LABORATORY MANUAL

Python Language Programming Lab

Department of Computer Science and Engineering

LIST OF EXPERIMENTS

Exp. No.	Title of experiment	Corresponding CO
1.	Demonstrate the working of 'id' and 'type' functions.	C 220.1
2.	Write a Python program to find all prime numbers within a given range	C 220.1
3.	Write a Python program to print 'n terms of Fibonacci series using iteration	C 220.1
4.	Write a Python program demonstrate use of slicing in string.	C 220.1
5.	Write a Python program a) To add 'ing' at the end of a given string (length should be at least 3). If the given string already ends with 'ing' then add 'ly' instead. If the string length of the given string is less than 3, leave it unchanged. Sample String : 'abc' Expected Result : 'abcing' Sample String : 'string' Expected Result : 'stringly' b) To get a string from a given string where all occurrences of its first char have been changed to '$', except the first char itself.	C 220.2
6.	Write a Python program to a) Compute the frequency of the words from the input. The output should output after sorting the key alphanumerically. b) program that accepts a comma separated sequence of words as input and prints the words in a comma-separated sequence after sorting them alphabetically.	C 220.2
7.	Write a Python program that accepts a sequence of whitespace separated words as input and prints the words after removing all duplicate words and sorting them alphanumerically	C 220.2
8.	Write a Python program to demonstrate use of list & related functions	C 220.2
9.	Write a Python program to demonstrate use Dictionary& related functions.	C 220.2
10.	Write a Python program to demonstrate use tuple, set & related functions.	C 220.3
11.	Write a Python program to implement stack using list.	C 220.3

12.	Write a Python program to implement queue using list.	C 220.3
13.	Write a Python program to read and write from a file.	C 220.3
14.	Write a Python program copy a file.	C 220.3
15.	Write a Python program to demonstrate working of classes and objects.	C 220.3
16.	Write a Python program to demonstrate class method & static method.	C 220.4
17.	Write a Python program to demonstrate constructors.	C 220.4
18.	Write a Python program to demonstrate inheritance.	C 220.4
19.	Write a Python program to demonstrate aggregation/compositions	C 220.4
20.	Write a Python program to create a small GUI application for insert, update and delete in a table using Oracle as backend and front end for creating form	C 220.4

Content Beyond Syllabus		
21.	Write a Python program to compute area and circumference of a Triangle. Take input from user.	C 220.2
22.	Write a program to check if a number is Odd or even. Take input From user.	C 220.2
23.	Write a program to check that a given year is Leap Year or not.	C 220.2

INTRODUCTION

Python is a language with a simple syntax, and a powerful set of libraries. It is an interpreted language, with a rich programming environment, including a robust debugger and profiler. While it is easy for beginners to learn, it is widely used in many scientific areas for data exploration. This course is an introduction to the Python programming language for students without prior programming experience. We cover data types, control flow, object-oriented programming, and graphical user interface-driven applications. The examples and problems used in this course are drawn from diverse areas such as text processing, simple graphics creation and image manipulation, HTML and web programming, and genomics.

Scope & Objective:

1. Learn basic programming constructs –data types, decision structures, control structures in python
2. Know how to use libraries for string manipulation and user-defined functions.
3. Learn to use in-built data structures in python – Lists, Tuples, Dictionary and File handling.
4. Learn the fundamental principles of Object-Oriented Programming
5. Solve problems through application of OO concepts and using Files/database

Use Python Shell (using command line) and IDLE – Interactive development environment.

- To evaluate expression.
- To create a script.

Using IDLE

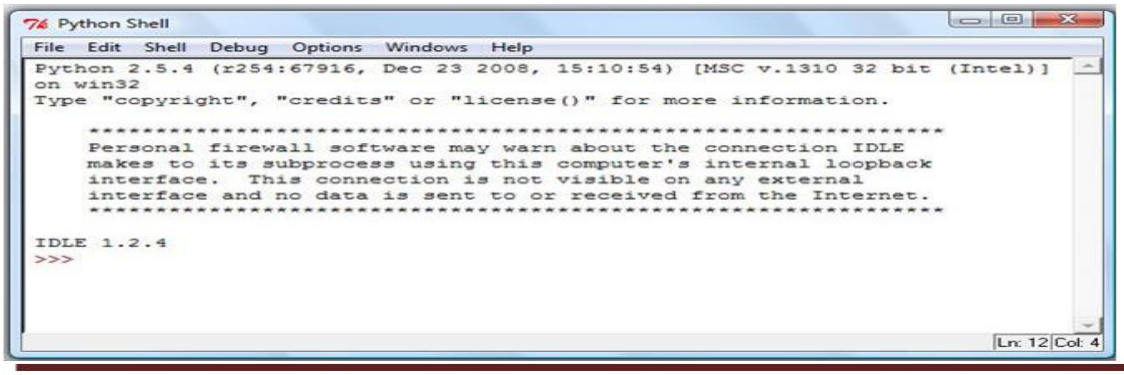
IDLE is the standard Python development environment. Its name is an acronym of "**I**ntegrated **D**eve**L**opment **E**nvironment". It works well on both Unix and Windows platforms.

It has a Python shell window, which gives you access to the Python interactive mode. It also has a file editor that lets you create and edit existing Python source files.

During the following discussion of IDLE's features, instead of passively reading along, you should start IDLE and try to replicate the screenshots.

Interactive Python shell

When you start up IDLE, a window with an interactive Python shell will pop up:



You can type Python code directly into this shell, at the '>>>' prompt. Whenever you enter a complete code fragment, it will be executed. For instance, typing:

```
>>> print "hello world"
```

and pressing ENTER, will cause the following to be displayed:

```
hello world
```

Try typing an underscore (_). Can you see it? On some operating systems, the bottoms of hanging letters such as 'g' or 'y', as well as underscores, cannot be seen in IDLE. If this is the case for you, go to Options -> Configure IDLE, and change the size of the default font to 9 or 11. This will fix the problem!

