://www.staredusolutions.org)

### Suggested experiments using Python:

1. Exploring basics of python like data types (strings, list, array, dictionaries, set, tuples) and control statements.
2. Creating functions, classes and objects using python. Demonstrate exception handling and inheritance.
3. Exploring Files and directories
  - a. Python program to append data to existing file and then display the entire file
  - b. Python program to count number of lines, words and characters in a file.
  - c. Python program to display file available in current directory
4. Creating GUI with python containing widgets such as labels, textbox, radio, checkboxes and custom dialog boxes.
5. Menu driven program for data structure using built in function for link list, stack and queues.
6. Program to demonstrate CRUD( create, read, update and delete) operations on database (SQLite/ MySQL) using python.
7. Creation of simple socket for basic information exchange between server and client.
8. Creating web application using Django web framework to demonstrate functionality of user login and registration (also validating user detail using regular expression).

### Suggested experiments using Perl:

10. Exploring various data type , loops and conditional statement in perl. And Creating functions, packages and modules in perl.
11. Program to demonstrate use of objects and classes in perl.
12. Program to demonstrate file handling, data manipulation and use of regular expression for text processing in perl
13. Program to send email and read content of URL.

---

## SARASWATI COLLEGE OF ENGINEERING

### COMPUTER ENGINEERING DEPARTMENT

Subject : Open Source Tech Lab (OSTL)

Class/Sem: SE/IV

Name of the Laboratory: Multimedia Lab

Year: 2020-21

---

#### LIST OF EXPERIMENTS

Experiment No.	Name of the Experiment
1	Program to swap two numbers and check if first number is positive or negative or zero.
2	Program to check if number and string is palindrome and find factorial of input number.
3	Menu driven program to demonstrate use of list in python. A. Put even and odd elements into two different lists. B. Merge and sort two lists. C. Update first element with x value and delete middle element of list. D. Find max and min element from list. E. Add n names into the existing number list and check if word python is present in list.
4	Menu driven program to demonstrate use of tuples in python. A. Add and show N student roll number, name and 3 subject marks in a list of tuples. B. Display student roll number and marks whose name is python. C. Demonstrate nested tuple and sort nested tuple by name.
5	Menu driven program to demonstrate use of set in python A. Accept two strings from user B. Display common letters in two input strings(set intersection) C. Display letters which are in the first string but not in the second.(set difference) D. Display set of all letters of both the strings.(set union) E. Displays letters which are in the two strings but not in both.(symmetric difference)
6	Menu driven program to demonstrate use of dictionary in python A. Create key/value pair dictionary. B. Update/concatenate and delete item of existing dictionary. C. Find a key and print its value. D. Map two list into dictionary
7	Design an employee class using python for reading and displaying the employee information.



---

8	Program to demonstrate single and multiple inheritance in python (with method overloading and overriding)
9	Exception handling A. Write a program to demonstrate exception handling using try, multiple exception and finally. B. Write a python program to create user defined exception.
10	Exploring files and directories. A. Python program to read the content of file and write in another file. B. Program to append data to existing file and then display entire file. C. Program to count number of lines, words and characters in a file. D. Program to display file available in current directory.
11	Create a package and module for data structure :stack and queues.
12	Creation of simple socket for basic info exchange between server and client.
13	Connecting to database and executing basic commands.

H/W Requirement	P-IV and above, Ram 128 MB, Printer, Internet Connection
S/W Requirement	Python IDLE

Practical Incharge

H. O. D.

---



---

## EXPERIMENT NO. 1

<u>Aim</u>	:	Program to swap two numbers and check if first number is positive or negative or zero
<u>Resources Required</u>	:	Consumables – Printer Pages for printouts.
<u>Theory</u>	:	<p>Python is a general-purpose interpreted, interactive, object-oriented, and high- level programming language. It was created by Guido van Rossum during 1985- 1990. Like Perl, Python source code is also available under the GNU General Public License (GPL).</p> <p>There are 3 ways of executing a Python program</p> <ol style="list-style-type: none"> <li>1. Using Python’s command line window.</li> <li>2. Using Python’s IDLE graphics window.</li> <li>3. Directly from System prompt.</li> </ol> <p>The first two are called interactive modes where we can type the program one line at a time and the compiler executes it immediately. The last one is called non-interactive mode where we ask compiler to execute our program after typing the entire program. Programmers can store Python script source code in a file with the '.py' extension, and use the interpreter to execute the contents of the file. To execute the script by the interpreter, you have to tell the interpreter the name of the file. Decision making is required when we want to execute a code only if a certain condition is satisfied. The if...elif...else statement is used in Python for decision making. Syntax for if statement as follows:</p> <ol style="list-style-type: none"> <li>1. if expression:           <div style="margin-left: 40px;">//execute your code</div> </li> <li>2. if expression:           <div style="margin-left: 40px;">//execute your code</div>           else:           <div style="margin-left: 40px;">//execute your code</div> </li> <li>3. if expression:           <div style="margin-left: 40px;">//execute your code</div>           elif expression:           <div style="margin-left: 40px;">//execute your code</div>           else:           <div style="margin-left: 40px;">//execute your code</div> </li> </ol>
<u>Conclusion</u>	:	We have studied how to run python program and swapping of two

---

---

	numbers, control statement if else for find the given number is positive, negative or zero. By Student - <u>Adnan Shaikh</u>
--	---

**Code:**

```
num1, num2 = map(int,input("Enter Your Numbers: ").strip().split())
print(f'Bepfore Swapping: {num1 }, {num2}')
num1, num2 = num2, num1
print(f'After Swapping: {num1 }, {num2}')
num1, num2 = num2, num1
if num1 > 0:
    print("First number is positive!!!")
elif num1 < 0:
    print("First number is negative!!!")
else:
    print("First number is zero!!!")
```

**Output:**

```
PS C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'> python exp1.py
Enter Your Numbers: 57 85
Bepfore Swapping: 57, 85
After Swapping: 85, 57
First number is positive!!!
PS C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'> □
```

---

---

## EXPERIMENT NO. 2

Aim	:	: Menu driven program to check if number and string is palindrome and find factorial of input number
Resources Required	:	Consumables – Printer Pages for printouts.
Theory	:	<p>Functions in python.</p> <p>The four steps to defining a function in Python are the following:</p> <ol style="list-style-type: none"><li>1. Use the keyword def to declare the function and follow this up with the function name.</li><li>2. Add arguments to the function: they should be within the parentheses of the function. End your line with a colon.</li><li>3. Add statements that the functions should execute.</li><li>4. End your function with a return statement if the function should output something. Without the return statement, your function will return an object None.</li><li>5. docstring is short for documentation string. It is used to explain in brief, what a function does.</li></ol> <p>Syntax for functions in python:</p> <pre>def function_name(parameters):     """docstring"""     statement(s)</pre> <p>A palindrome is a string which is same read forward or backwards. For example: "dad" is the same in forward or reverse direction. Another example is "aibohphobia" which literally means, an irritable fear of palindromes. The factorial of a number is the product of all the integers from 1 to that number. For example, the factorial of 6 (denoted as 6!) is <math>1*2*3*4*5*6 = 720</math>. Factorial is not defined for negative numbers and the factorial of zero is one, <math>0! = 1</math>.</p>
Conclusion	:	<p>We have studied how to write functions in python program to find palindrome and factorial of number.</p> <p>By Student – <u>Adnan Shaikh</u></p>

---

**Code:**

```
def factorial(n):
    if n <= 1:
        return 1
    else:
        return n*factorial(n-1)

def palindrome(str):
    if str == str[::-1]:
        print(f"{str} is palindrome!!!")
    else:
        print(f"{str} is not a palindrome!!!")

if __name__ == "__main__":

    while(1):
        num = int(input("*****Menu Driven*****\n1. To calculate factorial\n2. To
check given number or string is palindrome!!!!\n3. Exit\n"))
        if num == 1:
            n = int(input("Enter your number: "))
            z = factorial(n)
            print(f"{n}! = {z}")
        elif num == 2:
            palindrome(input("Enter number or a string: "))
        elif num == 3:
            break
        else:
            print("you pressed wrong key please try again")
```

**Output:**

```
PS C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp2> python exp2.py
*****Menu Driven*****
1. To calculate factorial
2. To check given number or string is palindrome!!!!
3. Exit
1
Enter your number: 10
10! = 3628800
*****Menu Driven*****
1. To calculate factorial
2. To check given number or string is palindrome!!!!
3. Exit
2
Enter number or a string: 565
565 is palindrome!!!
*****Menu Driven*****
1. To calculate factorial
2. To check given number or string is palindrome!!!!
3. Exit
2
Enter number or a string: maam
maam is palindrome!!!
*****Menu Driven*****
1. To calculate factorial
2. To check given number or string is palindrome!!!!
3. Exit
2
Enter number or a string: hello
hello is not a palindrome!!!
*****Menu Driven*****
1. To calculate factorial
2. To check given number or string is palindrome!!!!
3. Exit
4
you pressed wrong key please try again
*****Menu Driven*****
1. To calculate factorial
2. To check given number or string is palindrome!!!!
3. Exit
3
PS C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp2> □
```

