Savitribai Phule Pune University

T. Y. B. B. A. (C. A.) Semester-V (CBCS 2019 Pattern)

Core Java, MongoDB / Python CA-506: Lab Book

Student Name: ____

College Name: ____

Roll No.: ____________ **Division:** ____________ **Seat No:** ____________

Academic Year: ______________________________

CERTIFICATE

This is to certify that Mr./Ms.________________________________ Seat Number ___________ of T.Y.B.B.A. (C.A) Sem-V has successfully completed Laboratory course (Core Java, Mongo DB/ Python in the year ____________________. He/She has scored _____mark out of 10 (For Lab Book).

Subject Teacher **H.O.D./Coordinator**

Internal Examiner **External Examiner**

Introduction

1. **About the work book:**
   This workbook is intended to be used by T.Y.B.B.A. (C.A.) Semester-V students for Core java, MongoDB and Python Practical assignments. This workbook is designed by considering all the practical topics mentioned in syllabus.

2. **The objectives of this workbook are:**
   - Defining the scope of the course.
   - To bring the uniformity in the practical conduction and implementation in all colleges affiliated to SPPU.
   - To have continuous assessment of the course and students.
   - Providing ready reference for the students during practical implementation.
   - Provide more options to students so that they can have good practice before facing the examination.
   - Catering to the demand of slow and fast learners and accordingly providing the practice assignments to them.

3. **How to use this workbook:**
   The workbook is divided into two sections. Section-I is related to Core Java assignments, Section-II is related to MongoDB assignments or Python assignments.

   Section-I: Core java is divided into five assignments.
   Section-II: MongoDB is divided into five assignments.
   OR
   Section-II: Python is divided into eight assignments.

   Students have to perform practical assignments of selected elective subject from Section-II.
   Each assignment of all sections has three SETs-A, B and C. It is mandatory for students to complete SET A and SET B in lab. It also includes practice set which are expected to be solved by students as home assignments and to be evaluated by subject teachers.

4. **Instructions to the students:**
   Please read the following instructions carefully and follow them during practical.
   - Students are expected to carry this workbook every time they come to the lab for computer practical.
   - Students should prepare for the assignment by reading the relevant material which is mentioned in ready reference and the concepts taught in class.
   - Instructor will specify which problems to solve in the lab during the allotted slot and student should complete them and get verified by the instructor. However, student should spend additional hours in Lab and at home to cover all workbook assignments if needed.
   - Students will be assessed for each exercise on a scale from 0 to 5.

Not Done	0
Incomplete	1
Late Complete	2
Needs improvement	3
Complete	4
Well Done	5

5. Instruction to the Instructors:

Make sure that students should follow above instructions.

- Explain the assignment and related concepts in around ten minutes using whiteboard if required or by demonstrating the software.
- Evaluate each assignment carried out by a student on a scale of 5 as specified above by ticking appropriate box.
- The value should also be entered on assignment completion page of the respective Lab course.

6. Instructions to the Lab administrator:

You have to ensure appropriate hardware and software is made available to each student.

The operating system and software requirements on server side and also client side are as given below:

- Operating System - Windows
- Python 3.0
- MongoDB Community Edition
- JDK

Assignment Completion Sheet

Section-I: Core Java			
Sr. No.	**Assignment Name**	**Marks (out of 5)**	**Teacher's Sign**
1	Introduction to Java		
2	Classes, Objects and Methods		
3	Inheritance, Package and Collection		
4	File and Exception Handling		
5	Applet, AWT, Event & Swing Programming		
Total (Out of 25)			
Total (Out of 5)			

Instructor Signature:

Section-II: MongoDB			
Sr. No.	**Assignment Name**	**Marks (out of 5)**	**Teacher's Sign**
1	MongoDB Basics		
2	MongoDB Operators		
3	Update and Delete operation in MongoDB		
4	MongoDB Cursor		
5	MongoDB Index and Aggregation		
Total (Out of 25)			
Total (Out of 5)			

'OR'

Section-II: Python			
Sr. No.	**Assignment Name**	**Marks (out of 5)**	**Teacher's Sign**
1	Introduction to Basic Python		
2	Working with Strings and List		
3	Working with Tuples, Sets and Dictionaries		
4	Working with Functions, Modules and Packages		
5	Python Classes and Objects		
6	Inheritance		
7	Exception Handling		
8	Python GUI Programming using Tkinter		
Total (Out of 40)			
Total (Out of 5)			

Instructor Signature:

Section – I
Core Java

Assignment No. 1: Introduction to Java

Jdk (Java Development Kit) Tools:

Java environment includes a number of development tools, classes and methods. The development tools are part of the system known as Java Development Kit (JDK) and the classes and methods are part of the Java Standard Library (JSL), also known as the Application Programming Interface (API).

Java Development kit (JDK) – The JDK comes with a set of tools that are used for developing and running Java program. It includes:

1) **appletviewer:** It is used for viewing the applet
2) **javac:** It is a Java Compiler
3) **java:** It is a java interpreter
4) **javap:** It is Java diassembler, which convert byte code into program description.
5) **javah:** It is for java C header files.
6) **javadoc:** It is for creating HTML document.
7) **jdb:** It is Java debugger.

Data Types:

Type	Description(Range)
boolean	These have values of either true or false.
byte	7-bit 2s-compliment integer with values between –27 and 27-1 (–128 to 127).
short	16-bit 2s-compliment integer with values between –215 and 215-1 (–32,768 to 32,767).
char	16-bit Unicode characters. For alphanumerics, these are the same as ASCII with the high byte set to 0. The numerical values are unsigned 16-bit values between 0 and 65535.
int	32-bit 2s-compliment integer with values between –231 and 231-1 (–2,147,483,648 to 2, 147,483,647).
long	64-bit 2s-compliment integer with values between –263 and 263-1 (–9223372036854775808 to 9223372036854775807).
float	32-bit single precision floating point numbers using the IEEE 754-1985 standard (+/– about 1039).
double	64-bit double precision floating point numbers using the IEEE 754-1985 standard (+/– about 10317).

Structure of java Program:

A Java program may contain many classes of which only one class defines a main method. Classes contain data members and methods that operate on the data members of the class. Methods may contain data type declarations and executable statements.

Documentation Section
Package Statement
Import Statements
Interface Statements
Class Definitions

Main Method Class
{
  Main Method Definition
}

Command Line Arguments:

In Java, The command-line arguments allow the programmers to pass the arguments during the execution of a program. The users can pass the arguments during the execution by passing the command-line arguments inside the main () method.

For Example:

```
class CommandLineDemo
{
        public static void main(String args[ ])
        {
                System.out.println("The command-line arguments are:\n");
                for (int i = 0; i < args.length; i++)
                        System.out.println( args[ i ]);
        }
}
```

Steps to run the above program:

To compile and run a java program on command prompt:

1. Save the program as CommandLineDemo.java
2. Open the command prompt window and compile the program using
   ```
   javac CommandLineDemo.java
   ```
3. After a successful compilation of the program, execute the program using
   ```
   java CommandLineDemo   Argument-list
   ```

For example – *java CommandLineDemo Hello*

Press Enter and you will get the desired output.

Output: Hello

Java Array

Array is a collection of similar type of elements that have contiguous memory location. Java array is an object that contains elements of similar data type. Only fixed set of elements can be stored in a java array.

Syntax:

There are two sections in the syntax of an array:

Declaration:

```
dataType[] arr; (or)
dataType []arr; (or)
dataType arr[];
```

Instantiation:

```
arrayRefVar=new datatype[size];
```

Use of length property of an array:

To calculate size of an array.

Size=arrayname.length;

For Example: Java Program for demonstration of an array using command line arguments.

```
class Demo
{
public static void main (String args [])
{
   int i,n;
   n=args.length;
   int a[]=new int[n];
   for(i=0;i<n;i++)
   {
      a[i]=Integer.parseInt(args[i]);
   }
   //printing array
     for (int i=0;i<n;i++)
        System.out.println (a[i]);
 }}
```

String:

The set of characters are collectively called String. It is Wrapper class. Anything, if we declare with String class, java compiler considered it as an object. It is immutable.

The List of functions of String:

Sr. No	Methods with Description
1	**char charAt(int index)** Returns the character at the specified index.
2	**int compareTo(Object o)** Compares this String to another Object.

3	**int compareTo(String anotherString)** Compares two strings lexicographically.
4	**int compareToIgnoreCase(String str)** Compares two strings lexicographically, ignoring case differences.
5	**String concat(String str)** Concatenates the specified string to the end of this string.
6	**boolean contentEquals(StringBuffer sb)** Returns true if and only if this String represents the same sequence of characters as the specified StringBuffer.
7	**static String copyValueOf(char[] data)** Returns a String that represents the character sequence in the array specified.
8	**static String copyValueOf(char[] data, int offset, int count)** Returns a String that represents the character sequence in the array specified.
9	**boolean endsWith(String suffix)** Tests if this string ends with the specified suffix.
10	**boolean equals(Object anObject)** Compares this string to the specified object.
11	**boolean equalsIgnoreCase(String anotherString)** Compares this String to another String, ignoring case considerations.
12	**byte getBytes()** Encodes this String into a sequence of bytes using the platform's default charset, storing the result into a new byte array.
13	**byte[] getBytes(String charsetName)** Encodes this String into a sequence of bytes using the named charset, storing the result into a new byte array.
14	**void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)** Copies characters from this string into the destination character array.
15	**int hashCode()** Returns a hash code for this string.
16	**int indexOf(int ch)** Returns the index within this string of the first occurrence of the specified character.
17	**int indexOf(int ch, int fromIndex)** Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.
18	**int indexOf(String str)** Returns the index within this string of the first occurrence of the specified substring.
19	**int indexOf(String str, int fromIndex)** Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.
20	**String intern()** Returns a canonical representation for the string object.
21	**int lastIndexOf(int ch)** Returns the index within this string of the last occurrence of the specified character.
22	**int lastIndexOf(int ch, int fromIndex)**

	Returns the index within this string of the last occurrence of the specified character, searching backward starting at the specified index.
23	**int lastIndexOf(String str)** Returns the index within this string of the rightmost occurrence of the specified substring.
24	**int lastIndexOf(String str, int fromIndex)** Returns the index within this string of the last occurrence of the specified substring, searching backward starting at the specified index.
25	**int length()** Returns the length of this string.
26	**boolean matches(String regex)** Tells whether or not this string matches the given regular expression.
27	**boolean regionMatches(boolean ignoreCase, int toffset, String other, int ooffset, int len)** Tests if two string regions are equal.
28	**boolean regionMatches(int toffset, String other, int ooffset, int len)** Tests if two string regions are equal.
29	**String replace(char oldChar, char newChar)** Returns a new string resulting from replacing all occurrences of oldChar in this string with newChar.
30	**String replaceAll(String regex, String replacement)** Replaces each substring of this string that matches the given regular expression with the given replacement.
31	**String replaceFirst(String regex, String replacement)** Replaces the first substring of this string that matches the given regular expression with the given replacement.
32	**String[] split(String regex)** Splits this string around matches of the given regular expression.
33	**String[] split(String regex, int limit)** Splits this string around matches of the given regular expression.
34	**boolean startsWith(String prefix)** Tests if this string starts with the specified prefix.
35	**boolean startsWith(String prefix, int toffset)** Tests if this string starts with the specified prefix beginning a specified index.
36	**Char Sequence subSequence(int beginIndex, int endIndex)** Returns a new character sequence that is a subsequence of this sequence.
37	**String substring(int beginIndex)** Returns a new string that is a substring of this string.
38	**String substring(int beginIndex, int endIndex)** Returns a new string that is a substring of this string.
39	**char[] toCharArray()** Converts this string to a new character array.
40	**String toLowerCase()** Converts all of the characters in this String to lower case using the rules of the default locale.

41	**String toLowerCase(Locale locale)** Converts all of the characters in this String to lower case using the rules of the given Locale.
42	**String toString()** This object (which is already a string!) is itself returned.
43	**String toUpperCase()** Converts all of the characters in this String to upper case using the rules of the default locale.
44	**String toUpperCase(Locale locale)** Converts all of the characters in this String to upper case using the rules of the given Locale.
45	**String trim()** Returns a copy of the string, with leading and trailing whitespace omitted.
46	**static String valueOf(primitive data type x)** Returns the string representation of the passed data type argument.

Example: Java Program to display the files having extension .java (Use Command Argument).

```
class FileDisp
{
    public static void main(String args[])
    {
        int i,n;
        n=args.length;
        for(i=0;i<n;i++)
        {
            if(args[i].endsWith(".java"))
            {
                System.out.println(args[i]);
            }
        }
    }
```

Built In Packages:

The Java Standard Library (or APl) intrudes hundreds of classes and methods grouped into several functional packages. Most commonly used packages are as follows:

1) **Language Support Package:**
   A collection of classes and methods required for implementing basic features of Java.
2) **Utilities Package:**
   It is a collection of classes to provide utility functions such as date and time functions.
3) **Input / Output Package:**
   It is a collection of classes required for input/output manipulation.
4) **Networking Package:**
   It is a collection of classes for communicating with other computers via Internet.
5) **AWT Package:**

It is the Abstract Window Tool Kit package contains classes that implements platform independent graphical user interface.

6) **Applet Package:**
This includes an act of classes that allows us to create Java applets.

For adding the packages in an application, import statement is used.

Syntax:

```
import package_Name;
```

Example 1: Java program to accept the data and display it.(Use Scanner Class)

```
import java.util.*;
class Emp
{
  public static void main(String args[])
  {
      int e;
      String en;
      float s;
      Scanner ob=new Scanner(System.in);
      System.out.println("Eno Ename and Salary");
      e=ob.nextInt();
      en=ob.next();
      s=ob.nextFloat ();
      System.out.println("Emp No Is " + e);
      System.out.println("Emp Name" + en);
      System.out.println("Salary Is " + s);
  }
}
```

Example 2: Java Program to display date and time of a system.

```
        import  java.util.*;
        class DateTime
        {
        public static void main(String args[])
        {
              Date d=new Date();
              System.out.println("Date and Time of a System is " + d);
         }
    }
```

Practice Set:

1. Write a java Program to check whether given String is Palindrome or not.
2. Write a Java program which accepts three integer values and prints the maximum and minimum.
3. Write a Java program to accept a number from command prompt and generate multiplication table of a number.
4. Write a Java program to display Fibonacci series.
5. Write a Java program to calculate sum of digits of a number.
6. Write a Java program to accept a year and check if it is leap year or not.
7. Write a Java program to display characters from A to Z using loop.
8. Write a Java program to accept two numbers using command line argument and calculate addition, subtraction, multiplication and division.
9. Write a java Program to calculate the sum of first and last digit of a number.
10. Write a java program to calculate the sum of even numbers from an array.

Set A:

1. Write a java Program to check whether given number is Prime or Not.
2. Write a java Program to display all the perfect numbers between 1 to n.
3. Write a java Program to accept employee name from a user and display it in reverse order.
4. Write a java program to display all the even numbers from an array. (Use Command Line arguments)
5. Write a java program to display the vowels from a given string.

Set B:

1. Write a java program to accept n city names and display them in ascending order.
2. Write a java program to accept n numbers from a user store only Armstrong numbers in an array and display it.
3. Write a java program to search given name into the array, if it is found then display its index otherwise display appropriate message.
4. Write a java program to display following pattern:
```
5
4 5
3 4 5
2 3 4 5
1 2 3 4 5
```
5. Write a java program to display following pattern:
```
1
0 1
0 1 0
1 0 1 0
```

Set C:

1. Write a java program to count the frequency of each character in a given string.
2. Write a java program to display each word in reverse order from a string array.
3. Write a java program for union of two integer array.
4. Write a java program to display transpose of given matrix.
5. Write a java program to display alternate character from a given string.

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: WellDone [ ]

Signature of Instructor

Assignment No. 2: Classes, Objects and Methods

Class: Collection of objects is called class.

Syntax to create a Class:

A keyword class is used to create a class.

```
class Classname
{
        //Variable declaration
        //Method declaration
}
```

For Example: Number.java

```
class Number
{
        int x=10;               //variable declaration
}
```

Object: Objects have states and behaviors. It is defined as an instance of a class. An object contains an address and takes up some space in memory. Classes create objects and objects use methods to communicate between them.

Syntax to create an Object:

An object is created for a class by using the class name and new keyword.

For Example:

ClassName objectName=new ClassName();

Number Obj=new Number();

Instance variables and methods are accessed by using objects.

```
objectname.variableName;   //Accessing variable
objectname.methodName(); // Accessing a class method
```

For Example: Java Program to demonstrate Class & Object.

```
class Number
{
        int a,b;
        void disp()
        {
                System.out.println(a+ " " + b);
        }
}
class CDemo           // Main Class
{
        public static void main(String args[])
        {
                Number obj=new Number();
                obj.a=10;
                obj.b=20;
                obj.disp();
        }
}
```

Constructor:
A constructor in Java is a **special method** that is used to initialize objects of the class. The constructor is called when an object of a class is created. Constructor has same name as the class itself. Constructors have no explicit return type.

Types of Constructor:

a. **Default/No-argument constructor:** A constructor that has no parameter is known as default constructor.
b. **Parameterized constructor:** A constructor that takes parameter is known as parameterized constructor.

Constructor Overloading:
Constructor overloading is a technique of having more than one constructor with different parameter lists. The compiler differentiates these constructors depending on the number of parameters in the list and their types.

For Example: Java Program to demonstrate Constructor overloading.

```
class Number
{
        int num;
        Number()                //Default Constructor
        {
                num=10;
        }
        Number(int a)           //Parameterized Constructor
        {
                num=a;
        }
        void Display()
        {
                        System.out.println("num = " +num);
                }
}
```

```
class CDemo                     // Main Class
{
        public static void main(String args[])
        {
                Number obj1=new Number();          //invokes Default constructor
                Number obj2=new Number(50);        //invokes Parameterized constructor
                obj1.Display();
                obj2.Display();
        }
}
```

Method Overloading:

A class having two or more methods with the same name but different parameters is called as method overloading. It is used to perform similar task but on different input parameters.

For Example: Java Program to implement Method Overloading concept.

```
class Overload
    {
         int add(int a, int b)
        {
                int ans=a+b;
                System.out.println("Result= " +ans);
        }
        int add(int a, int b, int c)
        {
                int ans=a+b+c;
                System.out.println("Result= " +ans);

        }
}
class MethodOverloading
{
        public static void main (String args[])
        {
                Overload obj = new Overload();
                obj.add(10,20);
                obj.add(10, 20,30);
        }
}
```

Recursion:

A method that calls itself is known as a recursive method and this process is known as recursion. In java Recursion is a process in which a method calls itself continuously.

```
class RecFibbonacci
{
        int fib(int n)
        {
                if(n==0)
                        return 0;
                else if (n==1)
                        return 1;
                else
                        return fib(n-1)+fib(n-2);
        }

        public static void main(String args[])
        {
                int n,i;
                n=Integer.parseInt(args[0]);
                RecFibbonacci R=new RecFibbonacci();
                for(i=0;i<=n;i++)
                {
                        System.out.println(R.fib(i));
                }

        }
}
```

Passing Object as Parameter:

When primitive type is passed to a method, it is passed by value. But when an object is passed to a method, then it is known as call-by-reference. The updation done with in method will be reflected in the object.

Syntax:

```
        methodName(Object obj)
```

Returning Objects:

In java, a method can return any type of data, including objects.

Syntax:

```
        ObjectName methodName();
```

The new operator:
The new operator is used in Java to create new objects. It can also be used to create an array object.
Syntax to create object using new operartor

```
ClassName object=new ClassName();
```

The Array of Objects
Syntax :

```
Class_name [] objArray;
        or
Class_name objArray[];
```

Example:

```
Student[] S;
   or
Student S[] ;
```

Declare and instantiate the array of objects

```
Class obj[]= new Class[array_length]
```

Ex. Student S[] = new student[2];
It will create an array of objects 'S' with 2 elements/object references.
Imp Note that once an array of objects is instantiated; the individual elements of the array of objects need to be created using new.

For Example: Java Program to demonstrate Array of Object.

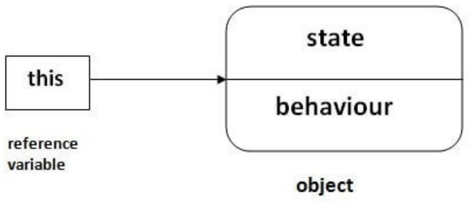
```
import java.util.*;
class Student
{
        int sid;
        String Sname;
        void accept(int id, String name)
        {
                sid=id;
                Sname=name;
        }
        void display()
        {
                System.out.println("Sid = " +sid);
                System.out.println("Sname= " + Sname);
        }
}
class ArrayObjectEx
{
  public static void main(String args[])
```

```
 {
        Scanner S=new Scanner(System.in);
        Student S1[]=new Student[3];

        for(int i=0; i<3;i++)
        {
                S1[i]=new Student();
                System.out.println("Enter id");
                S1[i].sid=S.nextInt();
                System.out.println("Enter Sname");
                S1[i].Sname=S.next();
                S1[i].accept(S1[i].sid, S1[i].Sname);
        }
        for(int i=0; i<3;i++)
        {
                S1[i].display();
        }
 }
}
```

this keyword:
In java, this is a **reference variable** that refers to the current object. If there are many objects and user don't know which object is currently active then this keyword is used to refer current object.
Syntax:

```
        this.varName;
```

where, varName is the name of an instance variable.



For Example: Java Program to demonstrate this keyword.

```
public class Number
{
        int x;
        public Number (int x)
        {
                this.x = x;
        }
        public static void main(String[] args)
        {
                Number Obj = new Number (5);
                System.out.println("Value of x = " + Obj.x);
        }
}
```

Static Keyword:

The static keyword is used for memory management. It is applied with variables, methods, blocks and nested classes. Static keyword belongs to the class than an instance of the class.

The static is used with variable and method.

Static Variable:

Static variable is declared using static keyword. The static variable gets memory only once in the class area at the time of class loading and it is shared by all objects of its class.

Syntax:

```
        static datatype varName;
```

For Example: Java Program to demonstrate static variable.

```
class Number
{
        static int count=0;

        Demo()
        {
                count++;
                System.out.println(count);
        }
        public static void main(String args[])
        {
                Number N1=new Number ();
                Number N2=new Number ();
                Number N3=new Number ();
         }
}
```

Static Method:

Static method is declared using static keyword. It is called using class name. A static method can access static data member and can change the value of it.

For Example: Java Program to get the cube of a given number using the static method.

```
class Number
{
        static int cube(int x)
        {
                return x*x*x;
        }

        public static void main(String args[])
        {
                int result;
                result = Number.cube(5);
                System.out.println(result);
        }
}
```

finalize() method:

finalize() is the method of Object class**(java.lang.Object)**. This method is called just before an object is garbage collected. finalize() method overrides to dispose system resources, perform clean-up activities and minimize memory leaks. finalize() method releases system resources before the garbage collector runs for a specific object. JVM allows finalize() to be invoked only once per object.

Syntax:

```
        protected void finalize() throws Throwable
```

For Example: Java Program to demonstrate finalize() method.

```
public class Number
{
        public static void main(String[] args)
        {
                Number obj = new Number();
                System.out.println(obj.hashCode());
                obj = null;
                System.gc();            // calling garbage collector
                System.out.println("end of garbage collection");
        }
        protected void finalize()
        {
                System.out.println("finalize method called");
        }
}
```

Nested class:

A class defined within another class is called as nested class. A nested class is a member of its Outer class. Nested /Inner class can access all the members of outer class including private data members and methods. Outer class does not have access to the members of the nested/Inner class.

Syntax:

```
	class OuterClass
	{
		class NetsedClass
		{
			---
			---
		 }
	}
```

For Example: Java Program to demonstrate nested class concept.

```
class Outer
{
	int x =10;
	class Inner
	{
	      int y=20;
	      void display()
	      {
	        System.out.println("x=  " + x);
	        System.out.println("y=  " + y);
	       }
	}
	void show()
	{
		Inner I =new Inner();
		System.out.println("Outer Show ");
		I.display();
	}
}
class NestedEx
{
	public static void main(String args[])
	{
		Outer O =new Outer();
		O.show();
	}
}
```

Inner class:
A non-static class that is created inside a class but outside a method is called member inner class. Inner class can access the private data members & methods of outer class directly. It is used to group classes and interfaces in one place.

For Example: Java Program to demonstrate Inner class concept.
```
class Outer
{
        int x=30;
        class Inner
        {
                void disp()
                {
                    System.out.println("Inner class " +x);              //x is acessed directly
                }
        }
      void show()
      {
                Inner o=new Inner();
                o.disp();
      }
}
 class InnerDemo
 {
        public static void main(String args[])
        {
                Outer obj=new Outer();
                obj.show();
                Outer.Inner obj2=new Inner();
                obj2.disp();
        }
}
```

Anonymous Classes:
A class that has no name is known as anonymous inner class. Anonymous class is always inner class. It is defined inside another class. It is always child class of Some Parent class. They can extend a class or implements an interface. It can be used to override method of class or interface.
For Example: Java Program to demonstrate Anonymous class concept.
```
class Number
{
        public void display()
        {
                System.out.println("I am in Number class");
        }
}

class AnonymousDemo
{
        public void createC()
        {
                Number N = new Number ()
                {
                        public void display()
                        {
                                System.out.println("I am in anonymous class.");
                        }
                };
        N.display();
        }
}
class Demo
{
        public static void main(String[] args)
        {
                AnonymousDemo obj = new AnonymousDemo();
                obj.createC();
        }
}
```

Practice Set:

1. Write a Java program to for the implementation of reference variable.
2. Write a Java program to keep the count of object created of a class. Display the count each time when the object is created.
3. Write a Java program to convert integer primitive data. (use toString()).
4. Write a Java program to calculate sum of digits of a number using Recursion.
5. Write a Java program to for the implementation of this keyword.

Set A:

1. Write a Java program to calculate power of a number using recursion.
2. Write a Java program to display Fibonacci series using function.
3. Write a Java program to calculate area of Circle, Triangle & Rectangle.(Use Method Overloading)
4. Write a Java program to Copy data of one object to another Object.

5. Write a Java program to calculate factorial of a number using recursion.

Set B:

1. Define a class person(pid,pname,age,gender). Define Default and parameterised constructor. Overload the constructor. Accept the 5 person details and display it.(use this keyword).
2. Define a class product(pid,pname,price). Write a function to accept the product details, to display product details and to calculate total amount. (use array of Objects)
3. Define a class Student(rollno,name,per). Create n objects of the student class and Display it using toString().(Use parameterized constructor)
4. Define a class MyNumber having one private integer data member. Write a default constructor to initialize it to 0 and another constructor to initialize it to a value. Write methods isNegative, isPositive. Use command line argument to pass a value to the object and perform the above tests.

Set C:

1. Define class Student(rno, name, mark1, mark2). Define Result class(total, percentage) inside the student class. Accept the student details & display the mark sheet with rno, name, mark1, mark2, total, percentage. (Use inner class concept)
2. Write a java program to accept n employee names from user. Sort them in ascending order and Display them.(Use array of object nd Static keyword)
3. Write a java program to accept details of ‘n’ cricket players(pid, pname, totalRuns, InningsPlayed, NotOuttimes). Calculate the average of all the players. Display the details of player having maximum average.
4. Write a java program to accept details of ‘n’ books. And Display the quantity of given book.

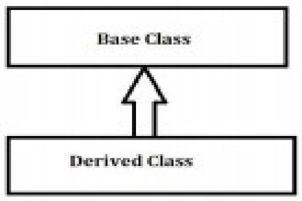
Assignment Evaluation:

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: WellDone [ ]

Signature of Instructor

Assignment No. 3: Inheritance, Package and Collection

The mechanism of deriving a new class from an old class is called as Inheritance. Old class is called as Superclass or Base class and the new derived class is called as Subclass or Derived class. It is also defined as the process where one class acquires the properties (methods and fields) of another class. The keyword **extends** is used to inherit the properties of a base/super class in derived/sub class.



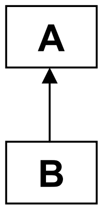
Syntax:

```
class Superclass
{
        // Superclass data variables
         // Superclass member functions
}
class Subclass extends Superclass
{
         // Subclass data variables
         // Subclass member functions
}
```

Types of inheritance:

A) Single Inheritance:

One subclass is derived from only one superclass is called as single inheritance.



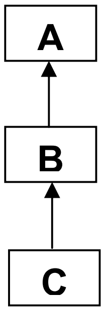
Syntax:

```
class ClassA
{
        ..... .....
}
class ClassB extends ClassA
{
        ..... .....
```

}

B) Multilevel Inheritance:

A subclass is derived from another derived class is called as Multilevel inheritance.

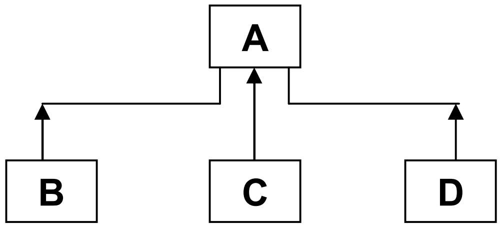


Syntax:

```
class ClassA
{         ..... ..... }
class ClassB extends ClassA
{ ..... ..... }
class ClassC extends ClassB
{ ..... ..... }
```

C) Hierarchical Inheritance:

More than one clasees are derived from a single superclass is called as Hierarchical Inheritance.



Syntax:

```
class ClassA
{         ..... .....  }
class ClassB extends ClassA
{ ..... ..... }
class ClassC extends ClassA
{ ..... ..... }
class ClassD extends ClassA
{ ..... ..... }
```

Inheritance in constructor:

In Java, constructor of base class with no argument gets automatically called in derived class constructor. The parameterized constructor of parent class is called explicitly by using super keyword. Base class constructor call must be the first line in derived class.

For example: Java Program to demonstrate Inheritance in Constructor.

```
class Superclass
{
        int x;
        Superclass (int m)
        {
           x = m;
        }
}
 class Subclass extends Superclass
{
        int y;
        Subclass (int m, int n)
        {
                super(m);
                y = n;
        }
         void Display()
        {
              System.out.println("x = " + x + " y = " + y);
        }
}
 public class Main
{
        public static void main(String[] args)
        {
                Subclass obj = new Subclass (10, 20);
                obj.Display();
         }
}
```

Method Overriding in Java:

The subclass contains the same method name and type as that of the Superclass is called as method overriding. It is used to provide specific implementation of a particular method of superclass. It helps to achieve runtime polymorphism.

For example: Java Program to demonstrate overriding.

```
class Superclass
{
	int a;
	Superclass(int x)
	{
		a=x;
	}
	void Display()
	{
		System.out.println("a= " +a);
	}
}
class Subclass extends Superclass
{
	int b;
	Subclass(int x, int y)
	{
		super(x);
		b = y;
	}
	void Display()
	{
		System.out.println("a= " + a + "b= " + b);
	}
}
class MethodOverDemo
{
	public static void main(String arg[])
	{
		Subclass obj = new Subclass(10, 20);
		obj.Display();
	}
}
```

Use of super

Super keyword is used by subclass to refer its immediate superclass.

Super keyword is used -

1. **To invoke the superclass variables:** To access the data members of super class when both superclass and subclass contains member with same name.
   Syntax: super.variableName;

   For example: Java Program to invoke the superclass variable.
   ```
   class SuperClass
   {
   	int x=100;
   }
   ```

```
class SubClass extends SuperClass
{
        int x=200;
        void display()
        {
                System.out.println("Super Class: x = " + super.x);
                System.out.println("Sub Class: x = " + x);
        }
}
class SuperVariable
{
        public static void main(String args[])
        {
                SubClass S2= new SubClass();
                S2.display();
        }
}
```

2. **To invoke the superclass methods:** To access the method of superclass when subclass has overridden that method.
   Syntax: super.methodName(arguments);

   For example: Java Program to invoke the superclass method.

```
class Superclass
{
        int x=100;
        void display()
        {
                System.out.println("Super Class: x = " + x);
        }
}
class Subclass extends Superclass
{
        int x=200;
        void display()
        {
                super.display();
                System.out.println("Sub Class: x = " + x);
        }
}
```

```
class SuperMethod
{
        public static void main(String args[])
        {
                Subclass S2= new Subclass();
                S2.display();
        }
}
```

3. **To invoke the superclass constructor:** To explicitly call the constructor of superclass.
   Syntax: super.constructorName(arguments);

   For example: Java Program to invoke the superclass constructor.

```
class Superclass
{
        int a;
        Superclass(int x)
        {
                a=x;
        }
        void Display()
        {
                System.out.println("a= " +a);
        }
}
class Subclass extends Superclass
{
        int b;
        Subclass(int x, int y)
        {
                super(x);
                b = y;
        }
        void Display()
        {
                System.out.println("a= " + a + "b= " + b);
        }
}
class SuperConstructor
{
        public static void main(String arg[])
        {
```

```
                        Subclass obj = new Subclass(10, 20);
                        obj.Display();
                }
        }
```

final keyword:

In java final keyword is used with:

1. Variable: If any variable is declared as final, the value of final variable cannot be changed throughout the program. It works as constant.
   For Example: final int SIZE=10;
2. Method: If any method is declared final, it canot be overridden in subclass.
   For Example: final void Display(){…}
3. Class: If any class is declared as final, it cannot be extended/inherited.
   For Example: final class Superclass{…}

   For example: Java Program to demonstrate final variable.

```
class FinalVarDemo
{
 public static void main(String args[])
 {
        final int x=10;
        System.out.println("Final x= " + x);
         x=20;          // Error: It will raise an error because final variable cannot be
                        reinitialized.
        System.out.println("Final x= " + x);
 }
}
```

   For example: Java Program to demonstrate final method.

```
class Superclass
{
        final void display()
        {
                System.out.println("Super Class: this cannot be overiden");
        }
}
class Subclass extends Superclass
{
        void display1()
        {
                System.out.println("Sub Class");
        }
}
```

```
class FinalMethodDemo
{
        public static void main(String args[])
        {
                Subclass obj=new Subclass();
                obj.display();
                obj.display1();
        }
}
```

For example: Java Program to demonstrate final class.

```
final class Superclass
{
        void display()
        {
                System.out.println("This is a final method.");
        }
}
class Subclass extends Superclass
{
        void display()
        {
                System.out.println("The final method is overridden.");
        }
}
class FinalClass
{
        public static void main(String args[])
        {
                Subclass obj = new Subclass();
                obj.display();
        }
}
```

Output will be:

```
C:\Program Files\Java\jdk1.8.0_221\bin>javac FinalClass.java
FinalClass.java:9: error: cannot inherit from final Superclass
class Subclass extends Superclass
                       ^
1 error
```

Abstract class:

An abstract class is a class in which one or more methods are declared, but not defined i.e it contains abstract methods. Abstract class cannot be instantiated. An abstract class can contain variables, meth

For example: Java Program to demonstrate abstract class.

```
abstract class Superclass
{
        abstract void display();        //abstract method
}

class Subclass extends Superclass
{
        Subclass ()
        {
                System.out.println("Sub Class Constructor ");
        }
        void display()
        {
                System.out.println("Subclass ");
        }
}
class AbstractClassDemo
{
        public static void main(String args[])
        {
                Subclass obj=new Subclass ();
                obj.display();
        }
}
```

Interface:

An interface looks like a class but it is not a class. Is is a collection of only abstract methods and static final variables. Interface cannot be instantiated. An interface does not contain constructor. Interface is used to achieve Multiple Inheritance & abstraction.