IDLE can also be used as a calculator:

```
>>> 4+4
8
>>> 8**3
512
```

Addition (+), subtraction (-), multiplication (*), division (/), modulo (%) and power (**) operators are built into the Python language. This means you can use them right away. If you want to use a square root in your calculation, you can either raise something to the power of 0.5 or you can import the math module
Below are two examples of square root calculation:

```
>>> 16**0.5
4.0
>>> import math
>>> math.sqrt(16)
4.0
```

The math module allows you to do a number of useful operations:

```
>>> math.log(16, 2)
4.0
>>> math.cos( 0 )
1.0
```

Note that you only need to execute the import command once after you start IDLE; however you will need to execute it again if you restart the shell, as restarting resets everything back to how it was when you opened IDLE.

Creating scripts

1. save your hello.py program in the ~/pythonpractice folder.
2. Open up the terminal program. ...
3. Type cd ~/pythonpractice to change directory to your pythonpractice folder, and hit Enter.
4. Type chmod a+x hello.py to tell Linux that it is an executable program.
5. Type ./hello.py to run your program!

Program to add two integers.

Take input from user.

```
number1 = input(" Please Enter the First Number: ")
number2 = input(" Please Enter the second number: ")

# Using arithmetic + Operator to add two
numbers sum = float(number1) + float(number2)
print('The sum of {0} and {1} is {2}'.format(number1, number2, sum))
```

Program to calculate area of a triangle:

(a) Given Base and height of the triangle. Take input from user.

(b) Given Three sides of a triangle (Make sure that it forms a triangle). Take input from user.

```
# Python Program to find Area of a Triangle

a = float(input('Please Enter the First side of a Triangle: '))
b = float(input('Please Enter the Second side of a Triangle: '))
c = float(input('Please Enter the Third side of a Triangle: '))

# calculate the Perimeter
Perimeter = a + b + c

# calculate the semi-
perimeter s = (a + b + c) / 2

# calculate the area
Area = (s*(s-a)*(s-b)*(s-c)) ** 0.5

print("\n The Perimeter of Traiangle = %.2f" %Perimeter);
print(" The Semi Perimeter of Traiangle = %.2f" %s); print("
The Area of a Triangle is %0.2f" %Area)
```

PREFACE

This manual will introduce you to the Python programming language. It's aimed at beginning programmers, but even if you've written programs before and just want to add Python to your list of languages, It will get you started.

Python is a powerful high-level, object-oriented programming language created by Guido van Rossum. It has simple easy-to-use syntax, making it the perfect language for someone trying to learn computer programming for the first time.

This practical manual will be helpful for students of Computer Science & Engineering for understanding the course from the point of view of applied aspects.

Though all the efforts have been made to make this manual error free, yet some errors might have crept in inadvertently. Suggestions from the readers for the improvement of the manual are most welcomed

DO'S AND DONT'S

DO's

1. Conform to the academic discipline of the department.
2. Enter your credentials in the laboratory attendance register.
3. Read and understand how to carry out an activity thoroughly before coming to the laboratory.
4. Ensure the uniqueness with respect to the methodology adopted for carrying out the experiments.
5. Shut down the machine once you are done using it.

DONT'S

1. Eatables are not allowed in the laboratory.
2. Usage of mobile phones is strictly prohibited.
3. Do not open the system unit casing.
4. Do not remove anything from the computer laboratory without permission.
5. Do not touch, connect or disconnect any plug or cable without your faculty/laboratory technician's permission.

GENERAL SAFETY INSTRUCTIONS

1. Know the location of the fire extinguisher and the first aid box and how to use them in case of an emergency.

2. Report fire or accidents to your faculty /laboratory technician immediately.

3. Report any broken plugs or exposed electrical wires to your faculty/laboratory technician immediately.

4. Do not plug in external devices without scanning them for computer viruses.

DETAILS OF THE EXPERIMENTS CONDUCTED

(TO BE USED BY THE STUDENTS IN THEIR RECORDS)

S. No	DATE OF CONDUCTION	EXPT. No	TITLE OF THE EXPERIMENT	PAGE No.	MARKS AWARDED (20)	FACULTY SIGNATURE WITH REMARK
1						
2						
3						
4						
5						
6						
7						
8						
9						
10						
11						

PYTHON LANGUAGE PROGRAMMING LAB FILE (RCS 454)

Name	
Roll No.	
Section- Batch	

INDEX

Experiment No.	Experiment Name	Date of Conduction	Date of Submission	Faculty Signature

EXPERIMENT 1

OBJECTIVE:- Demonstrate the working of 'id' and 'type' functions

THEORY:-

The id() function returns identity (unique integer) of an object.

The syntax of id() is:

```
id(object)
```

As we can see the function accepts a single parameter and is used to return the identity of an object. **This identity has to be unique and constant for this object during the lifetime.** Two objects with non-overlapping lifetimes may have the same id() value. If we relate this to C, then they are actually the memory address, here in Python it is the unique id. This function is generally used internally in Python.

Examples:

The output is the identity of the object passed. This is random but when running in the same program, it generates unique and same identity.

Input : id(1025)
Output : 140365829447504
Output varies with different runs

Input : id("geek")
Output : 139793848214784

```
# This program shows various
identities str1 = "geek"
print(id(str1))

str2 = "geek"
print(id(str2))

# This will return True
print(id(str1) == id(str2))

# Use in Lists
list1 = ["aakash", "priya", "abdul"]
print(id(list1[0]))
print(id(list1[2]))

# This returns false
print(id(list1[0])==id(list1[2]))
```

Return Value from id():

The id() function returns identity of the object. This is an integer which is unique for the given object and remains constant during its lifetime.

Example:

```
print('id of 5 =',id(5))
a = 5
print('id of a =',id(a))
b = a
print('id of b =',id(b))
c = 5.0
print('id of c =',id(c))
```

The 'type' function:

Python have a built-in method called as type which generally come in handy while figuring out the type of variable used in the program in the runtime.

The type function returns the datatype of any arbitrary object. The possible types are listed in the types module. This is useful for helper functions that can handle several types of data.