---

---

### EXPERIMENT NO. 3

Aim	:	Write a Menu driven program to demonstrate use of list in python. A. Put even and odd elements into two different lists. B. Merge and sort two lists. C. Update first element with x value and delete middle element of list. D. Find max and min element from list. E. Add n names into the existing number list and check if word python is present in list.
Resources Required	:	Consumables – Printer Pages for printouts.
Theory	:	<p>Lists:</p> <p>Lists are positionally ordered collections of arbitrarily typed objects, and they have no fixed size and they are mutable. Lists are contained in square brackets []. Lists can contain numbers, strings, nested sublists, or nothing</p> <p>Examples: L1 = [0,1,2,3],               L2 = ['zero', 'one'],               L3 = [0,1,[2,3],'three',['four,one']],               L4 = [].</p> <p>List indexing works just like string indexing. Lists are mutable: individual elements can be reassigned in place. Moreover, they can grow and shrink in place.</p> <p>Example:</p> <pre>&gt;&gt;&gt; L1 = [0,1,2,3] &gt;&gt;&gt; L1[0] = 4 &gt;&gt;&gt; L1[0] 4</pre> <p>Basic List Operations Lists respond to the + and * operators much like strings; they mean concatenation and repetition here too, except that the result is a new list, not a string. Some of basic operations of list are as follows.</p>

		Python includes the following list functions –		
		Sr. No.	Function	Description
		1	cmp(list1, list2)	Compares elements of both lists.
		2	len(list)	Gives the total length of the list.
		3	max(list)	Returns item from the list with max value.
		4	min(list)	Returns item from the list with min value.
		5	list(seq)	Converts a tuple into list.
		Python includes following list methods:		
		Sr.No	Methods	Description
		1	list.append(obj)	Appends object obj to list 2
		2	list.count(obj)	Returns count of how many times obj occurs in list
		3	list.extend(seq)	Appends the contents of seq to list
		4	list.index(obj)	Returns the lowest index in list that obj appears
		5	list.insert(index, obj)	Inserts object obj into list at offset index
		6	list.pop(obj=list[-1])	Removes and returns last object or obj from list
		7	list.remove(obj)	Removes object obj from list
		8	list.reverse()	Reverses objects of list in place
Conclusion	:	: Hence from above experiment student can understand that basic concept of list and list related various operation that create the list, update the list, merge the list etc. By Student - <u>Adnan Shaikh</u>		

---

**Code:**

```
def even_odd(l = []):
    even, odd = [], []
    for x in l:
        if x%2 == 0:
            even.append(x)
        else:
            odd.append(x)
    return even,odd

def merge_sort(l1 = [],l2 = []):
    lis = [*l1,*l2]
    return sorted(lis)

def update_delete(l = []):
    l[0] = input("Enter First value to update: ")
    del l[len(l)//2]
    return l

def minmax(l = []):
    return min(l),max(l)

def searching(l = []):
    num = input("Do you want to add element Y/N:").strip()
    if num.lower() == "y":
        l.append(input("Enter the element you want to append: "))
        searching(l)
    elif num.lower() == "n":
        if "python" in l:
            print("python is present in the list at position: ",l.index("python"))
        elif "Python" in l:
            print("python is present in the list at position: ",l.index("Python"))
        else:
            print("Python is not present in the list")
    else:
        print("You pressed wrong key please try again")
        searching(l)

if __name__ == "__main__":

    while(1):
        num = int(input("*****Menu Driven*****\n1. Separate Even and odd elements\n2. Merge and sort two lists \n3. Updating first element and deleting middle element\n4. Minimum and maximum value of a list\n5. Add names into list and check for python\n6. Exit\n"))
        if num == 1:
            l = list(map(int,input("Enter the element of the list: ").strip().split()))
            l1,l2 = even_odd(l)
            print(f"Even list: {l1}\nOdd list: {l2}")
        elif num == 2:
            l1 = list(map(int,input("Enter the element of first list: ").strip().split()))
            l2 = list(map(int,input("Enter the element of second list: ").strip().split()))
            l = merge_sort(l1,l2)
            print("Merged and sorted list: {}".format(l))
        elif num == 3:
```

---



```

l = list(map(str,input("Enter the element of the list: ").strip().split()))
l = update_delete(l)
print(f"Modified list = {l}")
elif num == 4:
    l = list(map(int,input("Enter the element of the list: ").strip().split()))
    mini, maxi = minmax(l)
    print(f"Maximum element of the list: {maxi}\nMinimum element of the list:{mini}")
elif num == 5:
    searching([])
elif num == 6:
    break
else:
    print("you pressed wrong key please try again")

```

### Output:

```

PS C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python\exp3> python exp3.py
*****Menu Driven*****
1. Separate Even and odd elements
2. Merge and sort two lists
3. Updating first element and deleting middle element
4. Minimum and maximum value of a list
5. Add names into list and check for python
6. Exit
1
Enter the element of the list: 56 89 2 6 546 464 5
Even list: [56, 2, 6, 546, 464]
Odd list: [89, 5]
*****Menu Driven*****
1. Separate Even and odd elements
2. Merge and sort two lists
3. Updating first element and deleting middle element
4. Minimum and maximum value of a list
5. Add names into list and check for python
6. Exit
2
Enter the element of first list: 46 11 78 22 662 45 798 223 69
Enter the element of second list: 12 46 889 2 54
Merged and sorted list: [2, 11, 12, 22, 45, 46, 46, 54, 69, 78, 223, 662, 798, 889]
*****Menu Driven*****
1. Separate Even and odd elements
2. Merge and sort two lists
3. Updating first element and deleting middle element
4. Minimum and maximum value of a list
5. Add names into list and check for python
6. Exit
3
Enter the element of the list: apple mango orange pineapple watermelon
Enter First value to update: grapes
Modified list = ['grapes', 'mango', 'pineapple', 'watermelon']
*****Menu Driven*****
1. Separate Even and odd elements
2. Merge and sort two lists
3. Updating first element and deleting middle element
4. Minimum and maximum value of a list
5. Add names into list and check for python
6. Exit
4
Enter the element of the list: 89 56 1 -999 36 56 45 2
Maximum element of the list: 89
Minimum element of the list:-999

```

```

PS C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python\exp3> python exp3.py
*****Menu Driven*****
1. Separate Even and odd elements
2. Merge and sort two lists
3. Updating first element and deleting middle element
4. Minimum and maximum value of a list
5. Add names into list and check for python
6. Exit
5
Do you want to add element Y/N: y
Enter the element you want to append: Ruby on rail
Do you want to add element Y/N:Y
Enter the element you want to append: R
Do you want to add element Y/N:Y
Enter the element you want to append: Java
Do you want to add element Y/N:d
You pressed wrong key please try again
Do you want to add element Y/N:Y
Enter the element you want to append: Java script
Do you want to add element Y/N:Y
Enter the element you want to append: Python
Do you want to add element Y/N:n
python is present in the list at position: 4
*****Menu Driven*****
1. Separate Even and odd elements
2. Merge and sort two lists
3. Updating first element and deleting middle element
4. Minimum and maximum value of a list
5. Add names into list and check for python
6. Exit
[]

```

---

---

## EXPERIMENT NO. 4

Aim	:	Menu driven program to demonstrate use of tuples in python. A. Add and show N student roll number, name and 3 subject marks in a list of tuples. B. Display student roll number and marks whose name is python. C. Demonstrate nested tuple and sort nested tuple by name.
Resources Required	:	Consumables – Printer Pages for printouts.
Theory	:	<p>A tuple is a sequence of immutable Python objects. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets. Creating a tuple is as simple as putting different comma-separated values. Optionally you can put these comma-separated values between parentheses also.</p> <p>For example</p> <pre>tup1 = ('physics', 'chemistry', 1997, 2000); tup2 = (1, 2, 3, 4, 5 ); tup3 = "a", "b", "c", "d";</pre> <p>Advantages of Tuple over List</p> <p>However, there are certain advantages of implementing a tuple over a list. Below listed are some of the main advantages:</p> <ul style="list-style-type: none"><li>• We generally use tuple for heterogeneous (different) datatypes and list for homogeneous (similar) datatypes.</li><li>• Since tuple are immutable, iterating through tuple is faster than with list. So there is a slight performance boost.</li><li>• Tuples that contain immutable elements can be used as key for a dictionary. With list, this is not possible.</li><li>• If you have data that doesn't change, implementing it as tuple will guarantee that it remains write-protected.</li></ul> <p>1. Creating a Tuple</p> <p>A tuple is created by placing all the items (elements) inside a parentheses (), separated by comma. The parentheses are optional but is a good practice to write it. A tuple can have any number of items and they may be of different types (integer, float, list, string etc.).</p> <p>2. Accessing Elements in a Tuple There are various ways in</p>