Syntax to define an interface in java:

```
interface InterfaceName
{
        ...
        //members declaration;
        //method declaration;
        ...
}
```

Syntax to implement inteface:

```
class className implements InterfaceName
{
        ...
        body-of-the-class
        ...
}
```

"**implements**" keyword is used to inherit interface in interface/class.

For example: Java Program to demonstrate interface.

```
interface I1
{
        void display();
}

class Demo implements I1
{
        public void display()
        {
                System.out.println("Intreface Demo ....");
        }
}

class InterfaceDemo
{
        public static void main(String args[])
        {
                Demo D=new Demo();
                D.display();
        }
}
```

Interface inheritance

Interface inheritance is used when a class wants to implement more than one interface. List of interfaces are seprated using comma implemented by the class. Multiple inheritance is possible using interfaces but not by using class.

Syntax to implement multiple interfaces:

```
class className implements InterfaceName1 extends InterfaceName2, InterfaceName3 ...
{
          ...
          body-of-the-class
          ...
}
```

For example: Java Program to demonstrate Interface inheritance.

```
interface Interface1
{
        public void Display1();
}

interface Interface2
{
        public void Display2();
}

interface Interface3 extends Interface1,Interface2
{
  public void Display3();
}

public class InfInheritanceDemo implements Interface3
{
         public void Display1()
        {
                System.out.println("Display1");
        }
        public void Display2()
        {
                System.out.println("Display2");
        }
        public void Display3()
        {
                System.out.println("Display3");
        }
        public static void main(String args[])
        {
                Interface3 obj = new InfInheritanceDemo();
                obj.Display1();
                obj.Display2();
                obj.Display3();
        }
}
```

Dynamic method dispatch:

Dynamic method dispatch is the mechanism in which a call to an overridden method is resolved at run time instead of compile time. This is an important concept because of how Java implements run-time polymorphism.

Java uses the principle of 'a superclass reference variable can refer to a subclass object' to resolve calls to overridden methods at run time. When a superclass reference is used to call an overridden method, Java determines which version of the method to execute based on the type of the object being referred to at the time call.

For Example: Java Program to demonstrate Dynamic Method Dispatch.

```
class A
{
        void Display1()
        {
                System.out.println("I am in A's Display method");
         }
}

class B extends A
{
        void Display1()         // overriding Display1()
        {
                System.out.println("I am in B's Display method");
        }
}

class C extends A
{
        void Display1()                 // overriding Display1()
        {
                System.out.println("I am in C's Display method");
        }
}

class DispatchDemo
{
        public static void main(String args[])
        {
                A a = new A();
                B b = new B();
                C c = new C();

                A ref;                  // obtain a reference of type A
                ref = a;                // ref refers to an A object
                ref.Display1();         // calling A's version of Display1()

                ref = b;                // now ref refers to a B object
                ref.Display1();         // calling B's version of Display1()

                 ref = c;               // now ref refers to a C object
                ref.Display1();         // calling C's version of Display1()
   }
}
```

Access Control:
Access modifiers define the scope of the class and its members (data and methods).

	Private	**No Modifier**	**Protected**	**Public**
Same Class	Yes	Yes	Yes	Yes
Same package subclass	No	Yes	Yes	Yes
Same package non-subclass	No	Yes	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

Packages:
A java package is a group of similar types of classes, interfaces and sub-packages.In Java, Package is categorized in two forms:

a. User-defined package
b. Predefined package

User-defined package: The package created by user is called user-defined package. To create user defined package, Java uses a file system directory to store them just like folders on your computer:

```
Example
 └── root/bin
       └── mypackage
             └── MyPackageClass.java
```

First we create a directory mypackage (name should be same as the name of the package).
Then create the MyPackageClass.java inside the directory with the first statement being the package names. To create a package, package keyword is used.
Syntax:

```
package packagename;
```

For example:

```
package mypackage;
class MyPackageClass
{
   public static void main(String[] args)
     {
        System.out.println("This is my package!");
     }
 }
```

Keyword **import** is used to use a class or a package from the library.
Syntax:

```
import package.name.Class;          // Import a single class
import package.name.*;              // Import the whole package
```

For Example: Java Program to demonstrate package concept.

```
package abc;
public class A
{
	public void disp()
	{
		System.out.println("A Class");
	}
}

package abc;
public class B
{
	public void disp()
	{
		System.out.println("B Class");
	}
}

import abc.*;
class PackageDemo
{
	public static void main(String args[])
	{
		A obj1 =new A();
		obj1.disp();
		B obj2 =new B();
		obj2.disp();
	}
}
```

Predefined packages: Predefined packages in java are developed by Sun Microsystem. It is also called as built-in packages. It contains large number of predefined classes, interfaces, and methods that are used by the programmer to perform any task in the program.

Key points about predefined Packages:

1. Java predefined supports a group of packages that contains a group of classes and interfaces. These classes and interfaces consist of a group of methods.
   For example, Java language contains a package called java.lang which contains string class, StringBuffer class, StringBuilder class, all wrapper classes, runnable interface, etc. String class contains a number of methods such as length(), toUpperCase(), toLowerCase() etc.
2. Java contains 14 predefined packages which are main packages. These 14 predefined packages contain nearly 150 sub-packages that consist of a minimum of 7 thousand classes. These 7 thousand classes contain approx 7 lakhs methods.

Collection:

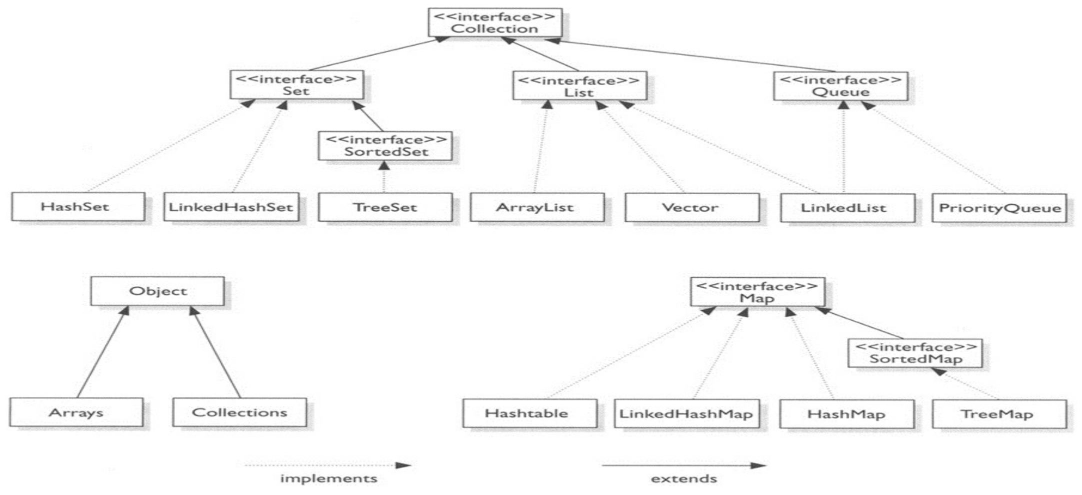
A collection is an object that groups multiple elements into a single unit i.e. single unit of objects. Collections are used to store, retrieve, manipulate, and communicate aggregate data.

Collection Framework:

The collection framework is the collection of classes and interfaces.

A Java collection framework provides architecture to store and manipulate a group of objects. A Java collection framework includes the following:

1. Interfaces
2. Classes
3. Algorithm



The List of interfaces:

- Collection
- Set
- List
- Map
- SortedMap
- Enumeration

The List of classes:

- ArrayList
- LinkedList
- HashSet
- TreeSet

Algorithm: Algorithm refers to the methods which are used to perform operations such as searching and sorting, on objects that implement collection interfaces.

Interfaces: Collection, List, Set

Collection Interface:

The group of data or the set of data which is binding in a unit in object form that unit is called Collection. Collection is an interface present at topmost position in collection framework.

For the implementation of collection interface any class can be used which is at last level as leaf node in collection framework.

For Example:

```
Collection c=new HashSet()
Collection c=new ArrayList()
Collection c=new Vector()
```

Sr. No.	Methods with Description
1.	**boolean add(Object obj)** Adds obj to the invoking collection. Returns true if obj is added to the collection. Returns false if obj is already a member of the collection, or if the collection does not allow duplicates.
2.	**boolean addAll(Collection c)** Adds all the elements of c to the invoking collection. Returns true if the operation succeeds (i.e., the elements were added). Otherwise, returns false.
3.	**void clear( )** Removes all elements from the invoking collection.
4.	**boolean contains(Object obj)** Returns true if obj is an element of the invoking collection. Otherwise, returns false.
5.	**boolean containsAll(Collection c)** Returns true if the invoking collection contains all elements of **c**. Otherwise, returns false.
6.	**boolean equals(Object obj)** Returns true if the invoking collection and obj are equal. Otherwise, returns false.
7.	**int hashCode( )** Returns the hash code for the invoking collection.
8.	**boolean isEmpty( )** Returns true if the invoking collection is empty. Otherwise, returns false.
9.	**Iterator iterator( )** Returns an iterator for the invoking collection.
10.	**boolean remove(Object obj)**

	Removes one instance of obj from the invoking collection. Returns true if the element was removed. Otherwise, returns false.
11.	**boolean removeAll(Collection c)** Removes all elements of c from the invoking collection. Returns true if the collection changed (i.e., elements were removed). Otherwise, returns false.
12.	**boolean retainAll(Collection c)** Removes all elements from the invoking collection except those in c. Returns true if the collection changed (i.e., elements were removed). Otherwise, returns false.
13.	**int size( )** Returns the number of elements held in the invoking collection.
14.	**Object[ ] toArray( )** Returns an array that contains all the elements stored in the invoking collection. The array elements are copies of the collection elements.
15.	**Object[ ] toArray(Object array[ ])** Returns an array containing only those collection elements whose type matches that of array.

For Example: Java Program to accept n city names from user and display it.

```
import java.util.*;
public class CollDemo
{
        public static void main(String args[])
        {
                int i,n;
                String cn;
                Collection c=new ArrayList();
                Scanner ob=new Scanner(System.in);
                System.out.println ("How Many");
                n=ob.nextInt ();
                for(i=1;i<=n;i++)
                {
                        System.out.println ("City Name");
                        cn=ob.next();
                        c.add(cn);
                }
                System.out.println ("Entered Data is " + c);
        }
}
```

List interface:

The List interface extends Collection and declares the behavior of a collection that stores a sequence of elements. Elements can be inserted or accessed by their position in the list, using a zero-based index. A list may contain duplicate elements.

Sr. No.	Methods with Description
1.	**void add(int index, Object obj)** Inserts obj into the invoking list at the index passed in index. Any pre-existing elements at or beyond the point of insertion are shifted up. Thus, no elements are overwritten.
2.	**boolean addAll(int index, Collection c)** Inserts all elements of c into the invoking list at the index passed in index. Any pre-existing elements at or beyond the point of insertion are shifted up. Thus, no elements are overwritten. Returns true if the invoking list changes and returns false otherwise.
3.	**Object get(int index)** Returns the object stored at the specified index within the invoking collection.
4.	**int indexOf(Object obj)** Returns the index of the first instance of obj in the invoking list. If obj is not an element of the list, .1 is returned.
5.	**int lastIndexOf(Object obj)** Returns the index of the last instance of obj in the invoking list. If obj is not an element of the list, .1 is returned.
6.	**ListIterator listIterator( )** Returns an iterator to the start of the invoking list.
7.	**ListIterator listIterator(int index)** Returns an iterator to the invoking list that begins at the specified index.
8.	**Object remove(int index)** Removes the element at position index from the invoking list and returns the deleted element. The resulting list is compacted. That is, the indexes of subsequent elements are decremented by one
9.	**Object set(int index, Object obj)** Assigns obj to the location specified by index within the invoking list.
10.	**List subList(int start, int end)** Returns a list that includes elements from start to end. in the invoking list. Elements in the returned list are also referenced by the invoking object.

List interface has been implemented in various classes like ArrayList or LinkedList, etc.

For Example: Java Program to demonstrate List Interface.

```
import java.util.*;
class  ListDemo
```

```
{
  public static void main(String args[])
  {
          List C=new ArrayList();

          C.add("aa");
          C.add("bb");
          C.add("cc");
          C.add(2,"ee");

          System.out.println(C);
          System.out.println("get  = "+ C.get(2));
          System.out.println("IndexOf   = "+ C.indexOf("ee"));
          System.out.println("Last IndexOf  = " + C.lastIndexOf("ee"));
          C.remove(2);
          System.out.println("After Remove element at 4" +C);

          C.set(1,"xx");
          System.out.println("After replace " +C);
          System.out.println("Size " + C.size());
          System.out.println(C.subList(2,3));
  }
}
```

Set interface:

A Set is a Collection that cannot contain duplicate elements. It models the mathematical set abstraction. The Set interface contains only methods inherited from Collection and adds the restriction that duplicate elements are prohibited.

Sr. No.	Methods with Description
1.	**add( )** Adds an object to the collection
2.	**clear( )** Removes all objects from the collection
3.	**contains( )** Returns true if a specified object is an element within the collection
4.	**isEmpty( )** Returns true if the collection has no elements
5.	**iterator( )** Returns an Iterator object for the collection which may be used to retrieve an object
6.	**remove( )** Removes a specified object from the collection
7.	**size( )** Returns the number of elements in the collection

Set have its implementation in various classes like HashSet, TreeSet, LinkedHashSet.

For Example: Java Program to demonstrate Set Interface.

```
import java.util.*;
class SetDemo
{
        public static void main(String[] args)
        {
                 Set C1 = new HashSet ();
                C1.add(2);
                 C1.add(3);
                 System.out.println("Set1: " + C1);

                 Set C2 = new HashSet ();
                 C2.add(1);
                C2.add(2);
                System.out.println("Set2: " + C2);

                 C2.addAll(C1);
                System.out.println("Union is: " + C2);
   }
}
```

Navigation: Enumeration, Iterator, ListIterator

Enumeration:

The enumeration() method of java.util.Collections class is used to return an enumeration over the specified collection. This method takes the collection c as a parameter for which an enumeration is to be returned. This method returns an enumeration over the specified collection.

Syntax:

```
        public static  Enumeration enumeration(Collection C)
```

For Example: Java program to demonstrate enumeration() method.

```
import java.util.*;
public class EnumDemo
{
   public static void main(String[] argv) throws Exception
   {
      try
         {
                List<Integer> arrlist = new ArrayList<Integer>();
                arrlist.add(20);
                 arrlist.add(30);
                arrlist.add(40);
                 System.out.println("List: " + arrlist);
                Enumeration<Integer> e = Collections.enumeration(arrlist);
                System.out.println("\nEnumeration over list: ");
                while (e.hasMoreElements())
                        System.out.println("Value is: " + e.nextElement());
            }

            catch (IllegalArgumentException e)
            {
                System.out.println("Exception thrown : " + e);
            }
            catch (NoSuchElementException e)
            {
                System.out.println("Exception thrown : " + e);
            }
        }
}
```

Iterator interface:
'Iterator' is an interface which belongs to collection framework. It allows us to traverse the collection, access the data element and remove the data elements of the collection. java.util package has public interface Iterator and contains following methods:

Sr. No.	Methods with Description
1.	**boolean hasNext()** It returns true if Iterator has more element to iterate.
2.	**Object next()** It returns the next element in the collection until the hasNext () method return true. This method throws NoSuchElementException' if there is no next element.
3.	**void remove()** It removes the current element in the collection. This method throws 'IllegalStateException' if this function is called before next( ) is invoked.

For Example: Java program to accept n employee names through command line and display them.

```
import java.util.*;
class IteratorDemo
{
    public static void main(String args[])
    {
        int i,n;
        String str;
        Integer obj;
        Collection c=new ArrayList ();
        n=args.length;
        for(i=0;i<n;i++)
        {
                c.add(args[i]);
        }
        Iterator it=c.iterator ();
        while(it.hasNext())
        {
                str = (String)it.next();
                System.out.println (str);
        }
    }
  }
```

ListIterator:

Iterator interface is used to traverse List or Set interface in forward direction only, ListIterator interface traverse List or Set interface in both the directions (backward and forward). ListIterator interface is inherited from Iterator interface.

Sr. No.	Methods with Description
1.	**void add(E e):** Inserts the specified element into the list.
2.	**boolean hasNext():** Returns true if this ListItrator has more elements when traversing the list in the forward direction. It returns the next element in the collection until the hasNext () method return true. This method throws NoSuchElementException' if there is no next element.
3.	**boolean hasPrevious():** Returns true if this ListIterator has more elements when traversing the list in the reverse direction.
4.	**E next():** Returns the next element in the list and advances the cursor position.
5.	**int nextIndex():** Returns the index of the element that would be returned by a subsequent call to next().

6.	**E previous():**
	Returns the previous element in the list and moves the cursor position backwards.
	int previousIndex():
	Returns the index of the element that would be returned by a subsequent call to previous().

7.	**void remove():** Removes from the list the last element that was returned by next() or previous().
8.	**void set(E e):** Replaces the last element returned by next() or previous() with the specified element.
9.	**E next():** Returns the next element in the list and advances the cursor position.

For Example: Java program to accept N students names from user and display them in reverse order.

```
import java.util.*;
class ListDemo
{
        public static void main(String args[])
        {
                        int i,n;
                        String sn;
                        Scanner ob=new Scanner (System.in);
                        List l=new LinkedList();
                        System.out.println ("How Many");
                        n=ob.nextInt ();
                        for(i=1;i<=n;i++)
                        {
                                System.out.println ("Student Name");
                                sn=ob.next();
                                l.add(sn);
                        }
                        ListIterator lst=l.listIterator ();
                        while(lst.hasPrevious())
                        {
                                sn=(String)lst.previous();
                        }
        }
}
```

Classes: LinkedList, ArrayList, Vector, HashSet
LinkedList:
The LinkedList class extends AbstractSequentialList and implements the List interface. It provides a linked-list data structure.
The LinkedList class supports two constructors.

- LinkedList( ) : Builds an empty linked list:

- LinkedList(Collection c) : Builds a linked list that is initialized with the elements of the collection c.

Sr. No.	Methods with Description
1.	**void add(int index, Object element)** Inserts the specified element at the specified position index in this list. Throws IndexOutOfBoundsException if the specified index is out of range (index < 0 || index > size()).
2.	**boolean add(Object o)** Appends the specified element to the end of this list.
3.	**boolean addAll(Collection c)** Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator. Throws NullPointerException if the specified collection is null.
4.	**boolean addAll(int index, Collection c)** Inserts all of the elements in the specified collection into this list, starting at the specified position. Throws NullPointerException if the specified collection is null.
5.	**void addFirst(Object o)** Inserts the given element at the beginning of this list.
6.	**void addLast(Object o)** Appends the given element to the end of this list.
7.	**void clear()** Removes all of the elements from this list.
8.	**Object clone()** Returns a shallow copy of this LinkedList.
9.	**boolean contains(Object o)** Returns true if this list contains the specified element. More formally, returns true if and only if this list contains at least one element e such that (o==null ? e==null : o.equals(e)).
10.	**Object get(int index)** Returns the element at the specified position in this list. Throws IndexOutOfBoundsException if the specified index is out of range (index < 0 || index >= size()).
11.	**Object getFirst()** Returns the first element in this list. Throws NoSuchElementException if this list is empty.
12.	**Object getLast()** Returns the last element in this list. Throws NoSuchElementException if this list is empty.
13.	**int indexOf(Object o)** Returns the index in this list of the first occurrence of the specified element,

	or -1 if the List does not contain this element.
14.	**int lastIndexOf(Object o)** Returns the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element.
15.	**ListIterator listIterator(int index)** Returns a list-iterator of the elements in this list (in proper sequence), starting at the specified position in the list. Throws IndexOutOfBoundsException if the specified index is out of range (index < 0 || index >= size()).
16.	**Object remove(int index)** Removes the element at the specified position in this list. Throws NoSuchElementException if this list is empty.
17.	**boolean remove(Object o)** Removes the first occurrence of the specified element in this list. Throws NoSuchElementException if this list is empty. Throws IndexOutOfBoundsException if the specified index is out of range (index < 0 || index >= size()).
18.	**Object removeFirst()** Removes and returns the first element from this list. Throws NoSuchElementException if this list is empty.
19.	**Object removeLast()** Removes and returns the last element from this list. Throws NoSuchElementException if this list is empty.
20.	**Object set(int index, Object element)** Replaces the element at the specified position in this list with the specified element. Throws IndexOutOfBoundsException if the specified index is out of range (index < 0 || index >= size()).
21.	**int size()** Returns the number of elements in this list.
22.	**Object[] toArray()** Returns an array containing all of the elements in this list in the correct order. Throws NullPointerException if the specified array is null.
23.	**Object[] toArray(Object[] a)** Returns an array containing all of the elements in this list in the correct order; the runtime type of the returned array is that of the specified array.

For Example: Java program to demonstrate LinkedList interface.

```
import java.util.*;
class LinkedListDemo
{
        public static void main(String args[])
        {
                int i,n;
                String sn;
                Scanner ob=new Scanner (System.in);
                List l=new LinkedList ();
                System.out.println ("How Many");
                n=ob.nextInt ();
                for(i=1;i<=n;i++)
                {
                        System.out.println ("Enter Data");
                        sn=ob.next();
                        l.add(sn);
                }
                System.out.println ("Entered Data " + l);
        }
}
```

ArrayList:

The ArrayList class extends AbstractList and implements the List interface. ArrayList supports dynamic arrays that can grow as needed. Standard Java arrays are of a fixed length. After arrays are created, they cannot grow or shrink, which means that user must know in advance how many elements an array will hold. Array lists are created with an initial size. When this size is exceeded, the collection is automatically enlarged. When objects are removed, the array may be shrunk.

The ArrayList class supports three constructors.

- ArrayList( ) : Builds an empty array list.
- ArrayList(Collection c) : Builds an array list that is initialized with the elements of the collection c.
- ArrayList(int capacity) : Builds an array list that has the specified initial capacity. The capacity is the size of the underlying array that is used to store the elements. The capacity grows automatically as elements are added to an array list.

Sr. No.	Methods with Description
1.	**void add(int index, Object element)** Inserts the specified element at the specified position index in this list. Throws IndexOutOfBoundsException if the specified index is out of range (index < 0 || index > size()).
2.	**boolean add(Object o)** Appends the specified element to the end of this list.
3.	**boolean addAll(Collection c)** Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator. Throws NullPointerException if the specified collection is null.

4.	**boolean addAll(int index, Collection c)** Inserts all of the elements in the specified collection into this list, starting at the specified position. Throws NullPointerException if the specified collection is null.
5.	**void clear()** Removes all of the elements from this list.
6.	**Object clone()** Returns a shallow copy of this ArrayList.
7.	**boolean contains(Object o)** Returns true if this list contains the specified element. More formally, returns true if and only if this list contains at least one element e such that (o==null ? e==null : o.equals(e)).
8.	**void ensureCapacity(int minCapacity)** Increases the capacity of this ArrayList instance, if necessary, to ensure that it can hold at least the number of elements specified by the minimum capacity argument.
9.	**Object get(int index)** Returns the element at the specified position in this list. Throws IndexOutOfBoundsException if the specified index is out of range (index < 0 || index >= size()).
10.	**int indexOf(Object o)** Returns the index in this list of the first occurrence of the specified element, or -1 if the List does not contain this element.
11.	**int lastIndexOf(Object o)** Returns the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element.
12.	**Object remove(int index)** Removes the element at the specified position in this list. Throws IndexOutOfBoundsException if index out of range (index < 0 || index >= size()).
13.	**protected void removeRange(int fromIndex, int toIndex)** Removes from this List all of the elements whose index is between fromIndex, inclusive and toIndex, exclusive.
14.	**Object set(int index, Object element)** Replaces the element at the specified position in this list with the specified element. Throws IndexOutOfBoundsException if the specified index is out of range (index < 0 || index >= size()).
15.	**int size()** Returns the number of elements in this list.
16.	**Object[] toArray()** Returns an array containing all of the elements in this list in the correct

	order. Throws NullPointerException if the specified array is null.
17.	**Object[] toArray(Object[] a)** Returns an array containing all of the elements in this list in the correct order; the runtime type of the returned array is that of the specified array.
18.	**void trimToSize()** Trims the capacity of this ArrayList instance to be the list's current size.

For Example: Java program to demonstrate ArrayList interface using command Line argument.

```
import java.util.*;
class ArrayListDemo
{
        public static void main (String args[])
        {
                int i,n;
                n=args.length;
                ArrayList l=new ArrayList ();
                for (i=0; i<n; i++)
                {
                        l.add(args[i]);
                }
                System.out.println (l);
        }
}
```

Vector:

Vector is a class belongs to package java.util used to store the data in object form.The generic dynamic array is called Vector. Vector implements a dynamic array. It is similar to ArrayList, but with two differences:

- Vector is synchronized.
- Vector contains many legacy methods that are not part of the collections framework.

Vector proves to be very useful if you don't know the size of the array in advance or you just need one that can change size over the life time of a program.

Sr. No.	Constructor
1.	**Vector( )** This constructor creates a default vector, which has an initial size of 10.
2.	**Vector(int size)** This constructor accepts an argument that equals to the required size, and creates a vector whose initial capacity is specified by size.
3.	**Vector(int size, int incr)** This constructor creates a vector whose initial capacity is specified by size and whose increment is specified by incr. The increment specifies the number of elements to allocate each time that a vector is resized upward.

4.	**Vector(Collection c)** This constructor creates a vector that contains the elements of collection c.

Methods:

Sr. No.	Methods with Description
1.	**void add(int index, Object element)** Inserts the specified element at the specified position in this Vector.
2.	**boolean add(Object o)** Appends the specified element to the end of this Vector.
3.	**boolean addAll(Collection c)** Appends all of the elements in the specified Collection to the end of this Vector, in the order that they are returned by the specified Collection's Iterator.
4.	**boolean addAll(int index, Collection c)** Inserts all of the elements in in the specified Collection into this Vector at the specified position.
5.	**void addElement(Object obj)** Adds the specified component to the end of this vector, increasing its size by one.
6.	**int capacity()** Returns the current capacity of this vector.
7.	**void clear()** Removes all of the elements from this vector.
8.	**Object clone()** Returns a clone of this vector.
9.	**boolean contains(Object elem)** Tests if the specified object is a component in this vector.
10.	**boolean containsAll(Collection c)** Returns true if this vector contains all of the elements in the specified Collection.
11.	**void copyInto(Object[] anArray)** Copies the components of this vector into the specified array.
12.	**Object elementAt(int index)** Returns the component at the specified index.
13.	**Enumeration elements()** Returns an enumeration of the components of this vector.

For Example: Java program to display all the files having extension .java.

```
import java.util.*
```

```
class VectDemo
{
        public static void main(String args[])
        {
                int i,n;
                n=args.length;
                Vector v=new Vector ();
                String str[]=new String[n];
                for(i=0;i<n;i++)
                {
                        v.addElement (args[i]);
                }

                v.copyInto (str);
                for(i=0;i<n;i++)
                {
                    if(str[i].endsWith(".java")==0)
                            {
                                System.out.println(str[i]);
                            }
                        }
          }
   }
```

Map Classes:
Following are the two main classes of the Map Class:

A) HashMap:
A HashMap contains values based on the key. It implements the Map interface and extends AbstractMap class. It contains only unique elements. It may have one null key and multiple null values. It maintains no order.

For Example: Java program to demonstrate HashMap.

```
import java.util.*;
class TestCollection
{
    public static void main(String args[])
    {
            HashMap<Integer,String> hm=new HashMap<Integer,String>();
            hm.put(100,"Amit");
            hm.put(101,"Vijay");
            hm.put(102,"Rahul");
            for(Map.Entry m:hm.entrySet())
```

```
                {
                        System.out.println(m.getKey()+" "+m.getValue());
                 }
        }
    }
```

B) TreeMap:

A TreeMap contains values based on the key. It implements the NavigableMap interface and extends AbstractMap class. It contains only unique elements. It cannot have null key but can have multiple null values. It is same as HashMap instead maintains ascending order.

For Example: Java program to demonstrate TreeMap.

```
import java.util.*;
class TestCollection
{
  public static void main(String args[])
  {
            TreeMap<Integer,String> hm=new TreeMap<Integer,String>();
            hm.put(100,"Amit");
            hm.put(102,"Ravi");
            hm.put(101,"Vijay");
            hm.put(103,"Rahul");
            for(Map.Entry m:hm.entrySet())
            {
                    System.out.println(m.getKey()+" "+m.getValue());
            }
    }
}
```

For Example: Java program to demonstrate hash table.

```
import java.util.*;

class HashtableSalary
{
  public static void main(String args[])
  {
        Hashtable H=new Hashtable();

        H.put("Amit", "10000");
        H.put("Sumit", "20000");
        H.put("Akshay","30000");

        Enumeration E;
        E=H.keys();
        while(E.hasMoreElements())
        {
                String k=(String)E.nextElement();
                System.out.println( k + " :  " + H.get(k) );
        }

        Scanner S=new Scanner(System.in);
        System.out.println("enter Employee name to search : ");
        String Ename=S.next();
        E=H.keys();
        while(E.hasMoreElements())
        {
                String s=(String)E.nextElement();

                if(Ename.equals(s))
                {
                        System.out.println(s+" value="+H.get(s));
                        break;
                }
        }

  }
}
```

Practice Set:

1. Create abstract class Shape with abstract method area().Write a Java program to calculate are of Rectangle and Triangle.(Inherit Shape class in classes Rectangle and Triangle)
2. Create a class Teacher(Tid, Tname, Designation, Salary, Subject). Write a Java program to accept the details of 'n' teachers and display the details of teacher who is teaching Java Subject.(Use array of Object)
3. Create a class Doctor(Dno, Dname, Qualification, Specialization). Write a Java program to accept the details of 'n' doctors and display the details of doctor in ascending order by doctor name.
4. Write a Java program to accept 'n' employee names through command line, Store them in vector. Display the name of employees starting with character 'S'.
5. Create a package Mathematics with two classes Maximum and Power. Write a java program to accept two numbers from user and perform the following operations on it:
    a. Find Maximum of two numbers.
    b. Calculate the power(X^Y);

Set A:

1. Write a java program to calculate area of Cylinder and Circle.(Use super keyword)

2. Define an Interface Shape with abstract method area(). Write a java program to calculate an area of Circle and Sphere.(use final keyword)
3. Define an Interface “Integer” with a abstract method check().Write a Java program to check whether a given number is Positive or Negative.
4. Define a class Student with attributes rollno and name. Define default and parameterized constructor. Override the toString() method. Keep the count of Objects created. Create objects using parameterized constructor and Display the object count after each object is created.
5. Write a java program to accept ‘n’ integers from the user & store them in an ArrayList collection. Display the elements of ArrayList collection in reverse order.

Set B:

1. Create an abstract class Shape with methods calc_area() & calc_volume(). Derive two classes Sphere(radius)& Cone(radius, height) from it. Calculate area and volume of both. (Use Method Overriding)
2. Define a class Employee having private members-id, name, department, salary. Define default & parameterized constructors. Create a subclass called Manager with private member bonus. Define methods accept & display in both the classes. Create n objects of the manager class & display the details of the manager having the maximum total salary(salary+bonus).
3. Construct a Linked List containg name: CPP, Java, Python and PHP. Then extend your program to do the following:
    i. Display the contents of the List using an iterator
    ii. Display the contents of the List in reverse order using a ListIterator.
4. Create a hashtable containing employee name & salary. Display the details of the hashtable. Also search for a specific Employee and display salary of that employee.
5. Write a package game which will have 2 classes Indoor & Outdoor. Use a function display() to generate the list of player for the specific game. Use default & parameterized constructor.

Set C:

1. Create a hashtable containing city name & STD code. Display the details of the hashtable. Also search for a specific city and display STD code of that city.
2. Construct a Linked List containing name: red, blue, yellow and orange. Then extend your program to do the following:
   Display the contents of the List using an iterator
   Display the contents of the List in reverse order using a ListIterator.
   Create another list containing pink & green. Insert the elements of this list between blue & yellow.
3. Define an abstract class Staff with members name &address. Define two sub classes FullTimeStaff(Departmet, Salary) and PartTimeStaff(numberOfHours, ratePerHour). Define appropriate constructors. Create n objects which could be of either FullTimeStaff or PartTimeStaff class by asking the user’s choice. Display details of FulltimeStaff and PartTimeStaff.

4. Derive a class Square from class Rectangle. Create one more class Circle. Create an interface with only one method called area(). Implement this interface in all classes. Include appropriate data members and constructors in all classes. Write a java program to accept details of Square, Circle & Rectangle and display the area.

5. Create a package named Series having three different classes to print series:
   i. Fibonacci series
   ii. Cube of numbers
   iii. Square of numbers

   Write a java program to generate 'n' terms of the above series.

Assignment Evaluation:

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: WellDone [ ]

Signature of Instructor

Assignment No. 4 : File and Exception Handling

Exception:

Exceptions are generated when an error condition occur during the execution of a method. It is possible that a statement might throw more than one kind of exception. Exception can be generated by Java-runtime system or they can be manually generated by code. Error-Handling becomes a necessary while developing an application to account for exceptional situations that may occur during the program execution, such as

- **a)** Run out of memory
- **b)** Resource allocation Error
- **c)** Inability to find a file
- **d)** Problems in Network connectivity

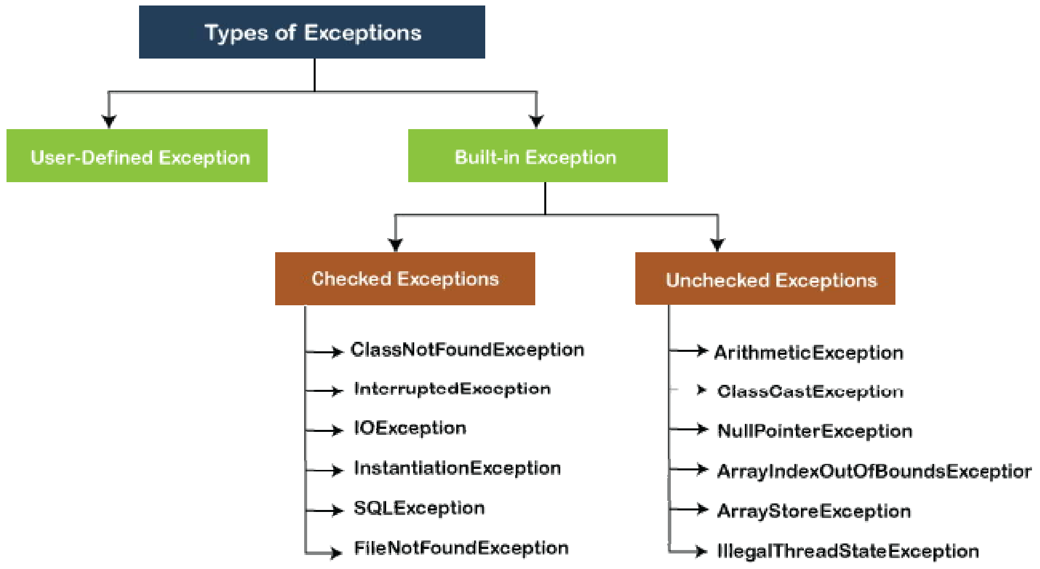
Abnormal condition of a program that terminates its execution is called Exception.

Exception Types:

In Java, exception is an event that occurs during the execution of a program and disrupts the normal flow of the program's instructions. Bugs or errors that we don't want and restrict our program's normal execution of code are referred to as exceptions.