Example : Introducing type

```
>>> type(1)
<type 'int'>
>>> li = []
>>> type(li)
<type 'list'>
>>> import odbchelper
>>> type(odbchelper)
<type 'module'>
>>> import types
>>> type(odbchelper) == types.ModuleType
True
```

- type takes anything -- and I mean anything -- and returns its data type. Integers, strings, lists, dictionaries, tuples, functions, classes, modules, even types are acceptable.
- type can take a variable and return its datatype.
- type also works on modules.

You can use the constants in the types module to compare types of objects. This is what the info function does, as you'll see shortly.

VIVA QUESTIONS:

1.What data types are used in Python?

2.What is the difference between functions used in python and C? Are both the same?

EXPERIMENT -2

OBJECTIVE:-To find all prime numbers within a given range.

THEORY:-

Prime numbers: A prime number is a natural number greater than 1 and having no positive divisor other than 1 and itself.

For example: 3, 7, 11 etc are prime numbers.

Composite number: Other natural numbers that are not prime numbers are called composite numbers.

For example: 4, 6, 9 etc. are composite numbers.

Here is source code of the Python Program to check if a number is a prime number.

```
r=int(input("Enter upper limit: "))
for a in range(2,r+1):
   k=0
   for i in range(2,a//2+1):
      if(a%i==0):
         k=k+1
   if(k<=0):
      print(a)
```

VIVA QUESTIONS:

1.Why don’t we use semi colon in this program?

2.What are Composite numbers?

EXPERIMENT-3

OBJECTIVE:- To print ‘n terms of Fibonacci series using iteration.

THEORY: The Fibonacci numbers are the numbers in the following integer sequence.
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ……..

A Fibonacci sequence is the integer sequence of 0, 1, 1, 2, 3, 5, 8....

The first two terms are 0 and 1. All other terms are obtained by adding the preceding two terms. This means to say the nth term is the sum of (n-1)th and (n-2)th term.

In mathematical terms, the sequence Fn of Fibonacci numbers is defined by the recurrence relation

Fn = Fn-1 + Fn-2 with initial valu F0 = 0 and F1 = 1

```
# Program to display the Fibonacci sequence up to n-th term where n is provided by the user
# change this value for a different result nterms = 10
# uncomment to take input from the user
nterms = int(input("How many terms? "))

# first two
terms n1 = 0
n2 = 1
count = 0

# check if the number of terms is
valid if nterms <= 0:
  print("Please enter a positive integer")
elif nterms == 1:
  print("Fibonacci sequence upto",nterms,":")
  print(n1)
else:
  print("Fibonacci sequence upto",nterms,":")
  while count < nterms:
     print(n1)
     nth = n1 + n2
     # update values
     n1 = n2
     n2 = nth
     count += 1
```

VIVA QUESTIONS

1.What is the difference between iteration and recursion?

2.How looping technique is different from Recursion?

EXPERIMENT-4

OBJECTIVE:- To demonstrate use of slicing in string.

THEORY:- Like other programming languages, it's possible to access individual characters of a string by using array-like indexing syntax. In this we can access each and every element of string through their index number and the indexing starts from 0. Python does index out of bound checking.

So, we can obtain the required character using syntax, **string_name[index_position]**:

The positive index_position denotes the element from the starting(0) and the negative index shows the index from the end(-1).

EXAMPLE-

```
# A python program to illustrate slicing in

strings x = "Geeks at work"

# Prints 3rd character beginning from 0
print x[2]

# Prints 7th
character print x[6]

# Prints 3rd character from rear beginning from -
1 print x[-3]

# Length of string is 10 so it is out of bound
print x[15]
```

Slicing in string

We use the slice notation on strings. In this example, we omit the first index to start at the beginning, and then consume four characters total. We extract the first four letters.

```
Python program that slices string

word = "something"

# Get first four characters.

part = word[:4] print(part)
```

Copy. With slicing, we can copy sequences like lists. We assign a new list variable to a slice with no specified start or stop values. So the slice copies the entire list and returns it.

To extract substring from the whole string then we use the syntax like

string_name[beginning: end : step]

- end denotes the end index of string which is not inclusive
- steps denotes the distance between the two words. o

beginning represents the starting index of string

VIVA QUESTIONS

1.Are Strings immutable? Explain.

2.What are the ways to create a string?

EXPERIMENT-5

OBJECTIVE:-

a. To add 'ing' at the end of a given string (length should be at least 3). If the given string already ends with 'ing' then add 'ly' instead. If the string length of the given string is less than 3, leave it unchanged.

Sample String : 'abc' Expected Result : 'abcing' Sample String : 'string' Expected Result : 'stringly'

b. To get a string from a given string where all occurrences of its first char have been changed to '$', except the first char itself.

Sample String : 'restart' E**xpected Result :** **'resta$t'** **a. Python Code**:

```
def add_string(str1):
        length =
        len(str1)
    if length >
        2:
        if str1[-3:] ==
            'ing':
        str1 +=
      'ly' else:
        str1 +=
   'ing' return str1
print(add_string('ab'))
print(add_string('abc'))
print(add_string('string'
))
```

b. PYTHON CODE:

```
def change_char(str1):
  char = str1[0] length
  = len(str1)
  str1 = str1.replace(char,
  '$') str1 = char + str1[1:]
return str1
print(change_char('restart'
))
```

VIVA QUESTIONS-

1. What are the built-in type does python provides?
2. In Python what are iterators?

EXPERIMENT-6

OBJECTIVE:-

a. To compute the input. The frequency of the words from the output should output after sorting the key alphanumerically.

b. Write a program that accepts a comma separated sequence of words as input and prints the words in a comma-separated sequence after sorting them alphabetically.

a. Python code:

```
def word_count(str):
    counts = dict()
    words = str.split()

    for word in words:
        if word in counts:
            counts[word] += 1
        else:
            counts[word] = 1

    return counts

print( word_count('the quick brown fox jumps over the lazy dog.'))
```

b.

```
items = input("Input comma separated sequence of
words") words = [word for word in items.split(",")]
print(",".join(sorted(list(set(words)))))
```

VIVA QUESTIONS:

1.Define string operations.
2.What is concatenation function?