		<p>which we can access the elements of a tuple.</p> <p>a. Indexing</p> <p>We can use the index operator [] to access an item in a tuple where the index starts from 0. So, a tuple having 6 elements will have index from 0 to 5. Trying to access an element other than (6, 7,...) will raise an IndexError. The index must be an integer, so we cannot use float or other types. This will result into TypeError.</p> <p>b. Negative Indexing</p> <p>Python allows negative indexing for its sequences. The index of -1 refers to the last item, -2 to the second last item and so on.</p> <p>c. Changing a Tuple</p> <p>Unlike lists, tuples are immutable. This means that elements of a tuple cannot be changed once it has been assigned. But, if the element is itself a mutable datatype like list, its nested items can be changed.</p> <p>d. Python Tuple Methods</p> <p>Methods that add items or remove items are not available with tuple. Only the following two methods are available.</p> <table><tr><th>Method</th><th>Description</th></tr><tr><td>count(x)</td><td>Return the number of items that is equal to x</td></tr><tr><td>index(x)</td><td>Return index of first item that is equal to x</td></tr></table>	Method	Description	count(x)	Return the number of items that is equal to x	index(x)	Return index of first item that is equal to x
Method	Description							
count(x)	Return the number of items that is equal to x							
index(x)	Return index of first item that is equal to x							
Conclusion	:	<p>Hence from this experiment we learned the various functions of tuples like delete, modify or insert elements of tuples since tuple are immutable, so for that how to create a new tuple and store the updated elements in it.</p> <p>By Student - <u>Adnan Shaikh</u></p>						

---

**Code:**

```
class Student:
    def __init__(self):
        self.tup = tuple()

    def addinfo(self):
        roll, name = int(input("Enter roll no: ")), input("Enter name: ")
        sub1, sub2, sub3 = map(int, input("Enter Three subjects marks: ").strip().split())
        l = list(self.tup)
        l.append((roll, name, sub1, sub2, sub3))
        self.tup = tuple(l)
        print("Inserted Tuple:", self.tup[len(self.tup)-1])

    def fetchinfo(self):
        print(*[str(x)+"\n" for x in self.tup if x[1].lower() == "python"])

    def display(self):
        print(*[str(x)+"\n" for x in sorted(self.tup, key= lambda x: x[1])])

if __name__ == "__main__":
    stud = Student()
    while True:
        num = int(input("*****Menu Driven*****\n1. To add student information\n2. To fetch all students where, name == python\n3. To display all student information in sorted order.\n4. Exit\n"))
        if num == 1:
            stud.addinfo()
        elif num == 2:
            stud.fetchinfo()
        elif num == 3:
            stud.display()
        elif num == 4:
            break
        else:
            print("You pressed wrong key please try again")
```

## Output:

```
PS C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp4> python exp4.py
*****Menu Driven*****
1. To add student information
2. To fetch all students where, name == python
3. To display all student information in sorted order.
4. Exit
1
Enter roll no: 55
Enter name: Json
Enter Three subjects marks: 98 95 96
Inserted Tuple: (55, 'Json', 98, 95, 96)
*****Menu Driven*****
1. To add student information
2. To fetch all students where, name == python
3. To display all student information in sorted order.
4. Exit
1
Enter roll no: 15
Enter name: Adnan
Enter Three subjects marks: 55 65 60
Inserted Tuple: (15, 'Adnan', 55, 65, 60)
*****Menu Driven*****
1. To add student information
2. To fetch all students where, name == python
3. To display all student information in sorted order.
4. Exit
1
Enter roll no: 77
Enter name: Python
Enter Three subjects marks: 65 55 45
Inserted Tuple: (77, 'Python', 65, 55, 45)
*****Menu Driven*****
1. To add student information
2. To fetch all students where, name == python
3. To display all student information in sorted order.
4. Exit
1
Enter roll no: 88
Enter name: python
Enter Three subjects marks: 88 95 99
Inserted Tuple: (88, 'python', 88, 95, 99)

*****Menu Driven*****
1. To add student information
2. To fetch all students where, name == python
3. To display all student information in sorted order.
4. Exit
3
(15, 'Adnan', 55, 65, 60)
(55, 'Json', 98, 95, 96)
(77, 'Python', 65, 55, 45)
(88, 'python', 88, 95, 99)

*****Menu Driven*****
1. To add student information
2. To fetch all students where, name == python
3. To display all student information in sorted order.
4. Exit
2
(77, 'Python', 65, 55, 45)
(88, 'python', 88, 95, 99)

*****Menu Driven*****
1. To add student information
2. To fetch all students where, name == python
3. To display all student information in sorted order.
4. Exit
4
PS C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp4> 
```

---

---

## EXPERIMENT NO. 5

Aim	:	Write a menu driven program to demonstrate use of set in python A. Accept two strings from user B. Display common letters in two input strings (set intersection) C. Display letters which are in the first string but not in the second (set difference) D. Display set of all letters of both the strings (set union) E. Displays letters which are in the two strings but not in both (symmetric difference)
Resources Required	:	Consumables – Printer Pages for printouts.
Theory	:	<p>A Set is an unordered collection data type that is iterable, mutable and has no duplicate elements. Python's set class represents the mathematical notion of a set. The major advantage of using a set, as opposed to a list, is that it has a highly optimized method for checking whether a specific element is contained in the set. This is based on a data structure known as a <a href="#">hash table</a>.</p> <p>Methods for Sets:</p> <ol style="list-style-type: none"><li>1. add(x) Method: Adds the item x to set if it is not already present in the set.</li><li>2. union(s) Method: Returns a union of two set. Using the ' ' operator between 2 sets is the same as writing set1.union(set2)</li><li>3. intersect(s) Method: Returns an intersection of two sets. The '&amp;' operator comes can also be used in this case.</li><li>4. difference(s) Method: Returns a set containing all the elements of invoking set but not of the second set. We can use '-' operator here.</li><li>5. clear() Method: Empties the whole set.</li></ol>
Conclusion	:	<p>Hence from this experiment we learned the various operations of sets in python.</p> <p>By Student - <u>Adnan Shaikh</u></p>

---

**Code:**

#A accept 2 input string form user

```
def intake():
    str1 =input("Enter first string: ")
    str2 =input("Enter second string: ")
    print("first string is { } and second string is { }".format(str1, str2))
    print(str1, str2)
```

#B print common letter from input string(set insertion)

```
def com():
    str1 =input("Enter first string: ")
    str2 =input("Enter second string: ")
    setA=set(str1)
    setB=set(str2)
    common=list(setA&setB)
    print("Common letters in both the set is: ")
    for element in common:
        print(element)
```

#C Display letters which are in the first string but not in the second(set difference)

```
def diff():
    str1 =input("Enter first string: ")
    str2 =input("Enter second string: ")
    setA = set(str1)
    setB = set(str2)
    difference=list(setA-setB)
    print("Elements which are in A but not in B :")
    for element in difference:
        print(element)
```

#D Display set of common letters in both the strings (set union)

```
def un():
    str1 =input("Enter first string: ")
    str2 =input("Enter second string: ")
    setA = set(str1)
    setB = set(str2)
    union=list(setA|setB)
    print("Elements in both the sets: ")
    for element in union:
        print(element)
```

#E Displays letters which are in the two strings but not in both(symmetric difference)

```
def uncommon():
    str1 =input("Enter first string: ")
    str2 =input("Enter second string: ")
    setA = set(str1)
    setB = set(str2)
    uncommon=list(setA^setB)
    print("Element present in the strings but not in both the string: ")
    for element in uncommon:
        print(element)
```

```
if __name__=="__main__":
```

```
    while True:
```

---

```
num = int(input("*****Menu Driven*****\n1. TO input strings \n2.To check  
common letters between two strings \n3. To check letters which are in the first string but not in the  
second.\n4. To check set of common letters in both the strings \n5. To check letters which are in the  
both the strings but not in both \n6. Exit\n"))
```

```
if num == 1:  
    intake()  
elif num == 2:  
    com()  
elif num == 3:  
    diff()  
elif num ==4:  
    un()  
elif num==5:  
    uncommon()  
elif num==6:  
    break  
  
else:  
    print("Wrong key")
```