Exceptions can be categorized into two ways:

1. **Built-in Exceptions**
   - Checked Exception
   - Unchecked Exception
2. **User-Defined Exceptions**



Built-in Exception

Exceptions that are already available in Java libraries are referred to as built-in exception. These exceptions are able to define the error situation so that we can understand the reason of getting this error. It can be categorized into two broad categories, i.e., checked exceptions and unchecked exception.

Checked Exception

Checked exceptions are called compile-time exceptions because these exceptions are checked at compile-time by the compiler. The compiler ensures whether the programmer handles the exception or not. The programmer should have to handle the exception; otherwise, the system has shown a compilation error.

Unchecked Exception

The unchecked exceptions are just opposite to the checked exceptions. The compiler will not check these exceptions at compile time. In simple words, if a program throws an unchecked exception, and even if we didn't handle or declare it, the program would not give a compilation error. Usually, it occurs when the user provides bad data during the interaction with the program.

Exception Handling:

There are two ways to handle an exception:

a) One can try the "risky" code, catch the exception, and do something about it, after which the transmission of the exception come to an end

b) One can mark that this method throws that exception, in which case the Java runtime engine will throw the exception back to the method.

So, if one uses a method in code that is marked as throwing a particular exception, the compiler will not allow that code unless user handle the exception. If the exception occurs in a try block, the JVM looks to the catch block(s) that follow to see if any of them equivalent the exception type. The first one that matches will be executed. If none match, then this methods ends, and execution jumps to the method that called this one, at the point the call was made.

Using try…catch:

If a method is going to resolve potential exception internally, the line of code that could generate the exception is placed inside a try block there may be other code inside the try block, before and/or after the risky line(s) - any code that depends upon the risky code's success should be in the try block, since it will automatically be skipped if the exception occurs.

Syntax of try:

```
try
{
    code
    risky/unsafe code
    code that depends on the risky code succeeding
}
```

There is usually at least one catch block immediately after the try block a catch block must specify what type of exception it will catch.

Syntax of catch:

```
catch (ExceptionClassName exceptionObjectName)
{
    code using methods from exceptionObjectName
}
```

Example 1: Java Program to count number of valid and invalid Integers.

```
class ExcDemo
{
        public static void main (String args[])
        {
                int i, n, vcnt=0, icnt=0,m;
                n=args.length;
                for(i=0;i<n;i++)
                {
                        try
                        {
                                m=Integer.parseInt (args[i]);
                                    vcnt++;
                           }
                        catch(Exception obj)
                        {
                                icnt++;
                           }
                  }
                  System.out.println ("Valid Integers are " + vcnt);
                  System.out.println ("Invalid Integers are " + icnt);
        }
 }
```

Example 2: Java program to accept a number from user and calculate the sum of 1st and last digit of that number.

```
import java.io.*;
class SumDemo
{
        public static void main(String args[])
        {
                int n, a, b, s=0;
                try
                {
                    BufferedReader br=new BufferedReader (new InputStreamReader
                                                          (System.in));
                   System.out.println ("Enter the Number");
                   n=Integer.parseInt (br.readLine ());
                   a=n%10;
                   while (n>0)
                   {
                           b=n%10;
                           n=n/10;
                   }
                        s=a+b;
                        System.out.println ("Sum Is " + s);
               }
       }
```

finally Block:

To guarantee that a line of code runs, whether an exception occurs or not, use a finally block after the try and catch blocks. The code in the finally block will almost always execute, even if an unhandled exception occurs; in fact, even if a return statement is encountered

Syntax:

```
  try
  {
      Risky code/ unsafe code block
  }
  catch (ExceptionClassName exceptionObjectName)
  {
      Code to resolve problem
  }
  finally
  {
      Code that will always execute
  }
```

Example: Java program to demonstrate finally block.

```
class FinDemo
 {
      public static void main(String args[])
      {
           int a=5,b=0,c;
           try
           {
                c=a/b;
```

```
                        System.out.println(c);
                }
                catch ( Exception obj)
                {
                        System.out.println ("Error Is " + obj);
                }
                finally
                {
                        System.out.println ("You are in finally Block");
                }
        }
}
```

throw:

It is used to throw a user defined exception. User Defined Exception can be defined by inheriting Exception class in User defined Exception class.

class MyException extends Exception {}

Syntax:

It can be thrown by using throw keyword.

```
        throw ThrowableObj;
```

Example 1: Java program to check whether given name is valid or not.

```
import java.io.*;
class NameValExc extends Exception{}
class ExcDemo
{
        public static void main(String args[])
        {
                int i,n,flag=0;
                String nm;
                char ch;
                try
                {
BufferedReader br=new BufferedReader (new InputStreamReader
                                        (System.in));
                  System.out.println ("Enter Your Name");
                  nm=br.readLine();
                  n=nm.length();
                  for(i=0;i<n;i++)
                  {
```

```
                                        ch=nm.charAt(i);
                                        if(!(Character.isLetter(ch)))
                                        {
                                                throw new NameValExc();
                                                flag=0;
                                        }
                                        else
                                                flag=1;
                                  }
                                  if(flag==1)
                                        System.out.println ("Name Is Valid");
                                }
                                catch(NameValExc obj)
                                {
                                        System.out.println ("Name Is Invalid");
                                }
                        }
              }
```

Example 2: Java program to check given number is valid or not. If it is valid then display its factors, otherwise display appropriate message.

```
            import java.io.*;
            class ZeroNumExc extends Exception{}
            class ZeroNumDemo
            {
                  public static void main(String args[])
                {
                   int n,i;
                   try
                    {
                BufferedReader br=new BufferedReader(new InputStreamReader (System.in));
                        System.out.println("Enter a Number");
                        n=Integer.parseInt(br.readLine());
                        if(n==0)
                        {
                                throw new ZeroNumExc();
                        }
                        else
                        {
                                for(i=1;i<n;i++)
                                {
                                        if(n%i==0)
                                      System.out.println (i);
                                }
                        }
                   }
                  catch (ZeroNumExc obj)
                {
                        System.out.println("Number Is Zero");
                }
                catch (Exception ob)
                {
                        System.out.println ("Number Is Invalid");
                }
        }
 }
```

throws:

The throws statement is used by a method to specify the types of exceptions the method throws. If a method is capable of raising an exception that it does not handle, the method must specify that the exception have to be handled by the calling method. This is done using the throws statement.

Syntax:

[< access specifier >] [< modifier >] < return type > < method name >
[< arg list >] [throws <exception list >]

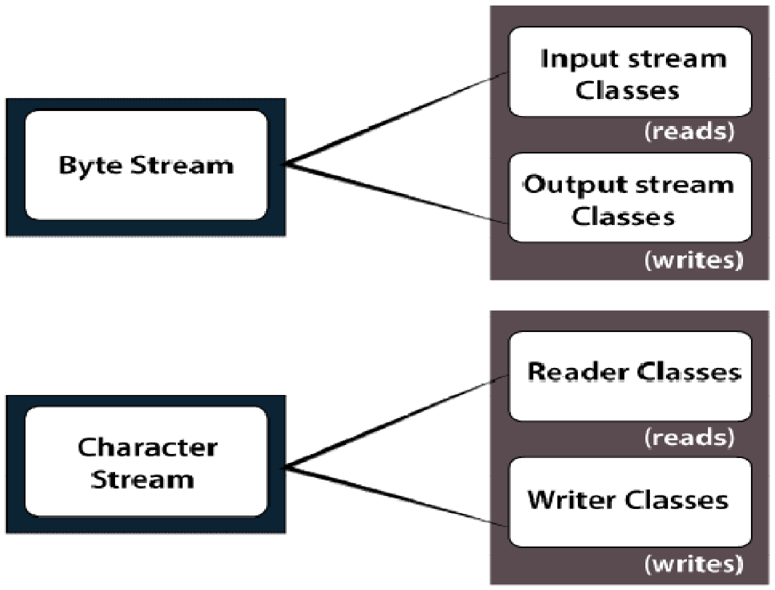
File Handling:

File: It is collection of data in byte form. File is used to store the data in proper way. File handling in Java is defined as reading and writing data to a file. The particular file class from the package called **java.io** allows us to handle and work with different formats of files.

In Java, a File is an abstract data type. A named location used to store related information is known as a File. There are several File Operations like creating a new File, getting information about File, writing into a File, reading from a File and deleting a File.

Stream

A series of data is referred to as a stream. In Java, Stream is classified into two types, i.e., Byte Stream and Character Stream.



Brief classification of I/O streams

Byte Stream

Byte Stream classes are used to read bytes from the input stream and write bytes to the output stream. In other words, we can say that ByteStream classes read/write the data of 8-bits. We can store video, audio, characters, etc., by using ByteStream classes. These classes are part of the java.io package.

The ByteStream classes are divided into two types of classes, i.e., InputStream and OutputStream. These classes are abstract and the super classes of all the Input/Output stream classes.

The List of Byte Stream classes are :

Stream class	**Description**
BufferedInputStream	Used for Buffered Input Stream.
BufferedOutputStream	Used for Buffered Output Stream.
DataInputStream	Contains method for reading java standard datatype
DataOutputStream	An output stream that contain method for writing java standard data type
FileInputStream	Input stream that reads from a file
FileOutputStream	Output stream that write to a file.
InputStream	Abstract class that describe stream input.
OutputStream	Abstract class that describe stream output.
PrintStream	Output Stream that contain print() and println() method

Methods of InputStream Classes:

Method	**Description**
read()	This method is abstract in the Input Stream class, so it has to be defined in a subway. This method returns the next byte available as stream int. Once the stream is getting towards an end, the method gives -1. An exception of type IOException will be thrown if an I/O error occurs.
read(byte[] array)	It reads byte from the successive streams to the array. The maximum of array.length bytes will be read. This method will not return the data until the stream gets to the end. If the method will reach the end then it will not return the end of bytes.
read(byte[] array, int offset, int length)	It works the same as the previous methods except the length.

Methods of OutputStream Classes:

Sr.No.	**Method & Description**
void close()	This method closes this output stream and releases any system resources associated with this stream.
void flush()	This method flushes this output stream and forces any buffered output bytes to be written out.
void write(byte[] b)	This method writes *b.length* bytes from the specified byte array to this output stream.
void write(byte[] b, int off, int len)	This method writes *len* bytes from the specified byte array starting at offset *off* to this output stream.
abstract void write(int b)	This method writes the specified byte to this output stream.

Example 1: Java program to display the data from a file. (Use command Line argument)

```
import java.io.*;
class FileRead
{
        public static void main (String args[]) throws Exception
        {
                int b;
                FileInputStream fin=new FileInputStream (args[0]);
                while((b=fin.read())!=-1)
                    {
                        System.out.print ((char)b);
                    }
                    fin.close();
            }
    }
```

Example 2: Java program to copy the data from one file into another file.

```
 import java.io.*;
class FileReadWrite
{
        public static void main (String args[]) throws Exception
        {
                int b;
                FileInputStream fin=new FileInputStream (args[0]);
                FileOutputStream fout=new FileOutputStream (args[1]);
                while((b=fin.read())!=-1)
                    {
                        fout.write(b);
                    }
                    fin.close();
                    fout.close();
            }
    }
```

Character Stream Classes:

The java.io package provides CharacterStream classes to overcome the limitations of ByteStream classes, which can only handle the 8-bit bytes and is not compatible to work directly with the Unicode characters. CharacterStream classes are used to work with 16-bit Unicode characters. They can perform operations on characters, char arrays and Strings.

However, the CharacterStream classes are mainly used to read characters from the source and write them to the destination. For this purpose, the CharacterStream classes are divided into two types of classes, I.e., Reader class and Writer class.

The List of Character Stream classes:

Stream class	**Description**
BufferedReader	Handles buffered input stream.
BufferedWriter	Handles buffered output stream.
FileReader	Input stream that reads from file.
FileWriter	Output stream that writes to file.
InputStreamReader	Input stream that translate byte to character
OutputStreamReader	Output stream that translate character to byte.
PrintWriter	Output Stream that contain print() and println() method.
Reader	Abstract class that define character stream input
Writer	Abstract class that define character stream output

Methods of Reader classes are:

Method	**Description**
int read()	This method returns the integral representation of the next character present in the input. It returns -1 if the end of the input is encountered.
int read(char buffer[])	This method is used to read from the specified buffer. It returns the total number of characters successfully read. It returns -1 if the end of the input is encountered.
int read(char buffer[], int loc, int nChars)	This method is used to read the specified nChars from the buffer at the specified location. It returns the total number of characters successfully read.
void mark(int nchars)	This method is used to mark the current position in the input stream until nChars characters are read.
void reset()	This method is used to reset the input pointer to the previous set mark.
long skip(long nChars)	This method is used to skip the specified nChars characters from the input stream and returns the number of characters skipped.
boolean ready()	This method returns a boolean value true if the next request of input is ready. Otherwise, it returns false.
void close()	This method is used to close the input stream. However, if the program attempts to access the input, it generates IOException.

Methods of Writer classes are:

Method	**Description**
void write()	This method is used to write the data to the output stream.
void write(int i)	This method is used to write a single character to the output stream.
void write(char buffer[])	This method is used to write the array of characters to the output stream.
void write(char buffer [],int loc, int nChars)	This method is used to write the nChars characters to the character array from the specified location.
void close ()	This method is used to close the output stream. However, this generates the IOException if an attempt is made to write to the output stream after closing the stream.
void flush ()	This method is used to flush the output stream and writes the waiting buffered characters.

Example 1: Java program to read the data from a file.

```
import java.io.*;
class FileRead
{
  public static void main(String args[])
  {
    char[] array = new char[100];
     try
    {
          FileReader input = new FileReader ("input.txt");
          input.read(array);
          System.out.println ("Data in the file:");
          System.out.println(array);
          input.close();
    }
    catch (Exception e)
    {
      e.getStackTrace ();
    }
  }
}
```

Example 2 : Java program to write data into the file.

```
import java.io.*;
class FileWrite
{
  public static void main(String args[])
  {
     String data = "This is the data in the output file";
     try
     {
        FileWriter output = new FileWriter("output.txt");
        output.write (data);
        System.out.println("Data is written to the file.");
        output.close ();
     }
    catch (Exception e)
     {
     e.getStackTrace ();
     }
  }
}
```

File class:
It is related with characteristics of a file such as name, size, location etc.
Syntax:

```
File f=new File ("Path");
```

Methods of File Class:

Method	**Description**
createTempFile(String prefix, String suffix)	It creates an empty file in the default temporary-file directory, using the given prefix and suffix to generate its name.
createNewFile()	It atomically creates a new, empty file named by this abstract pathname if and only if a file with this name does not yet exist.
canWrite()	It tests whether the application can modify the file denoted by this abstract pathname.String[]
canExecute()	It tests whether the application can execute the file denoted by this abstract pathname.

canRead()	It tests whether the application can read the file denoted by this abstract pathname.
isAbsolute()	It tests whether this abstract pathname is absolute.
isDirectory()	It tests whether the file denoted by this abstract pathname is a directory.
isFile()	It tests whether the file denoted by this abstract pathname is a normal file.
getName()	It returns the name of the file or directory denoted by this abstract pathname.
getParent()	It returns the pathname string of this abstract pathname's parent, or null if this pathname does not name a parent directory.
toPath()	It returns a java.nio.file.Path object constructed from the this abstract path.
toURI()	It constructs a file: URI that represents this abstract pathname.
listFiles()	It returns an array of abstract pathnames denoting the files in the directory denoted by this abstract pathname
getFreeSpace()	It returns the number of unallocated bytes in the partition named by this abstract path name.
list(FilenameFilter filter)	It returns an array of strings naming the files and directories in the directory denoted by this abstract pathname that satisfy the specified filter.
mkdir()	It creates the directory named by this abstract pathname.

Example 1: Java Program to create a new File.

```
import java.io.*;
class FileDemo
{
   public static void main (String args[])
   {
      try
      {
         File file = new File ("DYP.txt");
         if (file.createNewFile())
         {
                System.out.println ("New File is created!");
          }
         else
         {
             System.out.println ("File already exists.");
          }
       }
        catch (IOException e)
       {
         System.out.println (e);
       }
    }
}
```

Example 2: Java program to display details of files from a given directory.

```
import java.io.*;
class FileDemo
{
     public static void main(String args[])
     {
          File dir=new File(“E:/Satyavan");
          File files[]=dir.listFiles();
          for(File file:files)
          {
                    System.out.println (file.getName()+" Can Write: "+file.canWrite()+"
                   Is Hidden: "+file.isHidden ()+" Length: "+file.length()+" bytes");
           }
     }
}
```

Example 3: Write a java program to delete a given file.

```
          import java.io.*;
          class FileDel
          {
              public static void main (String[] args)
              {
                  File file = new File("file.txt");
                  boolean value = file.delete();
                  if(value)
                  {
                             System.out.println ("The File is deleted.");
```

```
            }
            else
            {
                System.out.println ("The File is not deleted.");
            }
        }
    }
```

Example 4: Java program to rename a given file.

```
import java.io.*;
class FileRename
{
   public static void main(String args[])
   {
        File file = new File("oldName");
        try
        {
                file.createNewFile();
        }
        catch(Exception e)
        {
             e.getStackTrace();
        }
             File newFile = new File("newName");
             boolean value = file.renameTo(newFile);
             if(value)
             {
                  System.out.println("The name of the file is changed.");
             }
           else
           {
                  System.out.println("The name cannot be changed.");
           }
   }
}
```

Practice Set:

1. Write a java program to accept the data from a user and write it into the file.
2. Write a java program to display ASCII values of the characters from a file.
3. Write a java program to count number of digits, spaces and characters from a file.
4. Write a java program to accept a number from user if it is non-zero then check whether it is Armstrong or not, otherwise throws user defined Exception "Number is Invalid".
5. Write a java program to check whether given file is hidden or not.
6. Write a java program to display name and size of the given files.

Set A:

1. Write a java program to count the number of integers from a given list.(Use command line arguments).
2. Write a java program to check whether given candidate is eligible for voting or not. Handle user defined as well as system defined Exception.
3. Write a java program to calculate the size of a file.
4. Write a java program to accept a number from a user, if it is zero then throw user defined Exception "Number is Zero". If it is non-numeric then generate an error "Number is Invalid" otherwise check whether it is palindrome or not.
5. Write a java program to accept a number from user, If it is greater than 100 then throw user defined exception "Number is out of Range" otherwise do the addition of digits of that number. (Use static keyword)

Set B:

1. Write a java program to copy the data from one file into another file, while copying change the case of characters in target file and replaces all digits by '*' symbol.
2. Write a java program to accept string from a user. Write ASCII values of the characters from a string into the file.
3. Write a java program to accept a number from a user, if it less than 5 then throw user defined Exception "Number is small", if it is greater than 10 then throw user defined exception "Number is Greater", otherwise calculate its factorial.
4. Write a java program to display contents of a file in reverse order.
5. Write a java program to display each word from a file in reverse order.

Set C:

1. Write a java program to accept list of file names through command line. Delete the files having extension .txt. Display name, location and size of remaining files.
2. Write a java program to display the files having extension .txt from a given directory.
3. Write a java program to count number of lines, words and characters from a given file.
4. Write a java program to read the characters from a file, if a character is alphabet then reverse its case, if not then display its category on the Screen. (whether it is Digit or Space)

5. Write a java program to validate PAN number and Mobile Number. If it is invalid then throw user defined Exception “Invalid Data”, otherwise display it.

Assignment Evaluation:

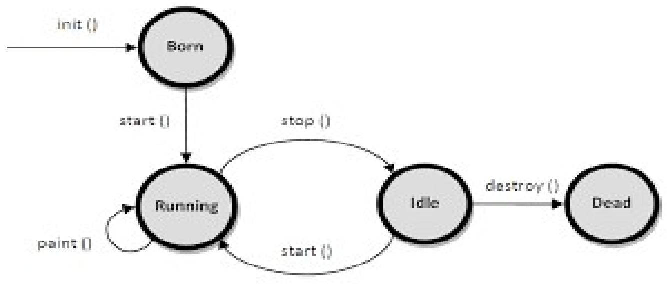
0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: WellDone [ ]

Signature of Instructor

Assignment No. 5: Applet, AWT, Event and Swing Programming

Graphical User Interface elements are implemented in two java packages – AWT and Swing. Swing is the newer package and swing classes are based on AWT classes. In this assignment we will learn three important concepts of java Applet, AWT, Event and Swing Programming

APPLET: Applets are small java programs which are executed and displayed in a java compatible web browser.



Applet Lifecycle

Creating an applet

All applets are subclasses of the java.applet.Applet class. You can also create an applet by extending the javax.swing.JApplet class. The syntax is:

```
class MyApplet extends Applet
{
   //applet methods
}
```

Applet methods:

Method	Description	Example

init( )	Automatically called to perform initialization of the applet. Executed only once.	public void init()
		{
		//initialisation
		}

start( )	Called every time the applet moves into sight on the Web browser to allow the applet to start up its normal operations.	public void start()
		{
		//Code
		}

stop( )	Called every time the applet moves out of sight on the Web browser to allow the applet to shut off expensive operations.	public void stop()
		{
		//Code
		}

destroy( )	Called when the applet is being unloaded from the page to perform final release of resources when the applet is no longer used	public void destroy()
		{
		// Code
		}

paint()	Called each time the applets output needs to be redrawn.	public void paint(Graphics g)
		{
		//Display Statements
		}

Running an applet

1. Compile the applet code using javac
2. Use the java tool – appletviewer to view the applet (embed the APPLET tag in comments in the code)
3. Use the APPLET tag in an HTML page and load the applet in a browser

Using appletviewer:

1. Write the HTML APPLET tag in comments in the source file.
2. Compile the applet source code using javac.
3. Use appletviewer ClassName.class to view the applet.

Using browser:

1. Create an HTML file containing the APPLET tag.
2. Compile the applet source code using javac.
3. In the web browser, open the HTML file.

The APPLET tag

```
< APPLET
        [CODEBASE    = appletURL]
        CODE         = appletClassFile
        [ALT         = alternateText]
        [ARCHIVE     = archiveFile]
        [NAME        = appletInstanceName]
        WIDTH        = pixels
        HEIGHT       = pixels
        [ALIGN       = alignment]
        [VSPACE      = pixels]
        [HSPACE      = pixels]
>
[< PARAM NAME = AttributeName VALUE = AttributeValue />]
</APPLET>
```

Attribute	Value	Meaning
align	left right top bottom middle baseline	Specifies the alignment of an applet according to surrounding elements
alt	text	Specifies an alternate text for an applet
archive	URL	Specifies the location of an archive file
code	URL	Specifies the file name of a Java applet
codebase	URL	Specifies a relative base URL for applets specified in the code attribute
height	pixels	Specifies the height of an applet
hspace	pixels	Defines the horizontal spacing around an applet
name	name	Defines the name for an applet (to use in scripts)
vspace	pixels	Defines the vertical spacing around an applet
width	pixels	Specifies the width of an applet

The mandatory attributes are CODE, HEIGHT and WIDTH.
Examples:

1. <applet code=MyApplet width=200 height=200 archive="files.jar">
   </applet>

2. <applet code=Simple.class width=100 height=200 codebase="example/">
   </applet>

Passing parameters to applets

The PARAM tag allows us to pass information to an applet when it starts running. A parameter is a NAME – VALUE pair. Every parameter is identified by a name and it has a value.
< PARAM NAME = AttributeName VALUE = AttributeValue />

Example:

```
<APPLET     NAME = "MyApplet.class" WIDTH = 100 HEIGHT = 100>
<PARAM NAME = "ImageSource" VALUE = "project/images/">
<PARAM NAME = "BackgroundColor" VALUE = "0xc0c0c0">
<PARAM NAME = "FontColor" VALUE = "Red">
</APPLET>
```

Example: Using getParameter()

```
String dirName = getParameter("ImageSource");
Color c = new Color( Integer.parseInt(getParameter("BackgroundColor")));
```

Sr No	Method	Description
1	public abstract void drawString(String str, int x, int y)	Used to draw specified string.
2	public void drawRect(int x, int y, int width, int height)	Used to draw a rectangle of specified width and height.
3	public abstract void fillRect(int x, int y, int width, int height)	Used to draw a rectangle with a default colourof specified width and height.
4	public abstract void drawOval(int x, int y, int width, int height)	Used to draw oval of specified width and height.
5	public abstract void fillOval(int x, int y, int width, int height)	Used to draw oval with a default colour of specified width and height.
6	public abstract void drawLine(int x1, int y1, int x2, int y2)	Used for drawing lines between the point (x1, x1) and (x2, y2).
7	public abstract booleandrawImage(Image img, int x, int y, ImageObserver observer)	Used for drawing a specified image.
8	public abstract void drawArc(int x, int y, int width, int height, intstartAngle, intarcAngle)	Used for drawing a circular arc.
9	public abstract void fillArc(int x, int y, int width, int height, intstartAngle, intarcAngle)	Used for filling circular arc.
10	public abstract void setColor(Color c)	Used to set a colour to the object.
11	public abstract void setFont(Font font)	Used to set font.
12	public void paint(Graphics g)	The paint() method draws the applet.
13	public void repaint()	The repaint() method is used to force redrawing of the applet.
14	public void update()	The update() method redraws only a portion of the applet.
15	String getParameter(String Parameter);	The Applet can retrieve information about the parameters using the getParameter() method.

Example 1: Program to demonstrate Applet Lifecycle.

```
import java.applet.*;
```

```
 public class AppletLifeCycle extends Applet
{
        public void init()
        {
                System.out.println("Applet is Initialized");
        }
        public void start()
        {
                System.out.println("Applet is being Executed");
        }
        public void stop()
        {
                System.out.println("Applet execution has Stopped");
        }
        public void paint(Graphics g)
        {
                System.out.println("Painting the Applet...");
        }
        public void destroy()
        {
                System.out.println("Applet has been destroyes..!”); ha
        }
}
```

```
// AppletLifecycle.html
<html>
<body>
<applet code =” AppletLifeCycle” width=200 height=60>
</applet>
<//body>
</html>
```

- Compile the above program using

javac AppletLifeCycle.java

- Execute the applet using

appletviewer AppletLifeCycle.html

Example 2: Java Program to demonstrate simple applet.

```
import java.applet.Applet;
import java.awt.Graphics;
// HelloWorld class extends Applet
```

```
public class HelloWorldApplet extends Applet
{
	// Overriding paint() method
	public void paint (Graphics g)
	{
		g.drawString ("Hello World", 20, 20);
	}
}
```

Example 3: Java Program to show status using applet.

```
import java.awt.*;
import java.applet.*;
/*
<applet code ="StatusWindow" width=300 height=50>
</applet>
*/
public class StatusWindow extends Applet
{
	public void init ()
	{
		setBackground (Color.cyan);
	}
	public void paint (Graphics g)
	{
		g.drawString ("This is in the applet window", 10, 20);
		showStatus ("This is shown in the status window.");
	}
}
```

Example 4: Sample Program Passing Parameters to Applet.

```
import java.applet.Applet;
import java.awt.Graphics;
/*
<applet code="ParamDemo" width="300" height="300">
<param name=fontName value=Welcome>
</applet>
*/
public class ParamDemo extends Applet
{
	String fontName;
	public void init ()
	{
```

```
                fontName = getParameter ("pname");
                if (fontName == null)
                {
                        fontName = "Welcome to Applet Window";
                        fontName = "Hello " + fontName;
                }
        }
        public void paint (Graphics g)
        {
                g.drawString (fontName, 0, 10);
        }
}
```

Example 5: Program to draw rectangle.

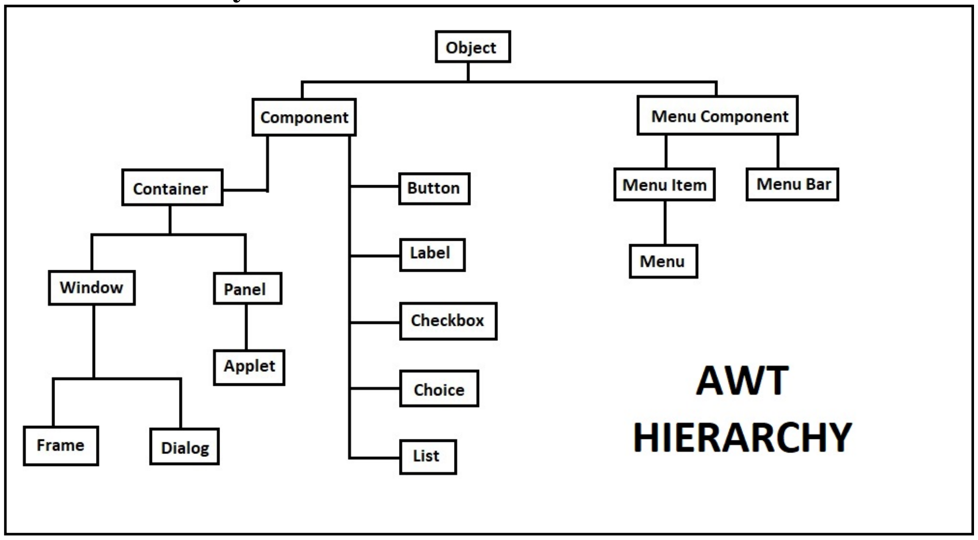
```
import java.applet.*;
import java.awt.*;
public class MyApplet extends Applet
{
        int height, width;
        public void init()
        {
                height = getSize().height;
                width = getSize().width;
                setName("MyApplet");
        }
        public void paint(Graphics g)
        {
                g.drawRoundRect(10, 30, 120, 120, 2, 3);
        }
}
```

Example 6: Program to draw different Shapes.

```
import java.applet.Applet;
import java.awt.*;
public class GraphicsDemo1 extends Applet
{
  public void paint(Graphics g)
  {
        g.setColor(Color.black);
        g.drawString("Welcome to TYBBA_CA",50, 50);
        g.setColor(Color.blue);
        g.fillOval(170,200,30,30);
```

```
            g.drawArc(90,150,30,30,30,270);
            g.fillArc(270,150,30,30,0,180);
            g.drawLine(21,31,20,300);
            g.drawRect(70,100,30,30);
            g.fillRect(170,100,30,30);
            g.drawOval(70,200,30,30);
  }
}
```

AWT Hierarchy:



AWT Controls in Java: Controls are components that allow a user to interact with your application in various ways. The AWT supports the following types of controls:

1. **Labels:** The easiest control to use is a label. A label contains a string and is an object of type Label. Labels are passive controls that do not support any interaction with the user.

Creating Label : Label l = new Label(String);

Label Constructors:

1. **Label() throws HeadlessException:** It creates a blank label.
2. **Label(String str) throws HeadlessException:** It creates a label that contains the string specified by str.
3. **Label(String str, int how):** It creates a label that contains the string specified by str using the alignment specified by how. The value of how must be one of these three constants: **LEFT**, **Label.RIGHT, Label.CENER**.

Label Methods:

1. **void setText(String str):** It is used to set or change the text in a label by using the setText() method. Here, str specifies the new label.
2. **String getText()**: It is used to obtain the current label by calling getText() method. Here, the current label is returned.
3. **void setAlignment(int how):** It is used to set the alignment of the string within the label by calling setAlignment() method. Here, how is one of the alignment constants?

4. **int getAlignment():** It is used to obtain the current alignment, getAlignment() is called.

2. AWT Buttons Control: The most widely used control is Button. A button is a component that contains a label and that generates an event when it is pressed.
Creating Button : Button b = new Button(String label);

Button Constructors:
1. **Button() throws HeadlessException:** It creates an empty button.
2. **Button(String str) throws HeadlessException**: It creates a button that contains str as a label.

Button Methods :
1. **void setLabel(String str):** You can set its label by calling setLabel(). Here, str is the new Label for the button.
2. **String getLabel():** You can retrieve its label by calling getLabel() method.

3. Canvas Control in java: Canvas encapsulates a blank window upon which you can draw in an application or receive inputs created by the user.
Canvas Constructor:
1. **Canvas() :** Constructs a new Canvas.
2. **Canvas (GraphicsConfiguration config) :** Constructs a new Canvas given a GraphicsConfiguration object.

Canvas Methods:
1. **void addNotify():** It is used to create the peer of the canvas.
2. **void createBufferStrategy(int numBuffers):** It is used to create a new strategy for multi-buffering on this component.
3. **BufferStrategy getBufferStrategy():** It is used to return the BufferStrategy used by this component.

4. Checkbox Control in java: This component is used to display Checkbox.
Creating Checkbox : Checkbox cb = new Checkbox(Label);

Checkbox Constructor
1. **Checkbox() throws HeadlessException:** Creates a checkbox whose label is initially blank. The state of the checkbox is unchecked.
2. **Checkbox(String str) throws HeadlessException:** Creates a checkbox whose label is specified by str. The state of the checkbox is unchecked.
3. **Checkbox(String str, Boolean on) throws HeadlessException:** It allows you to line the initial state of the checkbox. If one is true, the checkbox is initially checked; otherwise, it's cleared.
4. **Checkbox(String str, Boolean on, CheckboxGroup cbGroup) throws HeadlessException or Checkbox(String str, CheckboxGroup cbGroup, Boolean on) throws HeadlessException:** It creates a checkbox whose label is specified by str and whose group is specified by cbGroup. If this checkbox isn't a part of a gaggle, then cbGroup must be null. the worth of on determines the initial state of the checkbox.

Methods of Checkbox
1. **boolean getState():** To retrieve the present state of a checkbox.

2. **void setState(boolean on):** To line its state, call setState(). Here, if one is true, the box is checked. If it's false, the box is cleared.
3. **String getLabel():** you'll obtain the present label related to a checkbox by calling getLabel().
4. **void setLabel(String str):** To line the label, call setLabel(). The string passed in str becomes the new label related to the invoking checkbox.

5. **CheckboxGroup: Radio Buttons**

Creating Radiobutton :

CheckboxGroup cbg = new CheckboxGroup();
Checkbox rb = new Checkbox(Label, cbg, boolean);

CheckboxGroup Methods

1. **Checkbox getSelectedCheckbox():** You can determine which checkbox in a group is currently selected by calling getSelectedCheckbox().
2. **void setSelectedCheckbox(Checkbox which):** You can set a checkbox by calling setSelectedCheckbox(). Here, which is the checkbox that you simply want to be selected. The previously selected checkbox is going to be turned off.

6. AWT Choice Control in Java: This Component is use to create a dropdown list.
Note: Choice only defines the default constructor, which creates an empty list.
Creating Choice : Choice ch = new Choice();

Choice Methods

1. **void add(String name):** To add a selection to the list, use add(). Here, the name is the name of the item being added.
2. **String getSelectedItem():** It determines which item is currently selected. It returns a string containing the name of the item.
3. **int getSelectedIndex():** It determines which item is currently selected. It returns the index of the item.
4. **int getItemCount():** It obtains the number of items in the list.
5. **void select(int index):** It is used to set the currently selected item with a zero-based integer index.
6. **void select(String name):** It is used to set the currently selected item with a string that will match a name in the list.
7. **String getItem(int index):** It is used to obtain the name associated with the item at the given index. Here, the index specifies the index of the desired items.

7. AWT List Control in Java: This component will display a group of items as a drop-down menu from which a user can select only one item. The List class provides a compact, multiple-choice, scrolling selection list.
Creating List : List l = new List(int, Boolean);
List Constructor

1. **List() throws HeadlessException:** It creates a list control that allows only one item to be selected at any one time.
2. **List(int numRows) throws HeadlessException:** Here, the value of numRows specifies the number of entries in the list that will always be visible.