EXPERIMENT-7

OBJECTIVE:- Write a program that accepts a sequence of whitespace separated words as input and prints the words after removing all duplicate words and sorting them alphanumerically.

THEORY:-

Suppose the following input is supplied to the program:

hello world and practice makes perfect and hello world again
Then, the output should be:
again and hello makes perfect practice world

Hints:
In case of input data being supplied to the question, it should be assumed to be a console input.
We use set container to remove duplicated data automatically and then use sorted() to sort the data.

Solution:

```
s = raw_input()
words = [word for word in s.split(" ")]
print " ".join(sorted(list(set(words))))
#----------------------------------------#

#----------------------------------------#
```

VIVA QUESTIONS:

1.What is Dynamic programming?

2**.** Mention what are the rules for local and global variables in Python?

EXPERIMENT-8

OBJECTIVE:- To demonstrate use of list & related functions

THEORY:- Python has a number of built-in data structures, including lists. Data structures provide us with a way to organize and store data, and we can use built-in methods to retrieve or manipulate that data.

list.append()

The method list.append(x) will add an item (x) to the end of a list. We'll start with a list of our fish that are dispersed throughout the aquarium.

```
fish = ['barracuda','cod','devil ray','eel']
```

This list is comprised of 4 string items, and their index numbers range from 'barracuda' at 0 to 'eel' at index 3.

We just got a new fish into the aquarium today, and we would like to add that fish to our list. We'll pass the string of our new fish type, 'flounder' into the list.append() method, and then print out our modified list to confirm that the item was added.

```
fish.append('flounder')
print(fish)
```

Output

```
['barracuda', 'cod', 'devil ray', 'eel', 'flounder']
```

Now, we have a list of 5 string items that ends with the item we passed to the .append() function.

List.insert()

The list.insert(i,x) method takes two arguments, with i being the index position you would like to add an item to, and x being the item itself.

Our aquarium acquired another new fish, an anchovy. You may have noticed that so far the list fish is in alphabetical order. Because of this, we don't want to just add the string 'anchovy' to the end of fishwith the list.append() function. Instead, we'll use list.insert() to add 'anchovy' to the beginning of this list at index position 0:

```
fish.insert(0,'anchovy')
print(fish)
```

Output

```
['anchovy', 'barracuda', 'cod', 'devil ray', 'eel', 'flounder']
```

In this case, we added the string item to the front of the list. Each of the successive items will now be at a new index number as they have all moved down. Therefore, 'barracuda' will be at index 1, 'cod' will be at index 2, and 'flounder' — the last item — will be at index 5.

If, at this point, we are bringing a damselfish to the aquarium and we wanted to maintain alphabetical order based on the list above, we would put the item at
index 3: fish.insert(3,'damselfish').

list.extend()

If we want to combine more than one list, we can use the list.extend(L) method, which takes in a second list as its argument.

Our aquarium is welcoming four new fish from another aquarium that is closing. We have these fish together in the list more_fish:

```
more_fish = ['goby','herring','ide','kissing gourami']
```

We'll now add the items from the list more_fish to the list fish and print the list to ensure that the second list was incorporated:

```
fish.extend(more_fish)
print(fish)
```

list.remove()

When we need to remove an item from a list, we'll use the list.remove(x) method which removes the first item in a list whose value is equivalent to x.

A group of local research scientists have come to visit the aquarium. They are doing research on the kissing gourami species of fish. They have requested for us to loan our kissing gourami to

'kissing gourami'

```
fish.remove('kissing gourami')
print(fish)
```

Following the use of the list.remove() method, our list no longer has the 'kissing gourami' item.

VIVA QUESTIONS:

1.How do you remove duplicates froma list?

2. How To Determine The Size Of Your List in Python?

EXPERIMENT-9

OBJECTIVE:-How To Determine The Size Of Your List in Python

THEORY:- Python dictionary is an unordered collection of items. While other compound data types have only value as an element, a dictionary has a key: value pair.

Dictionaries are optimized to retrieve values when the key is known

How to create a dictionary?

Creating a dictionary is as simple as placing items inside curly braces {} separated by comma. An item has a key and the corresponding value expressed as a pair, key: value.

While values can be of any data type and can repeat, keys must be of immutable type (string,number or tuple with immutable elements) and must be unique.

```
# empty dictionary
my_dict = {}
# dictionary with integer keys
my_dict = {1: 'apple', 2: 'ball'}
# dictionary with mixed keys
my_dict = {'name': 'John', 1: [2, 4, 3]}
# using dict()
my_dict = dict({1:'apple', 2:'ball'})
# from sequence having each item as a
pair my_dict = dict([(1,'apple'), (2,'ball')])
```

How to access elements from a dictionary?

While indexing is used with other container types to access values, dictionary uses keys. Key can be used either inside square brackets or with the `get()` method.

The difference while using `get()` is that it returns `None` instead of `KeyError`, if the key is not found.

Python code:

```
my_dict = {'name':'Jack', 'age': 26}

# Output: Jack
print(my_dict['name'])

# Output: 26
print(my_dict.get('age')

# Trying to access keys which doesn't exist throws error
# my_dict.get('address')
# my_dict['address']
```

How to change or add elements in a dictionary?

Dictionary are mutable. We can add new items or change the value of existing items using assignment operator.

If the key is already present, value gets updated, else a new key: value pair is added to the dictionary.

Python code:

```
my_dict = {'name':'Jack', 'age': 26}

# update value
my_dict['age'] = 27

#Output: {'age': 27, 'name': 'Jack'}
print(my_dict)

# add item
my_dict['address'] = 'Downtown'


# Output: {'address': 'Downtown', 'age': 27, 'name':
'Jack'} print(my_dict)
```

How to delete or remove elements from a dictionary?