### Output:

```
PS C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp5> python exp-5.py  
*****Menu Driven*****  
1. TO input strings  
2.To check common letters between two strings  
3. To check letters which are in the first string but not in the second.  
4. To check set of common letters in both the strings  
5. To check letters which are in the both the strings but not in both  
6. Exit  
1  
Enter first string: Star Platinum  
Enter second string: Za Wurdoo  
first string is Star Platinum and second string is Za Wurdoo  
Star Platinum Za Wurdoo  
*****Menu Driven*****  
1. TO input strings  
2.To check common letters between two strings  
3. To check letters which are in the first string but not in the second.  
4. To check set of common letters in both the strings  
5. To check letters which are in the both the strings but not in both  
6. Exit  
2  
Enter first string: I, Giorno Giovanna, have a dream  
Enter second string: Kono Giorno Giovanna niwa yumegarui  
Common letters in both the set is:  
r  
a  
v  
  
o  
G  
e  
i  
m  
n
```



```

*****Menu Driven*****
1. TO input strings
2.To check common letters between two strings
3. To check letters which are in the first string but not in the second.
4. To check set of common letters in both the strings
5. To check letters which are in the both the strings but not in both
6. Exit
3
Enter first string: Korega Requiem Desuka
Enter second string: This is Requiem
Elements which are in A but not in B :
D
r
a
o
K
k
g
*****Menu Driven*****
1. TO input strings
2.To check common letters between two strings
3. To check letters which are in the first string but not in the second.
4. To check set of common letters in both the strings
5. To check letters which are in the both the strings but not in both
6. Exit
4
Enter first string: Jonathan Joester
Enter second string: Joseph Joester
Elements in both the sets:
t
o
h
e
p
n
r
a
s
J
*****Menu Driven*****
1. TO input strings
2.To check common letters between two strings
3. To check letters which are in the first string but not in the second.
4. To check set of common letters in both the strings
5. To check letters which are in the both the strings but not in both
6. Exit
5
Enter first string: Dempunki Berserk
Enter second string: Band of Hawk
Element present in the strings but not in both the string:
w
D
r
a
s
u
o
f
e
m
H
p
i
d
*****Menu Driven*****
1. TO input strings
2.To check common letters between two strings
3. To check letters which are in the first string but not in the second.
4. To check set of common letters in both the strings
5. To check letters which are in the both the strings but not in both
6. Exit
6
PS C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp5> 

```

---

---

## EXPERIMENT NO. 6

Aim	:	Menu driven program to demonstrate use of dictionary in python A. Create key/value pair dictionary. B. Update/concatenate and delete item of existing dictionary. C. Find a key and print its value. D. Map two lists into dictionary.
Resources Required	:	Consumables – Printer Pages for printouts.
Theory	:	<p>Dictionary in python consists of keys and their values.</p> <ol style="list-style-type: none"><li>1. Create a Python Dictionary Here is an example of how we can create a dictionary in Python :  <pre>&gt;&gt;&gt; myDict = {"A": "Apple", "B": "Boy", "C": "Cat"}</pre><p>In the above example: A dictionary is created. This dictionary contains three elements. Each element constitutes of a key value pair. This dictionary can be accessed using the variable myDict.</p></li><li>2. Access Dictionary Elements Once a dictionary is created, you can access it using the variable to which it is assigned during creation. For example, in our case, the variable myDict can be used to access the dictionary elements. Here is how this can be done :  <pre>&gt;&gt;&gt; myDict["A"] 'Apple' &gt;&gt;&gt; myDict["B"] 'Boy' &gt;&gt;&gt; myDict["C"] 'Cat'</pre><p>So you can see that using the variable myDict and Key as index, the value of corresponding key can be accessed. For those who have C/C++ background, it's more like accessing the value kept at a particular index in an array. If you just type the name of the variable myDict, all the key value pairs in the dictionary will be printed.</p><pre>&gt;&gt;&gt; myDict { 'A': 'Apple', 'C': 'Cat', 'B': 'Boy' }</pre><p>Only dictionary keys can be used as indexes. This means that myDict["A"] would produce 'Apple' in output but myDict["Apple"] cannot produce 'A' in the output.</p><pre>&gt;&gt;&gt; myDict["Apple"]</pre></li></ol>

		<p>Traceback (most recent call last): File "&lt;stdin&gt;", line 1, in &lt;module&gt; KeyError: 'Apple' So we see that the python compiler complained about 'Apple' being used as index.</p> <p>3. Update Dictionary Elements Just the way dictionary values are accessed using keys, the values can also be modified using the dictionary keys. Here is an example to modify python dictionary element:</p> <pre>&gt;&gt;&gt; myDict["A"] = "Application" &gt;&gt;&gt; myDict["A"] 'Application' &gt;&gt;&gt; myDict {'A': 'Application', 'C': 'Cat', 'B': 'Boy'}</pre> <p>You can see that in the example shown above, the value of key 'A' was changed from 'Apple' to 'Application' easily. This way we can easily conclude that there could not be two keys with same name in a dictionary.</p> <p>4. Delete Dictionary Elements Individual elements can be deleted easily from a dictionary. Here is an example to remove an element from dictionary.</p> <pre>&gt;&gt;&gt; myDict {'A': 'Application', 'C': 'Cat', 'B': 'Boy'} &gt;&gt;&gt; del myDict["A"] &gt;&gt;&gt; myDict {'C': 'Cat', 'B': 'Boy'}</pre> <p>So you can see that by using 'del' an element can easily be deleted from the dictionary.</p> <p>If you want to delete complete dictionary i.e all the elements in the dictionary then it can be done using the clear() function. Here is an example :</p> <pre>&gt;&gt;&gt; myDict {'C': 'Cat', 'B': 'Boy'} &gt;&gt;&gt; myDict.clear() &gt;&gt;&gt; myDict {}</pre> <p>So you see that all the elements were deleted making the dictionary empty.</p> <p>1. Dictionaries are Unordered</p> <pre>&gt;&gt;&gt; myDict = {"A": "Apple", "B": "Boy", "C": "Cat"} &gt;&gt;&gt; myDict {'A': 'Apple', 'C': 'Cat', 'B': 'Boy'}</pre> <p>You can observe that the order of elements while the dictionary was being created is different from the order in which they are actually stored and displayed.</p>
--	--	---

		<p>Even if you try to add other elements to python dictionary:</p> <pre>&gt;&gt;&gt; myDict["D"] = "Dog" &gt;&gt;&gt; myDict {'A': 'Apple', 'C': 'Cat', 'B': 'Boy', 'D': 'Dog'} &gt;&gt;&gt; myDict["E"] = "Elephant" &gt;&gt;&gt; myDict {'A': 'Apple', 'C': 'Cat', 'B': 'Boy', 'E': 'Elephant', 'D': 'Dog'}</pre> <p>You'll observe that it's not necessary that elements will be stored in the same order in which they were created.</p> <p>2. Dictionary Keys are Case Sensitive. Same key name but with different case are treated as different keys in python dictionaries. Here is an example :</p> <pre>&gt;&gt;&gt; myDict["F"] = "Fan" &gt;&gt;&gt; myDict["f"] = "freeze" &gt;&gt;&gt; myDict {'A': 'Apple', 'C': 'Cat', 'B': 'Boy', 'E': 'Elephant', 'D': 'Dog', 'F': 'Fan', 'f': 'freeze'}</pre>
Conclusion	:	<p>Hence from this experiment we learned the various operation of dictionary like create, modify or insert elements of dictionary and how two lists can merge in the dictionary.</p> <p>By Student - <u>Adnan Shaikh</u></p>

#### Code:

```
# Create key/value pair dictionary
#global dictionary
my_dict={1: "java", 2: "c", 3: "python"}
```

```
#update dictionary
def update():
    key=int(input("enter the key to be updated: "))
    value=input("Enter value: ")
    my_dict[key]=value
    print("updated dictionary: ",my_dict)
```

```
#concatenate dictionary
def concatenate():
    key_lst = []
    value_lst = []
    n = int(input("Enter number of pairs: "))
    # iterating till the range
    for i in range(0, n):
        keys = int(input("enter key: "))
        key_lst.append(keys)
        value = (input("enter value: "))
        value_lst.append(value)
```

---

```

my_dict_2= dict(zip(key_lst, value_lst))
my_dict.update(my_dict_2)
print("after concatenation:= ", my_dict)
return my_dict

def delete():
    #delete value from dictionary
    key=int(input("enter the key to be deleted:"))
    del my_dict[key]
    print("Dictionary after deleting key { } associated with its value is:= { }".format(key, my_dict))

def find():
    #find a key and print its value
    key_lst=list(my_dict.keys())
    value_lst=list(my_dict.values())
    print("keys from dictionary my_dict",key_lst)
    x=int(input("Enter the key to be viewed:"))
    value=key_lst.index(x)
    print("associated value to the selected key { } is : {}".format(x, value_lst[value]))

def combine_list():
    key_lst = []
    n = int(input("Enter number of elements : "))
    # iterating till the range
    for i in range(0, n):
        element1 = int(input("enter key list:"))
        key_lst.append(element1)
    value_lst = []
    # iterating till the range
    for i in range(0, n):
        value= (input("enter value list: "))
        value_lst.append(value)
    #map two lists into dictionary
    combined_list=dict(zip(key_lst, value_lst))
    print("after combining two list of keys(key_lst) and values(value_lst)", combined_list)

if __name__=="__main__":
    print(my_dict)
    while True:
        num = int(input("*****Menu Driven*****\n1. TO update dictionary \n2.To
concatenate dictionary \n3. To delete values from dictionary.\n4. To find value from dictionary \n5.
To create dictionary from lists. \n6. Exit\n"))
        if num == 1:
            update()
        elif num == 2:
            concatenate()
        elif num == 3:
            delete()
        elif num ==4:
            find()
        elif num==5:
            combine_list()
        elif num==6:

```

```

        break
    else:
        print("Wrong key")

```

### Output:

```

PS C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python\exp6> python exp-6.py
{1: 'java', 2: 'c', 3: 'python'}
*****Menu Driven*****
1. TO update dicitonary
2.To concatenate dicitonary
3. To delete values from dictionary.
4. To find value from dictionary
5. To create dictionary from lists.
6. Exit
1
enter the key to be updated: 1
Enter value: Java Script
updated dictionary: {1: 'Java Script', 2: 'c', 3: 'python'}
*****Menu Driven*****
1. TO update dicitonary
2.To concatenate dicitonary
3. To delete values from dictionary.
4. To find value from dictionary
5. To create dictionary from lists.
6. Exit
2
Enter number of pairs: 3
enter key: 4
enter value: Java
enter key: 5
enter value: R
enter key: 6
enter value: C#
after concatenation:= {1: 'Java Script', 2: 'c', 3: 'python', 4: 'Java', 5: 'R', 6: 'C#'}
*****Menu Driven*****
1. TO update dicitonary
2.To concatenate dicitonary
3. To delete values from dictionary.
4. To find value from dictionary
5. To create dictionary from lists.
6. Exit
3
enter the key to be deleted:=2
Dictionary after deleting key 2 associated with its value is:= {1: 'Java Script', 3: 'python', 4: 'Java', 5: 'R', 6: 'C#'}
*****Menu Driven*****
1. TO update dicitonary
2.To concatenate dicitonary
3. To delete values from dictionary.
4. To find value from dictionary
5. To create dictionary from lists.
6. Exit
4
keys from dictionary my_dict [1, 3, 4, 5, 6]
Enter the key to be viewed:=4
associated value to the selected key 4 is : Java
*****Menu Driven*****
1. TO update dicitonary
2.To concatenate dicitonary
3. To delete values from dictionary.
4. To find value from dictionary
5. To create dictionary from lists.
6. Exit
5
Enter number of elements : 4
enter key list:1
enter key list:2
enter key list:9
enter key list:6
enter value list: JSON
enter value list: REACT
enter value list: Angular JS
enter value list: Vue Js
after combining two list of keys(key_list) and values(value_list) {1: 'JSON', 2: 'REACT', 9: 'Angular JS', 6: 'Vue Js'}
*****Menu Driven*****
1. TO update dicitonary
2.To concatenate dicitonary
3. To delete values from dictionary.
4. To find value from dictionary
5. To create dictionary from lists.
6. Exit
6
PS C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python\exp6> 

```

---

---

## EXPERIMENT NO. 7

Aim	:	Design an employee class using python for reading and displaying the employee information.
Resources Required	:	Consumables – Printer Pages for printouts.
Theory	:	<p>Creating Classes:</p> <p>The class statement creates a new class definition. The name of the class immediately follows the keyword class followed by a colon as follows –</p> <pre>class ClassName:     'Optional class documentation string'     class_suite</pre> <ul style="list-style-type: none"><li>• The class has a documentation string, which can be accessed via <code>ClassName.doc</code>.</li><li>• The <code>class_suite</code> consists of all the component statements defining class members, data attributes and functions.</li></ul> <p>Creating Instance Objects:</p> <p>To create instances of a class, you call the class using class name and pass in whatever arguments its <code>__init__</code> method accepts.</p> <p>Accessing Attributes</p> <p>You access the object's attributes using the dot operator with object.</p>
Conclusion	:	<p>Hence from this experiment we learned how to create classes and its objects in python.</p> <p>By Student - <u>Adnan Shaikh</u></p>

---

**Code:**

```
class Employee:
    count = 0

    def __init__(self, name, desig, salary):
        self.name = name
        self.desig = desig
        self.salary = salary
        Employee.count += 1

    def displayCount(self):
        print("There are %d employees" % Employee.count)

    def displayDetails(self):
        print("Name:", self.name, ", Designation:", self.desig, ", Salary:", self.salary)

if __name__ == "__main__":

    e1 = Employee("John", "Manager", 80000)
    e2 = Employee("Mike", "Team Leader", 50000)
    e3 = Employee("Ramesh", "Programmer", 30000)
    e4 = Employee("Raj", "Assistant", 25000)
    e4.displayCount()
    print("Details of all employee:")
    e1.displayDetails()
    e2.displayDetails()
    e3.displayDetails()
    e4.displayDetails()
```

**Output:**

```
[Running] python -u "c:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp7\employee.py"
There are 4 employees
Details of all employee:
Name: John , Designation: Manager , Salary: 80000
Name: Mike , Designation: Team Leader , Salary: 50000
Name: Ramesh , Designation: Programmer , Salary: 30000
Name: Raj , Designation: Assistant , Salary: 25000

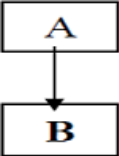
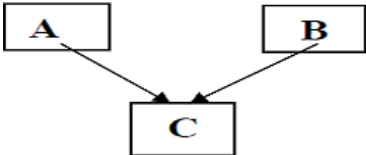
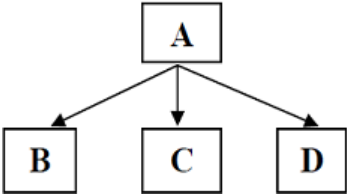
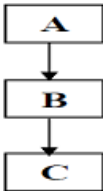
[Done] exited with code=0 in 0.115 seconds
```



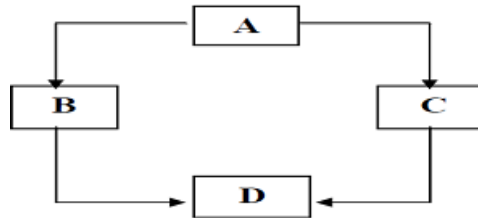
---

---

## EXPERIMENT NO. 8

Aim	:	Program to demonstrate single and multiple inheritance in python (with method overloading and overriding)
Resources Required	:	Consumables – Printer Pages for printouts.
Theory	:	<p>Inheritance means deriving properties from other classes.</p> <p>Types of Inheritance:</p> <ol style="list-style-type: none"><li>1. Single Inheritance: A derived class with only one base class is called as Single Inheritance.</li></ol>  <pre>graph TD; A[A] --&gt; B[B]</pre> <ol style="list-style-type: none"><li>2. Multiple Inheritances: A derived class with several base classes is called Multiple Inheritance.</li></ol>  <pre>graph TD; A[A] --&gt; C[C]; B[B] --&gt; C[C]</pre> <ol style="list-style-type: none"><li>3. Hierarchical Inheritance: More than one class may inherit the features of one class this process is called as Hierarchical Inheritance.</li></ol>  <pre>graph TD; A[A] --&gt; B[B]; A[A] --&gt; C[C]; A[A] --&gt; D[D]</pre> <ol style="list-style-type: none"><li>4. Multilevel Inheritance: The mechanism of deriving a class from another derived class is called Multilevel Inheritance.</li></ol>  <pre>graph TD; A[A] --&gt; B[B]; B[B] --&gt; C[C]</pre>

5. Hybrid Inheritance: There could be situations where we need to apply one or more types of inheritances to design a program this process is called Hybrid Inheritance.



Method overloading: In Python you can define a method in such a way that there are multiple ways to call it. Given a single method or function, we can specify the number of parameters ourselves. Depending on the function definition, it can be called with zero, one, two or more parameters. This is known as method overloading.

Example: class operation:

```
def abc(self, a=None):  
    if a is not None:  
        print ('first num is' + a)  
    else:  
        print ('no number assigned ')  
        obj = operation()  
        obj.abc ()  
        obj.abc('10')
```

Method overriding: Overriding is the ability of a class to change the implementation of a method provided by one of its ancestors. In Python method overriding occurs simply defining in the child class a method with the same name of a method in the parent class.