3. **List(int numRows, boolean multipleSelect) throws HeadlessException:** If multipleSelect is true, then the user may select two or more items a time. If it's false, then just one item could also be selected.

Method of Lists

1. **void add(String name):** To add a selection to the list, use add(). Here, the name is that the name of the item being added. It adds items to the end of the list.
2. **void add(String name, int index):** It also adds items to the list but it adds the items at the index specified by the index.
3. **String getSelectedItem():** It determines which item is currently selected. It returns a string containing the name of the item. If more than one item is selected, or if no selection has been made yet, null is returned.
4. **int getSelectedIndex():** It determines which item is currently selected. It returns the index of the item. The first item is at index 0. If more than one item is selected, or if no selection has yet been made, -1 is returned.
5. **String[] getSelectedItems():** It allows multiple selections. It returns an array containing the names of the currently selected items.
6. **int[] getSelectedIndexes():** It also allows multiple selections. It returns an array containing the indexes of the currently selected items.
7. **int getItemCount():** It obtains the number of items in the list.
8. **void select(int index):** It is used to set the currently selected item with a zero-based integer index.
9. **String getItem(int index):** It is used to obtain the name associated with the item at the given index. Here, the index specifies the index of the desired items.

8. AWT Scroll Bar Control in java: This component is used to create a ScrollBar.

Creating Scrollbar : Scrollbar sb = new Scrollbar();

Scrollbar Constructor:

1. **Scrollbar() throws HeadlessException:** It creates a vertical scrollbar.
2. **Scrollbar(int style) throws HeadlessException:** It allows you to specify the orientation of the scrollbar. If style isScrollbar.VERTICAL, a vertical scrollbar is created. If a style is Scrollbar.HORIZONTAL, the scroll bar is horizontal.
3. **Scrollbar(int style, int initialValue, int thumbSize, int min, int max) throws HeadlessException:** Here, the initial value of the scroll bar is passed in initialValue. The number of units represented by the peak of the thumb is passed in thumbSize. The minimum and maximum values for the scroll bar are specified by min and max.

Scrollbar Methods:

1. **void setValues(int initialValue, int thumbSize, int min, int max):** It is used to set the parameters of the constructors.
2. **int getValue():** It is used to obtain the current value of the scroll bar. It returns the current setting.
3. **void setValue(int newValue):** It is used to set the current value. Here, newValue specifies the new value for the scroll bar. When you set a worth, the slider box inside the scroll bar is going to be positioned to reflect the new value.
4. **int getMaximum():** It is used to retrieve the minimum values. They return the requested quantity. By default, 1 is the increment added to the scroll bar.
5. **int getMaximun():** It is used to retrieve the maximum value. By default, 1 is the increment subtracted from the scroll bar.

9. **AWT TextComponent Control in Java:** The TextComponent class is the superclass of any component that permits the editing of some text. A text component embodies a string of text. There are two types of TextComponent: **TextField, TextArea**

   1. **TextField:** The TextField component will allow the user to enter some text. It is used to implements a single-line text-entry area, usually called an edit control.

      Creating TextField : TextFielf tf = new TextField(size);

      TextField Constructors:

      1. **TextField() throws HeadlessException:** It creates a default textfield.
      2. **TextField(int numChars) throws HeadlessException:** It creates a textfield that is numChars characters wide.
      3. **TextField(String str) throws HeadlessException:** It initializes the textfield with the string contained in str.
      4. **TextField(String str, int numChars) throws HeadlessException:** It initializes a text field and sets its width.

      TextField Methods:

      1. **String getText():** It is used to obtain the string currently contained in the text field.
      2. **void setText(String str):** It is used to set the text. Here, str is the new String.
      3. **void select(int startIndex, int endIndex):** It is used to select a portion of the text under program control. It selects the characters beginning at startIndex and ending at endIndex-1.
      4. **String getSelectedText():** It returns the currently selected text.
      5. **boolean isEditable():** It is used to determine editability. It returns true if the text may be changed and false if not.
      6. **void setEditable(boolean canEdit):** It is used to control whether the contents of a text field may be modified by the user. If canEdit is true, the text may be changed. If it is false, the text cannot be altered.
      7. **void setEchoChar(char ch):** It is used to disable the echoing of the characters as they are typed. This method specifies a single character that the TextField will display when characters are entered.
      8. **boolean echoCharIsSet():** By this method, you can check a text field to see if it is in this mode.
      9. **char getEchochar():** It is used to retrieve the echo character.

   2. **TextArea:** Sometimes one line of text input isn't enough for a given task. To handle these situations, the AWT includes an easy multiline editor called TextArea.

      Creating TextArea : TextArea ta = new TextArea();

      TextArea Constructor:

      1. **TextArea() throws HeadlessException:** It creates a default textarea.
      2. **TextArea(int numLines, int numChars) throws HeadlessException:** It creates a text area that is numChars characters wide. Here, numLines specifies the height, in lines of the text area.
      3. **TextArea(String str) throws HeadlessException:** It initializes the text area with the string contained in str.

4. **TextArea(String str, int numLines, int numChars) throws HeadlessException:** It initializes a text field and sets its width. Initial text can be specified by str.
5. **TextArea(String str, int numLines, int numChars, int sBars) throws HeadlessException:** Here, you can specify the scroll bars that you want the control to have. sBars must be one of these values :
    1. **SCROLLBARS_BOTH**
    2. **SCROLLBARS_NONE**
    3. **SCROLLBARS_HORIZONTAL_ONLY**
    4. **SCROLLBARS_VERTICAL_ONLY**

TextArea Methods: TextArea is a subclass of TextComponent. Therefore, it supports the **getText( ), setText( ), getSelectedText( ), select( ), isEditable( ),** and **setEditable( )** methods described in the TextField section. **TextArea** adds the following methods:

1. **void append(String str):** It appends the string specified by str to the end of the current text.
2. **void insert(String str, int index):** It inserts the string passed in str at the specified index.
3. **voidReplaceRange(String str, int startIndex, int endIndex):** It is used to replace the text. It replaces the characters from startIndex to endIndex-1.

Example: Java Program to display a simple calculator using AWT

```
import java.awt.*;
import java.awt.event.*;
public  class Calculator extends Frame implements ActionListener
{
 Button on,clear,addn,subn,muln,divn,end,equal;
 TextField input;
 Panel pan1,pan2;
 int no1,no2,ans;
        char oper;
        public Calculator ()
        {
                pan1= new Panel();
                pan2=new Panel();
                addn=new Button("+");
                subn=new Button("-");
                muln=new Button("*");
                divn=new Button("/");
                equal=new Button("=");
                end=new Button("Exit");
                on=new Button("On");
                clear=new Button("Clear");
                input=new TextField();
                pan1.add(addn);
                pan1.add(subn);
                pan1.add(muln);
```

```
		pan1.add(divn);
		pan1.add(equal);

		pan2.add(on);
		pan2.add(clear);
		pan2.add(end);

		setLayout(new BorderLayout());
		add(input,"North");
		add(pan1,"Center");
		add(pan2,"South");

		input.setEnabled(false);
		addn.setEnabled(false);
		subn.setEnabled(false);
		muln.setEnabled(false);
		divn.setEnabled(false);

		addn.addActionListener(this);
		subn.addActionListener(this);
		muln.addActionListener(this);
		divn.addActionListener(this);
		equal.addActionListener(this);
		end.addActionListener(this);
		on.addActionListener(this);
		clear.addActionListener(this);

		setTitle("ArithMetic Calculator");
		setSize(400,400);
		setVisible(true);
	}
	public static void main(String args[])
	{
		new Ass15().show();

	}
	public void actionPerformed(ActionEvent e)
	{
		Button btn= (Button)e.getSource();
		if(btn==end)
		{
			System.exit(0);
		}
		if(btn==addn || btn==subn || btn==muln || btn==divn)
		{
			no1=Integer.parseInt(input.getText());
			oper=btn.getLabel().charAt(0);
		}
```

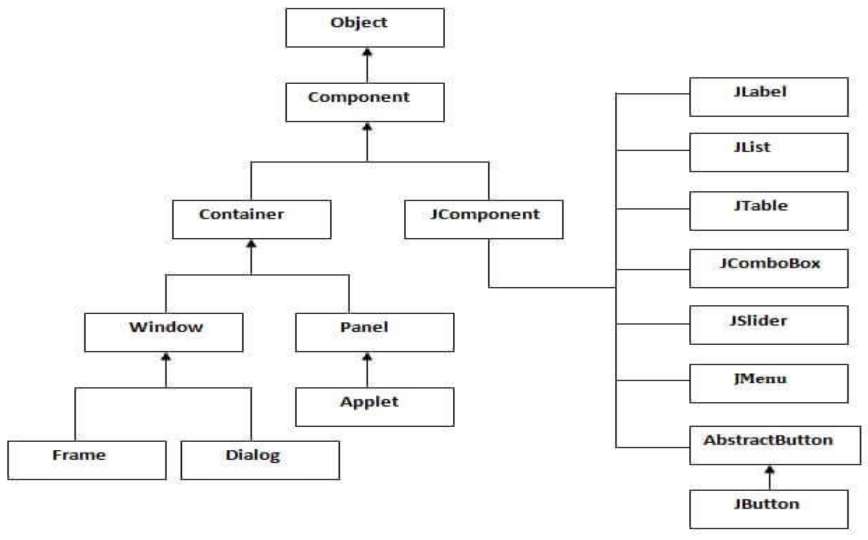
```
            if(btn==equal)
            {
                    no2=Integer.parseInt(input.getText());
                    switch(oper)
                    {
                            case '+' :
                                            ans=no1 + no2;
                                            break;
                            case '-' :
                                            ans=no1 - no2;
                                            break;
                            case '*' :
                                            ans=no1 * no2;
                                            break;
                            case '/' :
                                            ans=no1 / no2;
                                            break;
                    }
                    input.setText(Integer.toString(ans));
             }
        if(btn==clear)
        {
                input.setText("");
                input.requestFocus();
        }
        if(btn==on)
        {
            input.setEnabled(true);
                addn.setEnabled(true);
                subn.setEnabled(true);
                muln.setEnabled(true);
                divn.setEnabled(true);

        }
}
}
```

Swing Architecture:

The design of the Swing component classes is based on the Model-View-Controller architecture, or MVC.

1. The model stores the data.
2. The view creates the visual representation from the data in the model.
3. The controller deals with user interaction and modifies the model and/or the view.

Swing Class Heirarchy:



Layout Manager: The job of a layout manager is to arrange components on a container. Each container has a layout manager associated with it. To change the layout manager for a container, use the setLayout() method.

Syntax:

```
setLayout(LayoutManager obj)
```

Types of Layout Managers: AWT package provides following types of Layout Managers

1. Flow Layout
2. Border Layout
3. Card Layout
4. Grid Layout
5. Grid Bag Layout

Examples:

```
JPanel p1 = new JPanel()
p1.setLayout(new FlowLayout());
p1.setLayout(new BorderLayout());
p1.setLayout(new GridLayout(3,4));
```

1. Flow Layout: This layout will display the components in sequence from left to right, from top to bottom.

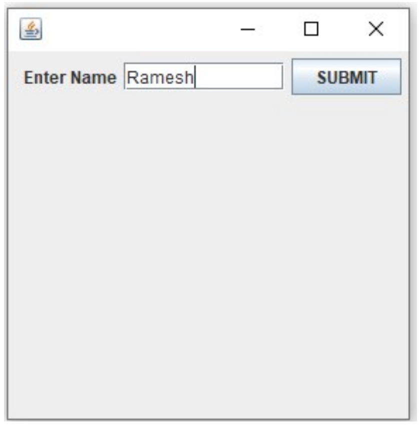
Costructor:

```
FlowLayout f1 = new FlowLayout();
FlowLayout f1 = new FlowLayout(int align);
FlowLayout f1 = new FlowLayout(int align, int hgap, int vgap);
```

For Example : Java Program to demonstrate Flow Layout Manager

```
import java.awt.*;
import javax.swing.*;
public class FlowLayoutDemo
{
        JFrame f;
        FlowLayoutDemo ()
        {
                f = new JFrame ();
                JLabel l1 = new JLabel ("Enter Name");
                JTextField tf1 = new JTextField (10);
                JButton b1 = new JButton ("SUBMIT");
                f.add (l1);
                f.add (tf1);
                f.add (b1);
                f.setLayout (new FlowLayout (FlowLayout.RIGHT));
                //setting flow layout of right alignment
                f.setSize (300, 300);
                f.setVisible (true);
        }
        public static void main (String[]args)
        {
                new FlowLayoutDemo ();
        }
}
```

Output:



2. Border Layout: This layout will display the components along the border of the container. This layout contains five locations. Locations are North, South, East, west, and Center. The default region is the center.

Costructor:

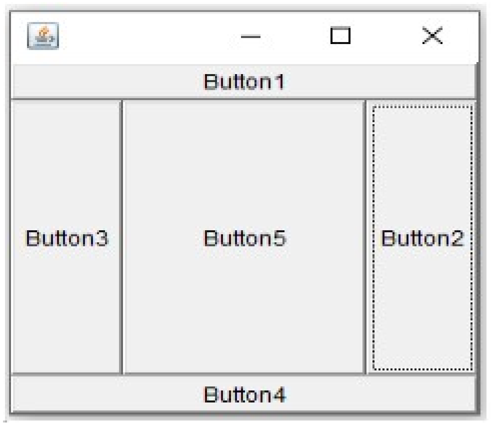
BorderLayout bl = new BorderLayout();

BorderLayout bl = new BorderLayout(int vgap, int hgap);

For Example : Java Program to demonstrate Border Layout Manager.

```
import java.awt.*;
public class BorderLayoutDemo
{
   public static void main (String[]args)
   {
      Frame f1 = new Frame ();
      f1.setSize (250, 250);
      Button b1 = new Button ("Button1");
      Button b2 = new Button ("Button2");
      Button b3 = new Button ("Button3");
      Button b4 = new Button ("Button4");
      Button b5 = new Button ("Button5");
      f1.add (b1, BorderLayout.NORTH);
      f1.add (b2, BorderLayout.EAST);
      f1.add (b3, BorderLayout.WEST);
      f1.add (b4, BorderLayout.SOUTH);
      f1.add (b5);
      f1.setVisible (true);
   }
}
```

Output:



3.Card Layout: A card layout represents a stack of cards displayed on a container. At a time only one card can be visible and each can contain the only component.

Costructor

CardLayout cl = new CardLayout();

CardLayout cl = new CardLayout(int hgap, int vgap);

To add the components in CardLayout we use add method:

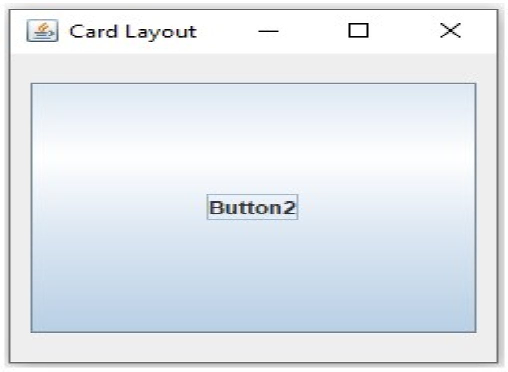
add(“Cardname”, Component);

Methods of CardLayout

1. first(Container): It is used to flip to the first card of the given container.
2. last(Container): It is used to flip to the last card of the given container.
3. next(Container): It is used to flip to the next card of the given container.
4. previous(Container): It is used to flip to the previous card of the given container.
5. show(Container, cardname): It is used to flip to the specified card with the given name.

For Example: Java Program to demonstrate Card Layout Manager.

```
import java.awt.*;
import javax.swing.*;
import javax.swing.JButton;
import java.awt.event.*;
public class CardLayoutDemo extends JFrame implements ActionListener
{
   JButton b1, b2, b3, b4, b5;
   CardLayout cl;
   Container c;
   CardLayoutDemo ()
   {
      b1 = new JButton ("Button1");
      b2 = new JButton ("Button2");
      b3 = new JButton ("Button3");
      b4 = new JButton ("Button4");
      b5 = new JButton ("Button5");
      c = this.getContentPane ();
      cl = new CardLayout (10, 20);
      c.setLayout (cl);
      c.add ("Card1", b1);
      c.add ("Card2", b2);
      c.add ("Card3", b3);
      b1.addActionListener (this);
      b2.addActionListener (this);
      b3.addActionListener (this);
      setVisible (true);
      setSize (400, 400);
      setTitle ("Card Layout");
      setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
   }
   public void actionPerformed (ActionEvent ae)
   {
      cl.next (c);
   }
   public static void main (String[]args)
   {
      new CardLayoutDemo ();
   }
}
```
Output:



4. Grid Layout: The layout will display the components in the format of rows and columns statically. The container will be divided into a table of rows and columns.

Costructor:

GridLayout gl = new GridLayout(int rows, int cols);

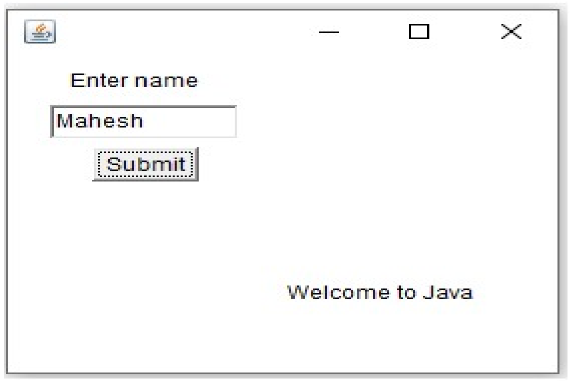
GridLayout gl = new GridLayout(int rows, int cols, int vgap, int hgap);

For Example: Java Program to demonstrate Grid Layout Manager.

```
 import java.awt.*;
import javax.swing.*;
public class GridLayoutDemo
{
   public static void main (String[]args)
   {
      Frame f1 = new Frame ();
      f1.setSize (250, 250);
      GridLayout ob = new GridLayout (2, 2);
      f1.setLayout (ob);
      Panel p1 = new Panel ();
      Label l1 = new Label ("Enter name");
      TextField tf = new TextField (10);
      Button b1 = new Button ("Submit");
      p1.add (l1);
      p1.add (tf);
      p1.add (b1);
      f1.add (p1);
      Panel p2 = new Panel ();
      f1.add (p2);
```

```
        Panel p3 = new Panel ();
        f1.add (p3);
        Label l2 = new Label ("Welcome to Java");
        f1.add (l2);
        f1.setVisible (true);
    }
}
```

Output:



5. Grid Bag Layout: This layout is the most efficient layout that can be used for displaying components by specifing the location, the size, etc.

Costructor:

    GridBagLayout gbl = new GridBagLayout();

Instance variables to manipulate the GridBagLayout object Constraints are:

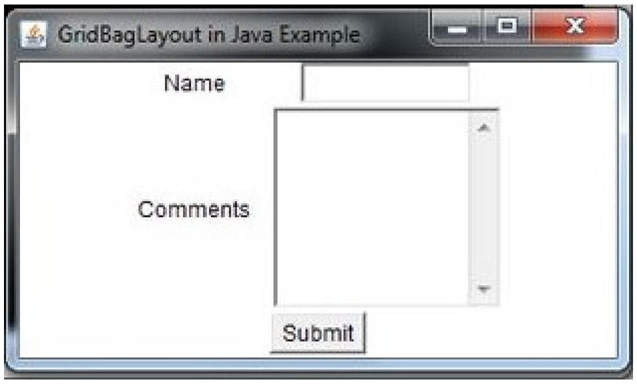
Variable	Role
gridx and gridy	These contain the coordinates of the origin of the grid. They allow a at a specific position of a component positioning. By default, they have GrigBagConstraint. RELATIVE value which indicates that a component can be stored to the right of previous
gridwidth, gridheight	Define how many cells will occupy component (height and width). by The default is 1. The indication is relative to the other components of the line or the column. The GridBagConstraints.REMAINDER value specifies that the next component inserted will be the last of the line or the current column. the value GridBagConstraints. RELATIVE up the component after the last component of a row or column.
fill	Defines the fate of a component smaller than the grid cell. GridBagConstraints. NONE retains the original size: Default GridBagConstraints. HORIZONTAL expanded horizontally GridBagConstraints.VERTICAL GridBagConstraints.BOTH expanded vertic ally expanded to the dimensions of the cell
ipadx, ipady	Used to define the horizontal and vertical expansion f components. not works if expansion is required by fill. The default value is (0,0).

anchor	When a component is smaller than the cell in which it is inserted, it can be positioned using this variable to define the side from which the control should be aligned within the cell. Possible variables NORTH, NORTHWEST, NORTHEAST, SOUTH, SOUTHWEST, SOUTHEAST, WEST and EAST
weightx, weighty	Used to define the distribution of space in case of change of dimension

For Example: Java Program to demonstrate GridBag Layout Manager.

```
import java.awt.*;
 class GridBagLayoutExample extends Frame
 {
     GridBagLayoutExample()
     {
        Label lblName = new Label("Name");
        TextField txtName =new TextField(10);
        Label lblcomments = new Label("Comments");
        TextArea TAreaComments=new TextArea(6,15);
        Button btnSubmit = new Button("Submit");
        setLayout(new GridBagLayout());
        GridBagConstraints gc =new GridBagConstraints();
        add(lblName,gc,0,0,1,1,0,0);
        add(txtName,gc,1,0,1,1,0,20);
        add(lblcomments,gc,0,1,1,1,0,0);
        add(TAreaComments,gc,1,1,1,1,0,60);
        add(btnSubmit,gc,0,2,2,1,0,20);
     }
    void add(Component comp,GridBagConstraints gc,int x,int y,int w,int h,int wx,int wy)
     {
        gc.gridx = x;
        gc.gridy = y;
        gc.gridwidth = w;
        gc.gridheight= h;
        gc.weightx = wx;
        gc.weighty = wy;
        add(comp,gc);
     }
 }
   class GridBagLayoutJavaExample
   {
       public static void main(String args[])
       {
            GridBagLayoutExample frame = new GridBagLayoutExample();
            frame.setTitle("GridBagLayout in Java Example");
            frame.setSize(300,200);
            frame.setVisible(true);
        }
    }
```

Output:



Container in Swing:

1. **JFrame** – This is a top-level container which can hold components and containers like panels.

Constructors:

JFrame()
JFrame(String title)

Methods:

Method	Description
setSize(int width, int height)	Specifies size of the frame in pixels
setLocation(int x, int y)	Specifies upper left corner
setVisible(boolean visible)	Set true to display the frame
setTitle(String title)	Sets the frame title
setDefaultCloseOperation(int mode)	Specifies the operation when frame is closed. The modes are: JFrame.EXIT_ON_CLOSE JFrame.DO_NOTHING_ON_CLOSE JFrame.HIDE_ON_CLOSE JFrame.DISPOSE_ON_CLOSE
pack()	Sets frame size to minimum size required to hold components

2. <u>JPanel</u> – This is a middle-level container which can hold components and can be added to other containers like frame and panels.

Constructors:

public javax.swing.JPanel(java.awt.LayoutManager, boolean);
public javax.swing.JPanel(java.awt.LayoutManager);
public javax.swing.JPanel(boolean);
public javax.swing.JPanel();

<u>Component used in Swing:</u>

1. Label : With the JLabel class, you can display unselectable text and images.

Constructors-

JLabel(Icon i)	JLabel(Icon I , int n)
JLabel(String s)	JLabel(String s, Icon i, int n)
JLabel(String s, int n)	JLabel()

The int argument specifies the horizontal alignment of the label's contents within its drawing area; defined in the SwingConstants interface (which JLabel implements): LEFT (default), CENTER, RIGHT, LEADING, or TRAILING.

Methods-

Method	Description
void setText(String), String getText()	Set or get the text displayed by the label
void setIcon(Icon) , Icon getIcon()	Set or get the image displayed by the label.
void setDisabledIcon(Icon) Icon getDisabledIcon()	Set or get the image displayed by the label when it's disabled. If you don't specify a disabled image, then the look-and-feel creates one by manipulating the default image.
void setHorizontalAlignment(int) void setVerticalAlignment(int) int getHorizontalAlignment() int getVerticalAlignment()	Set or get where in the label its contents should be placed. For vertical alignment: TOP, CENTER (the default), and BOTTOM.

2. **Button:** A Swing button can display both text and an image. The underlined letter in each button's text shows the mnemonic which is the keyboard alternative.

Constructors-

JButton(Icon I)
JButton(String s)
JButton(String s, Icon I)

Methods-

void setDisabledIcon(Icon)	void setPressedIcon(Icon)
void setSelectedIcon(Icon)	void setRolloverIcon(Icon)
String getText()	void setText(String)

Event- ActionEvent

3. Check boxes : This component allows the user to select multiple items from a group of items. It is used to create a CheckBox. It is used to turn an option ON or OFF.

Constructors-

JCheckBox(Icon i)
JCheckBox(Icon i,booean state)
JCheckBox(String s)
JCheckBox(String s, boolean state)
JCheckBox(String s, Icon i)
JCheckBox(String s, Icon I, boolean state)

Methods-

void setSelected(boolean state)	String getText()
void setSelected(boolean state)	String getText()
void setText(String s)	

Event- ItemEvent

4. Radio Buttons: This component allows the user to select only one item from a group item. By using the JRadioButton component you can choose one option from multiple options.

Constructors-

JRadioButton(): It is used to create an unselected radio button with no text.
JRadioButton(Label): It is used to create an unselected radio button with specified text.
JRadioButton(Label, boolean): It is used to create a radio button with the specified text and selected status.

<u>Creating a ButtonGroup:</u>

ButtonGroup(): This class is used to place multiple RadioButton into a single group. So the user can select only one value from that group. We can add RadioButtons to the ButtonGroup by using the add method.

Constructor: ButtonGroup bg = new ButtonGroup();

Methods:

void add(AbstractButton)	Adds a button to the group
void remove(AbstractButton)	Removes a button from the group.

Example:

```
 JRadioButton jrb = new JRadioButton();
ButtonGroup bg = new ButtonGroup();
add(jrb);
```

5. Combo Boxes: This component will display a group of items as a drop-down menu from which one item can be selected. At the top of the menu the choice selected by the user is shown. It basically inherits JComponent class. We can add the items to the ComboBox by using the addItem() method.

Constructor:

JComboBox(): It is used to create a JComboBox with a default data model.

JComboBox(Object[] items): It is used to create a JComboBox that contains the elements in the specified array.

JComboBox(Vector<?> items): It is used to create a JComboBox that contains the elements in the specified Vector.

Methods:

void addItem(Object)	int getItemCount()
Object getItemAt(int)	Object getSelectedItem()

Event- ItemEvent

6. List: The object of List class represents a list of text items. With the help of the List class, user can choose either one item or multiple items. It inherits the Component class.

Constructor- JList(ListModel)

List models-

1. SINGLE_SELECTION - Only one item can be selected at a time. When the user selects an item, any previously selected item is deselected first.
2. SINGLE_INTERVAL_SELECTION- Multiple, contiguous items can be selected. When the user begins a new selection range, any previously selected items are deselected first.
3. MULTIPLE_INTERVAL_SELECTION- The default. Any combination of items can be selected. The user must explicitly deselect items.

Methods-

boolean isSelectedIndex(int)	void setSelectedIndex(int)
void setSelectedIndices(int[])	void setSelectedValue(Object, boolean)
void setSelectedInterval(int, int)	int getSelectedIndex()
int getMinSelectionIndex()	int getMaxSelectionIndex()
int[] getSelectedIndices()	Object[] getSelectedValues()

Example-

```
listModel = new
DefaultListModel();
listModel.addElement("India");
listModel.addElement("Japan");
listModel.addElement("France");
listModel.addElement("Denmark");
list = new JList(listModel);
```

Event- ActionEvent

7. Text Classes: All text related classes are inherited from JTextComponent class

1. JTextField: The JTextField component allows the user to type some text in a single line. It basically inherits the JTextComponent class.

Constructors-

- ✓ JTextField(): It is used to create a new Text Field.
- ✓ JTextField(String text): It is used to create a new Text Field initialized with the specified text.
- ✓ JTextField(String text, int columns): It is used to create a new Text field initialized with the specified text and columns.
- ✓ JTextField(int columns): It is used to create a new empty TextField with the specified number of columns.

Example:

```
JTextField jtf = new JTextField();
JTextField jtf= new JTextField (20);
```

2. JPasswordField : Creates a password field. When present, the int argument specifies the desired width in columns. The String argument contains the field's initial text. The Document argument provides a custom document for the field.

Constructors-

- ✓ JPasswordField(): It is used to construct a new JPasswordField, with a default document, null starting text string, and column width.
- ✓ JPasswordField(int columns): It is used to construct a new empty JPasswordField with the specified number of columns.
- ✓ JPasswordField(String text): It is used to construct a new JPasswordField initialized with the specified text.

Example:

```
JPasswordField jpf = new JPasswordField();
```

Methods:

void setText(String), String getText()	Set or get the text displayed by the text field.
char[] getPassword()	Set or get the text displayed by the text field.

void setEditable(boolean), boolean isEditable()	Set or get whether the user can edit the text in the text field.
void setColumns(int); int getColumns()	Set or get the number of columns displayed by the text field. This is really just a hint for computing the field's preferred width.
int getColumnWidth()	Get the width of the text field's columns. This value is established implicitly by the font.
void setEchoChar(char), char getEchoChar()	Set or get the echo character i.e. the character displayed instead of the actual characters typed by the user.

Event- ActionEvent

3. **JTextArea :** Represents a text area which can hold multiple lines of text

Constructors-

JTextArea (int row, int cols)
JTextArea (String s, int row, int cols)

Methods-

void setColumns (int cols)	void setRows (int rows)
void append(String s)	void setLineWrap (boolean)

Example:

JTextArea jta = new JTextArea();
JTextArea jta = new JTextArea (3, 20);

8. Dialog Boxes:

Types-

1. **Modal**- won't let the user interact with the remaining windows of application until first deals with it. Ex- when user wants to read a file, user must specify file name before prg. can begin read operation.
2. **Modeless dialog box**- Lets the user enters information in both, the dialog box & remainder of application ex- toolbar.

Swing has a JOptionPane class, that lets you put a simple dialog box. Methods in JOption Class

1. **static void showMessageDialog()**- Shows a message with ok button.
2. **static int showConfirmDialog()**- shows a message & gets users options from set of options.
3. **static int showOptionDialog**- shows a message & get users options from set of options.
4. **String showInputDialog()**- shows a message with one line of user input.

9. Menu: Following table gives idea about Menu componenet

Creating and Setting Up Menu Bars	
Constructor or Method	Description
JMenuBar()	Creates a menu bar.
JMenu add(JMenu)	Creates a menu bar.
void setJMenuBar(JMenuBar) JMenuBar getJMenuBar()	Sets or gets the menu bar of an applet, dialog, frame, internal frame, or root pane.
Creating and Populating Menus	
JMenu() JMenu(String)	Creates a menu. The string specifies the text to display for the menu.
JMenuItem add(JMenuItem) JMenuItem add(Action) JMenuItem add(String)	Adds a menu item to the current end of the menu. If the argument is an Action object, then the menu creates a menu item. If the argument is a string, then the menu automatically creates a JMenuItem object that displays the specified text.
void addSeparator()	Adds a separator to the current end of the menu.
JMenuItem insert(JMenuItem, int) JMenuItem insert(Action, int) void insert(String, int) void insertSeparator(int)	Inserts a menu item or separator into the menu at the specified position. The first menu item is at position 0, the second at position 1, and so on. The JMenuItem, Action, and String arguments are treated the same as in the corresponding add methods.
void remove(JMenuItem) void remove(int) void removeAll()	Removes the specified item(s) from the menu. If the argument is an integer, then it specifies the position of the menu item to be removed.
Implementing Menu Items	
JMenuItem() JMenuItem(String) JMenuItem(Icon) JMenuItem(String, Icon) JMenuItem(String, int)	Creates an ordinary menu item. The icon argument, if present, specifies the icon that the menu item should display. Similarly, the string argument specifies the text that the menu item should display. The integer argument specifies the keyboard mnemonic to use. You can specify any of the relevant VK constants defined in the KeyEvent class. For example, to specify the A key, use KeyEvent.VK_A.
JCheckBoxMenuItem() JCheckBoxMenuItem(String) JCheckBoxMenuItem(Icon) JCheckBoxMenuItem(String, Icon) JCheckBoxMenuItem(String,	Creates a menu item that looks and acts like a check box. The string argument, if any, specifies the text that the menu item should display. If you specify true for the boolean argument, then the menu item is initially selected (checked). Otherwise, the menu item is initially unselected.

boolean) JCheckBoxMenuItem(String, Icon, boolean)	

JRadioButtonMenuItem()
JRadioButtonMenuItem(String)
JRadioButtonMenuItem(Icon)
JRadioButtonMenuItem(String, Icon)
JRadioButtonMenuItem(String, boolean)
JRadioButtonMenuItem(Icon, boolean)
JRadioButtonMenuItem(String, Icon, boolean)
	Creates a menu item that looks and acts like a radio button. The string argument, if any, specifies the text that the menu item should display. If you specify true for the boolean argument, then the menu item is initially selected. Otherwise, the menu item is initially unselected.

void setState(boolean) boolean getState() (in JCheckBoxMenuItem)	Set or get the selection state of a check box menu item.
void setEnabled(boolean)	If the argument is true, enable the menu item. Otherwise, disable the menu item.

Example: Java JMenuItem and JMenu

```
import javax.swing.*;
class MenuExample
{       JMenu menu, submenu;
        JMenuItem i1, i2, i3, i4, i5;
        MenuExample()
        {
           JFrame f= new JFrame("Menu and MenuItem Example");
          JMenuBar mb=new JMenuBar();
          menu=new JMenu("Menu");
          submenu=new JMenu("Sub Menu");
           i1=new JMenuItem("Item 1");
           i2=new JMenuItem("Item 2");
           i3=new JMenuItem("Item 3");
           i4=new JMenuItem("Item 4");
           i5=new JMenuItem("Item 5");
          menu.add(i1); menu.add(i2); menu.add(i3);
          submenu.add(i4); submenu.add(i5);
          menu.add(submenu);
          mb.add(menu);
          f.setJMenuBar(mb);
```

```
                f.setSize(400,400);
                f.setLayout(null);
                f.setVisible(true);
            }
             public static void main(String args[])
            {
                new MenuExample();
            }
    }
```

10. **JTable Class:** he JTable class is used to display data in tabular form. It is composed of rows and columns.

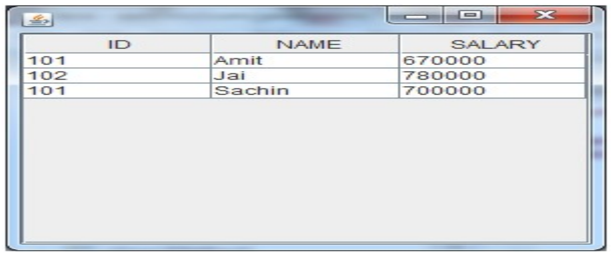
Constructors:

JTable():Creates a table with empty cells.
JTable(Object[][] rows, Object[] columns): Creates a table with the specified data

Example:

```
import javax.swing.*;
public class TableExample
 {
        JFrame f;
        TableExample()
         {
                f=new JFrame();
                String data[][]={ {"101","Amit","670000"},
                                  {"102","Jai","780000"},
                                   {"101","Sachin","700000"}
                                };
                    String column[]={"ID","NAME","SALARY"};
                     JTable jt=new JTable(data,column);
                     jt.setBounds(30,40,200,300);
                    JScrollPane sp=new JScrollPane(jt);
                    f.add(sp);
                    f.setSize(300,400);
                    f.setVisible(true);
        }
        public static void main(String[] args)
        {
                new TableExample();
        }
}
```

Output:



ID	NAME	SALARY
101	Amit	670000
102	Jai	780000
101	Sachin	700000

Event Handling: Event handling is an important part of GUI based applications. Events are generated by event sources. A mouse click, Window closed, key typed etc. are examples of events. All java events are sub-classes of java.awt.AWTEvent class.