We can remove a particular item in a dictionary by using the method `pop()`. This method removes as item with the provided key and returns the value.

The method, `popitem()` can be used to remove and return an arbitrary item (key, value) form the dictionary. All the items can be removed at once using the `clear()` method.

We can also use the `del` keyword to remove individual items or the entire dictionary itself.

Python code:

```
# create a dictionary

squares = {1:1, 2:4, 3:9, 4:16, 5:25}

# remove a particular item
# Output: 16
print(squares.pop(4))

# Output: {1: 1, 2: 4, 3: 9, 5: 25}
print(squares)

# remove an arbitrary item
# Output: (1, 1)
print(squares.popitem())
```

```
# Output: {2: 4, 3: 9, 5: 25}
print(squares)

# delete a particular item
del squares[5]

# Output: {2: 4, 3: 9}
print(squares)

# remove all items
squares.clear()


# Output: {}
print(squares)

# delete the dictionary itself
del squares

# Throws Error
# print(squares)
```

VIVA QUESTIONS:-

1.What is Dictionary?

2. When does a dictionary is used instead of a list?

EXPERIMENT-10

OBJECTIVE:-To demonstrate use of tuple, set& related functions

THEORY: In Python programming, a tuple is similar to a list. **The** difference between the two is that we cannot change the elements of a tuple once it is assigned whereas in a list, elements can
be changed.

Creating a Tuple

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.).

Python code:

```
# empty tuple
# Output: ()
my_tuple = ()
print(my_tuple
)

# tuple having integers
# Output: (1, 2, 3)
my_tuple = (1, 2,
3) print(my_tuple)

# tuple with mixed datatypes
# Output: (1, "Hello", 3.4)
my_tuple = (1, "Hello",
3.4) print(my_tuple)

# nested tuple
# Output: ("mouse", [8, 4, 6], (1, 2, 3))
my_tuple = ("mouse", [8, 4, 6], (1, 2,
3)) print(my_tuple)

# tuple can be created without parentheses
# also called tuple packing
# Output: 3, 4.6, "dog"

my_tuple = 3, 4.6,
"dog" print(my_tuple)
```

```
# tuple unpacking is also possible
# Output:
# 3
# 4.6
# dog
a, b, c = my_tuple
print(a)
print(b)
print(c)
```

Accessing Elements in a Tuple

There are various ways in which we can access the elements of a tuple.

1. **Indexing**

   Python code:

```
my_tuple = ('p','e','r','m','i','t')

# Output: 'p'
print(my_tuple[0])

# Output: 't'
print(my_tuple[5])

# index must be in range
# If you uncomment line 14,
# you will get an error.
# IndexError: list index out of range

#print(my_tuple[6])

# index must be an integer
# If you uncomment line 21,
# you will get an error.
# TypeError: list indices must be integers, not float

#my_tuple[2.0]

# nested tuple
n_tuple = ("mouse", [8, 4, 6], (1, 2, 3))

# nested index
# Output: 's'
print(n_tuple[0][3])
```

```
# nested index
# Output: 4
print(n_tuple[1][1])
```

2. Deleting a Tuple

Deleting a tuple entirely is possible using the keyword del.

```
my_tuple = ('p','r','o','g','r','a','m','i','z')

# can't delete items
# if you uncomment line 8,
# you will get an error:
# TypeError: 'tuple' object doesn't support item deletion

#del my_tuple[3]

# can delete entire tuple
# NameError: name 'my_tuple' is not
defined del my_tuple
my_tuple
```

VIVA QUESTIONS

1.What is a tuple?

2.Differentiate between a tuple and a list?

EXPERIMENT NO 11

OBJECTIVE:-To implement stack using list.

THEORY

Stack is a linear data structure which follows a particular order in which the operations are performed. The order may be LIFO(Last In First Out) or FILO(First In Last Out).
Mainly the following three basic operations are performed in the stack:

- **Push:** Adds an item in the stack. If the stack is full, then it is said to be an Overflow condition.
- **Pop:** Removes an item from the stack. The items are popped in the reversed order in which they are pushed. If the stack is empty, then it is said to be an Underflow condition.
- Peek or Top: Returns top element of stack.
- **isEmpty:** Returns true if stack is empty, else false.

How to understand a stack practically?

There are many real life examples of stack. Consider the simple example of plates stacked over one another in canteen. The plate which is at the top is the first one to be removed, i.e. the plate which has been placed at the bottommost position remains in the stack for the longest period of time. So, it can be simply seen to follow LIFO/FILO order.

```
# Python program for implementation of stack
# import maxsize from sys module
# Used to return -infinite when stack is
empty from sys import maxsize

# Function to create a stack. It initializes size of stack as
0 def createStack():
        stack = []
        return stack

# Stack is empty when stack size is 0
def isEmpty(stack):
        return len(stack) == 0

# Function to add an item to stack. It increases size by
1 def push(stack, item):
        stack.append(item)
        print("pushed to stack " + item)

# Function to remove an item from stack. It decreases size by
1 def pop(stack):
        if (isEmpty(stack)):
                return str(-maxsize -1) #return minus
        infinite return stack.pop()
```

VIVA QUESTIONS

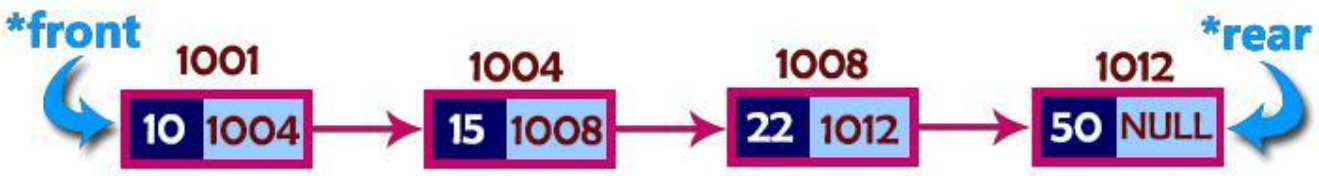
a) What is Stack and how implementation of Stack in c is different from python?

b) Difference between queue and stack?