Example

```
class Parent(object):  
    def __init__(self):  
        self.value = 5  
    def get_value(self): ### overridden method  
        return self.value  
class Child(Parent):  
    def get_value(self):  
        return self.value + 1
```

---

Conclusion	:	From this experiment we studied how to write a class and use the concept of inheritance to achieve the specified requirements. By Student - <u>Adnan Shaikh</u>
------------	---	--

### Single Inheritance:

#### **Code:**

```
class Animal:
    def __init__(self):
        self.kingdomname = "Animalia"
    def kingdom(self):
        print("Kingdom: ",self.kingdomname)
```

```
class Cow(Animal):
    def __init__(self):
        super().__init__()
        self.name = "Cow"
        self.voc = "Mooooo"

    def naam(self):
        print("Name: ",self.name)
```

```
    def voice(self):
        print(f"{self.name} {self.voc}")
```

```
if __name__ == "__main__":
    cow1 = Cow()
    cow1.kingdom()
    cow1.naam()
    cow1.voice()
```

#### **Output:**

```
PS C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp8> python sinheritane.py
Kingdom: Animalia
Name: Cow
Cow Mooooo
PS C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp8> █
```

### Multiple Inheritance:

#### **Code:**

```
class Father:
    fathername = ""

    def show_father(self):
        print(self.fathername)
```

---

```

class Mother:
    mothername = ""

    def show_mother(self):
        print(self.mothername)

class Son(Father, Mother):
    def show_parent(self):
        print("Father :", self.fathername)
        print("Mother :", self.mothername)

if __name__ == "__main__":

    s1 = Son() # Object of Son class
    s1.fathername = "Joseph Joester"
    s1.mothername = "Lily"
    s1.show_parent()

```

### Output:

```

PS C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp8> python Minheritance.py
Father : Joseph Joester
Mother : Lily
PS C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp8> 

```

### Hierarchical Inheritance:

#### Code:

```

class Parent:
    parentname = ""
    childname = ""

    def show_parent(self):
        print("Parent Name: ",self.parentname)

class Son(Parent):
    def show_child(self):
        print("Son Name: ",self.childname)

class Daughter(Parent):
    def show_child(self):
        print("Daughter Name: ",self.childname)

if __name__ == "__main__":
    s1 = Son()
    s1.parentname = "Cujoh"
    s1.childname = "Jotaro"

```

---

```
s1.show_parent()
s1.show_child()

d1 = Daughter()
d1.childname = "Jolyne"
d1.parentname = "Jotaro"
d1.show_parent()
d1.show_child()
```

**Output:**

```
PS C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp8> python hierarchical.py
Parent Name:  Cujoh
Son Name:    Jotaro
Parent Name:  Jotaro
Daughter Name: Jolyne
PS C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp8> █
```

Multilevel Inheritance:

**Code:**

```
class Family:
    def show_family(self):
        print("This is Cujoh Family:")

class Father(Family):
    fathername = ""

    def show_father(self):
        print(self.fathername)

class Mother(Family):
    mothername = ""

    def show_mother(self):
        print(self.mothername)

class Child(Father, Mother):
    def __init__(self):
        super().__init__()
        self.childname = ""
    def show_parent(self):
        print("Son name: ",self.childname)
        print("Father :", self.fathername)
        print("Mother :", self.mothername)

if __name__ == "__main__":
    s1 = Child()
    S.E. IV [OSTL]
```

---

```
s1.childname = "Jotaro Cujoh"  
s1.fathername = "Sadao Cujoh"  
s1.mothername = "Holy Cujoh"  
s1.show_family()  
s1.show_parent()
```

**Output:**

```
PS C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp8> python multilevel.py  
This is Cujoh Family:  
Son name: Jotaro Cujoh  
Father : Sadao Cujoh  
Mother : Holy Cujoh  
PS C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp8> █
```

## EXPERIMENT NO. 9

Aim	:	Exception handling A. Write a program to demonstrate exception handling using try, multiple exception and finally. B. Write a python program to create user defined exception.																													
Resources Required	:	Consumables – Printer Pages for printouts.																													
Theory	:	<p>Python provides two very important features to handle any unexpected error in your Python programs and to add debugging capabilities in them:</p> <ul style="list-style-type: none"><li>a. Exception Handling</li><li>b. Assertions</li></ul> <p>List of Standard Exceptions –</p> <table><tr><th>Sr.No</th><th>Exception Name</th><th>Description</th></tr><tr><td>1</td><td>Exception</td><td>Base class for all exceptions</td></tr><tr><td>2</td><td>StandardError</td><td>Base class for all built-in exceptions except StopIteration and SystemExit.</td></tr><tr><td>3</td><td>ArithmeticError</td><td>Base class for all errors that occur for numeric calculation.</td></tr><tr><td>4</td><td>OverflowError</td><td>Raised when a calculation exceeds maximum limit for a numeric type.</td></tr><tr><td>5</td><td>NameError</td><td>Raised when an identifier is not found in the local or global namespace.</td></tr><tr><td>6</td><td>IOError</td><td>Raised when an input/ output operation fails, such as the print statement or the open() function when trying to open a file that does not exist.</td></tr><tr><td>7</td><td>TypeError</td><td>Raised when an operation or function is attempted that is invalid for the specified data type.</td></tr><tr><td>8</td><td>ValueError</td><td>It is raised when the built-in function for a data type has the valid type of arguments, but the arguments have invalid values specified.</td></tr></table>			Sr.No	Exception Name	Description	1	Exception	Base class for all exceptions	2	StandardError	Base class for all built-in exceptions except StopIteration and SystemExit.	3	ArithmeticError	Base class for all errors that occur for numeric calculation.	4	OverflowError	Raised when a calculation exceeds maximum limit for a numeric type.	5	NameError	Raised when an identifier is not found in the local or global namespace.	6	IOError	Raised when an input/ output operation fails, such as the print statement or the open() function when trying to open a file that does not exist.	7	TypeError	Raised when an operation or function is attempted that is invalid for the specified data type.	8	ValueError	It is raised when the built-in function for a data type has the valid type of arguments, but the arguments have invalid values specified.
Sr.No	Exception Name	Description																													
1	Exception	Base class for all exceptions																													
2	StandardError	Base class for all built-in exceptions except StopIteration and SystemExit.																													
3	ArithmeticError	Base class for all errors that occur for numeric calculation.																													
4	OverflowError	Raised when a calculation exceeds maximum limit for a numeric type.																													
5	NameError	Raised when an identifier is not found in the local or global namespace.																													
6	IOError	Raised when an input/ output operation fails, such as the print statement or the open() function when trying to open a file that does not exist.																													
7	TypeError	Raised when an operation or function is attempted that is invalid for the specified data type.																													
8	ValueError	It is raised when the built-in function for a data type has the valid type of arguments, but the arguments have invalid values specified.																													

---

Conclusion	:	From this experiment we studied how to use standard exceptions and create user defined exceptions in python.  By Student - <u>Adnan Shaikh</u>
------------	---	--

#### Code:

A:

```
import math
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)

number_list = [6,-9,420.69,'banana']

for number in number_list:
    try:
        number_factorial = math.factorial(number)
    except TypeError:
        print("Factorial is not supported for given input type. \n")
    except ValueError:
        print(f"Factorial only accepts whole values. {number} is not a whole value. \n")
    else:
        print(f"The factorial of {number} is {number_factorial}\n")
    finally:
        print("Release any resources in use. \n")
```

#### Output:

```
PS C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python\exp9> python exceptionhandling.py
The factorial of 6 is 720

Release any resources in use.

Factorial only accepts whole values. -9 is not a whole value.

Release any resources in use.

Factorial only accepts whole values. 420.69 is not a whole value.

Release any resources in use.

Factorial is not supported for given input type.

Release any resources in use.
```



---

```
B:
#numberGuessing
class ValueTooSmallError(Exception):
    #Raised when the input value is too small
    pass
class ValueTooLargeError(Exception):
    #Raised when the input value is too large
    pass

number = 10
while True:
    try:
        i_num = int(input("Guess the number: "))
        if i_num < number:
            raise ValueTooSmallError
        elif i_num > number:
            raise ValueTooLargeError
        break
    except ValueTooSmallError:
        print("This value is too small, try again!")
        print()
    except ValueTooLargeError:
        print("This value is too large, try again!")
        print()

print("Congratulations! You guessed it correctly.")
```

**Output:**

```
PS C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp9> python userdefinedexception.py
Guess the number: 56
This value is too large, try again!

Guess the number: 99
This value is too large, try again!

Guess the number: 69
This value is too large, try again!

Guess the number: 7
This value is too small, try again!

Guess the number: 9
This value is too small, try again!

Guess the number: 11
This value is too large, try again!

Guess the number: 10
Congratulations! You guessed it correctly.
```