Java has two types of events:
1. Low-Level Events: Low-level events represent direct communication from user. A low level event is a key press or a key release, a mouse click, drag, move or release, and so on. Following are low level events.

Event	Description
ComponentEvent	Indicates that a component object (e.g. Button, List, TextField) is moved, resized, rendered invisible or made visible again.
FocusEvent	Indicates that a component has gained or lost the input focus.
KeyEvent	Generated by a component object (such as TextField) when a key is pressed, released or typed.
MouseEvent	Indicates that a mouse action occurred in a component. E.g. mouse is pressed, releases, clicked (pressed and released), moved or dragged.
ContainerEvent	Indicates that a container's contents are changed because a component was added or removed.
WindowEvent	Indicates that a window has changed its status. This low level event is generated by a Window object when it is opened, closed, activated, deactivated, iconified, deiconified or when focus is transferred into or out of the Window.

2. **High-Level Events:** High-level (also called as semantic events) events encapsulate the meaning of a user interface component. These include following events.

Event	Description
ActionEvent	Indicates that a component-defined action occurred. This high-level event is generated by a component (such as Button) when the component-specific action

	occurs (such as being pressed).
AdjustmentEvent	The adjustment event is emitted by Adjustable objects like scrollbars.
ItemEvent	Indicates that an item was selected or deselected. This high-level event is generated by an ItemSelectable object (such as a List) when an item is selected or deselected by the user.
TextEvent	Indicates that an object's text changed. This high-level event is generated by an object (such as TextComponent) when its text changes.

The following table lists the events, their corresponding listeners and the method to add the listener to the component.

Event	**Event Source**	**Event Listener**	**Method to add listener to event source**
		Low-level Events	
ComponentEvent	Component	ComponentListener	addComponentListener()
FocusEvent	Component	FocusListener	addFocusListener()
KeyEvent	Component	KeyListener	addKeyListener()

MouseEvent	Component	MouseListener
		MouseMotionListener
			addMouseListener()
			addMouseMotionListener()

ContainerEvent	Container	ContainerListener	addContainerListener()
WindowEvent	Window	WindowListener	addWindowListener()
		High-level Events	
ActionEvent	Button List MenuItem TextField	ActionListener	addActionListener()
ItemEvent	Choice CheckBox CheckBoxMenuItem List	ItemListener	addItemListener()
AdjustmentEvent	Scrollbar	AdjustmentListener	addAdjustmentListener()

TextEvent	TextField TextArea	TextListener	addTextLIstener()

Listener Methods:

Methods	Description
ComponentListener	
componentResized(ComponentEvent e)	Invoked when component's size changes.
componentMoved(ComponentEvent e)	Invoked when component's position changes.
componentShown(ComponentEvent e)	Invoked when component has been made visible.
componentHidden(ComponentEvent e)	Invoked when component has been made invisible.
FocusListener	
focusGained(FocusEvent e)	Invoked when component gains the keyboard focus.
focusLost(FocusEvent e)	Invoked when component loses the keyboard focus.
KeyListener	
keyTyped(KeyEvent e)	Invoked when a key is typed.
keyPressed(KeyEvent e)	Invoked when a key is pressed.
keyReleased(KeyEvent e)	Invoked when a key is released.
MouseListener	
mouseClicked(MouseEvent e)	Invoked when a mouse button is clicked (i.e. pressed and released) on a component.
mousePressed(MouseEvent e)	Invoked when a mouse button is pressed on a component.
mouseReleased(MouseEvent e)	Invoked when a mouse button is released on a component.
mouseEntered(MouseEvent e)	Invoked when a mouse enters a component.
mouseExited(MouseEvent e)	Invoked when a mouse exits a component.
MouseMotionListener	

mouseDragged(MouseEvent e)	Invoked when a mouse button is pressed on a component and then dragged.
mouseMoved(MouseEvent e)	Invoked when a the mouse cursor is moved on to a component but mouse button is not pressed.
ContainerListener	
componentAdded(ContainerEvent e)	Invoked when a component is added to the container.
componentRemoved(ContainerEvent e)	Invoked when a component is removed from the container.
WindowListener	
windowOpened(WindowEvent e)	Invoked the first time a window is made visible
windowClosing(WindowEvent e)	Invoked when the user attempts to close the window from the window's system menu.
windowClosed(WindowEvent e)	Invoked when a window has been closed as the result of calling dispose on the window.
windowIconified(WindowEvent e)	Invoked when a window is changed from a normal to a minimized state.
windowDeiconified(WindowEvent e)	Invoked when a window is changed from minimized to normal state.
windowActivated(WindowEvent e)	Invoked when the window is set to be the active window.
windowDeactivated(WindowEvent e)	Invoked when the window is no longer the active window.
ActionListener	
actionPerformed(ActionEvent e)	Invoked when an action occurs.
ComponentListsner	
itemStateChanged(ActionEvent e)	Invoked when anitem has been selected oe deselected by the user.
AdjustmentListener	
adjustmentValueChanged(ActionEvent	Invoked when the value of the adjustable has changed.

e)	
TextListener	
textValueChanged(ActionEvent e)	Invoked when the value of the text has changed.

Adapter Classes: All high level listeners contain only one method to handle the high-level events. But most low level event listeners are designed to listen to multiple event subtypes (i.e. the MouseListener listens to mouse-down, mouse-up, mouse-enter, etc.). AWT provides a set of abstract "adapter" classes, which implements each listener interface. These allow programs to easily subclass the Adapters and override only the methods representing event types they are interested in, instead of implementing all methods in listener interfaces.

The Adapter classes provided by AWT are as follows:
java.awt.event.ComponenentAdapter
java.awt.event.ContainerAdapter
java.awt.event.FocusAdapter
java.awt.event.KeyAdapter
java.awt.event.MouseAdapter
java.awt.event.MouseMotionAdapter java.awt.event.WindowAdapter

Example: Program to close window

```
// importing the necessary libraries
import java.awt.*;
import java.awt.event.*;

public class AdapterExample
{
   Frame f;    // object of Frame
  // class constructor
   AdapterExample()
   {
         // creating a frame with the title
           f = new Frame ("Window Adapter");
         // adding the WindowListener to the frame overriding the windowClosing () method
          f.addWindowListener (new WindowAdapter()
          {
                    public void windowClosing (WindowEvent e)
                    {
                             f.dispose();
                    }
```

```
            });
         // setting the size, layout and
        f.setSize (400, 400);
        f.setLayout (null);
        f.setVisible (true);
    }

public static void main(String[] args)
 {
    new AdapterExample();
}
}
```

Example 1: Sample Program to understand JLabel Swing Control in Java:

```
import javax.swing.*;
import java.awt.*;

public class JLabelDemo extends JFrame
{
    JLabel jl;
    JLabelDemo ()
    {
        jl = new JLabel ("Good Morning");
        Container c = this.getContentPane ();
        c.setLayout (new FlowLayout ());
        c.setBackground (Color.blue);
        Font f = new Font ("arial", Font.BOLD, 34);
        jl.setFont (f);
        jl.setBackground (Color.white);
        c.add (jl);
        this.setVisible (true);
        this.setSize (400, 400);
        this.setTitle ("Label");
        this.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
    }
    public static void main (String[]args)
    {
        new JLabelDemo ();
    }
}
```

Example 2: Java Program to understand the above-discussed Swing Controls.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class SwingDemo2 extends JFrame implements ActionListener
{
   JRadioButton eng, doc;
   ButtonGroup bg;
   JTextField jtf;
   JCheckBox bcd, ccb, acb;
   JTextArea jta;
   SwingDemo2 ()
   {
      eng = new JRadioButton ("Engineer");
      doc = new JRadioButton ("Doctor");
      bg = new ButtonGroup ();
      bg.add (eng);
      bg.add (doc);
      jtf = new JTextField (20);
      bcd = new JCheckBox ("Bike");
      ccb = new JCheckBox ("Car");
      acb = new JCheckBox ("Aeroplane");
      jta = new JTextArea (3, 20);
      Container c = this.getContentPane ();
      c.setLayout (new FlowLayout ());

      // Registering the listeners with the components
      eng.addActionListener (this);
      doc.addActionListener (this);
      bcd.addActionListener (this);
      ccb.addActionListener (this);
      acb.addActionListener (this);
      c.add (eng);
      c.add (doc);
      c.add (jtf);
      c.add (bcd);
      c.add (ccb);
      c.add (acb);
      c.add (jta);
      this.setVisible (true);
      this.setSize (500, 500);
      this.setTitle ("Selection Example");
```

```
        this.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
    }
    public void actionPerformed (ActionEvent ae)
    {
        if (ae.getSource () == eng)
        {
              jtf.setText ("You are an Engineer");
        }
        if (ae.getSource () == doc)
        {
            jtf.setText ("You are an Doctor");
        }
        String str = " ";
        if (bcd.isSelected ())
        {
               str += "Bike\n";
        }
        if (ccb.isSelected ())
        {
              str += "Car\n";
        }
        if (acb.isSelected ())
        {
         str += "Aeroplane\n";
        }
        jta.setText (str);
    }
    public static void main (String[]args)
    {
        new SwingDemo2 ();
    }
}
```

Example 3: Java Program to understand JButton and Border controls.

```
import javax.swing.*;
import java.awt.*;
public class JButtonDemo extends JFrame
{
    private JButton button[];
    private JPanel panel;
    public JButtonDemo ()
    {
        setTitle ("JButton Borders");
        panel = new JPanel ();
        panel.setLayout (new GridLayout (7, 1));
        button = new JButton[7];
        for (int count = 0; count < button.length; count++)
        {
                button[count] = new JButton ("Button " + (count + 1));
                panel.add (button[count]);
        }
        button[0].setBorder (BorderFactory.createLineBorder (Color.blue));
        button[1].setBorder (BorderFactory.createBevelBorder (0));
        button[2].setBorder (BorderFactory.createBevelBorder (1, Color.red, Color.blue));
        button[3].setBorder (BorderFactory.createBevelBorder (1, Color.green,
        Color.orange,Color.red, Color.blue));
        button[4].setBorder (BorderFactory.createEmptyBorder (10, 10, 10, 10));
        button[5].setBorder (BorderFactory.createEtchedBorder (0));
        button[6].setBorder (BorderFactory.createTitledBorder ("Titled Border"));
        add (panel, BorderLayout.CENTER);
        setSize (400, 300);
        setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo (null);
        setVisible (true);
    }
    public static void main (String[]args)
    {
        new JButtonDemo ();
    }
}
```

Example 4: Sample Program to understand JComboBox control in Java

```
import javax.swing.*;
public class JComboBoxDemo
{
    JFrame f;
    JComboBoxDemo ()
    {
        f = new JFrame ("ComboBox Example");
        String country[] ={ "Hyderabad", "Chennai", "Bengaluru", "Mumbai", "Delhi" };
        JComboBox cb = new JComboBox (country);
        cb.setBounds (50, 50, 90, 20);
        f.add (cb);
        f.setLayout (null);
        f.setSize (400, 500);
        f.setVisible (true);
    }
    public static void main (String[]args)
    {
        new JComboBoxDemo ();
    }
}
```

Example 5: Sample Program to understand JTabbedPane control in Java

```
import javax.swing.*;
import java.awt.*;
public class JTabbedPaneDemo
{
    public static void main (String args[])
    {
        JFrame frame = new JFrame ("Technologies");
        JTabbedPane tabbedPane = new JTabbedPane ();
        JPanel panel1, panel2, panel3, panel4, panel5;
        panel1 = new JPanel ();
        panel2 = new JPanel ();
        panel3 = new JPanel ();
        panel4 = new JPanel ();
        panel5 = new JPanel ();
        tabbedPane.addTab ("Cricket", panel1);
        tabbedPane.addTab ("Badminton ", panel2);
        tabbedPane.addTab ("Football", panel3);
        tabbedPane.addTab ("Basketball ", panel4);
        tabbedPane.addTab ("Tennis", panel5);
        frame.add (tabbedPane);
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.setSize (550, 350);
        frame.setVisible (true);
    }
}
```

Example 6: Sample Program to understand JPasswordFiled Swing control in Java

```
import javax.swing.*;
public class JPasswordFieldDemo
{
    public static void main (String[]args)
    {
        JFrame f = new JFrame ("Password Field Example");
        JPasswordField value = new JPasswordField ();
        JLabel l1 = new JLabel ("Password:");
        l1.setBounds (20, 100, 80, 30);
        value.setBounds (100, 100, 100, 30);
        f.add (value);
        f.add (l1);
        f.setSize (300, 300);
        f.setLayout (null);
        f.setVisible (true);
    }
}
```

Practice Set:

1. Develop an applet that draws a circle. The dimension of the applet should be 500 * 300 pixels the circle should be centered the applet and have a radius of 100 pixels. Display your name centered in a circle(using drawOval() method)
2. Write a program to create a frame using AWT. Implement mouseClicked(), mouseEntered() and mouseExited() events. Frame should become visible when mouse enters it.
3. Write a program to display a string in frame window with pink colour as background.
4. Write a program to create two buttons named “Red” and “Blue”. When a button is pressed the background colour should be set to the colour named by the button’s label.
5. Write a program which responds to KEY_TYPED event and updates the status window with message (“Typed character is: X”). Use adapter class for other two events.
6. Write a program to create two buttons labeled ‘GetInfo’ and ‘GetCGPA’. When button ‘GetInfo’ is pressed, it displays your personal information (Name, Course, Roll No, College) and when button ‘GetCGPA’ is pressed, it displays your CGPA in previous semester.

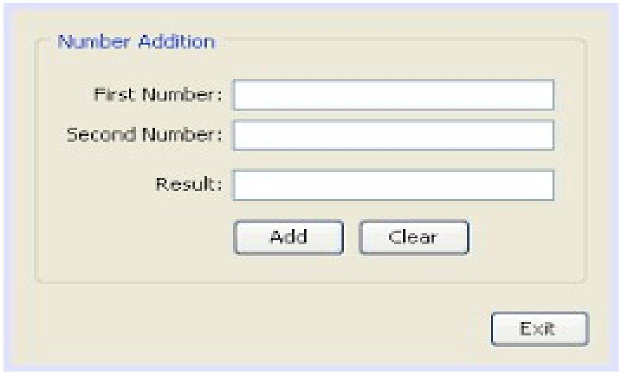
Set A:

1. Write a program that asks the user's name, and then greets the user by name. Before outputting the user's name, convert it to upper case letters. For example, if the user's name is Raj, then the program should respond "Hello, RAJ, nice to meet you!".
2. Write a program that reads one line of input text and breaks it up into words. The words should be output one per line. A word is defined to be a sequence of letters. Any characters in the input that are not letters should be discarded. For example, if the user inputs the line He said, "That's not a good idea." then the output of the program should be

```
He
said
thats
not
```

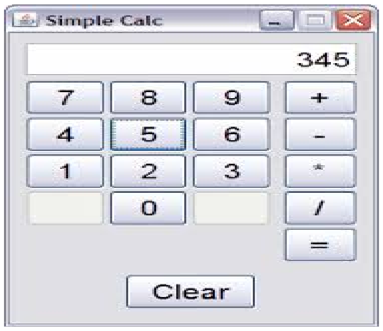
a
good
idea

3. Write a program that will read a sequence of positive real numbers entered by the user and will print the same numbers in sorted order from smallest to largest. The user will input a zero to mark the end of the input. Assume that at most 100 positive numbers will be entered.
4. Create an Applet that displays the x and y position of the cursor movement using Mouse and Keyboard. (Use appropriate listener).
5. Create the following GUI screen using appropriate layout managers.

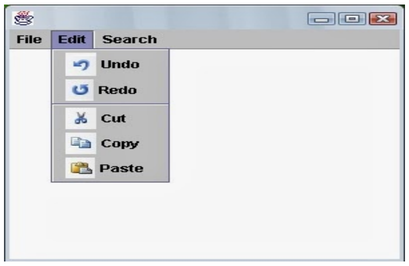


Set B:

1. Write a java program to implement a simple arithmetic calculator. Perform appropriate validations.



2. Write a java program to implement following. Program should handle appropriate events.

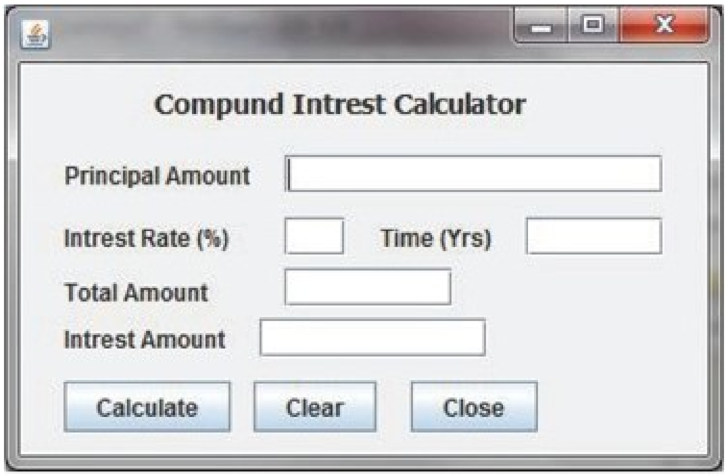


3. Write an applet application to draw Temple.

4. Write an applet application to display Table lamp. The color of lamp should get change in random color.
5. Write a java program to design email registration form.(Use maximum Swing component in form).

Set C:

1. Write a java program to accept the details of employee employee eno,ename, sal and display it on next frame using appropriate even .
2. Write a java program to display at least five records of employee in JTable.(Eno, Ename ,Sal).
3. Write a java Program to change the color of frame. If user clicks on close button then the position of frame should get change.
4. Write a java program to display following screen.



5. Write an applet application to display smiley and sad face.

Assignment Evaluation:

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: WellDone [ ]

Signature of Instructor

Section-II

MongoDB

Assignment No. 1: MongoDB Basics

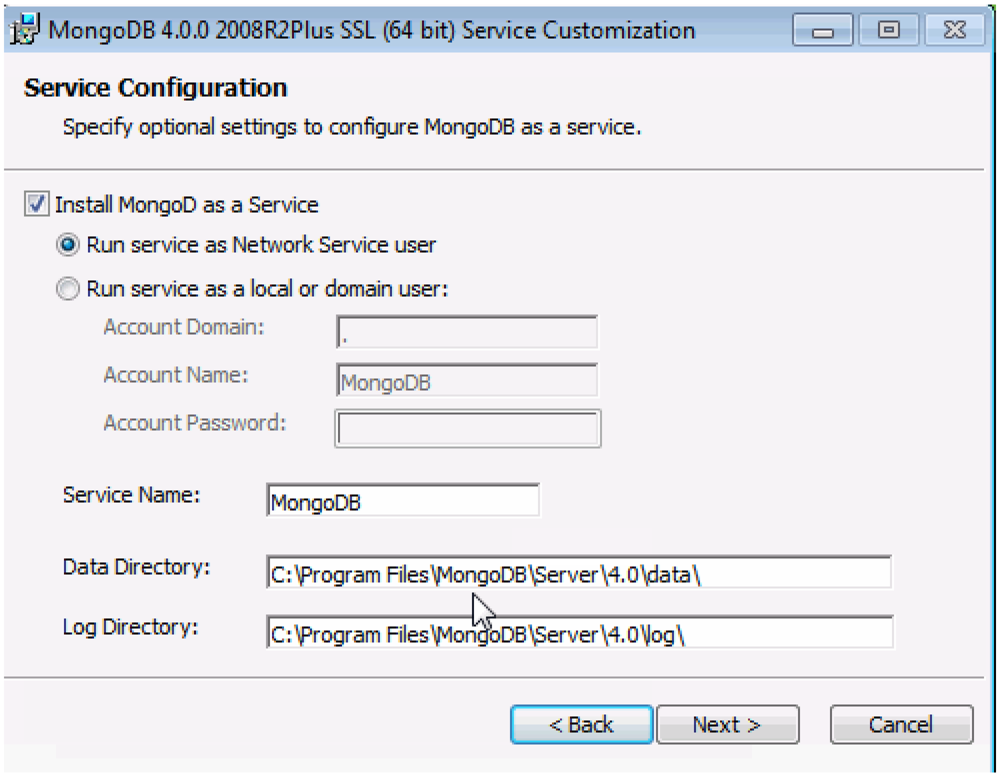
MongoDB Installation Steps:
Follow these steps to install MongoDB Community Edition using the MongoDB Installer wizard.
Download the installer.
Download the MongoDB Community .msi installer from the following link:
➢ MongoDB Download Center
(https://www.mongodb.com/try/download/community?tck=docs_server)
Run the MongoDB installer.
For example, from the Windows Explorer/File Explorer:

a. Go to the directory where you downloaded the MongoDB installer (.msi file). By default, this is your Downloads directory.
b. Double-click the .msi file.

Follow the MongoDB Community Edition installation wizard.
The wizard steps you through the installation of MongoDB and MongoDB Compass.

a. Choose Setup Type
You can choose either the **Complete** (recommended for most users) or **Custom** setup type. The **Complete** setup option installs MongoDB and the MongoDB tools to the default location. The **Custom** setup option allows you to specify which executables are installed and where.

b. Service Configuration
Starting in MongoDB 4.0(or current version), you can set up MongoDB as a Windows service during the install or just install the binaries.
MongoDB ServiceMongoDB
The following installs and configures MongoDB as a Windows service.
Starting in MongoDB 4.0(or current version), you can configure and start MongoDB as a Windows service during the install, and the MongoDB service is started upon successful installation.


MongoDB 4.0.0 2008R2Plus SSL (64 bit) Service Customization

Service Configuration

Specify optional settings to configure MongoDB as a service.

☑ Install MongoD as a Service

● Run service as Network Service user

○ Run service as a local or domain user:

Account Domain: .

Account Name: MongoDB

Account Password:

Service Name: MongoDB

Data Directory: C:\Program Files\MongoDB\Server\4.0\data\

Log Directory: C:\Program Files\MongoDB\Server\4.0\log\

- Select **Install MongoD as a Service** MongoDB as a service.
- Select either:
  - **Run the service as Network Service user** (Default)
    This is a Windows user account that is built-in to Windows
    or
  - **Run the service as a local or domain user**
    - For an existing local user account, specify a period (i.e. .) for the **Account Domain** and specify the **Account Name** and the **Account Password** for the user.
    - For an existing domain user, specify the **Account Domain**, the **Account Name** and the **Account Password** for that user.
- **Service Name**. Specify the service name. Default name is MongoDB. If you already have a service with the specified name, you must choose another name.
- **Data Directory**. Specify the data directory, which corresponds to the --dbpath. If the directory does not exist, the installer will create the directory and sets the directory access to the service user.
- **Log Directory**. Specify the Log directory, which corresponds to the --logpath. If the directory does not exist, the installer will create the directory and sets the directory access to the service user.

c. **Install MongoDB Compass**
Optional. To have the wizard install MongoDB Compass, select **Install MongoDB Compass** (Default).

d. When ready, click **Install**.

Database, Collection and Documents in MongoDB:
Databases, collections, documents are important parts of MongoDB without them you are not able to store data on the MongoDB server. A Database contains a collection, and a collection contains documents and the documents contain data in the form of key-value pair, they are related to each other.

Database
In MongoDB, a database contains the collections of documents. One can create multiple databases on the MongoDB server.
Command to view database : >show dbs
It gives list of databases that are present in your MongoDB server.
Command to create a database : >use database_name
This command actually switches you to the new database if the given name does not exist and if the given name exists, then it will switch you to the existing database. Now at this stage, if you use the show command to see the database list, you will find that your new database is not present in that database list because, in MongoDB, the database is actually created when you start entering data in that database.
Command to check currently selected database : >db
Command to delete database : >db.dropDatabase()
This command drop/delete a database. If you have not selected any database, then by default it will delete 'test' database. To delete specific database first switch to that database using 'use' command and then give 'dropDatabase' command.

Collection
Collections are just like tables in relational databases, they also store data, but in the form of documents. A single database is allowed to store multiple collections.
As we know that MongoDB databases are schemaless, it is not necessary in a collection that the schema of one document is similar to another document. Or in other words, a single collection contains different types of documents. Create a collection to store documents.
Command to create collection implicitly: >db.collection_name.insertOne({..})
Here, in a single command you can insert a document and create a collection implicitly. insertOne() function is used to store single document in the specified collection. And in the curly braces {} we store our data or document.
Command to create collection explicitly: >db.createCollection("student")
Command to display the collections from database: >show collections
Command to delete a Collection: >db.student.drop()

Document
In MongoDB, the data records are stored as BSON documents. Here, BSON stands for binary representation of JSON(JavaScript Object Notation) documents, although BSON contains more data types as compared to JSON. The document is created using key-value pairs and the value of the field can be of any BSON type.
Command to insert a single document at a time:
>db.collection_name.insertOne({field1: value1, field2: value2, ..., fieldN: valueN})
Command to insert many documents at a time:
>db.collection_name.insertMany([{document 1},{document 2}…{document N}])

Command to query data from MongoDB collection:

```
>db.collection_name.find(query, projection)
```

Here the **first parameter** is **query** which is **optional**. It specifies the query selection criteria using query operators. **Second parameter** is **projection** which is also **optional**, which specifies the fields return using projection operators. In projection parameter {field:1 or true} shows that the field is included in result and {field:0 or false} shows that the field is excluded from result.

Command to display first document of the collection:

```
>db.collection_name.findOne(query, projection)
```

Command to display the documents of the collection in a well-formatted way:

```
>db.collection_name.find().pretty()
```

Example:

Use the mongo command to connect with a MongoDB database

```
C:\Program Files\MongoDB\Server\5.0\bin>mongo
```

Use following command to list out databases that are present in your MongoDB server.

```
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
```

This command switches you to the new database which you want to create.

```
> use mydatabase
switched to db mydatabase
```

Using following command you can check currently selected database. Before creating collection and inserting document make sure that currently selected database is your own database. In MongoDB default database is test.

```
> db
mydatabase
```

After this, if you give a command "show dbs", it will not show your newly created database because in MongoDB the database is actually created when you start entering data in that database.

```
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
```

Use following command to create new collection named "student" under database "mydatabase" and to insert a new document inside it.

```
>db.student.insertOne({"_id":1, "Sname":"Manisha Kadam", "class":"TYBBA_CA",
"Contact_details":{"e-mail": "manisha_kadam_123@gmail.com", "phone":"9876543210"}})
{ "acknowledged" : true, "insertedId" : 1 }
```

Once data will get added inside your database “show dbs” command will list out your newly created database “mydatabase”.

```
> show dbs
admin      0.000GB
config     0.000GB
local      0.000GB
mydatabase  0.000GB
```

Use following command to display the documents of “student” collection in a well-formatted way.

```
> db.student.find().pretty()
{
        "_id" : 1,
        "Sname" : "Manisha Kadam",
        "class" : "TYBBA_CA",
        "Contact_details" : {
                "e-mail" : "manisha_kadam_123@gmail.com",
                "phone" : "9876543210"
        }
}
```

Use following command to display the list of collections present in currently selected database.

```
> show collections
student
```

MongoDB Data modeling:

The main challenge in data modeling is balancing the needs of the application, the performance characteristics of the database engine, and the data retrieval patterns. When designing data models, always consider the application usage of the data (i.e. queries, updates, and processing of the data) as well as the inherent structure of the data itself.

Flexible Schema

Unlike SQL databases, Data in MongoDB has a flexible schema. The documents in the same collection do not need to have the same set of fields or structure. Common fields in a collection’s documents may hold different types of data. To change the structure of the documents in a collection, such as add new fields, remove existing fields, or change the field values to a new type, update the documents to the new structure.

This flexibility facilitates the mapping of documents to an entity or an object. Each document can match the data fields of the represented entity, even if the document has substantial variation from other documents in the collection.

MongoDB Data Model Design

MongoDB provides two types of data models:

i) Embedded data model

ii) Normalized data model.

Based on the requirement, you can use either of the models while preparing your document.

i) Embedded Data Model

In this model, you can have (embed) all the related data in a single document, it is also known as de-normalized data model. Embedding provides better performance for read operations, as well as the ability to request and retrieve related data in a single database operation. Embedded data models make it possible to update related data in a single atomic write operation.

For example, assume we are getting the details of students in three different documents namely, Personal_details, Contact and, Address, you can embed all the three sub documents in a single one as shown below –

```
{
	_id: <ObjectId101>,
	Stud_ID: "S1001",
	Personal_details:{
		First_Name: "Pramod",
		Last_Name: "Mane",
		Date_Of_Birth: "1999-07-17"
	},
	Contact_details: {
		e-mail: "pramod_mane_123@gmail.com",
		phone: "9876543210"
	},
	Address: {
		city: "Pune",
		Area: "Wakad",
		State: "Maharashtra"
	}
}
```

ii) Normalized Data Model

In this model, you can refer the sub documents in the original document, using references. For example, you can re-write the above document in the normalized model as:

```
Employee:
{
	_id: <ObjectId101>,
	Stud_ID: "S1001"
}
Personal_details:
{
	_id: <ObjectId102>,
	studDocID: " ObjectId101",
	First_Name: "Pramod",
	Last_Name: "Mane",
	Date_Of_Birth: "1999-07-17"
}
```

```
Contact_details:
{
        _id: <ObjectId103>,
        studDocID: "ObjectId101",
        e-mail: "pramod_mane_123@gmail.com",
        phone: "9876543210"
}
Address:
{
        _id: <ObjectId104>,
        studDocID: " ObjectId101",
        city: "Pune",
        Area: "Wakad",
        State: "Maharashtra"
}
```

Use Embedded Data Models:

- when you have "contains" relationships between entities.
- when you have one-to-many relationships between entities. In these relationships the "many" or child documents always appear with or are viewed in the context of the "one" or parent documents.

Use Normalized Data Models:

- when embedding would result in duplication of data but would not provide sufficient read performance advantages to outweigh the implications of the duplication.
- to represent more complex many-to-many relationships.
- to model large hierarchical data sets.

Considerations while designing Schema in MongoDB

- Design your schema according to user needs.
- Combine objects into one document if you will use them together. Otherwise separate them, but while doing this make sure there should not be need of joins.
- Duplicate the data but limited, because disk space is cheap as compare to compute time.
- Do joins while write, not on read.
- Optimize your schema for most frequent use cases.
- Do complex aggregation in schema.

Analogy between RDBMS and MongoDB Data Model:

RDBMS	**MongoDB**
Database	Database
Table	Collection
Tuple/Row	Document
Column	Field
Table Join	Embedded Documents
Primary Key	Primary Key (Default Key _id provided by MongoDB itself)

Modeling relationships between documents:

One-to-one Relationship:

Method 1: Embedded Document
In the "student" collection, Contact_details and Class are often required along with the student's information. Hence it is embedded in the student document. Using this method, an application can retrieve the whole student information in single query.

```
>db.student.insertOne({"_id":1, "Sname":"Nitin Bhor", "class":"TYBBA_CA",
"Contact_details":{"e-mail": "nitin_bhor_123@gmail.com", "phone": "9876543210"}})
```

Method 2: Subset Pattern Method
The "student" collection contains basic information about students, whereas "student_details" collection contains less frequently accessed data about a student and it also contains a field "student_id" which is a reference to "_id" field of the "student" collection. This method improves read performance.

```
>db.student.insertOne({"_id":1, "Sname":"Nitin Bhor", "class":"TYBBA_CA",
"Contact_details":{e-mail: "nitin_bhor_123@gmail.com", phone: "9876543210"}})

>db.student_details.insertOne({"_id":101, "student_id":1,"age":20,"city": "Pune"})
```

One-to-Many Relationship:

Method 1: Embedded Document
If the application often requires the contact details of student along with name, then the best choice would be to embed the Contact_details data in student data. Here student has two contact details.

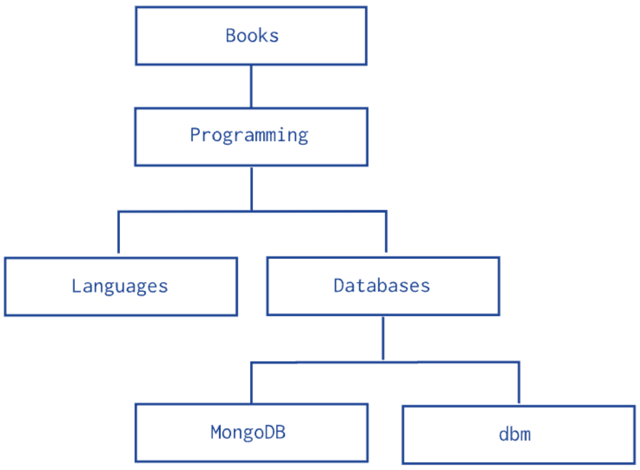
```
>db.student.insertOne({"_id":1, "Sname":"Priya Kale", "class":"TYBBA_CA",
"Contact_details":[{e-mail: "priya_kale_123@gmail.com", phone: "9876543210"},
                         {e-mail: "priyakale@gmail.com", phone: "9876501234"}]})
```

Method 2: Subset Pattern method
Instead of storing all the contact details of a student in a document, you can split the information in two collections, namely student and student_contact. Here a reference to student collection is maintained by adding student_id field in student_contact collection and student has two contact details.