EXPERIMENT NO 12

OBJECTIVE :- To implement queue using list

THEORY:- In a Queue data structure, we maintain two pointers, front and rear. The front points the first item of queue and rear points to last item.
enQueue() This operation adds a new node after rear and moves rear to the next node.
deQueue() This operation removes the front node and moves front to the next node.




```
# Python code to demonstrate Implementing
# Queue using deque and
list from collections
import deque
queue = deque(["Ram", "Tarun", "Asif", "John"])
print(queue)
queue.append("Ak
bar") print(queue)
queue.append("Bir
bal") print(queue)
print(queue.poplef
t())
print(queue.poplef
t()) print(queue)
```

VIVA QUESTIONS

c) What is queue and how implementation of queue in c is different from python?
d) Difference between queue and stack?

EXPERIMENT NO 13

OBJECTIVE :-To read and write from a file

THEORY:- When you're working with Python, you don't need to import a library in order to read and write files. It's handled natively in the language, albeit in a unique manner.

Python provides inbuilt functions for creating, writing and reading files. There are two types of files that can be handled in python, normal text files and binary files (written in binary language,0s and 1s).

- **Text files:** In this type of file, Each line of text is terminated with a special character called EOL (End of Line), which is the new line character ('\n') in python by default.
- **Binary files:** In this type of file, there is no terminator for a line and the data is stored after converting it into machine understandable binary language.

In this article, we will be focusing on opening, closing, reading and writing data in a text file.

File Access Modes

Access modes govern the type of operations possible in the opened file. It refers to how the file will be used once its opened. These modes also define the location of the **File Handle** in the file. File handle is like a cursor, which defines from where the data has to be read or written in the file. There are 6 access modes in python.

- **Read Only ('r') :** Open text file for reading. The handle is positioned at the beginning of the file. If the file does not exists, raises I/O error. This is also the default mode in which file is opened.
- **Read and Write ('r+') :** Open the file for reading and writing. The handle is positioned at the beginning of the file. Raises I/O error if the file does not exists.
- **Write Only ('w') :** Open the file for writing. For existing file, the data is truncated and over-written. The handle is positioned at the beginning of the file. Creates the file if the file does not exists.
- **Write and Read ('w+')** : Open the file for reading and writing. For existing file, data is truncated and over-written. The handle is positioned at the beginning of the file.
- **Append Only ('a')** : Open the file for writing. The file is created if it does not exist. The handle is positioned at the end of the file. The data being written will be inserted at the end, after the existing data.
- **Append and Read ('a+') :** Open the file for reading and writing. The file is created if it does not exist. The handle is positioned at the end of the file. The data being written will be inserted at the end, after the existing data.

Opening a File

```
# Open function to open the file "MyFile1.txt"
# (same directory) in append mode and
file1 = open("MyFile.txt","a")
```

```
# store its reference in the variable file1
# and "MyFile2.txt" in D:\Text in file2
file2 = open(r"D:\Text\MyFile2.txt","w+")
```

Closing a file

```
# Opening and Closing a file "MyFile.txt"
# for object name file1.
file1 = open("MyFile.txt","a")
file1.close()
```

Writing to a file

There are two ways to write in a file.

1. **write() :** Inserts the string str1 in a single line in the text

   ```
   file. File_object. write (str1)
   ```

2. **writelines() :** For a list of string elements, each string is inserted in the text file.Used to insert multiple strings at a single time.

   ```
   File_object.writelines(L) for L = [str1, str2, str3]
   ```

```
# Program to show various ways to read and
# write data in a file.
file1 = open("myfile.txt","w")
L = ["This is Delhi \n","This is Paris \n","This is London \n"]

# \n is placed to indicate EOL (End of
Line) file1.write("Hello \n")
file1.writelines(L)
file1.close() #to change file access modes

file1 = open("myfile.txt","r+")

print "Output of Read function is "
print file1.read()
print

# seek(n) takes the file handle to the nth
# bite from the beginning.
file1.seek(0)

print "Output of Readline function is "
print file1.readline()
print

file1.seek(0)

# To show difference between read and readline
```

```
print "Output of Read(9) function is "
print file1.read(9)
print file1.seek(0)
print "Output of Readline(9) function is "
print file1.readline(9)

file1.seek(0)
# readlines function
print "Output of Readlines function is "
print file1.readlines()
print
file1.close()
```

VIVA QUESTIONS

1.What is write command?

2.What is the difference between write and append?

Department of Computer Science & Engineering

EXPERIMENT NO 14

OBJECTIVE: Appending to a file

THEORY:- To add something at the end. For example, you can **append** one **file** to another or you can **append** a field to a record. Do not confuse **append** with insert. **Append**always means to add at the end. Insert means to add in between.

```
# Python program to illustrate
# Append vs write mode
file1 = open("myfile.txt","w")
L = ["This is Delhi \n","This is Paris \n","This is London
\n"] file1.close()

# Append-adds at last
file1 = open("myfile.txt","a")#append mode
file1.write("Today \n")
file1.close()

file1 = open("myfile.txt","r")
print "Output of Readlines after appending"
print file1.readlines()
print
file1.close()

# Write-Overwrites
file1 = open("myfile.txt","w")#write mode
file1.write("Tomorrow \n")
file1.close()

file1 = open("myfile.txt","r")
print "Output of Readlines after writing"
print file1.readlines()
print
file1.close()
```

VIVA QUESTIONS

1.What is the difference write and read mode?
2.What are the symbols used for the mode?

Department of Computer Science & Engineering

EXPERIMENT NO 15

OBJECTIVE :-To demonstrate working of classes and objects

THEORY: Like function definitions begin with the keyword def, **in** Python, we define a class using the keyword class.The first string is called docstring and has a brief description about the class. Although not mandatory, this is recommended.