---

---

## EXPERIMENT NO. 10

Aim	:	Exploring files and directories. A. Python program to read the content of file and write in another file. B. Program to append data to existing file and then display entire file. C. Program to count number of lines, words and characters in a file. D. Program to display file available in current directory.
Resources Required	:	Consumables – Printer Pages for printouts.
Theory	:	<p>Data is very important. Every organization depends on its data for continuing its business operations. To store the data in computer we need files. There are certain advantages of storing data in file:</p> <ol style="list-style-type: none"><li>to store huge amount of data When data is stored in a file, it is stored permanently.</li><li>It is possible to update the file data.</li><li>Files are highly useful.</li></ol> <p>In Python, there are two types of files. They are</p> <ol style="list-style-type: none"><li>Text File-: store the data in the form of characters</li><li>Binary File-: store the entire data in the form of bytes.</li></ol> <p>It is very important to know how to create files, store data in the files and retrieve the data from the files in python. To do any operations on files, first of all we should open the files.</p> <p>Opening a File in order to open a file for writing or use in Python, you must rely on the built-in open () function.</p> <p>As explained above, open ( ) will return a file object, so it is most commonly used with two arguments. An argument is nothing more than a value that has been provided to a function, which is relayed when you call it. So, for instance, if we declare the name of a file as “Test File,” that name would be considered an argument. The syntax to open a file object in Python is:</p> <p>file_object = open(“filename”, “mode”) where file_object is the variable to add the file object. Mode Including a mode argument is optional because a default value of ‘r’ will be assumed if it is omitted. The ‘r’ value stands for read mode, which is just one of many. The modes are:</p> <ul style="list-style-type: none"><li>•‘r’ – Read mode which is used when the file is only being read</li><li>•‘w’ – Write mode which is used to edit and write new information to the file (any existing files with the same name will be erased when this mode is activated)</li></ul>

		<ul style="list-style-type: none"> <li>•‘a’ – Appending mode, which is used to add new data to the end of the file; that is new information is automatically amended to the end</li> <li>•‘r+’ – Special read and write mode, which is used to handle both actions when working with a file.</li> </ul> <p>So, let’s take a look at a quick example.</p> <pre>f=open(“workfile”,”w”) Print f</pre> <p>This snippet opens the file named “workfile” in writing mode so that we can make changes to it. Just create the file named “testfile.txt” and leave it blank. If you need to extract a string that contains all characters in the file, then</p> <pre>file=open(“testfile.txt”, “r”) print file.read()</pre> <p>Using the File Write Method the file write method is that it only requires a single parameter, which is the string you want to be written. This method is used to add information or content to an existing file. To start a new line after you write data to the file, you can add an EOL character. Closing a File when you’re done working, you can use the fh.close() command to end things.</p>
Conclusion	:	<p>Hence from above experiment student will understand various file operations like creation of file, storing data in the files and retrieve data from the files in python.</p> <p>By Student - <u>Adnan Shaikh</u></p>

A:

**Code:**

```
file1 = open("dio.txt","w")
file1.write("KONO DIO DAAA")
file1.close()
file2 = open("wurdo.txt","w")
file2.write("You expected Output but it was me DIO")
file2.close()
file1 = open("dio.txt","r")
file2 = open("wurdo.txt","r")
file3 = open("output.txt","w")
file3.write(file1.read())
file3.write(file2.read())
file1.close()
file2.close()
file2.close()
print("Data copied successfully!!!")
```

---

## Output:

```
[Running] python -u "c:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp10\copyfiles.py"
Data copied successfully!!!

[Done] exited with code=0 in 0.13 seconds
```

## Content of files:

DIO.txt: KONO DIO DAAA

wurdo.txt: You expected Output but it was me DIO

output.txt: KONO DIO DAAAYou expected Output but it was me DIO

B:

## Code:

```
f=open("Adnan.txt","a+")
f.write(" Giorno Giovanna")
print("Data Appended Sucessfully")
f.close()
```

## Output:

```
[Running] python -u "c:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp10\second\append file.py"
Data Appended Sucessfully

[Done] exited with code=0 in 0.08 seconds
```

## Content of a file:

Before appending: Adnan Shaikh

After appending: Adnan Shaikh Giorno Giovanna

C:

## Code:

```
file = open("Count.txt", "r")

number_of_lines = 0
number_of_words = 0
number_of_characters = 0
for line in file:
    line = line.strip("\n")

    words = line.split()
    number_of_lines += 1
    number_of_words += len(words)
    number_of_characters += len(line)

file.close()

print("lines:", number_of_lines)
print("words:", number_of_words)
print("characters:", number_of_characters)
```

---

Output:

```
[Running] python -u "c:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp10\third\FileDataCount.py"
lines: 3
words: 14
characters: 80

[Done] exited with code=0 in 0.089 seconds
```

**Content of a file:**

Hello, this is Adnan  
Welcome to my woooooorld  
Ok, understandable have a great day.

D:

**Code:**

```
import os
from pathlib import Path
if os.path.exists("Adnan.txt"):
    print("YES, The file does exist")
else:
    print("NO, The file does not exist\nCreating a new file called Adnan.txt")
    f1 = open("Adnan.txt", "w+")
    print("Writing something in Adnan.txt")
    f1.write("Well well what do we have we here")
    print("Closing the file.....")
    f1.close()
```

**Output:**

```
[Running] python -u "c:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp10\fourth\FileExists.py"
NO, The file does not exist
Creating a new file called Adnan.txt
Writing something in Adnan.txt
Closing the file.....

[Done] exited with code=0 in 0.087 seconds

[Running] python -u "c:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp10\fourth\FileExists.py"
YES, The file does exist

[Done] exited with code=0 in 0.087 seconds
```

**Content of a file:** Well well what do we have we here

---

---

## EXPERIMENT NO. 11

Aim	:	Program to create a package and module for data structures like stack and queues.
Resources Required	:	Consumables – Printer Pages for printouts.
Theory	:	<p><b>Stack</b></p> <p>It is a sequence of items that are accessible at only one end of the sequence. Think of a stack as a collection of items that are piled one on top of the other, with access limited to the topmost item. A stack inserts item on the top of the stack and removes item from the top of the stack. It has LIFO (last-in / first-out) ordering for the items on the stack. Also, the inbuilt functions in Python make the code short and simple. To add an item to the top of the list, i.e., to push an item, we use <code>append()</code> function and to pop out an element we use <code>pop()</code> function. These functions work quite efficiently and fast in end operations.</p> <p><b>Queue</b></p> <p>A queue is a sequential storage structure that permits access only at the two ends of the sequence. We refer to the ends of the sequence as the front and rear. A queue inserts new elements at the rear and removes elements from the front of the sequence. You will note that a queue removes elements in the same order in which they were stored, and hence a queue provides FIFO (first-in / first-out), or FCFS (first-come / first-served), ordering. Implementing queue is a bit different. Time plays an important factor here. During the implementation of stack we used <code>append()</code> and <code>pop()</code> function which was efficient and fast because we inserted and popped elements from the end of the list, but in queue when insertion and pops are made from the beginning of the list, it is slow. This occurs due to the properties of list, which is fast at the end operations but slow at the beginning operations, as all other elements have to be shifted one by one. So, it's preferable to use <code>collections.deque</code> over list, which was specially designed to have fast appends and pops from both the front and back end.</p>

---

---

Conclusion	:	From this experiment we have studied built in functions like pop(), append() used to perform operations on data structures like stack and queue. By Student - <u>Adnan Shaikh</u>
------------	---	--

**Code:**

Package Content:

```
class Stack:
    def __init__(self):
        self.stack = []

    def push(self, val):
        self.stack.append(val)

    def pop(self):
        if len(self.stack) > 0:
            print(self.stack.pop())
        else:
            print("Stack is empty!!!")

    def display(self):
        print(self.stack)

class Queue:
    def __init__(self):
        self.queue = []

    def enqueue(self, val):
        self.queue.append(val)

    def dequeue(self):
        if len(self.queue) > 0:
            print(self.queue.pop(0))
        else:
            print("Queue is empty!!!")

    def display(self):
        print(self.queue)
```

Implementing Package:

```
from StackQueue.stack_queue import Stack, Queue

if __name__ == "__main__":

    while True:
        num = int(input("*****Menu Driven*****\n1.Implementing
Stack\n2.Implementing Queue\n3.Exit\n"))
        if num == 1:
            s = Stack()
            while True:
                n = int(input("1.Push\n2.Pop\n3.Display\n4.Back\n"))
```

---

```

if n == 1:
    s.push(input("Enter element you want to push: "))
elif n == 2:
    s.pop()
elif n == 3:
    s.display()
elif n == 4:
    break
else:
    print("You pressed wrong key please try agin")
elif num == 2:
    q = Queue()
    while True:
        n = int(input("1.Enqueue\n2.Dequeue\n3.Display\n4.Back\n"))
        if n == 1:
            q.enqueue(input("Enter element you want to enqueue: "))
        elif n == 2:
            q.dequeue()
        elif n == 3:
            q.display()
        elif n == 4:
            break
        else:
            print("You pressed wrong key please try agin")
    elif num == 3:
        break
    else:
        print("You pressed wrong key please try again")

```

### Output:

```

PS C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp11> python exp11.py
*****Menu Driven*****
1.Implementing Stack
2.Implementing Queue
3.Exit
1
1.Push
2.Pop
3.Display
4.Back
1
Enter element you want to push: Hello
1.Push
2.Pop
3.Display
4.Back
1
Enter element you want to push: World
1.Push
2.Pop
3.Display
4.Back
3
['Hello', 'World']
1.Push
2.Pop
3.Display
4.Back
2
World
1.Push
2.Pop
3.Display
4.Back
3
['Hello']