```
>db.student.insertOne({"_id":1, "Sname":"Priya Kale", "class":"TYBBA_CA"})

>db.student_contact.insertOne({"_id":101, "student_id":1, "e-mail":
"priya_kale_123@gmail.com", "phone": "9876543210"})
>db.student_contact.insertOne({"_id":102, "student_id":1, e-mail: "priyakale@gmail.com",
phone: "9876501234"})
```

Modeling tree structures in MongoDB:
MongoDB allows different ways to use tree data structures to model large hierarchical or nested data relationship.

Consider the following hierarchy of categories for all following models:



Model Tree Structures with Parent References
This model describes a tree-like structure in MongoDB documents by storing references to "parent" nodes in children nodes. The *Parent References* pattern stores each tree node in a document; in addition to the tree node, the document stores the id of the node's parent.

The following example models the tree using *Parent References*, storing the reference to the parent category in the **field parent:**
>db.categories.insertMany([
{ _id: **"MongoDB"**, parent: **"Databases"** },
{ _id: **"dbm"**, parent: **"Databases"** },
{ _id: **"Databases"**, parent: **"Programming"** },
{ _id: **"Languages"**, parent: **"Programming"** },
{ _id: **"Programming"**, parent: **"Books"** },
{ _id: **"Books"**, parent: null }
])

- The query to retrieve the parent of a node is fast and straightforward:
  >db.categories.findOne({ _id: **"MongoDB"** }).parent
- You can query by the **parent** field to find its immediate children nodes:
  >db.categories.find({ parent: **"Databases"** })

Model Tree Structures with Child References

This model describes a tree-like structure in MongoDB documents by storing references in the parent-nodes to children nodes. The *Child References* pattern stores each tree node in a document; in addition to the tree node, document stores in an array the id(s) of the node's children.

The following example models the tree using *Child References*, storing the reference to the node's children in the **field children:**

```
>db.categories.insertMany( [
{ _id: "MongoDB", children: [] },
{ _id: "dbm", children: [] },
{ _id: "Databases", children: [ "MongoDB", "dbm" ] },
{ _id: "Languages", children: [] },
{ _id: "Programming", children: [ "Databases", "Languages" ] },
{ _id: "Books", children: [ "Programming" ] }
] )
```

- The query to retrieve the immediate children of a node is fast and straightforward:
  ```
  >db.categories.findOne( { _id: "Databases" } ).children
  ```
- You can query for a node in the **children field** to find its parent node as well as its siblings:
  ```
  >db.categories.find( { children: "MongoDB" } )
  ```

The *Child References* pattern provides a suitable solution to tree storage as long as no operations on subtrees are necessary. This pattern may also provide a suitable solution for storing graphs where a node may have multiple parents.

Model Tree Structures with Array of Ancestors

This model describes a tree-like structure in MongoDB documents using references to parent nodes and an array that stores all ancestors. The *Array of Ancestors* pattern stores each tree node in a document; in addition to the tree node, document stores in an array the id(s) of the node's ancestors or path.

The following example models the tree using *Array of Ancestors*. In addition to the **ancestors field**, these documents also store the reference to the immediate parent category in the **parent field.**

```
>db.categories.insertMany( [
{ _id: "MongoDB", ancestors: [ "Books", "Programming", "Databases" ], parent: "Databases" },
{ _id: "dbm", ancestors: [ "Books", "Programming", "Databases" ], parent: "Databases" },
{ _id: "Databases", ancestors: [ "Books", "Programming" ], parent: "Programming" },
{ _id: "Languages", ancestors: [ "Books", "Programming" ], parent: "Programming" },
{ _id: "Programming", ancestors: [ "Books" ], parent: "Books" },
{ _id: "Books", ancestors: [ ], parent: null }
] )
```

- The query to retrieve the ancestors or path of a node is fast and straightforward:
  ```
  >db.categories.findOne( { _id: "MongoDB" } ).ancestors
  ```
- You can query by the field ancestors to find all its descendants:
  ```
  >db.categories.find( { ancestors: "Programming" } )
  ```

The *Array of Ancestors* pattern provides a fast and efficient solution to find the descendants and the ancestors of a node by creating an index on the elements of the ancestors field. This makes *Array of Ancestors* a good choice for working with subtrees. The *Array of Ancestors* pattern is slightly slower than the Materialized Paths pattern but is more straightforward to use.

Model Tree Structures with Materialized Paths

This model describes a tree-like structure in MongoDB documents by storing full relationship paths between documents.

The *Materialized Paths* pattern stores each tree node in a document; in addition to the tree node, document stores as a string the id(s) of the node's ancestors or path. Although the *Materialized Paths* pattern requires additional steps of working with strings and regular expressions, the pattern also provides more flexibility in working with the path, such as finding nodes by partial paths.

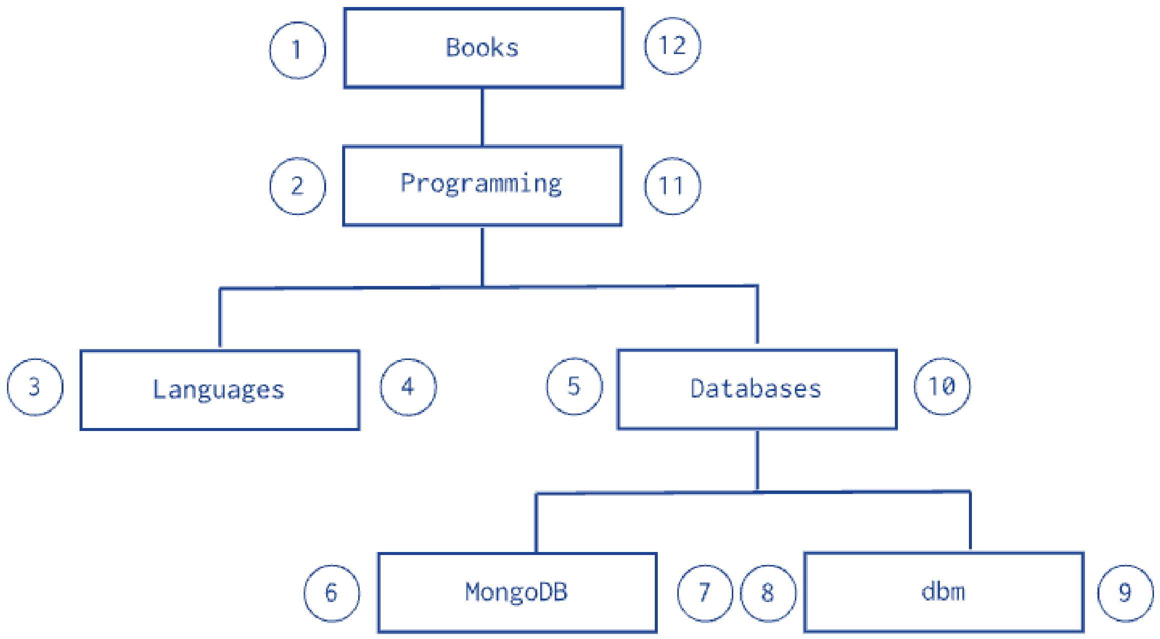
The following example models the tree using *Materialized Paths*, storing the path in the **field path**; the path string uses the comma ‘,’ as a delimiter:

```
>db.categories.insertMany( [
{ _id: "Books", path: null },
{ _id: "Programming", path: ",Books," },
{ _id: "Databases", path: ",Books,Programming," },
{ _id: "Languages", path: ",Books,Programming," },
{ _id: "MongoDB", path: ",Books,Programming,Databases," },
{ _id: "dbm", path: ",Books,Programming,Databases," }
] )
```

- You can query to retrieve the whole tree, sorting by the **field path**:
  `>db.categories.find().sort( { path: 1 } )`
- You can use regular expressions on the **path field** to find the descendants of **Programming**:
  `>db.categories.find( { path: /,Programming,/ } )`
- You can also retrieve the descendants of **Books** where the **Books** is also at the topmost level of the hierarchy:
  `>db.categories.find( { path: /^,Books,/ } )`

Model Tree Structures with Nested Sets

This model describes a tree like structure that optimizes discovering subtrees at the expense of tree mutability. The *Nested Sets* pattern identifies each node in the tree as stops in a round-trip traversal of the tree. The application visits each node in the tree twice; first during the initial trip, and second during the return trip. The *Nested Sets* pattern stores each tree node in a document; in addition to the tree node, document stores the id of node's parent, the node's initial stop in the **left field**, and its return stop in the **right field**.



The following example models the tree using *Nested Sets*:

```
>db.categories.insertMany( [
{ _id: "Books", parent: 0, left: 1, right: 12 },
{ _id: "Programming", parent: "Books", left: 2, right: 11 },
{ _id: "Languages", parent: "Programming", left: 3, right: 4 },
{ _id: "Databases", parent: "Programming", left: 5, right: 10 },
{ _id: "MongoDB", parent: "Databases", left: 6, right: 7 },
{ _id: "dbm", parent: "Databases", left: 8, right: 9 }
] )
```

You can query to retrieve the descendants of a node:

```
>var databaseCategory = db.categories.findOne( { _id: "Databases" } );
>db.categories.find( { left: { $gt: databaseCategory.left }, right: { $lt: databaseCategory.right } } );
```

The *Nested Sets* pattern provides a fast and efficient solution for finding subtrees but is inefficient for modifying the tree structure. As such, this pattern is best for static trees that do not change.

MongoDB Data Types:
MongoDB supports many data types. Some of them are

- **String** − This is the most commonly used data type to store the data. String in MongoDB must be UTF-8 valid.
- **Integer** − This type is used to store a numerical value. Integer can be 32 bit or 64 bit depending upon your server.
- **Boolean** − This type is used to store a Boolean (true/ false) value.
- **Double** − This type is used to store floating point values.
- **Arrays** − This type is used to store arrays or list or multiple values into one key. The array is created using square brackets.
- **Timestamp** − This type is used to store timestamp and value of this data type is 64 bit. This can be handy for recording when a document has been modified or added.
- **Object** − This data type is used for embedded documents.
- **Null** − This type is used to store a Null value.
- **Date** − This data type is used to store the current date or time in UNIX time format. You can specify your own date time by creating object of Date and passing day, month, year into it.
- **Object ID** − This data type is used to store the document's ID.
- **Binary data** − This data type is used to store binary data.
- **Code** − This data type is used to store JavaScript code into the document.
- **Regular expression** − This data type is used to store regular expression.

Inserting and Saving Documents:

The save() method uses either the insert or the update command internally. If you don't specify _id in the document then save() method will work same as insert() method. If you specify _id then it will replace whole data of document containing _id as specified in save() method.

Saving a New Document without Specifying an _id Field
If the document does not contain an _id field, then the save() method calls the insert() method. During the operation, the mongo shell will create an ObjectId and assign it to the _id field.

In the following example, save() method performs an insert since the document passed to the method does not contain the _id field:
>db.student.save({sname: "Vinaya Sane", percentage:90})

During the insert, the shell will create the _id field with a unique ObjectId value, The ObjectId values are specific to the machine and time when the operation is run. As such, your values may differ from those in the example.

Saving a New Document Specifying an _id Field
If the document contains an _id field, then the save() method is equivalent to an update with the upsert option set to true and the query predicate on the _id field.

In the following example, save() performs an update with upsert:true since the document contains an _id field:

```
>db.student.save({_id:100, sname: "Vinaya Sane", percentage:90})
```

If the _id field holds a value that *does not* exist in the collection, the update operation results in an insertion of the document. The results of these operations are identical to an update() method with the upsert option set to true.

Bulk Insert:

In MongoDB, the **Bulk.insert()** method is used to perform insert operations in bulk or in other words it is used to insert multiple documents in one go.

Unordered Insertion of documents:

Use following commands to make an object of Bulk using initializeUnorderedBulkOp() and use Bulk.insert() to insert multiple documents. Here, the initializeUnorderedBulkOp() method is used to generate an unordered list that MongoDB runs in bulk. If an error occurs during the processing of one of the write operations, MongoDB will continue to process remaining write operations in the list.

```
var bulk = db.student.initializeUnorderedBulkOp();
bulk.insert( { first_name: "Maya", last_name: "Jadhav" } );
bulk.insert( { first_name: "Pooja", last_name: "Patil" } );
bulk.insert( { first_name: "Ram", last_name: "Pawar" } );
bulk.insert( { first_name: "Vedant", last_name: "Jagtap" } );
bulk.insert( { first_name: "Supriya", last_name: "Mane" } );
bulk.execute();
```

With an *unordered* operations list, MongoDB can execute in parallel the write operations in the list and in any order. If the order of operations matter, use db.collection.initializeOrderedBulkOp() instead.

Ordered Insertion of documents:

Use following commands to make an object of Bulk using initializeOrderedBulkOp() and use Bulk.insert() to insert multiple documents. Here, the initializeOrderedBulkOp() method is used to generate an ordered list that MongoDB runs in bulk. If an error occurs during the processing of one of the write operations, MongoDB will return without processing any remaining write operations in the list.

```
var bulk = db.student.initializeOrderedBulkOp();
bulk.insert( { first_name: "Maya", last_name: "Jadhav" } );
bulk.insert( { first_name: "Pooja", last_name: "Patil" } );
bulk.insert( { first_name: "Ram", last_name: "Pawar" } );
bulk.insert( { first_name: "Vedant", last_name: "Jagtap" } );
bulk.insert( { first_name: "Supriya", last_name: "Mane" } );
bulk.execute();
```

With an *ordered* operations list, MongoDB executes the write operations in the list serially.

Practice Set:

1. Create a database named ‘Society’. Create collection named ‘Buildings’ in it. This collection should contain a document. Inside this document, we have a field named ‘committee for buildings’ which contains another document and this document contain three fields(i.e, Manager, Secretary, Member) with their values.
2. Create a database in Mongo DB called DB. Create a collection Employee in it. Use Bulk.insert() to insert at least 5 documents. Check if all the documents are inserted correctly.
3. Create a collection to embed the 3 courses document inside the Institute document like
   i. (“Institute_name":
   ii. "Contact":
   iii. "Address":
   iv. "Courses":)

   that maintains all the related data in a single document. Courses should contain fields like Course_id, Course_name & Fees.

Set A:

1. Create a database in Mongo DB called usermanaged, drop it and create it again. Check which database you are currently in.
2. Create a collection called customers in usermanaged database created and insert the document below. Check if the document is inserted correctly.

```
{     "firstName":"John",
      "lastName":"West",
      "email":"john.west@mail.com",
      "phone":"032345432134",
      "BusinessType": ["Sell", "Sugar", "Drinks"],
      "Reference":100,
      "Company":"Coca-Cola"}
```

   Write a MongoDB query to display all the documents in the collection

3. Create a database named ‘tybbaca’. Create collection named ‘Courses’ in it. This collection should contain a document. Inside this document, we have a field named ‘name’ which contains another document and this document contain three fields (i.e, first, middle, last) with their values.

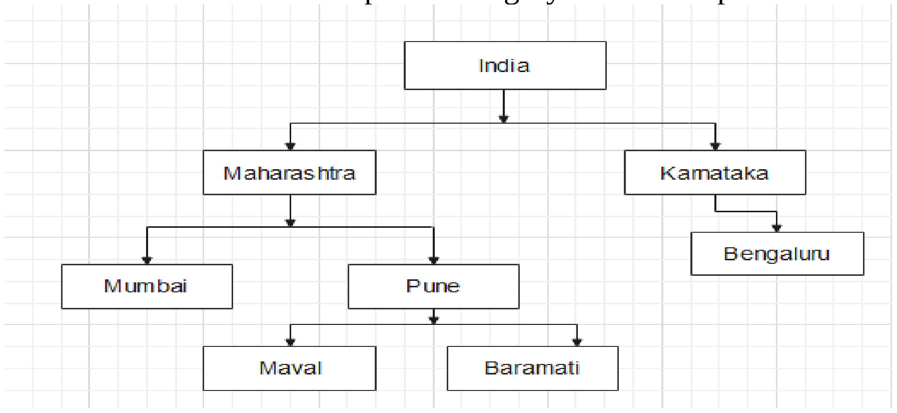
Set B:

1. Create a database in Mongo DB called myDatabase. Create a collection student in it. Use Bulk.insert() to insert at least 5 documents. Check if all the documents are inserted correctly.
2. Create a collection to embed the 2 address document inside the user document like

```
(“contact":
"dob":
"name":
"address": )
```

   that maintains all the related data in a single document.

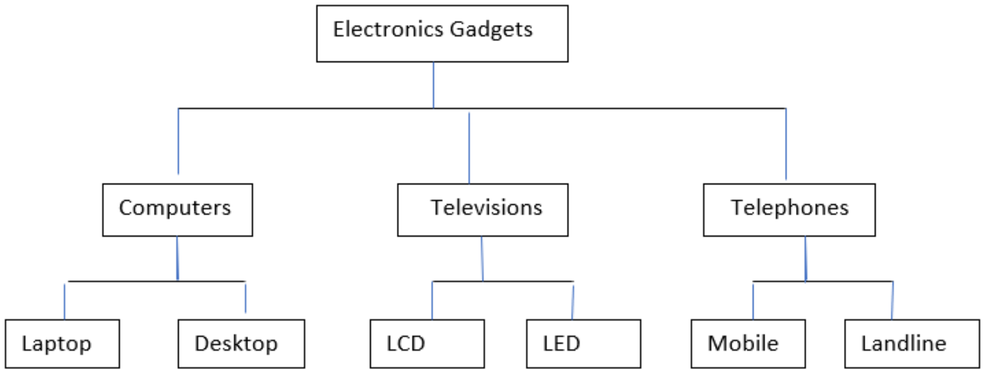
3. Write and run mongo DB query to models the following tree using *Parent References* and store the reference to the parent category in the field parent



Run a query to retrieve the descendants of a node Pune.

Set C:

1. Write and run mongo DB query to create following tree



For above tree query the parent field to list type of Electronics Gadgets

i) Write and run mongo DB queries to create tree structure with child references and ancestors for above tree.

ii) Write and run mongo DB query to retrieve the children of Televisions gadgets

iii) Write and run mongo DB query to find parent node and siblings of Laptop

Assignment Evaluation

0: Not Done [ ]	**1: Incomplete [ ]**	**2: Late Complete [ ]**
3: Needs Improvement [ ]	**4: Complete [ ]**	**5: Well Done [ ]**

Signature of Instructor

Assignment No. 2 : MongoDB Operators

MongoDB Operators:
Operators are special symbols that tell the compiler or interpreter to carry out specific mathematical or logical manipulations. Basically, all sorts of operators are available in MongoDB as we have in other programming languages.

Different Types of MongoDB Operators
1) MongoDB Comparison Operators
Comparison operators are used to compare two expressions and fetch data(documents) from the MongoDB database (collections) based on their usage in filters.
Various comparison operators in MongoDB are –

Name	Meaning	Description
$eq	Equal To	It matches values that are equal to the value specified in the query
$ne	Not Equal To (!=)	It matches all values that are not equal to the value specified in the query.
$gt	Greater Than (>)	It matches values that are greater than the value specified in the query.
$gte	Greater Than Equal To (>=)	It matches values that are greater than or equal to the value specified in the query.
$lt	Less Than (<)	It matches values that are less than the value specified in the query.
$lte	Less Than Equal To (<=)	It matches values that are less than or equal to the value specified in the query.
$in	In (Available in)	It matches any of the values that exist in an array specified in the query.
$nin	Not In (Not available in)	It matches values that do not exist in an array specified in the query.

Consider following student collection to understand MongoDB comparison, logical and element operators:

```
>db.student.insert({"stud_id":1, "stud_name": "Maya", "class": "TYBBA_CA",
"percentage":80})
>db.student.insert({"stud_id":2, "stud_name": "Pooja", "class": "TYBBA_CA",
"percentage":50})
>db.student.insert({"stud_id":3, "stud_name": "Ram", "class": "SYBBA_CA", "percentage":90})
>db.student.insert({"stud_id":4, "stud_name": "Vedant", "class": "SYBBA_CA",
"percentage":85})
>db.student.insert({"stud_id":5, "stud_name": "Supriya", "class": "TYBBA_CA",
"percentage":56})
>db.student.insert({"stud_id":6, "stud_name": "Smita", "class": "TYBBA_CA", "percentage":90,
"contact":9876543210})
```

a) $eq
Display the students from TYBBA_CA class.
```
>db.student.find({"class":{$eq:"TYBBA_CA"}})
```

b) $ne
Display Name of students not having TYBBA_CA class.
```
> db.student.find({"class":{$ne:"TYBBA_CA"}},{"_id":0, "stud_name":1})
```

c) $gt
Display the students having percentage greater than 80
```
>db.student.find({"percentage":{"$gt":80}})
```

d) $gte
Display the students having percentage greater than or equal to 80
```
>db.student.find({"percentage":{"$gte":80}})
```

e) $in
Display the students having percentage either 80 or 90.
```
>db.student.find({"percentage":{$in:[80,90]}})
```

f) $nin
Display the students who do not have percentage either 80 or 90.
```
>db.student.find({"percentage":{$nin:[80,90]}})
```

2) MongoDB Logical Operators

These MongoDB operators are used to perform logical operations on the given expressions. They help to make a decision based on multiple conditions.
Various logical operators in MongoDB are –

Name	Description
$and	It returns all documents that match the conditions of both expressions.
$or	It returns all documents that match the conditions of either expression.
$nor	It returns all documents that **do not** match the conditions of either expression
$not	It inverts the effect of a query expression and returns documents that **do *not*** match the query expression.

a) $and
Display students having percentage less than 80 and class TYBBA_CA
```
>db.student.find({$and:[{"percentage":{"$lt":80}},{"class":"TYBBA_CA"}]})
```

b) $or
Display students having percentage 80 or class SYBBA_CA.
```
>db.student.find({$or:[{"percentage":80},{"class":"SYBBA_CA"}]})
```

c) $nor
Display students not having percentage 80 or class SYBBA_CA.
>db.student.find({$nor:[{"percentage":80},{"class":"SYBBA_CA"}]})

d) $not
Display students not having the class as TYBBA_CA.
>db.student.find({"class":{$not:{$eq:"TYBBA_CA"}}})

3) MongoDB Element Operators

The next operators in MongoDB operators are element query operators. There are two operators under this category – $exists & $type.

Name	Description
$exists	It returns documents that have a specific field.
$type	It returns documents if field is of a specified type.

a) $exists
Display document having "contact" element.
>db.student.find({"contact":{"$exists":true}},{"_id":0})

b) $type
Display documents where "stud_id" is of type "String".
>db.student.find({"stud_id":{$type:"string"}})

4) MongoDB Array Operators

There are operators specified for projections for the documents having arrays. Below are the 3 operators in Array Query Operators –

Name	Description
$all	It returns documents from a collection if it matches all values in the specified array.
$size	It returns documents from the collection to match an array with the provided number of elements in it.
$elemMatch	It returns documents if they Match more than one component (all specified $elemMatch conditions) within an array element.

Consider following student collection to understand MongoDB array operators:

>db.classtest.insert({ "testResults" : [{ "subject" : "Maths", "marks" : 95 }, {"subject" : "English", "marks" : 80}], "tags" : ["Maths", "English"]})
>db.classtest.insert({ "testResults" : [{ "subject" : "Maths", "marks" : 90 }, {"subject" : "Science", "marks" : 80}], "tags" : ["Maths", "Science"]})

```
>db.classtest.insert({ "testResults" : [{ "subject" : "Maths", "marks" : 95 }, {"subject" :
"Science", "marks" : 80}, {"subject" : "English", "marks" : 80}], "tags" : ["Maths", "Science",
"English" ]})
```

a) $size
Display documents where there are 3 elements in testResults array.

```
> db.classtest.find({"testResults":{$size:3}})
```

b) $all
Display all the documents having Maths and English both in tags.

```
> db.classtest.find({"tags":{$all:["Maths","English"]}})
```

c) $elemMatch
Display documents where the subject is Maths and marks are greater than or equal to 95.

```
> db.classtest.find( { "testResults": { $elemMatch: { "subject" : "Maths", "marks":{$gte:95}}}})
```

Pattern Matching in MongoDB

MongoDB provides the functionality to search a pattern in a string during a query using following symbols.

Symbol	Syntax	Meaning
^	/^pattern/	It matches with the beginning of the string.
$	/pattern$/	It matches with the ending of the string.
.*	/.*pattern.*/	It matches with the substring appearing in the given field.

1) Display student details whose name start with “Ra” pattern.

```
>db.student.find({"stud_name": /^Ra/},{“_id”:0})
```

2) Display student details whose name end with “ya” pattern.

```
>db.student.find({"stud_name": /ya$/},{“_id”:0})
```

3) Display student details whose name has a substring “pri” in it.

```
>db.student.find({"stud_name": /.*pri.*/},{“_id”:0})
```

Document Validation:
To Create a document for the practical life we need to apply the document validators for the collection, means if we want to create the table with some rules and regulations like this field is accept only this data type, this field values must met some requirements like this then document validators are very essential.
MongoDB provides the capability to validate documents during updates and insertions. Validation rules are specified on a per-collection basis using the validator option, which takes a document that specifies the validation rules or expressions.

Example:
Create a collection name 'address' with the following rules

Field	Details
firstname	string type
lastname	string type
emailId	string type must be of tutorialtous.com domain
country	string only of (UK,INDIA)
pincode	String type min and max chars 5

Now validator is as follows
```
>db.createCollection("address",{
validator:{
firstname:{$type:"string"},
lastname:{$type:"string"},
emailid:{$type:"string",$regex:/@gmail\.com/},
country:{$in:["UK","INDIA"]},
pincode:{$type:"string",$regex:/...../}
}})
```

Valid Insertions for above collection:
```
>db.address.insert({firstname:"Maya",lastname:"Jadhav",emailid:"Maya@gmail.com",country:"
INDIA",pincode:"400001"})
```

Invalid Insertions for above collection:
```
>db.address.insert({firstname:"Dravid",lastname:"Rahul",emailid:"Maya_gmail.com",pincode:"
400001",country:"IN"})
```

Practice Set:

1) Write and run Mongo DB Query to create inventory collection with the following documents

   { _id: 1, item: { name: "ab", code: "123" }, qty: 15, tags: ["A", "B", "C"] }
   { _id: 2, item: { name: "cd", code: "123" }, qty: 20, tags: ["B"] }
   { _id: 3, item: { name: "ij", code: "456" }, qty: 25, tags: ["A", "B"] }
   { _id: 4, item: { name: "xy", code: "456" }, qty: 30, tags: ["B", "A"] }
   { _id: 5, item: { name: "mn", code: "000" }, qty: 20, tags: [["A", "B"], "C"] }

   Query the inventory collection to select all documents where the value of the qty field equals 20.
   Query the inventory collection to select all documents where the tags array equals exactly the specified array or the tags array contains an element that equals the array ["A", "B"].

Set A:

1. Write and run Mongo DB Query to create products collection with the following documents
   ([
   { "_id" : 1, "name" : "xPhone", "price" : 799, "releaseDate" : ISODate("2011-05-14T00:00:00Z"), "spec" : { "ram" : 4, "screen" : 6.5, "cpu" : 2.66 }, "color" : ["white", "black"], "storage" : [64, 128, 256] },
   { "_id" : 2, "name" : "xTablet", "price" : 899, "releaseDate" : ISODate("2011-09-01T00:00:00Z"), "spec" : { "ram" : 16, "screen" : 9.5, "cpu" : 3.66 }, "color" : ["white", "black", "purple"], "storage" : [128, 256, 512] },
   { "_id" : 3, "name" : "SmartTablet", "price" : 899, "releaseDate" : ISODate("2015-01-14T00:00:00Z"), "spec" : { "ram" : 12, "screen" : 9.7, "cpu" : 3.66 }, "color" : ["blue"], "storage" : [16, 64, 128] },
   { "_id" : 4, "name" : "SmartPad", "price" : 699, "releaseDate" : ISODate("2020-05-14T00:00:00Z"), "spec" : { "ram" : 8, "screen" : 9.7, "cpu" : 1.66 }, "color" : ["white", "orange", "gold", "gray"], "storage" : [128, 256, 1024] },
   { "_id" : 5, "name" : "SmartPhone", "price" : 599, "releaseDate" : ISODate("2022-09-14T00:00:00Z"), "spec" : { "ram" : 4, "screen" : 9.7, "cpu" : 1.66 }, "color" : ["white", "orange", "gold", "gray"], "storage" : [128, 256] }
   ])

   select all documents where the price is less than 699 or greater than 799.

2. For above collection Find documents where:
   i) the price field is not greater than 699.
   ii) do not contain the price field.

3. Create a collection named mycol in db and insert 3 documents like:

```
{
        title: "MongoDB Overview",
        description: "MongoDB is no SQL database",
        by: "M.Sc.(CA)",
        url: " http://www.dypacs.com ",
        tags: ["mongodb", "database", "NoSQL"],
        likes: 100
},
{
        title: "NoSQL Database",
        description: "NoSQL database doesn't have tables",
        by: "T.Y.B.C.A (Science)",
        url: "http://www.dypacs.com",
        tags: ["mongodb", "database", "NoSQL"],
        likes: 20,
        comments: [
                {
                        user:"user1",
                        message: "My first comment",
                        dateCreated: new Date(2021-11-10),
                        like: 10
                }
        ]
}
```

Set B:

1. Write MongoDB query for above collection to find all the tutorials written by ‘M. Sc. (CA)’ and whose title is 'MongoDB Overview'.

2. Write MongoDB query for above collection to show the documents that have likes greater than 10 and whose title is either 'MongoDB Overview' or by is ‘T.Y.B.C.A (Science)’.

3. Write MongoDB query to create MongoDB **posts** collection. Each **post** document has a title, author and a comments array field. Each element of the comments array represents a user with some basic information like country, name, isGuest, and country. Insert at least 5 documents in it with your own data. Retrieve all posts where at least one comment is made by a user with name John.

Set C:

1. Write Mongo DB query to create Employee collection which has the Field names of "Employeeid" and "EmployeeName". Let' also assume that we have the following documents in our collection.

Employee id	**Employee Name**
22	NewMartin
2	Mohan
3	Sameer
4	MohanR
100	Saurabh
6	Gurang

   i) Find all Employee Names which have the characters 'Sa' in it.
   ii) Display all documents in reversed sorted order

2. create a collection ***projects*** with required fields and validation as below:
   - name — string
   - status — enum [inProgress, Completed]
   - client — object

```
("projects", {
 validator:{
   $jsonSchema: {
     bsonType: "object",
     required: ["name", "status", "client"],
     properties: {
       name: {
         bsonType: "string",
         description: "Project name is required field"
       },
       status: {
         enum: ["inProgress", "Completed"],
         description: "Status can only be from one of the enum values"
       },
       client: {
         bsonType: "object",
         required: ["name"],
         properties:{
           name: {
             bsonType: "string",
             description: "Client name is required"
           },
           phone: {
             bsonType: "string",
             maximum: 10,
             minimum: 8,
             description: "Phone number maximum should be 10 digits long"
```

```
            }
          }
        }
      }
    }
  }
})
```

i) Try to inserts values which violates the validation
ii) Try to execute above query with valid values

Assignment Evaluation

0: Not Done [ ]	**1: Incomplete [ ]**	**2: Late Complete [ ]**
3: Needs Improvement [ ]	**4: Complete [ ]**	**5: Well Done [ ]**

Signature of Instructor

Assignment No. 3 : Update and Delete operation in MongoDB

Update Operations in MongoDB

The update operations are used to update or modify the existing document in the collection. You can perform update operations using the following methods provided by the MongoDB:

Method	Description
update()	It is used to update the values in the existing document in the collections of MongoDB
updateOne()	It is used to update a single document in the collection that satisfies the given criteria.
updateMany()	It is used to update multiple documents in the collection that satisfy the given criteria.
replaceOne()	It is used to replace single document in the collection that satisfy the given criteria.
findOneAndUpdate()	It is used to updates the first matched document in the collection that matches the selection criteria, it returns document.
findOneAndReplace()	It is used to find a matching element, replaces it with the provided element, and passes the returned doc to the callback.

update()
The **update()** method updates the values in the existing document in the collections of MongoDB. When you update your document the value of the _id field remains unchanged. By default, the db.collection.update() method updates a single document. Include the option multi: true to update all documents that match the given query. This method can be used for a single updating of documents as well as multi documents.
Example:
a) Update the name of the student whose name has “Maya” value to “Mayra”.
>db.student.update({stud_name:"Maya"},{$set:{stud_name:"Mayra"}})
Here, the first parameter is the document whose value to be changed {stud_name: “Maya”} and the second parameter is set keyword means to set(update) the following matched key value with the older key value. The value of the key must be of the same data type that was defined in the collection.
b) Update the percentage of the document whose stud_name is “Pooja” to 60.
>db.student.update({stud_name:"Pooja"},{$set:{percentage:60}}

updateOne()
In MongoDB, **updateOne()** method updates a first matched document within the collection based on the given query. This method updates one document at a time and can also add new fields in the given document.

Example:
a) Update the percentage of the student whose name is “Pooja”
>db.student.updateOne({stud_name: "Pooja"}, {$set:{percentage:60}})

b) Insert a new field in the document using the updateOne method
>db.student.updateOne({stud_name: "Supriya"}, {$set:{contact:9876543201}})
Here, a new field is added, i.e., contact:9876543201 in the document of a student whose name is "Supriya".

updateMany()
The **updateMany()** method updates all the documents in MongoDB collections that match the given query. This method can also add new fields in the document. Specify an empty document(**{}**) in the selection criteria to update all collection documents.

Example:
a) Update all the matched documents whose percentage is greater than 90 to remark: "Excellent"
>db.student.updateMany({percentage:{$gt:90}},{$set:{remark:"Excellent"}})

replaceOne()
The **replaceOne()** method replaces the first matching document in the collection that matches the given query, using the replacement document.
Example:
a) Replace the document with stud_id as 3 to new document having stud_name: "Ramesh", class: "TYBBA_CA", percentage:92.
>db.student.replaceOne({"stud_id":3}, {"stud_id":3, "stud_name": "Ramesh", "class": "TYBBA_CA", "percentage":92})

findOneAndUpdate()
The **findOneAndUpdate()** method updates the first matched document in the collection that matches the selection criteria. If more than one document matched the selection criteria then it updates only the first matched document. This method will return the original document but if we want to return the updated document then we have to set the value of the returnNewDocument parameter to true.
Difference between findOneAndUpdate() and updateOne() is, findOneAndUpdate returns a document whereas updateOne does not, it just returns the _id if it has created a new document.
Example:
a) Update the percentage of the student whose name is "Smita"
>db.student.findOneAndUpdate ({stud_name: "Smita"}, {$set:{percentage:95}})

Update the value of the embedded document:
Insert following document in student collection to update value of embedded document Marks. Here Marks is an embedded document.
>db.student.insert({"stud_id":7, "stud_name": "Pratibha", "class": "TYBBA_CA", "Marks":{"English": 90, "Science":80}})

b) Update the Science marks of the student whose name is "Pratibha" to 85.
>db.student.findOneAndUpdate({stud_name:"Pratibha"},{$set:{ "Marks.Science":85}})

findOneAndReplace()

The **findOneAndReplace()** method replaces the first matched document based on the given selection criteria. By default, this method returns the original document. To return the replacement document, set the value of the returnNewDocument option to true.

Example:

a) Replace the first matched document whose percentage is 90 and return replaced document.