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

class MyClass:
        "This is my second class"
                a = 10
        def func(self):
                print('Hello')

# Output: 10
print(MyClass.a)

# Output: <function MyClass.func at
0x0000000003079BF8> print(MyClass.func)

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

Creating an Object in Python

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

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

```
>>> ob = MyClass()
class MyClass:
        "This is my second
        class" a = 10
        def func(self):
                print('Hello')

# create a new MyClass
ob = MyClass()

# Output: <function MyClass.func at 0x000000000335B0D0>
```

```
print(MyClass.func)

# Output: <bound method MyClass.func of < main .MyClass object at 0x000000000332DEF0>>

print(ob.func)

# Calling function func()
# Output:
Hello ob.func()
```

VIVA QUESTIONS

1. What is class?How to define class in python?
2. What is object?How to use it in python?

Department of Computer Science & Engineering

EXPERIMENT NO 16

OBJECTIVE : To demonstrate class method & static method

THEORY:- Class Method

The @classmethod decorator, is a builtin function decorator that is an expression that gets evaluated after your function is defined. The result of that evaluation shadows your function definition.

A class method receives the class as implicit first argument, just like an instance method receives the instance

```
class C(object):
    @classmethod
    def fun(cls, arg1, arg2, ...):
        ....
```

fun: function that needs to be converted into a class method
returns: a class method for function.

- A class method is a method which is bound to the class and not the object of the class.
- They have the access to the state of the class as it takes a class parameter that points to the class and not the object instance.
- It can modify a class state that would apply across all the instances of the class. For example it can modify a class variable that will be applicable to all the instances.
- **Static Method**
    - A static method does not receive an implicit first argument.

```
class C(object):
    @staticmethod
    def fun(arg1, arg2, ...):
        ...
```

returns: a static method for function fun.

```
# Python program to demonstrate
# use of class method and static
method. from datetime import date

class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    # a class method to create a Person object by birth
    year. @classmethod
    def fromBirthYear(cls, name, year): return
        cls(name, date.today().year - year)
```

```
    # a static method to check if a Person is adult or not.
    @staticmethod
    def isAdult(age):
        return age > 18

person1 = Person('mayank', 21)
person2 = Person.fromBirthYear('mayank', 1996)

print person1.age
print person2.age

# print the result
print Person.isAdult(22)
```

VIVA QUESTIONS

1. What is static method in java?
2. What is auto and static in C?

EXPERIMENT NO. 17

OBJECTIVE :-To demonstrate constructors

THEORY

Class functions that begins with double underscore ( ) are called special functions as they have special meaning.

One particular interest is the init () function. This special function gets called whenever a new object of that class is instantiated.

This type of function is also called constructors in Object Oriented Programming (OOP). We normally use it to initialize all the variables.

```
class ComplexNumber:
    def __init__(self,r = 0,i = 0):
        self.real = r
        self.imag = i
    def getData(self):
        print("{0}+{1}j".format(self.real,self.imag))
# Create a new ComplexNumber
object c1 = ComplexNumber(2,3)
# Call getData() function
# Output: 2+3j
c1.getData()
# Create another ComplexNumber object
# and create a new attribute 'attr'
c2 = ComplexNumber(5)
c2.attr = 10
# Output: (5, 0, 10)
print((c2.real, c2.imag, c2.attr))

# but c1 object doesn't have attribute 'attr'
# AttributeError: 'ComplexNumber' object has no attribute
'attr' c1.attr
```

VIVA QUESTIONS

1. What is constructor in java ?

2. What is destructor in C++?

EXPERIMENT NO 18

OBJECTIVE :-To demonstrate inheritance

THEORY:- Inheritance is a powerful feature in object oriented programming.

It refers to defining a new class with little or no modification to an existing class. The new class is called **derived (or child) class** and the one from which it inherits is called the **base (or parent) class**.

SYNTAX
class BaseClass:
Body of base class
class DerivedClass(BaseClass):
Body of derived class
Derived class inherits features from the base class, adding new features to it. This results into re-usability of code.

```
class Polygon:
    def __init__(self, no_of_sides):
        self.n = no_of_sides
        self.sides = [0 for i in range(no_of_sides)]

    def inputSides(self):
        self.sides = [float(input("Enter side "+str(i+1)+" : ")) for i in range(self.n)]

    def dispSides(self):
        for i in range(self.n):
            print("Side",i+1,"is",self.sides[i])
```

This class has data attributes to store the number of sides, n and magnitude of each side as a list, sides.
Method inputSides() takes in magnitude of each side and similarly, dispSides() will display these properly.
A triangle is a polygon with 3 sides. So, we can created a class called Triangle which inherits from Polygon. This makes all the attributes available in class Polygon readily available in Triangle. We don't need to define them again (code re-usability). Triangle is defined as follows.

```
class Triangle(Polygon):
    def __init__(self):
        Polygon.__init__(self,3)
    def findArea(self):
        a, b, c = self.sides
        # calculate the semi-
        perimeter s = (a + b + c) / 2
        area = (s*(s-a)*(s-b)*(s-c)) ** 0.5
        print('The area of the triangle is %0.2f' %area)
```

VIVA QUESTIONS

1. What is Polymorphism?

2. What is multiple and multilevel inheritance?

EXPERIMENT NO 19

OBJECTIVE :-To demonstrate aggregation/composition

THEORY:- Composition. **Composition** is a specialized form of **aggregation** in which if the parent object is destroyed, the child objects would cease to exist. It is actually a strong type of **aggregation** and is also referred to as a "death" relationship. According to some formal definitions the term composition implies that the two objects are quite strongly linked – one object can be thought of as belonging exclusively to the other object. If the owner object ceases to exist, the owned object will probably cease to exist as well. If the link between two objects is weaker, and neither object has exclusive ownership of the other, it can also be called aggregation.

```
class Student:
    def __init__(self, name, student_number):
        self.name = name
        self.student_number = student_number
        self.classes = []

    def enrol(self, course_running):
        self.classes.append(course_running)
        course_running.add_student(self)


class Department:
    def __init__(self, name, department_code):
        self.name = name
        self.department_code = department_code
        self.courses = {}

    def add_course(self, description, course_code, credits):
        self.courses[course_code] = Course(description, course_code, credits, self)
        return self.courses[course_code]


class Course:
    def __init__(self, description, course_code, credits, department):
        self.description = description
        self.course_code = course_code
        self.credits = credits
        self.department = department
        self.department.add_course(self)

        self.runnings = []

    def add_running(self, year):
        self.runnings.append(CourseRunning(self,
        year)) return self.runnings[-1]
```

```
class CourseRunning:
    def __init__(self, course, year):
        self.course = course
        self.year = year
        self.students = []

    def add_student(self, student):
        self.students.append(student)


maths_dept = Department("Mathematics and Applied Mathematics", "MAM")
mam1000w = maths_dept.add_course("Mathematics 1000", "MAM1000W", 1)
mam1000w_2013 = mam1000w.add_running(2013)

bob = Student("Bob", "Smith")
bob.enrol(mam1000w_2013)
```