```

```

1.Push
2.Pop
3.Display
4.Back
2
Hello
1.Push
2.Pop
3.Display
4.Back
2
Stack is empty!!!
1.Push
2.Pop
3.Display
4.Back
4

```



```

*****Menu Driven*****
1.Implementing Stack
2.Implementing Queue
3.Exit
2
1.Enqueue
2.Dequeue
3.Display
4.Back
1
Enter element you want to enqueue: KONO
1.Enqueue
2.Dequeue
3.Display
4.Back
1
Enter element you want to enqueue: DIO
1.Enqueue
2.Dequeue
3.Display
4.Back
1
Enter element you want to enqueue: DA
1.Enqueue
2.Dequeue
3.Display
4.Back
3
['KONO', 'DIO', 'DA']
1.Enqueue
2.Dequeue
3.Display
4.Back
2
KONO
1.Enqueue
2.Dequeue
3.Display
4.Back
2
DIO
1.Enqueue
2.Dequeue
3.Display
4.Back
3
['DA']
1.Enqueue
2.Dequeue
3.Display
4.Back
2
DA
1.Enqueue
2.Dequeue
3.Display
4.Back
2
Queue is empty!!
1.Enqueue
2.Dequeue
3.Display
4.Back
4
*****Menu Driven*****
1.Implementing Stack
2.Implementing Queue
3.Exit
3
PS C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp11> 

```

---

---

## EXPERIMENT NO. 12

Aim	:	Program to create simple socket for basic info exchange between server and client.
Resources Required	:	Consumables – Printer Pages for printouts.
Theory	:	<p>Sockets are the endpoints of a bidirectional communications channel. Sockets may communicate within a process, between processes on the same machine, or between processes on different continents. Sockets may be implemented over a number of different channel types: Unix domain sockets, TCP, UDP, and so on. The socket library provides specific classes for handling the common transports as well as a generic interface for handling the rest. One socket(node) listens on a particular port at an IP, while other socket reaches out to the other to form a connection. Server forms the listener socket while client reaches out to the server. They are the real backbones behind web browsing. In simpler terms there is a server and a client. Socket programming is started by importing the socket library and making a simple socket.</p> <pre>import socket s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)</pre> <p>Here we made a socket instance and passed it two parameters. The first parameter is AF_INET and the second one is SOCK_STREAM. AF_INET refers to the address family ipv4. The SOCK_STREAM means connection oriented TCP protocol. Now we can connect to a server using this socket.</p> <p>Connecting to a server: Note that if any error occurs during the creation of a socket then a socket.error is thrown and we can only connect to a server by knowing its IP. You can find the ip of the server by using this: \$ ping <a href="http://www.google.com">www.google.com</a> You can also find the ip using python: <code>import socket ip = socket.gethostbyname('www.google.com') print ip</code></p> <p>Server Socket Methods</p>

		Method	Description
		s.bind()	This method binds address (hostname, port number pair) to socket.
		s.listen()	This method sets up and start TCP listener.
		s.accept()	This passively accept TCP client connection, waiting until connection arrives (blocking).
		Client Socket Methods	
		Method	Description
		s.recv()	This method receives TCP message
		s.send()	This method transmits TCP message
		s.recvfrom()	This method receives UDP message
		<p>Server :</p> <p>A server has a bind() method which binds it to a specific ip and port so that it can listen to incoming requests on that ip and port. A server has a listen() method which puts the server into listen mode. This allows the server to listen to incoming connections. And last a server has an accept() and close() method. The accept method initiates a connection with the client and the close method closes the connection with the client.</p> <p>Client :</p> <p>Now we need something with which a server can interact. We could tenet to the server like this just to know that our server is working. Type these commands in the terminal: # start the server \$ python server.py # keep the above terminal open # now open another terminal and type: \$ telnet localhost 12345</p>	
Conclusion	:	<p>From this experiment we studied the client server communication on a network with the help of socket.</p> <p>By Student <u>Adnan Shaikh</u></p>	

---

**Code:****Server Side:**

```
import socket

HOST = '127.0.0.1'
PORT = 65432

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind((HOST, PORT))
    s.listen()
    conn, addr = s.accept()
    with conn:
        print('Connected by', addr)
        while True:
            data = conn.recv(1024)
            if not data:
                break
            conn.sendall(data)
```

**Client Side:**

```
import socket

HOST = '127.0.0.1'
PORT = 65432

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((HOST, PORT))
    s.sendall(b'Hello, world')
    data = s.recv(1024)

print('Received', repr(data))
```

---

## Output:

### Server Side:

```
Command Prompt
Microsoft Windows [Version 10.0.19042.985]
(c) Microsoft Corporation. All rights reserved.

C:\Users\adnan>cd C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp12

C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp12>python server.py
Connected by ('127.0.0.1', 58842)

C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp12>
```

### Client Side:

```
Command Prompt
Microsoft Windows [Version 10.0.19042.985]
(c) Microsoft Corporation. All rights reserved.

C:\Users\adnan>cd C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp12

C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp12>python client.py
Received b'Hello, world'

C:\Users\adnan\OneDrive\Desktop\College\Sem 4\Python'\exp12>
```

---

---

### EXPERIMENT NO. 13

Aim	:	Demonstrate Database connectivity
Resources Required	:	Consumables – Printer Pages for printouts.
Theory	:	<p>Step1: Install python and set the path.</p> <p>Step2: Using any IDE or in terminal write the following command to install mysql connector:</p> <pre>pip install mysql-connector-python</pre> <p>or</p> <pre>Python -m pip install mysql-connector-python (if upper command doesn't work)</pre> <p>After installing above library we're ready to use it in our python files</p> <p>(Note: If you create Virtual Environment you need to install it in your Virtual Environment.)</p> <p>Step3: Create a Python file of your desired name(extension: .py)</p> <p>Step4: You need to write following code at the top to import mysql connector:</p> <pre>from mysql import connector</pre>

		<p>Step5: Now we need to connect to database using mysql.connector object, for that we have following syntax:</p> <pre>Object_name = connector.connect( host="hostname", username="db_username", password = "user_password", database = "db_name")</pre> <p>Hostname: In our case hostname is local host since we're not hosting it anywhere</p> <p>db_username: It is a database username it should exist or else code will raise an error.</p> <p>Password: user password should be correct or code will raise an error.</p> <p>Db_name: database name which you're going to use if not exist it will raise an error.</p> <p>Step6: If everything is correct in Step5 we have successfully connected to our database and we can write queries using our Object.</p>
Conclusion	:	<p>We have successfully demonstrate MySQL database connectivity in python.</p> <p>By Student <u>Adnan Shaikh</u></p>

**Code:**

```
from mysql import connector
```

```
mydb = connector.connect(
    host = "localhost",
    username = "root",
    password = "admin",
    database = "pythontemp"
)
```

```
mycursor = mydb.cursor()
```

```
mycursor.execute("CREATE TABLE IF NOT EXISTS customers(customer_id INT
AUTO_INCREMENT PRIMARY KEY,name VARCHAR(30))")
```

```
mycursor.execute("SHOW TABLES")
```

```
print(*[x for x in mycursor])
```

```
try:
```

```
    mycursor.execute("ALTER TABLE customers ADD address VARCHAR(255)")
```

```
except:
```

```
    print("Column already exist")
```

```
mycursor.execute("DESC customers")
```

---

```
print(*[x for x in mycursor])

sql = "INSERT INTO customers(name,address) values(%s,%s)"

val = [
    ("Levi Ackermann","Wall Rose"),
    ("Eren Jaeger","Wall Maria"),
    ("Lalatinna","Konosuba"),
    ("Kaneki Kun","Re"),
    ("Rias Gremory","DxD"),
]

mycursor.executemany(sql,val)

mydb.commit()

print(mycursor.rowcount," was inserted")

mycursor.execute("SELECT * FROM customers ORDER BY name")

result = mycursor.fetchall()

for x in result:
    print(x)
```

### Output:

```
[Running] python -u "c:\Users\adnan\OneDrive\Desktop\College\Sem 4\DBMS\Practicals\Practical-10\dbconnect.py"
('customers',)
('customer_id', b'int', 'NO', 'PRI', None, 'auto_increment') ('name', b'varchar(30)', 'YES', '', None, '') ('address', b'varchar(255)', 'YES', '', None, '')
5 was inserted
(2, 'Eren Jaeger', 'Wall Maria')
(4, 'Kaneki Kun', 'Re')
(3, 'Lalatinna', 'Konosuba')
(1, 'Levi Ackermann', 'Wall Rose')
(5, 'Rias Gremory', 'DxD')

[Done] exited with code=0 in 0.261 seconds
```