```
>db.student.findOneAndReplace({percentage:90},{"stud_id":3,      "stud_name":      "Ramesh",
"class": "TYBBA_CA", "percentage":92},{returnNewDocument:true})
```

MongoDB – Field Update Operators

MongoDB provides different types of field update operators to update the values of the fields of the documents that match the specified condition. The following table contains the field update operators:

Operator	Description
$currentDate	This operator is used to set the value of a field to current date, either as a Date or a Timestamp.
$inc	This operator is used to increment the value of the field by the specified amount.
$max	This operator is used only to update the field if the specified value is greater than the existing field value.
$min	This operator is used only to update the field if the specified value is less than the existing field value
$mul	This operator is used to multiply the value of the field by the specified amount.
$rename	This operator is used to rename a field.
$setOnInsert	This operator is used to set the value of a field if an update results in an insert of a document. It has no effect on update operations that modify existing documents.

Consider following Employee collection to understand MongoDB field update operators:

```
>db.Employee.insert({"emp_name":{"fname":"Varad", "lname":"Mane"},
"emp_details":{"age":30,"contact":11111111,"salary":40000},"dept":"Account","exp_yr":4,"doj
":ISODate(),"points":[6,8]});
>db.Employee.insert({"emp_name":{"fname":"Mahendra", "lname":"Pawar"},
"emp_details":{"age":25,"contact":22222222,"salary":30000},"dept":"HR","exp_yr":3,"doj":ISO
Date(),"points":[2,6,7]});
>db.Employee.insert({"emp_name":{"fname":"Pallavi", "lname":"Patil"},
"emp_details":{"age":29,"contact":33333333,"salary":25000},"dept":"Finance","exp_yr":3,"doj"
:ISODate(),"points":[5,7]});
>db.Employee.insert({"emp_name":{"fname":"Geeta", "lname":"Jagtap"},
"emp_details":{"age":28,"contact":44444444,"salary":30000},"dept":"Developement","exp_yr":
2,"doj":ISODate(),"points":[4,8,6]});
>db.Employee.insert({"emp_name":{"fname":"Samir", "lname":"Jadhav"},
"emp_details":{"age":30,"contact":55555555,"salary":40000},"dept":"Developement","exp_yr":
4,"doj":ISODate(),"points":[8,7]});
```

$currentDate operator:
In the example, we are updating the value of “doj” field of an employee’s document whose first name is Pallavi.
>db.Employee.updateOne({"emp_name.fname": "Pallavi"}, {$currentDate: {doj: true}})

$inc operator:
In this example, we are updating the salary field of an employee’s document whose name is Geeta.
>db.Employee.update({"emp_name.fname": "Geeta"}, {$inc: {"emp_details.salary": 5000}})

$max operator:
In this example, we are comparing values (or numbers) of the salary fields with the specified value, i.e., 40000. Here, the specified value is greater than the current value. So, $max operator updates the value of the salary field with the help of update() method to 40000.
>db.Employee.update({"emp_name.fname": "Mahendra"}, {$max: {"emp_details.salary": 40000}})

$min operator:
In this example, we are comparing values (or numbers) of the salary fields with the specified value, i.e, 35000. Here, the specified value is less than the current value. So, $min operator updates the value of the salary field with the help of update() method to 35000.
>db.Employee.update({"emp_name.fname": "Mahendra"}, {$min: {"emp_details.salary": 35000}})

$mul operator:
In this example, we are multiplying the value of salary field by 2 in the document who matches the specified condition, i.e., emp_name = “Samir”.
>db.Employee.update({"emp_name.fname": "Samir"}, {$mul: {"emp_details.salary": 2}})

$rename operator:
In this example, we are renaming the name of dept field to section in the employee’s document whose first name is “Varad”.
>db.Employee.update({"emp_name.fname": "Varad"}, {$rename: {"dept": "section"}})

$setOnInsert:
In this example, we are creating a new document in the Example collection with the help of update() method by setting the value of an upsert field to true and using $setOneInsert operator assign the values to embedded fields i.e., in the document.
>db.Employee.update({"emp_name.fname": "Pramod", "exp_yr": 5}, {$setOnInsert: {"emp_details.age": 27, "emp_details.contact": 66666666}},{upsert: true})

MongoDB - array update operators:
MongoDB provides different types of array update operators to update the values of the array fields in the documents.

Operator	Description
$push	It adds an item to an array.
$pop	It removes the first or last item of an array.
$pull	It removes all array elements that match a specified query.
$pullAll	It removes all matching values from an array.
$	It acts as a placeholder to update the first element that matches condition.
$addToSet	It adds elements to an array only if they do not already exist in the set.

a) MongoDB – $push Operator
$push operator is used to append a specified value to an array. We can use the following **modifiers** with the $push operator:

Modifier	Description
$each	It is used to append multiple values to the array field.
$slice	It is used to limit the number of array items and require the use of the $each modifier.
$sort	It is used to order items of the array and require the use of the $each modifier.
$position	It is used to add items in an array by specifying position and require the use of the $each modifier.

Consider following Contributor collection to understand MongoDB array update operators:

```
>db.Contributor.insert({"Cont_name":"Rahul", "Branch":"CSE", "Join_year":2020,
"Language":["C", "Java", "C++"],
"Articles":[{"Language":"C++","tArticles":30,"pArticles":20},{"Language":"Java","tArticles":6
0,"pArticles":40}], "Per_details": {"age":25, "Sem_Marks":[60,70,67,71]}});
>db.Contributor.insert({"Cont_name":"Amol", "Branch":"CSE", "Join_year":2019,
"Language":["Java", "C++","PHP"],
"Articles":[{"Language":"Java","tArticles":40,"pArticles":30},{"Language":"PHP","tArticles":5
0,"pArticles":50}], "Per_details": {"age":26, "Sem_Marks":[72,83,85,74]}});
>db.Contributor.insert({"Cont_name":"Dinesh", "Branch":"CSE", "Join_year":2020,
"Language":["Python","Java","PHP"],
"Articles":[{"Language":"Python","tArticles":40,"pArticles":40},{"Language":"Java","pArticles
":50}], "Per_details": {"age":24, "Sem_Marks":[92,84,81,88]}});
>db.Contributor.insert({"Cont_name":"Radha", "Branch":"CSE", "Join_year":2018,
"Language":["Java","C"],
"Articles":[{"Language":"Java","tArticles":50,"pArticles":40},{"Language":"C","tArticles":60}]
,"Per_details": {"age":25, "Sem_Marks":[78,82,91,90]}});
>db.Contributor.insert({"Cont_name":"Neha", "Branch":"CSE", "Join_year":2019,
"Language":["Python","Java"],
"Articles":[{"Language":"Python","tArticles":40,"pArticles":60},{"Language":"Java","tArticles"
:40,"pArticles":30}],"Per_details": {"age":26, "Sem_Marks":[77,79,85,83]}});
```

Appending a single value to an array:
In this example, we are appending a single value, i.e., “C” to an array field, i.e., Language field in the document whose Cont_name is “Neha”.
```
> db.Contributor.update({Cont_name: "Neha"}, {$push: {Language: "C"}})
```

Appending multiple values to an array:
In this example, we are appending multiple values, i.e., [“C++”, “PHP”] to an array field, i.e., Language field in the document whose Cont_name is “Radha”.
```
> db.Contributor.update({Cont_name: "Radha"}, {$push: {Language: {$each: ["C++",
"PHP"]}}})
```

Use of $each, $sort and $slice modifiers with $push operator:
In this example, we are using multiple modifiers like $each, $sort, and $slice with $push operator.
```
> db.Contributor.update({Cont_name: "Rahul"}, {$push: { Language: { $each: ["PHP",
"Python"],  $sort: 1, $slice:4}}})
```

Use of $position modifiers with $push operator:
$position modifier is used to specify the location in the array at which the $push operator inserts items. Without $position modifier $push operator insert items at the end of the array.

Adding items at the start of the array:
In this example, we are adding items, i.e., [“C”, “C++”] in the beginning (i.e., position 0)of the Language field.
```
>db.Contributor.update({Cont_name: "Neha"}, {$push: { Language: { $each: ["C", "C++"],
$position: 0}}})
```

Adding items to the middle of the array:
In this example, we are adding items, i.e., [“Java”] in the middle (i.e., position 2) of the Language field.
```
>db.Contributor.update({Cont_name: "Neha"},{$push: { Language: { $each: [ Java"],
$position: 2}}})
```

b) MongoDB – $pop Operator
$pop operator is used to remove the first or the last item from the array.

Removing first item from the array:
In this example, we are removing the first element of the Language field in the document whose Cont_name is “Rahul”, by setting the value of $pop operator to -1.
```
> db.Contributor.update({Cont_name: "Rahul"}, {$pop: { Language: -1}})
```

Removing last item from the array:
In this example, we are removing the last element of the Language field in the document whose Cont_name is “Rahul”, by setting the value of $pop operator to 1.
```
> db.Contributor.update({Cont_name: "Rahul"}, {$pop: {Language: 1}})
```

c) MongoDB – $pull Operator

$pull operator is used to remove all the instances of the value or the value that matches the specified condition from the existing array.

Removing all the elements that equal to the specified value:

In this example, we are removing the specified elements, i.e., [“C”, “C++”] from the Language field of all documents.

> db.Contributor.update({},{$pull: {Language: {$in: ["C", "C++"]}}}, {multi: true})

Removing all the elements that match the specified condition:

In this example, we are removing semester marks that are less than and equal to ($lte) 75 from the Per_details.Sem_Marks field in the document whose Cont_name is “Amol”.

>db.Contributor.update({Cont_name: "Amol"}, {$pull: {"Per_details.Sem_Marks": {$lte: 75}}})

Removing elements from the array of documents:

In this example, we are removing Language: “Java” and tArticles: 40 items from the array of documents, i.e., Articles field.

> db.Contributor.update({},{$pull: {Articles: {Language: "Java", tArticles: 40}}}, {multi: true})

d) MongoDB – $pullAll Operator

$pullAll operator is used to remove all instances of the specified values from an existing array. It is different from $pull operator, $pull operator removes items by specifying a query, whereas $pullAll operator removes items that matches the listed values.

Removing items from an array using $pullAll operator:

In this example, we are removing items specified in the list, i.e., [“Python”, “PHP”] from the language field with the help of $pullAll operator.

> db.Contributor.update({Cont_name: "Dinesh"}, {$pullAll: {Language: ["Python", "PHP"]}})

e) MongoDB – Positional Operator ($)

Positional operator (**$**) recognizes an element in an array to update without explicitly specifying the position of that item in the array.

Updating values in the array using $ operator:

In this example, we are updating the first item whose value is “Java” to “PHP” in the language field with the help of $ operator, because we don’t know the position of the item in the array.

> db.Contributor.updateOne({Cont_name: "Neha", Language: "Java"}, {$set: {"Language.$": "PHP"}})

Here, the $ operator is working as a placeholder for the first match of the update query document.

f) MongoDB – $addToSet Operator

If you use $each modifier with $addToSet operator, then it adds multiple values to an array field if the specified value does not present in the array field.

Using $each modifier with $addToSet operator:

In this example, we are updating a contributor's document whose Cont_name is Radha using $each modifier with $addToSet operator. Here, this operation only adds "PHP" in the Language field and does not add "C" because it is already exists in the Language field.

```
>db.Contributor.update({Cont_name: "Radha"},{$addToSet: {Language: {$each:
["C","PHP"]}}})
```

Delete Operations in MongoDB

The delete operation is used to delete or remove the documents from a collection. You can perform delete operations using the following methods provided by the MongoDB:

Method	**Description**
deleteOne()	It is used to delete a single document from the collection that satisfies the given criteria.
deleteMany()	It is used to delete multiple documents from the collection that satisfy the given criteria.
findOneAndDelete()	It is used to delete the first matching document in the collection that matches the filter
remove()	It can remove one or all documents that matches the filter.

deleteOne()

The **deleteOne()** method deletes the first document from the collection that matches the given selection criteria. It will delete/remove a single document from the collection

Example:

a) Delete the first matched document whose percentage is less than 40.

```
>db.student.deleteOne({percentage:{$lt:40}})
```

deleteMany()

In MongoDB, you are allowed to delete the existing documents from the collection using **deleteMany()** method. This method deletes multiple documents from the collection according to the filter.

Example:

a) Delete multiple documents whose percentage is less than 40:

```
>db.student.deleteMany({percentage:{$lt:40}})
```

b) Delete all documents from student collection.

```
>db.student.deleteMany({})
```

findOneAndDelete()

The **findOneAndDelete()** method deletes a single document based on the selection criteria from the collection. It deletes the first document from the collection that matches the given filter query expression.

Difference between findOneAndDelete() and deleteOne() is, findOneAndDelete() deletes single documents from the collection on the basis of a filter as well as it returns the deleted document whereas deleteOne() removes single document from the collection.

Example:

a) Find and Delete the first student document whose stud_name is “Maya”.

```
>db.student.findOneAndDelete({stud_name:"Maya"})
```

remove()

The **remove()** method removes documents from the database. It can remove one or all documents from the collection that matches the given query expression. If you pass an empty document(**{}**) in this method, then it will remove all documents from the specified collection.

Example:

a) Remove all the documents whose stud_name is “Pooja”.

```
>db.student.remove({stud_name: "Pooja"})
```

b) Remove all the documents from student collection.

```
>db.student.remove({})
```

DropMongoDB collection

In MongoDB, **drop()** method is used to drop a collection from a database. It completely removes a collection from the database and does not leave any indexes associated with the dropped collections.

Example:

```
>db.student.drop()
```

It will return true if student collection is deleted successfully.

Drop MongoDB database

In MongoDB **dropDatabase()** command is used to drop a existing database.

Example:

```
>db.dropDatabase()
```

We do not specify any database name in this command, because this command deletes the currently selected database. If you have not selected any database, then it will delete default 'test' database.

Practice Set:

For a collection "College" created in a database containing documents with fields {College_id, College_name, Faculty, Subjects, Salary}

With multiple documents

1. Update subject allocated to given faculty
2. Increase salary by Rs. 1000 of all faculty
3. Display all updated documents

Set A:

Consider a collection "***User***" which contains documents with the same structure like this one:

```
{
  "_id" : ObjectId("57cc58333d496dc219c09c2c"),
  "firstname" : "Ninad",
  "lastname" : "Nikam",
  "email" : "m.mustermann@example.com",
  "password" : "d9729feb74992cc3482b350163a1a010",
  "last_login" : "2015-01-07",
  "note" : "Always pays in time, very good customer!",
  "address" :
  {
    "country" : "India",
    "street" : "Nagar Road",
    "zip" : "62717"
  }
}
```

1. **Replace** a **single existing** document entirely with other data
2. Replace the document for current firstname field that you have taken in your document
3. Insert some other similar documents in it.

Set B:

In above Collection

1. Finds the first document whose firstname field is equal "Ninad" and updates the lastname field to "Patil".
2. Update multiple fields of a document.
3. Increase the zip code of a user by one.

Set C:

Consider the following document in the students collection whose grades element value is an array of embedded documents:

```
{
  _id: 4,
  grades: [
    { grade: 80, mean: 75, std: 8 },
    { grade: 85, mean: 90, std: 5 },
    { grade: 85, mean: 85, std: 8 }
  ]
}
```

1. Update the std field of the first array element that matches the grade equal to 85
2. Delete array element with grade 80

Assignment Evaluation

0: Not Done [ ]	**1: Incomplete [ ]**	**2: Late Complete [ ]**
3: Needs Improvement [ ]	**4: Complete [ ]**	**5: Well Done [ ]**

Signature of Instructor

Assignment No. 4: MongoDB Cursor

Cursor in MongoDB:

In MongoDB, when the find() method is used to find the documents present in the given collection, then this method returned a pointer which will points to the documents of the collection, now this pointer is known as cursor. Or in other words we can say that a *cursor is a pointer*, and using this pointer we can access the document. By default, cursor iterates automatically, but you can iterate a cursor manually also.

To display all the documents present in the student collection we use following query

```
>db.student.find().pretty()
```

This find() method return a cursor which contain all documents present in the student collection.

In MongoDB, the find() method return the cursor, now to access the document we need to iterate the cursor. In the mongo shell, if the cursor is not assigned to a var keyword then the mongo shell automatically iterates the cursor up to 20 documents. MongoDB also allows you to iterate cursor manually. So, to iterate a cursor manually simply assign the cursor return by the find() method to the var keyword Or JavaScript variable.

Example:

```
>var mycursor = db.student.find({stud_name: "Supriya"}).pretty()
```

Here, we iterate the cursor manually to find the document whose student name is “Supriya”. So, we assigned the cursor returned by the find() method to the JavaScript variable(i.e. mycursor).

a) next() and hasNext()

We can use cursor next() method to access the next document. Here hasNext() method returns true if the cursor has more documents and can be iterated.

Example:

```
>var mycursor = db.student.find({stud_id:{$gt:3}});
> while(mycursor.hasNext()){
... print(tojson(mycursor.next()))
... }
```

In this example we exclusively took the cursor to start with the stud_id > 3. So it skipped first three documents and retrieves the remaining documents. Here, print(tojson()) method is used to display the result. You can also use printjson() method to display the result.

b) forEach()

We can also use forEach() method to iterate the cursor. This function applies a JavaScript function to each document from the cursor.

Example:

```
>var mycursor = db.student.find({stud_name: "Supriya"})
>mycursor.forEach(printjson)
```

Here, first we store the cursor returned by the find() method in the mycursor variable. Now, we use forEach() method to iterate the cursor and display the resultant document using printjson.

c) toArray()
In mongo shell, you are allowed to iterate the cursor and display the resultant document in the array using toArray() method.
Example:
```
>var mycursor = db.student.find()
>var docs = mycursor.toArray()
>var resultdoc = docs[0]
```

Here, first we assign the returned cursor to the var mycursor, in the next we create an array from the resultant cursor using toArray() method and assign the result to the var docs. Now we access the documents according to their index e.g. var resultdoc = docs[0], here we display a document whose index is 0.

d) count()
To know how many documents are present in a collection use the count() method, which returns the total number of documents are present in the given collection.
Example:
```
>db.student.find().count()
```

e) limit()
The limit() method helps to fetch limited records from a collection. Suppose we have multiple documents, but we want to have topmost or only 2 documents, then by using the limit() method, we can achieve that.
Example:
```
>db.student.find().limit(2)
```
Here, we only display the first two documents from the student collection.

f) skip()
During the display of documents, we may require to skip few documents due to various reasons. Method skip() helps to display results after skipping a number of documents.
```
>db.student.find().skip(2)
```
Here, it will return the result set after skipping first two documents from student collection.

g) size()
The cursor.size() method will be helpful to return a count of the number of documents which got as the output from db.collection.find() query after applying any cursor.skip() and cursor.limit() methods. Basically, it will filter for the given condition and find the size of the cursor.
Example:
```
>db.student.find({class:"TYBBA_CA"}).size()
```

h) sort()
Use sort() method to sort the documents. It you want to sort the documents in ascending, then set the value of the field to 1 and in descending, then set it to -1.
Example:
```
>db.student.find().sort({stud_id:-1})
```
Here, we sort all the documents present in the student collection in descending order.

i) explain()
Use explain() method to report on the query execution plan for a cursor.
Example:
```
>db.student.find({stud_id:{$gt:3}}).explain('executionStats')
```

j) pretty()
Use pretty() method to display the result fetched by a cursor in well-formatted way.
Example:
```
>db.student.find({stud_id:{$gt:3}}).pretty()
```

Practice Set:

Insert the following data into the “Employee” collection with multiple documents containing following fields

```
{
     “emp_id”:
      "Name" :
      "City" :
      "Salary" :
}
```

i) Count all the documents of a collection

ii) Find the details of the top 2 employees living in a particular city.

iii) Skips the first 3 documents from the result and prints the remaining documents.

Set A:

1. For created database warehouse containing a collection of products with multiple documents having fields {“product_id” : ,"name": ,"brand":, "type": ,"price":, "warranty_years":, "available": "true"},
   i) Check if the cursor object has more documents to return or not
   ii) Return the next document in a cursor
   iii) Insert a new document having, name = “Cable TV Basic Service Package", "type" : "tv", "monthly_price" : 50, "term_years" : 2, "cancel_penalty" : 25, "sales_tax" : "true", "additional_tariffs" : [{ "kind" : "federal tariff", "amount" : { "percent_of_service" : 0.06 } }, { "kind" : "misc. tariff", "amount" : 2.25 }]
2. For above created database
   i) sort all the documents present in the products collection in ascending & descending order
   ii) Find how many documents are present in a collection
   iii) Display topmost or only 3 documents
3. For above created database
   i) Display the result fetched by a cursor in well-formatted way.
   ii) Report on the query execution plan for a cursor
   iii) Display all document by skipping first

Set B:

1. For created database warehouse containing a collection of products display the product_id which are not available in warehouse
2. Modify the availability of product having name phone
3. Replace the document with product_id 5 to a new document "name" : "Phone Extended Warranty", "type" : "warranty", "price" : 38, "warranty_years" : 2, "for" : ["ac3", "ac7", "ac9", "qp7", "qp8", "qp9"]

Set C:

1. For created database warehouse containing a collection of products remove the warranty of particular product
2. Delete all documents from collection which warranty_years less than 2 years

Assignment Evaluation

0: Not Done [ ]	**1: Incomplete [ ]**	**2: Late Complete [ ]**
3: Needs Improvement [ ]	**4: Complete [ ]**	**5: Well Done [ ]**

Signature of Instructor

Assignment No. 5 : MongoDB Index and Aggregation

MongoDB Index

Indexes provide users with an efficient way of querying data. When querying data without indexes, the query will have to search for all the records within a database to find data that match the query.
In MongoDB, querying without indexes is called a collection scan. A collection scan will:

- Result in various performance bottlenecks
- Significantly slow down your application

Fortunately, using indexes fixes both these issues. By limiting the number of documents to be queried, you'll increases the overall performance of the application.

Indexes are special data structures that store a small part of the Collection's data in a way that can be queried easily. Indexes store the values of the indexed fields outside the table or collection and keep track of their location in the disk. These values are used to order the indexed fields, this ordering helps to perform equality matches efficiently.

Consider following student collection to understand MongoDB Indexing:

```
>db.student.insert({"stud_id":1, "stud_name": "Maya", "class": "TYBBA_CA",
"percentage":80});
>db.student.insert({"stud_id":2, "stud_name": "Pooja", "class": "TYBBA_CA",
"percentage":50});
>db.student.insert({"stud_id":3, "stud_name": "Ram", "class": "SYBBA_CA",
"percentage":90});
>db.student.insert({"stud_id":4, "stud_name": "Vedant", "class": "SYBBA_CA",
"percentage":85});
>db.student.insert({"stud_id":5, "stud_name": "Supriya", "class": "TYBBA_CA",
"percentage":56});
```

Creating indexes

When creating documents in a collection, MongoDB creates a unique index using the _id field. MongoDB refers to this as the **Default _id Index**. This default index cannot be dropped from the collection.

When creating an index, you need to define the field to be indexed and the direction of the key (1 or -1) to indicate ascending or descending order. Another thing to keep in mind is the index names. By default, MongoDB will generate index names by concatenating the indexed keys with the direction of each key in the index using an underscore as the separator. For example: {stud_name: 1} will be created as stud_name_1. The best option is to use the name option to define a custom index name when creating an index.

Let’s create an index using the name field in the student collection and name it as **student name index**.

```
>db.student.createIndex(
{stud_name: 1},
{name: "student name index"}
)
```

Finding indexes
You can find all the available indexes in a MongoDB collection by using the getIndexes method. Following command will return all the indexes in a student collection.

```
>db.student.getIndexes()
```

The output contains the **default _id index** and the user-created index **student name index**.

Dropping indexes
To delete an index from a collection, use the **dropIndex** method while specifying the index name to be dropped.

Following command will remove the user-created index with the index name **student name index.**

```
>db.student.dropIndex("student name index")
```

or

```
>db.student.dropIndex({stud_name:1})
```

The **dropIndexes** command can also drop all the indexes excluding the default _id index.

```
>db.student.dropIndexes()
```

Common MongoDB index types
MongoDB provides different types of indexes that can be utilized according to user needs. Here are the most common ones:

- Single field index
- Compound index
- Multikey index

Single field index
These user-defined indexes use a single field in a document to create an index in an ascending or descending sort order (1 or -1). In a single field index, the sort order of the index key does not have an impact because MongoDB can traverse the index in either direction.

```
>db.student.createIndex({stud_name:1})
```

The above index will sort the data in ascending order using the stud_name field. You can use the **sort()** method to see how the data will be represented in the index.

```
>db.student.find({},{_id:0}).sort({stud_name:1})
```

Compound index
You can use multiple fields in a MongoDB document to create a compound index. This type of index will use the first field for the initial sort and then sort by the preceding fields.
>db.student.createIndex({class: 1, percentage: -1})

In the above compound index, MongoDB will:

- First sort by the class field
- Then, within each class value, sort by percentage

The index would create a data structure similar to the following:
>db.student.find({},{_id:0}).sort({class:1, percentage:-1})

Multikey index
MongoDB supports indexing array fields. When you create an index for a field containing an array, MongoDB will create separate index entries for every element in the array. These multikey indexes enable users to query documents using the elements within the array.
MongoDB will automatically create a multikey index when encountered with an array field without requiring the user to explicitly define the multikey type.

```
>db.student.insert({"stud_id":1, "stud_name": "Maya", "class": "TYBBA_CA",
"Sem_Marks":[84, 71, 99, 90] , "remark": "Excellent"});
>db.student.insert({"stud_id":2, "stud_name": "Pooja", "class": "TYBBA_CA", " Sem_Marks
":[74, 66, 45, 67]});
>db.student.insert({"stud_id":3, "stud_name": "Ram", "class": "SYBBA_CA", " Sem_Marks
":[70, 78, 81, 86]});
>db.student.insert({"stud_id":4, "stud_name": "Vedant", "class": "SYBBA_CA", " Sem_Marks
":[72, 93, 81, 90],"remark": "Excellent"});
>db.student.insert({"stud_id":5, "stud_name": "Supriya", "class": "TYBBA_CA", " Sem_Marks
":[90, 76, 97, 80],"remark": "Excellent"});
```

Now create an index using the Sem_Marks field.
>db.student.createIndex({Sem_Marks:1})

The above code will automatically create a Multikey index in MongoDB. When you query for a document using the array field (Sem_Marks), MongoDB will search for the first element of the array defined in the **find()** method and then search for the whole matching query.

>db.student.find({Sem_Marks: [70, 78, 81, 86]}, {_id: 0})
Initially, MongoDB will use the multikey index for searching documents where the Sem_Marks array contains the first element (70) in any position. Then, within those selected documents, the documents with all the matching elements will be selected.

Geospatial Index
MongoDB provides two types of indexes to increase the efficiency of database queries when dealing with geospatial coordinate data:

- 2d indexes that use planar geometry which is intended for legacy coordinate pairs used in MongoDB 2.2 and earlier.
- 2dsphere indexes that use spherical geometry.

```
>db.<collection_name>.createIndex( { <location Field> : "2dsphere" } )
```

Text index
The text index type enables you to search the string content in a collection.

```
>db.<collection_name>.createIndex( { <Index Field>: "text" } )
```

Hashed index
MongoDB Hashed index type is used to provide support for hash-based sharding functionality. This would index the hash value of the specified field.

```
>db.<collection_name>.createIndex( { <Index Field> : "hashed" } )
```

MongoDB index properties
You can enhance the functionality of an index further by utilizing index properties.
Sparse index
The MongoDB sparse property allows indexes to omit indexing documents in a collection if the indexed field is unavailable in a document and create an index containing only the documents which contain the indexed field.

```
>db.student.createIndex({remark:1},{sparse: true})
```

Here, if you create an index using the remark field, it will index only three documents as the remark field is present only in three documents.

Partial index
The partial index functionality allows users to create indexes that match a certain filter condition. Partial indexes use the **partialFilterExpression** option to specify the filter condition.

```
>db.student.createIndex({stud_name:1},
{partialFilterExpression: {percentage: { $gte: 80}}})
```

Here, command will create an index for the stud_name field but will only include documents in which the value of the percentage field is greater than or equal to 80.

Unique index
The unique property enables users to create a MongoDB index that only includes unique values. This will:

- Reject any duplicate values in the indexed field
- Limit the index to documents containing unique values

```
>db.student.createIndex({stud_name:1},{unique: true})
```

Here, created index will limit the indexing to documents with unique values in the stud_name field.

Aggregation in MongoDB

In MongoDB, aggregation operations process the data records/documents and return computed results. It collects values from various documents and groups them together and then performs different types of operations on that grouped data like sum, average, minimum, maximum, etc to return a computed result.

MongoDB provides three ways to perform aggregation

1. Aggregation pipeline
2. Map-reduce function
3. Single-purpose aggregation

Aggregation pipeline

In MongoDB, the aggregation pipeline consists of stages and each stage transforms the document. It is a multi-stage pipeline, so in each state, the documents taken as input and produce the resultant set of documents now in the next stage(id available) the resultant documents taken as input and produce output, this process is going on till the last stage.

Stages: Each stage starts from stage operators which are:

Operator	**Description**
$match	It is used for filtering the documents can reduce the amount of documents that are given as input to the next stage.
$project	It is used to select some specific fields from a collection.
$group	It is used to group documents based on some value.
$sort	It is used to sort the document that is rearranging them
$skip	It is used to skip ‘n’ number of documents and passes the remaining documents
$limit	It is used to pass first ‘n’ number of documents thus limiting them.
$unwind	It is used to unwind documents that are using arrays i.e. it deconstructs an array field in the documents to return documents for each element.
$out	It is used to write resulting documents to a new collection

Expressions: It refers to the name of the field in input documents for e.g. { $group : { _id : “**$id**”, total:{$sum: “**$fare**”}}} here **$id** and **$fare** are expressions.
Here in $group _id is Mandatory field, $out must be the last stage in the pipeline and $sum:1 will count the number of documents and $sum:”$fare” will give the sum of total fare generated per id.

Accumulators: These are basically used in the group stage

Accumulator	Description
sum	It sums numeric values for the documents in each group
count	It counts total numbers of documents
avg	It calculates the average of all given values from all documents
min	It gets the minimum value from all the documents
max	It gets the maximum value from all the documents

Example:

```
>db.student.insert({"stud_id":1, "stud_name": "Maya", "age":25,"section": "A",
"subjects":["Python","Java"]});
>db.student.insert({"stud_id":2, "stud_name": "Pooja", "age":20,"section": "B",
"subjects":["PHP"]});
>db.student.insert({"stud_id":3, "stud_name": "Ram", "age":27,"section": "A",
"subjects":["C++","Python","Java"]});
>db.student.insert({"stud_id":4, "stud_name": "Vedant", "age":35,"section": "B",
"subjects":["PHP","Java"]});
>db.student.insert({"stud_id":5, "stud_name": "Supriya", "age":28,"section": "A",
"subjects":["Python","Java","C++","PHP"]});
```

Display only student name and class field of student collection

```
>db.student.aggregate( [ { $project : { _id: 0, stud_name : 1 , age : 1 } } ] );
```

In this example the _id field is excluded and stud_name, and the age field is included in its output documents.

Displaying the total number of students in one section only

```
>db.student.aggregate([{$match:{section:"B"}},{$count:"Total student in sec:B"}])
```

In this example, for taking a count of the number of students in section B we first filter the documents using the **$match operator,** and then we use the **$count** accumulator to count the total number of documents that are passed after filtering from the $match.

Displaying the total number of students in both the sections and maximum age from both section

```
>db.student.aggregate([{$group: {_id:"$section", total_st: {$sum:1}, max_age:{$max:"$age"} }
}])
```

In this example, we use **$group** to group, so that we can count for every other section in the documents, here **$sum** sums up the document in each group and **$max** accumulator is applied on age expression which will find the maximum age in each document.

Displaying details of students whose age is greater than 30 using match stage

```
>db.student.aggregate([{$match:{age:{$gt:30}}}])
```

In this example, we display students whose age is greater than 30. So we use the **$match** operator to filter out the documents.

Sorting the students information on the basis of age

```
>db.student.aggregate([{'$sort': {'age': 1}}])
```

In this example, we are using the **$sort** operator to sort in ascending order we provide ‘age’:1 if we want to sort in descending order we can simply change 1 to -1 i.e. ‘age’:-1.

Displaying details of a student having the largest age in the section – B

```
>db.student.aggregate([{$match:{section:"B"}},{'$sort': {'age': -1}},{$limit:1}])
```

In this example, first, we only select those documents that have section B, so for that, we use the **$match** operator then we sort the documents in descending order using **$sort** by setting ‘age’:-1 and then to only show the topmost result we use **$limit**.

Unwinding students on the basis of subject

Unwinding works on array here in our collection we have array of subjects (which consists of different subjects inside it like Python, Java, PHP, etc) so unwinding will be done on that i.e. the array will be deconstructed and the output will have only one subject not an array of subjects which were there earlier.

```
>db.student.aggregate([{$unwind:"$subjects"}])
```

Map Reduce

Map reduce is used for aggregating results for the large volume of data. Map reduce has two main functions one is a **map** that groups all the documents and the second one is the **reduce** which performs operation on the grouped data.

Syntax:

```
>db.collectionName.mapReduce(mappingFunction, reduceFunction, {out:'Result'});
```

Example:

```
>db.student.insert({"stud_id":1, "stud_name": "Maya", "age":25, "class": "TYBBA_CA",
"percentage":80});
>db.student.insert({"stud_id":2, "stud_name": "Pooja", "age":20, "class": "TYBBA_CA",
"percentage":50});
>db.student.insert({"stud_id":3, "stud_name": "Ram", "age":27, "class": "SYBBA_CA",
"percentage":90});
>db.student.insert({"stud_id":4, "stud_name": "Vedant", "age":35, "class": "SYBBA_CA",
"percentage":85});
>db.student.insert({"stud_id":5, "stud_name": "Supriya", "age":20, "class": "TYBBA_CA",
"percentage":40});
```

Find total percentage in each age group.