VIVA QUESTIONS

1. What is aggregation?
2. What is generalization?

Department of Computer Science & Engineering

EXPERIMENT NO 20

OBJECTIVE:- To create a small GUI application for insert, update and delete in a table using Oracle as backend and front end for creating form

THEORY:- Installing MySQL Python connector is straightforward. For example, to install it in Windows environment, you use the following steps:

Unpack the download file into a temporary directory e.g., C:\temp

Start a console window and switch to the folder where you unpack the connector

> cd c:\temp
1
> cd c:\temp
Inside the c:\temp folder, use the following command:
c:\temp > python setup.py install
1
c:\temp > python setup.py install
Verifying MySQL Connector/Python installation

After installing the MySQL Python connector, you need to test it to make sure that it is working correctly and you are able to connect to MySQL database server without any issues. To verify the installation, you use the following steps:

Open Python command line

Type the following code

>>> import mysql.connector

>>> mysql.connector.connect(host='localhost',database='mysql',user='root',password='')

1
2
>>> import mysql.connector

>>> mysql.connector.connect(host='localhost',database='mysql',user='root',password='')

If the output is shown as below, you have been successfully installing the MySQL Python connector in your system.

<mysql.connector.connection.MySQLConnection object at

0x0187AE50> 1

<mysql.connector.connection.MySQLConnection object at 0x0187AE50>

EXPERIMENT NO 21

OBJECTIVE:- Write a program to compute area and circumference of a triangle. Take input from user.

THEORY : Area of Triangle is math.sqrt((s*(s-a)*(s-b)*(s-c)))

Python Code :

```
# Python Program to find Area of a Triangle using

Functions import math

def Area_of_Triangle(a, b, c):

    # calculate the Perimeter
    Perimeter = a + b + c
    # calculate the semi-
    perimeter s = (a + b + c) / 2

    # calculate the area
    Area = math.sqrt((s*(s-a)*(s-b)*(s-c)))

    print("\n The Perimeter of Traiangle = %.2f" %Perimeter);
    print(" The Semi Perimeter of Traiangle = %.2f" %s);
    print(" The Area of a Triangle is %0.2f" %Area)

Area_of_Triangle(6, 7, 8)
```

VIVA QUESTIONS

1. What is formula of area of traingle?
2. What is value of S?

EXPERIMENT NO 22

OBJECTIVE:- Write a program to check if a number is Odd or even. Take input from user.

THEORY : If the number is divisible by 2 , that number is called even number, else the number is odd.

Python Code :

```
# Python program to check if the input number is odd or even.
# A number is even if division by 2 give a remainder of 0.
# If remainder is 1, it is odd number.
num = int(input("Enter a number: "))
if (num % 2) == 0:
   print("{0} is Even".format(num))
else:
   print("{0} is Odd".format(num))
```

VIVA QUESTIONS

1. Which number is called odd number?
2. What is if… else condition ?

Computer Science & Engineering

EXPERIMENT NO 23

OBJECTIVE:-Write a program to check that a given year is Leap Year or not

THEORY : If the year is divisible by 4, 100 and 400 then that year is a leap year.

Python Code :

```
year = 2000
# To get year (integer input) from the user
# year = int(input("Enter a year: "))
if (year % 4) == 0:
   if (year % 100) == 0:
      if (year % 400) == 0:
         print("{0} is a leap year".format(year))
      else:
         print("{0} is not a leap year".format(year))
   else:
      print("{0} is a leap year".format(year))
else:
   print("{0} is not a leap year".format(year))
```

VIVA QUESTIONS :

1. What do you mean by Leap Year?
2. What is nested if.. else condition?

RCS454: PYTHON LANGUAGE PROGRAMMING LAB

Write a Python program to: -

1. Demonstrate the working of ‘id’ and ‘type’ functions
2. To find all prime numbers within a given range.
3. To print ‘n terms of Fibonacci series using iteration.
4. To demonstrate use of slicing in string
5. (a) To add 'ing' at the end of a given string (length should be at least 3). If the given string already ends with 'ing' then add 'ly' instead. If the string length of the given string is less than 3, leave it unchanged. Sample String : 'abc' Expected Result : 'abcing' Sample String : 'string' Expected Result : 'stringly'

   (b) To get a string from a given string where all occurrences of its first char have been changed to '$', except the first char itself.
6. (a) To compute the frequency of the words from the input. The output should output after sorting the key alphanumerically.

   (b) Write a program that accepts a comma separated sequence of words as input and prints the words in a comma-separated sequence after sorting them alphabetically.
7. Write a program that accepts a sequence of whitespace separated words as input and prints the words after removing all duplicate words and sorting them alphanumerically.
8. To demonstrate use of list & related functions
9. To demonstrate use of Dictionary& related functions
10. To demonstrate use of tuple, set& related functions
11. To implement stack using list
12. To implement queue using list
13. To read and write from a file
14. To copy a file
15. To demonstrate working of classes and objects
16. To demonstrate class method & static method
17. To demonstrate constructors
18. To demonstrate inheritance
19. To demonstrate aggregation/composition
20. To create a small GUI application for insert, update and delete in a table using Oracle as