```
>var mapfun = function(){emit(this.age, this.percentage)}
>var reducefun = function(key, values){return Array.sum(values)}
>db.student.mapReduce(mapfun, reducefun, {'out':'Result'})
```

Here, we will create two variables first mapfun which will emit age as a key (expressed as “_id” in the output) and percentage as value this emitted data is passed to our reducefun, which takes key and value as grouped data, and then it performs operations over it. After performing reduction the results are stored in a collection here in this case the collection is Result.

To view the output, give following command

```
>db.Result.find()
```

Single Purpose Aggregation

It is used when we need simple access to document like counting the number of documents or for finding all distinct values in a document. It simply provides the access to the common aggregation process using the count() and distinct().

Example:

Displaying distinct ages

```
>db.student.distinct("age")
```

Here, we use a distinct() method that finds distinct values of the specified field i.e., age.

Counting the total numbers of documents

```
>db.student.count()
```

Here, we use count() to find the total number of the document, unlike find() method it does not find all the document rather it counts them and return a number.

Practice Set:

For a collection, Movies created in your database with fields Movie_id, title, director, actors, num_of_award, box_office business in crores with multiple documents

i) Find movie which made highest business
ii) Find movies which received maximum awards
iii) Find movies which received no awards

Set A:

For a collection **people** created in your database containing the following type of documents:

```
{
  "_id": 1,
  "person": { name: "John", surname: "Brown" },
  "age": 34,
  "city": "New York"
}
```

1. Define, a single field index on the **age** field and also drop created index
2. Define, a multiple field index on the age field for descending and city field for ascending order.
3. Write mongo DB queries that will use the index both for retrieving the documents and for sorting. Also, retrieves all the indexes in the collection.

Set B:

In the collection created in your database you have the following data:

```
{
_id:
book_title: ,
description: ,
author:,
url:,
tags: ['mongodb', 'database', 'NoSQL'],
likes: 100
}
```

with multiple documents.

1. Display a list stating how many books are written by each author.
2. Gets the maximum likes of the corresponding values from all documents in the collection for each author.
3. Calculates the average likes of all given values from all documents in the collection for each author.

Set C:

1. For above collection created in database **use map reduce** function to
   i) Find total likes received to each author.
   ii) Calculate total likes received to authors except “Mongo DB” book.

Assignment Evaluation

0: Not Done [ ]	**1: Incomplete [ ]**	**2: Late Complete [ ]**
3: Needs Improvement [ ]	**4: Complete [ ]**	**5: Well Done [ ]**

Signature of Instructor

Section – II
Python Programming

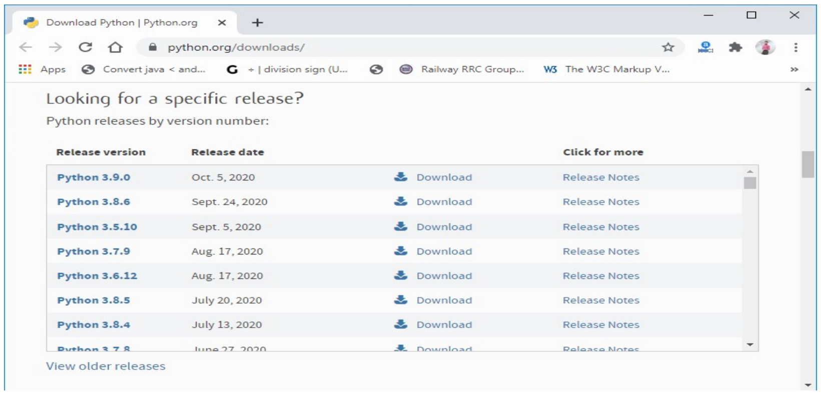
Assignment 1: Introduction to Basic Python

Basic Python:
Python is an interpreted high-level programming language for general-purpose programming. Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

How to Install Python (Environment Set-up):
In order to become Python developer, the first step is to learn how to install or update Python on a local machine or computer. Now, we will discuss the installation of Python on windows operating systems.

Visit the link https://www.python.org/downloads/ to download the latest release of Python. In this process, we will install Python 3.8.6 on our Windows operating system. When we click on the above link, it will bring us the following page.

Step - 1: Select the Python's version to download.
Click on the download button.



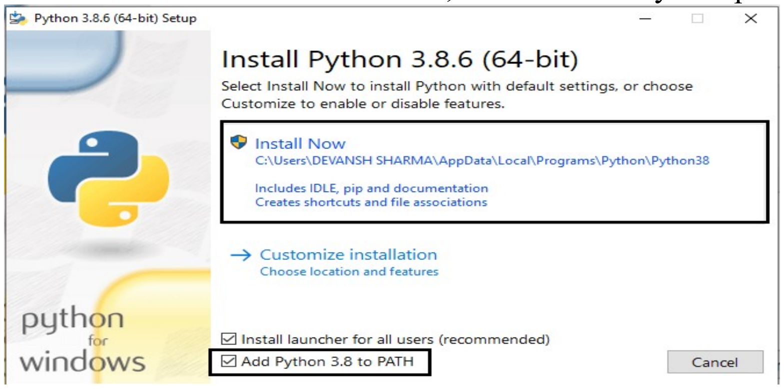
Looking for a specific release?
Python releases by version number:

Release version	Release date		Click for more
Python 3.9.0	Oct. 5, 2020	Download	Release Notes
Python 3.8.6	Sept. 24, 2020	Download	Release Notes
Python 3.5.10	Sept. 5, 2020	Download	Release Notes
Python 3.7.9	Aug. 17, 2020	Download	Release Notes
Python 3.6.12	Aug. 17, 2020	Download	Release Notes
Python 3.8.5	July 20, 2020	Download	Release Notes
Python 3.8.4	July 13, 2020	Download	Release Notes
Python 3.7.8	June 27, 2020	Download	Release Notes

View older releases

Step - 2: Click on the Install Now

Double-click the executable file, which is downloaded; the following window will open. Select Customize installation and proceed. Click on the Add Path check box, it will set the Python path automatically.



Install Python 3.8.6 (64-bit)
Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

Install Now
C:\Users\DEVANSH SHARMA\AppData\Local\Programs\Python\Python38

Includes IDLE, pip and documentation
Creates shortcuts and file associations

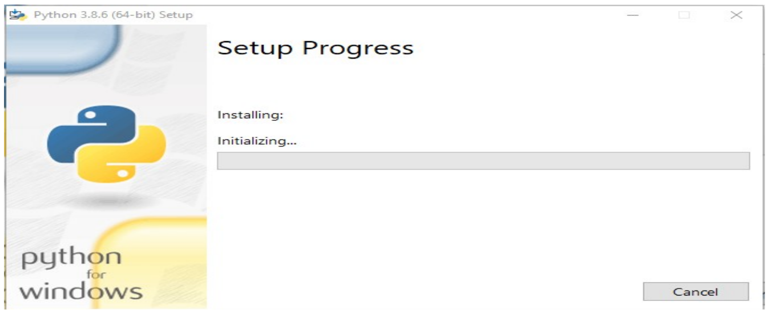
→ Customize installation
Choose location and features

☑ Install launcher for all users (recommended)
☑ Add Python 3.8 to PATH

Cancel

We can also click on the customize installation to choose desired location and features. Other important thing is install launcher for the all user must be checked.

Step - 3 Installations in Process



Starting the Interpreter:
After installation, the python interpreter lives in the installed directory. By default it is /usr/local/bin/python. X in Linux/Unix and C:\PythonXX in Windows, where the 'X' denotes the version number. To invoke it from the shell or the command prompt we need to add this location in the search path. Search path is a list of directories (locations) where the operating system searches for executables.
For example,
In Windows command prompt, we can type set path=%path%;c:\python33 (python33 means version 3.3, it might be different in your case) to add the location to path for that particular session.
Python Use:
Python is used by hundreds of thousands of programmers and is used in many places. Python has many standard library which is made up of many functions that come with Python when it is installed. On the Internet there are many other libraries available that make it possible for the Python language to do more things. These libraries make it a powerful language; it can do many different things.
Some things that Python is often used for are:

- Web development
- Game programming
- Desktop GUIs
- Scientific programming
- Network programming.

First Python Program:
This is a small example of a Python program. It shows "Hello World!" on the screen. Type the following code in any text editor or an IDE and save as helloWorld.py

```
print ("Hello world!")
```

Now at the command window, go to the location of this file. You can use the cd command to change directory. To run the script, type python helloWorld.py in the command window. We should be able to see the output as follows:

```
Hello world!
```

Immediate/Interactive mode:
Typing python in the command line will invoke the interpreter in immediate mode. We can directly type in Python expressions and press enter to get the output.
>>> is the Python prompt. It tells us that the interpreter is ready for our input. Try typing in 1 + 1 and press enter. We get 2 as the output. This prompt can be used as a calculator. To exit this mode type exit() or quit() and press enter.
Type the following text at the Python prompt and press the Enter

```
>>> print "Hello World"
```

2. Script Mode Programming
Invoking the interpreter with a script parameter begins execution of the script and continues until the script is finished. When the script is finished, the interpreter is no longer active.
Let us write a simple Python program in a script. Python files have extension .py. Type the following source code in a test.py file −

```
print"Hello World"
```

We assume that you have Python interpreter set in PATH variable. Now, try to run this program as follows

```
$ python test.py
```

This produces the following result −

```
Hello, Python!
```

3. Integrated Development Environment (IDE)
We can use any text editing software to write a Python script file.
We just need to save it with the .py extension. But using an IDE can make our life a lot easier. IDE is a piece of software that provides useful features like code hinting, syntax highlighting and checking, file explorers etc. to the programmer for application development. Using an IDE can get rid of redundant tasks and significantly decrease the time required for application development.
IDEL is a graphical user interface (GUI) that can be installed along with the Python programming language and is available from the official website. We can also use other commercial or free IDE according to our preference, the PyScripter IDE can be used for testing. It is free and open source.

Python Comments:
In Python, there are two ways to annotate your code.
Single-line comment – Comments are created simply by beginning a line with the hash (#) character, and they are automatically terminated by the end of line.
For example:

```
#This would be a comment in Python
```

Multi-line comment:
Multi lined comment can be given inside triple quotes.

```
‘’’This is
example of
multiline comments ‘’’
```

Indentation:
Most of the programming languages like C, C++, Java use braces { } to define a block of code. Python uses indentation. A code block (body of a function, loop etc.) starts with indentation and ends with the first unindented line. The amount of indentation is up to you, but it must be consistent throughout that block. The enforcement of indentation in Python makes the code look neat and clean. This results into Python programs that look similar and consistent. In Python, indentation is used to declare a block. If two statements are at the same indentation level, then they are the part of the same block.

Standard Data Types
Python has five standard data types −

- Numbers
- String

- List
- Tuple
- Dictionary

Python Numbers: Integers, floating point numbers and complex numbers falls under Python numbers category. They are defined as int, float and complex class in Python.
Python supports four different numerical types

- int (signed integers)
- long (long integers, they can also be represented in octal and hexadecimal)
- float (floating point real values)
- complex (complex numbers)

Python Strings: Strings in Python are identified as a contiguous set of characters represented in the quotation marks. Python allows for either pairs of single or double quotes.
Example: str='Hello all'

Python Lists:
Lists are the most versatile of Python's compound data types. A list contains items can be of different data types separated by commas and enclosed within square brackets ([]).
list_obj=['table', 59 ,2.69,"chair"]

Python Tuples:
A tuple is another sequence immutable data type that is similar to the list. A tuple consists of a number of values separated by commas and enclosed in parentheses (( )).
Example:
tuple_obj=(786,2.23, "college")

Python Dictionary
Python's dictionaries are kind of hash table type. They work like associative arrays or hashes found in Perl and consist of key-value pairs.
Dictionaries are enclosed by curly braces ({ })
Example:dict_obj={'roll_no': 15,'name':'xyz','per': 69.88}

Python Operators:
Python language supports the following types of operators.

- Arithmetic Operators
- Comparison (Relational) Operators
- Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

Arithmetic, logical, Relational operators supported by Python language are same as other languages like C,C++.

i. Arithmetic Operators:
The new arithmetic operators in python are,

a) ** (Exponent) - Performs exponential (power) calculation on operators
Example: a**b =10 to the power 20
b) // (Floor Division) - The division of operands where the result is the quotient in which the digits after the decimal point are removed. But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity)
Example: 9//2 = 4 and 9.0//2.0 = 4.0, -11//3 = -4, -11.0//3 = -4.0

ii. Logical operators:

Logical operators are the and, or, not operators.
a) and - True if both the operands are true
b) or - True if either of the operands is true
c) not - True if operand is false (complements the operand)

iii. Relational / Comparison operators:

== (equal to), != (not equal to), < (less than),<= (Less than or equal to), > (greater than) and >= (Greater than or equal to) are same as other language relational operators.
The new relational operator in python is, <>- If values of two operands are not equal, the condition becomes true.
Example: (a <> b) is true. This is similar to != operator.

iv. Operators: The following are operators in python which are same as in C, C++. =, +=, - =, *=, /=, %=, **=, //=

v. Bitwise Operators: The following are bitwise operators in python which are same as in C,C++. &(bitwise AND), |(bitwise OR) ,^ (bitwise XOR),~ (bitwise NOT),<<(bitwise left shift), >>(bitwise right shift)

vi. Membership operators:

in and not in are the membership operators; used to test whether a value or variable is in a sequence.
in - True if value is found in the sequence
not in - True if value is not found in the sequence

vii. Identity operators:

is and is not are the identity operators both are used to check if two values are located on the same part of the memory. Two variables that are equal does not imply that they are identical.
is - True if the operands are identical
is not - True if the operands are not identical

Decision making Statement:

Python programming language provides following types of decision making statements.

i. **If statement**: It is similar to that of other languages
Syntax

```
if expression:
        statement(s)
```

ii**. IF...ELIF...ELSE** Statements:
Syntax

```
if expression:
    statement(s)
else:
    statement(s)
```

iii. **nested IF** statements:
In a nested if construct, you can have an if...elif...else construct inside another if...elif...else construct.
Syntax

```
if expression1:
    statement(s)
elif expression2:
    statement(s)
elif expression3:
    statement(s)
else:
    statement(s)
```

Python Loops:

i. **while loop:**
A while loop statement in Python programming language repeatedly executes a target statement as long as a given condition is true.
Syntax

```
while expression:
    statement(s)
```

Example:

```
count=0
while(count <3):
    print('The count is:', count)
    count= count +1
```

ii**. for loop:**
It has the ability to iterate over the items of any sequence, such as a list or a string.
Syntax

```
for iterating_var in sequence:
    statements(s)
```

Example:

```
for x in'Hi':
    print x
```

Command Line Arguments:
You can get access to the command line parameters using the sys module. len(sys.argv) contains the number of arguments. To print all of the arguments simply execute str(sys.argv)

```
import sys
print('Arguments:', len(sys.argv))
print('List:', str(sys.argv))
```

Python Input and Output:
The functions like input() and print() are widely used for standard input and output operations respectively.

Python Output Using print() function:
We use the print() function to output data to the standard output device (screen).
For example

```
>>>print("Python is very easy")  Output: Python is very easy
>>> a=5
>>> print("The value of a is ",a)  Output: The value of a is 5
```

In the second print() statement, we can notice that a space was added between the string and the value of variable a. This is by default. The actual syntax of the print() function is
print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)

Here, objects is the value(s) to be printed.
The sep separator is used between the values. It defaults into a space character.
After all values are printed, end is printed. It defaults into a new line.
The file is the object where the values are printed and its default value is sys.stdout (screen).

```
>>>print(1,2,3,4)                    output: 1 2 3 4
>>>print(1,2,3,4,sep='*')            Output:1*2*3*4
>>>print(1,2,3,4,sep='#',end='&')    Output: 1#2#3#4&
```

Python Input: The function input() is used to accept the input from user Syntax of input function is input(string)

```
Eg. >>>num=input("Enter any number=")
>>> print(num)
Output: Enter  any number=10
10
```

In python each input is accepted in the form of string. To convert it into number we can use int() or float() function

```
Eg.     >>>num1=int(input("Enter any Number"))

        >>>num2=float(input("Enter anyNumber"))
```

Assignments:

Practice Set:

1. A cashier has currency notes of denomination 1, 5 and 10. Write python script to accept the amount to be withdrawn from the user and print the total number of currency notes of each denomination the cashier will have to give.
2. Write a python script to accepts annual basic salary of an employee and calculates and displays the Income tax as per the following rules.

Basic: < 2,50,000	Tax = 0
Basic: 2,50,000 to 5,00,000	Tax = 10%

Basic: > 5,00,000 Tax = 20

3. Write python script to accept the x and y coordinate of a point and find the quadrant in which the point lies.
4. Write a python script to accept the cost price and selling price from the keyboard. Find out if the seller has made a profit orloss and display how much profit or loss has been made.

Set A:

1. Write python script to calculate sum of digits of a given input number.
2. Write python script to check whether a input number is Armstrong number or not.
3. Write python script to check whether a input number is perfect number of not.
4. Write a program to calculate X^Y
5. Write a program to check whether a input number is palindrome or not.
6. Write a program to calculate sum of first and last digit of a number.

Set B:

1. Write a program to accept a number and count number of even, odd, zero digits within that number.
2. Write a program to accept a binary number and convert it into decimal number.
3. Write a program which accepts an integer value as command line and print "Ok" if value is between 1 to 50 (both inclusive) otherwise it prints "Out of range"
4. Write a program which accept an integer value 'n' and display all prime numbers till 'n'.
5. Write python script to accept two numbers as range and display multiplication table of all numbers within that range.

Set C:

1. Write a python script to generate the following pattern upto n lines

```
         1
      1  2  1
   1  2  3  2  1
1  2  3  4  3  2  1
```

Evaluation

0: Not Done [ ] **1: Incomplete [ ]** **2: Late Complete [ ]**

3: Need Improvement [ ] **4: Complete [ ]** **5: Well Done [ ]**

Signature of the Instructor

Assignment 2: Working with String and List

Python String:

In python string is sequence of characters enclosed either in single, double or triple quotes
Ex. str1="Python"; str2='Hi'; str3='''Hi''';
String "Python" will be stored from index 0 as shown in following figure

0	1	2	3	4	5
P	**y**	**t**	**h**	**o**	**n**

Following are the methods of accessing string in python

```
>>>print(str1)                  #output: Python
>>>print(str1[0])               #output: P
>>>for i in str:
            print(i,end=" ")              #output: P  y  t  h  o  n
>>>for i in range(len(str1)):
            print(str1[i],end=" ")        #output: P  y  t  h  o  n
```

Types of Strings:

There are two types of Strings supported in Python:

a) Single line String- Strings that are terminated within a single line are known as Single line Strings.
Eg:>>> text1='hello'

b) Multi line String: A piece of text that is spread along multiple lines is Multiple line String.
There are two ways to create Multiline Strings:

1. Adding black slash at the end of each line.

```
Eg:     >>> text1='hello\
                 user'
        >>> text1              #output: hellouser
```

2. Using triple quotation marks:

```
Eg:     >>> str2='''welcome
                 to
                 SSSIT'''
        >>> print str2
                 # output:     welcome
                               to
                               SSSIT
```

String Operators:

1. **+ :** It is concatenation operator used to connect two strings
Ex: str1="Hi"; str2="Hello";

```
 print(str1+str2)              #output: HiHello
 print(str2+str1)              #output: HelloHi
```

2. * : It is repetition operator used to connect multiple copies of the same string with itself
Ex: str="Python"

```
print(str*3)          #output: PythonPythonPython
```

3. **[ ] :** It is slice operator used to access character of a string from specific index
Ex: print(str[4])   #output: o

4. **[:]** :It is called as rang slice operator used to access substring of string from the specific range

Ex: print(str[2:4]) #output: th

5. **in:** It is membership operator which return True if substring is available in specified string, False otherwise
   Ex. str="Python";
   print("th" in str)   #output: True
   print("a" in str)   #output: False
6. **not in:** it is also membership operator which return true if a substring does not available in specified string, True otherwise
   Ex. print("Py" not in str)   #output: False
           print("xyz" not in str)   #output: True
7. **r/R:** It is used to specify raw string. Raw string are string which treat backslash(\) as a string literal.
   Ex. Print("Hi\nHello")   #output: Hi
                                                   Hello
           print(r"Hi\nHello")   #output: Hi\nHello

Escape Characters:
Following table is a list of escape or non-printable characters that can be represented with backslash notation.
An escape character gets interpreted; in a single quoted as well as double quoted strings.

Backslash	Notation
\a	Bell or alert
\b	Backspace
\cx or \C-x	Control-x
\f	Formfeed
\M-\C-x	Meta-Control-x
\n	Newline
\r	Carriage return
\s	Space
\t	tab
\nnn	Octal notation, where n is in the range 0.7
\v	Vertical tab
\x	Character x

Built in string Methods:

Method	**Description**
capitalize()	Converts the first character to upper case
casefold()	Converts string into lower case
center()	Returns a centered string
count()	Returns the number of times a specified value occurs in a string
encode()	Returns an encoded version of the string
endswith()	Returns true if the string ends with the specified value

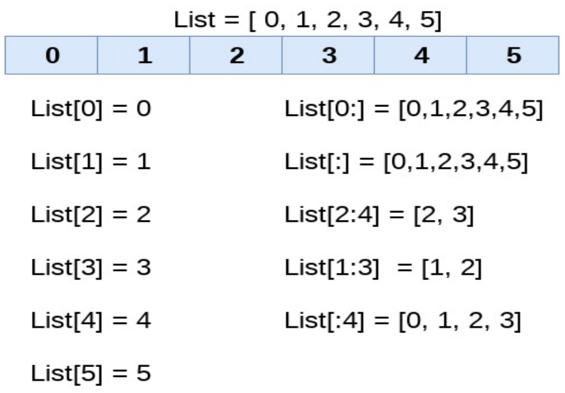
expandtabs()	Sets the tab size of the string
find()	Searches the string for a specified value and returns the position of where it was found
format()	Formats specified values in a string
format_map()	Formats specified values in a string
index()	Searches the string for a specified value and returns the position of where it was found
isalnum()	Returns True if all characters in the string are alphanumeric
isalpha()	Returns True if all characters in the string are in the alphabet
isascii()	Returns True if all characters in the string are ascii characters
isdecimal()	Returns True if all characters in the string are decimals
isdigit()	Returns True if all characters in the string are digits
isidentifier()	Returns True if the string is an identifier
islower()	Returns True if all characters in the string are lower case
isnumeric()	Returns True if all characters in the string are numeric
isprintable()	Returns True if all characters in the string are printable
isspace()	Returns True if all characters in the string are whitespaces
istitle()	Returns True if the string follows the rules of a title
isupper()	Returns True if all characters in the string are upper case
join()	Converts the elements of an iterable into a string
ljust()	Returns a left justified version of the string
lower()	Converts a string into lower case
lstrip()	Returns a left trim version of the string
maketrans()	Returns a translation table to be used in translations
partition()	Returns a tuple where the string is parted into three parts
replace()	Returns a string where a specified value is replaced with a specified value
rfind()	Searches the string for a specified value and returns the last position of where it was found
rindex()	Searches the string for a specified value and returns the last position of where it was found
rjust()	Returns a right justified version of the string
rpartition()	Returns a tuple where the string is parted into three parts

rsplit()	Splits the string at the specified separator, and returns a list
rstrip()	Returns a right trim version of the string
split()	Splits the string at the specified separator, and returns a list
splitlines()	Splits the string at line breaks and returns a list
startswith()	Returns true if the string starts with the specified value
strip()	Returns a trimmed version of the string
swapcase()	Swaps cases, lower case becomes upper case and vice versa
title()	Converts the first character of each word to upper case
translate()	Returns a translated string
upper()	Converts a string into upper case
zfill()	Fills the string with a specified number of 0 values at the beginning

Python List:
A list can be defined as a collection of values or items of different types.
The items in the list are separated with the comma (,) and enclosed with the square brackets [].

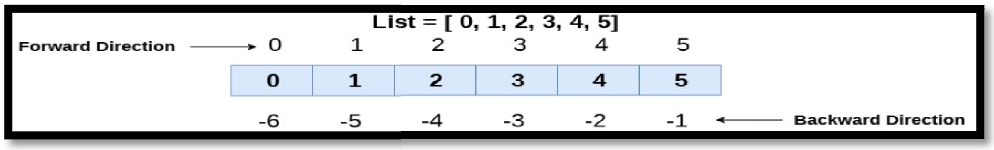
```
Ex.     >>> List=[10,20,"Hi","Hello",4.5]
        >>> print(List)                              #output: [10, 20, 'Hi', 'Hello', 4.5]
        >>> for i in List:
                        print(i,end=" “)             #output: 10 20 Hi Hello 4.5
        >>> for i in range(len(List)):
                        print(List[i],end=" “)       #output: 10 20 Hi Hello 4.5
        >>>print("%d,%d,%s,%s,%f"%(List[0], List[1], List[2], List[3], List[4]))
                                                 #output: 10,20,Hi,Hello,4.500000
```

List indexing and splitting:
The elements of the list can be accessed using the slice operator [].
The index starts from 0 and goes to length - 1. The first element of the list is stored at the 0th index, the second element of the list is stored at the 1st index, and so on.
Consider the following example.



```
List = [ 0, 1, 2, 3, 4, 5]

| 0 | 1 | 2 | 3 | 4 | 5 |

List[0] = 0          List[0:] = [0,1,2,3,4,5]
List[1] = 1          List[:] = [0,1,2,3,4,5]
List[2] = 2          List[2:4] = [2, 3]
List[3] = 3          List[1:3]  = [1, 2]
List[4] = 4          List[:4] = [0, 1, 2, 3]
List[5] = 5
```

The negative indices are counted from the right.
The last element (right most) of the list has the index -1, its adjacent left element is present at the index -2 and so on until the left most element is encountered.



```
Ex. >>> print(List[-1])                 #output: 5
```

Updating List Values: List values can be updated by using the slice and operator.

```
Ex.             >>>list=[1,2,3,4]
                >>>print(list)          #: [1,2,3,4]
                >>>list[2]=45
                >>>print(list)          #:[1,2,45,4]
                >>>list[1:3]=[55,56]
                >>>print(list)          #:[1,55,56,4]
```

Deleting List Element:
The list elements can be deleted by using the del keyword.

```
>>> list=[10,20,30,40]
>>> print(list)                 #[10, 20, 30, 40]
>>> del list[0]
>>> print(list)                 # [20, 30, 40]
>>> del list[2]
>>> print(list)                 # [20, 30]
```

Python List Operator:
The concatenation (+) and repetition (*) operator work in the same way as they were working with the strings.

```
Ex.     >>>L1=[10,20,30];
        >>>L2=[40,50]
        >>>print(L1*2)                  #[10,20,30,10,20,30]
        >>>print(L1+L2)                 #[10,20,30,40,50]
```

Membership operator in and not in can also be used to perform operation on list

```
Ex. >>>L1=[10,20,30]
        >>> print (20 in L1)                    #True
        >>>print (40 in L1)                     #False
        >>>print(50 not in L1)                  #True
        >>>print(30 not in L1)                  #False
```

Adding elements to the list:

- Python provides append() function by using which we can add an element to the list. However, the append() method can only add the value to the end of the list.
- Ex.>>>L1.append(40)

```
        >>>print(L1)            #[10,20,30,40]
```

Removing specific elements from the list:

- remove() method is used to delete specific element for list

```
Ex. >>>L1=[10,20,30]
        >>>print(L1)                    # [10,20,30]
```

```
>>> L1.remove(20)
>>>print(L1)                    #[10,30]
```

Python List Built-in functions:

1. cmp(list1, list2): used to compare two list. The function return 0 if both the list are same,-1 otherwise

   Ex. >>>L1=[10,20]; L2=[20,30]; L3=[10,20]
   ```
   >>>cmp(L1,L2)       # -1
   >>>cmp(L1,L3)       # 0
   ```
2. len(list): It is used to calculate the length of the list.
   ```
   >>>print(len(L1))      # 2
   ```
3. max(list): It returns the maximum element of the list.
4. min(list): It returns the minimum element of the list.
   ```
   Ex.   >>>L1=[16,3,24,12]
         >>>print(max(L1))            #24
         >>>print(min(L1))            # 3
   ```
5. list(seq): It converts any sequence to the list.
   ```
   Ex.   >>>str="Python"
         >>>L1=list(str)
         >>>print(L1)  #['P','y','t','h','o','n']
   ```
6. list.count(obj.): It returns the number of occurrences of the specified object in the list.

```
Ex. >>>L1=[10,20,40,10,20]
    >>>print(L1.count(20))        #2
```

7. list.extend(seq): The sequence represented by the object seq is extended to the list.

```
Ex. >>> L1=[10,20]; str="abc"
    >>>print(L1)            #[10,20]
    >>>print(L1.extend(str))       #[10,20,'a','b','c']
```

8. list.index(obj): It returns the lowest index in the list that object appears.

```
Ex.   >>>L1=[10,20,30,20]
      >>> print(L1.index(20))       # 1
```

9. list.insert(index, obj): The object is inserted into the list at the specified index.

```
Ex    >>>L1=[10,20,30]
      >>>L1.insert(2,40)
      >>>print(L1)                  #[10,20,40,30]
```

10. list.pop(): It removes and returns the last object of the list.

```
Ex    >>>L1=[10,20,30]
      >>print(L1.pop())             #30
```

11. list.reverse(): It reverses the list.

```
Ex.   >>>print(L1.reverse())        # [30 20 10]
```

12. list.sort(): It sorts the list

Indexing, Slicing:

we will focus on indexing and slicing operations over Python's lists.

Indexing:

In Python, list is just like arrays in other scripting languages. It allows you to store set of items in one place and access an item by its index. In python list, index starts form 0

Ex. City=['Pune', 'Mumbai', 'Nasik', 'Nagpur']

Each item in the list have value and index. First value of the list city is 'Pune' having index 0, second item in the list is 'Mumbai' having index 1 and so on

To accsess the element by index we use square brackets

```
>>> print(city[0], city[2])       #’Pune’  ‘Nasik’
```

Negative indexing:

Python list also support negative indexing system.

In negative indexing system last element of the list corresponds to index -1, second last element correspond to -2 and so on

```
>>> print(city[-1],city[-2])     # ‘Nagpur’  ‘Nasik’
```

Indexing not only used for accessing the content of a list but also it is possible to change list content using an operation

```
Ex. >>> print(city)     #[‘Pune’, ’Mumbai’, ‘Nasik’, ‘Nagpur’]
    >>> city[0]=“Pimpri’
    >>>print(city)     #[‘Pimpri’, ’Mumbai’, ‘Nasik’, ‘Nagpur’]
    >>> city[-1]=‘Jalgaon’
    >>> print(city)    #[‘Pimpri’, ’Mumbai’, ‘Nasik’, ‘Jalgaon’]
```

Deletion:

We can also easily delete any element from the list by using indexing and del statement

```
Ex. >>>print(city)      #[‘Pune’, ’Mumbai’, ‘Nasik’, ‘Nagpur’]
         >>> del city[0]
        >>> print(city)#[’Mumbai’, ‘Nasik’, ‘Nagpur’]
         >>> del city(-2)
        >>> print(city)  #[’Mumbai’,‘Nagpur’]
```

Slice Notation:

As it was shown, indexing allows you to access/change/delete only a single cell of a list.

Using slice notation we can access/change/delete sublist of the list.

The full slice syntax is [start: stop: step]

Start refers to the index of the element which is used as a start of our slice.

Stop refers to the index of the element we should stop just before to finish our slice.

Step allows you to take each nth-element within a start: stop range.

```
Ex.      >>>L=[10,20,30,40,50,60,70,80]
>>>print(L[2:6])                    # [30,40,50,60]
>>>print(L[:4])                     #[10,20,30,40]
>>>print(L[0:])                     #[10,20,30,40,50,60,70,80]
>>>print(L[:])              #[10,20,30,40,50,60,70,80]
>>>print(L[-6:4])           #[30,40]
>>>print(L[-3:])                    #[60,70,80]
>>>print(L[1:-1])           #[20,30,40,50,60,70]
>>>print(L[-5:-2])          #[40,50,60]
>>>print(L[1:8:2])          #[20,40,60]
>>>print(L[: :3])                   #[10,30,60]
>>>print(L[-2:1:-3])         #[70,40]
```

Slicing and Coping List:

```
>>>L1=[10,20,30,40,50,60]
>>>L2=L1[1:4]
>>>print(L2)  #[20,,30,40]
```

Slice :

```
>>>L1[:3]=[7,8,9]
>>>print(L1)  #[7,8,9,40,50,60]
```

Replace and Resize part of the list:
It's also possible to replace a bigger chunk with a smaller number of items:

```
>>>L1=[10,20,30,40,50,60]
>>>L1[:3]=[1]
>>>print(L1)  #[1,40,50,60]
```

We can also replace part of list with bigger chunk

```
>>>L1=[10,20,30,40,50,60]
>>>L1[:3]=[6,7,8,9,25]
>>>print(L1)  #[6,7,8,9,25,40,50,60]
```

Slice Deletion:
We can use del statement to remove a slice out of a list:

```
>>>L1=[10,20,30,40,50,60]
>>>del L1[2:5]
>>>print (L1)  #[10,20,60]
```

- We can also provide step parameter to slice and remove each n-th element:

```
>>>L1=[10,20,30,40,50,60,70,80]
>>>del L1[ : : 2]
>>>print(L1)  #[20,40,60,80]
```

Assignments:

Practice Set

1. Write a python script to create a list and display the list element in reverse order
2. Write a python script to display alternate characters of string from both the direction.
3. Write a python program to count vowels and consonants in a string.

Set A

1. Write a python script which accepts 5 integer values and prints "DUPLICATES" if any of the values entered are duplicates otherwise it prints "ALL UNIQUE". Example: Let 5 integers are (32, 45, 90, 45, 6) then output "DUPLICATES" to be printed.
2. Write a python script to count the number of characters (character frequency) in a string. Sample String : google.com'. Expected Result : {'o': 3, 'g': 2, '.': 1, 'e': 1, 'l': 1, 'm': 1, 'c': 1}
3. Write a Python program to remove the characters which have odd index values of a given string.
4. Write a program to implement the concept of stack using list
5. Write a Python program to get a string from a given string where all occurrences of its first char have been changed to '$', except the first char itself. Sample String: 'restart' Expected Result : 'resta$t'

Set B

1. Write a Python program to get a string made of the first 2 and the last 2 chars from a given a string. If the string length is less than 2, return instead of the empty string.
   Sample String : 'General12'
   Expected Result : 'Ge12'
   Sample String : 'Ka'
   Expected Result : 'KaKa'
   Sample String : ' K'

Expected Result : Empty String

2. Write a Python program to get a single string from two given strings, separated by a space and swap the first two characters of each string.
   Sample String : 'abc', 'xyz'
   Expected Result : 'xycabz'
3. Write a Python program to count the occurrences of each word in a given sentence.
4. Write a program to implement the concept of queue using list
5. Write a python program to count repeated characters in a string.
   Sample string: 'thequickbrownfoxjumpsoverthelazydog'
   Expected output:
   ```
   o 4
   e 3
   u 2
   h 2
   r 2
   t 2
   ```

Set C:

1. Write a binary search function which searches an item in a sorted list. The function should return the index of element to be searched in the list.

Evaluation

0: Not Done [ ] **1: Incomplete [ ]** **2: Late Complete [ ]**

3: Need Improvement [ ] **4: Complete [ ]** **5: Well Done [ ]**

Signature of the Instructor

Assignment 3: Working With Tuples, Sets and Dictionaries

Python Tuple:

A Tuple is a collection of Python objects separated by commas or written in round brackets.

```
Ex      >>>T1="Hi", "Hello"
        >>>print(T1)            #output: ( 'Hi', 'Hello')
        >>>T2=(10,20,4.5,"Monday")
        >>>print(T2)  #output: (10,20,4.5,'Mondy')
```

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.

```
        >>>print(T1[0])         #'Hi'
        >>>print(T1[-1])        #'Hello'
        >>>T1[0]="Bye"          #Error
```

Change Tuple Values:

Once a tuple is created, you cannot change its values. Tuples are unchangeable, or immutable

For changing tuple values you can convert the tuple into a list, change the list, and convert the list back into a tuple.

Ex

```
>>>T1=("Hi", "Hello")           #Tuple T1
>>> T1[0]="Bye"                 #Error
>>>T2=list(T1)                  #Converting Tuple T1 to list T2
>>>T2[0]="Bye"                  #Updating List T2
>>>T1=tuple(T2)                 #Converting List T2 to tuple T1
>>>print(T1)                    #('Bye', 'Hello')
```

The empty tuple is written as two parentheses containing nothing − >>> tup1 = ();

To write a tuple containing a single value you have to include a comma, even though there is only one value - >>>tup1 = (50,);

Like string indices, tuple indices start at 0, and they can be sliced, concatenated, and so on.

Different ways of accessing Tuple

```
        >>>T1=(10,20,30,40)
        >>>print(T1)                    #(10,20,30,40)
        >>>print(T1[0])                 #10
        >>>for i in T1:
                print(i,end= "  ")      #10     20      30      40
        >>>for i in range(len(T1)):
                print(T1[i],end= " ")   #10     20      30      40
        >>>print("%f"%T1[0])            #10.000
```

Python Tuple Operator:

The concatenation (+) and repetition (*) operator work in the same way as they were working with the strings.

```
        Ex.     >>>T1=(10,20,30);
                        >>>L2=(40,50)
                        >>>print(L1*2)          #(10,20,30,10,20,30)
                        >>>print(L1+L2)         #(10,20,30,40,50)
```

Membership operator in and not in can also be used to perform operation on list

```
Ex. >>>T1=[10,20,30]
        >>> print (20 in T1)          #True
        >>>print (40 in T1)           #False
        >>>print(50 not in T1)        #True
        >>>print(30 not in T1)        #False
```

Add Items:
Once a tuple is created, you cannot add items to it. Tuples are unchangeable.
Remove Items:
Note: You cannot remove items in a tuple.
Tuples are unchangeable, so you cannot remove items from it, but you can delete the tuple completely: The del keyword can delete the tuple completely

```
Ex.     >>>T1=(10,20)
        >>>del T1
```

In-Built Tuple Functions:

1. cmp(): Python tuple method cmp() compares elements of two tuples.
   syntax: cmp(tuple1, tuple2)
   cmp function return 0 if two tuples are same, 1 if elements of first tuple is greater than elements of second tuple, otherwise -1
2. len(tuple): It is used to calculate the length of the tuple
   ```
   >>>T1=(10,20,30)
   >>>print(len(T1))      # 3
   ```
3. max(Tuple): It returns the maximum element of the tuple.
4. min(Tuple): It returns the minimum element of the tuple.
   ```
   >>>print(max(T1)              #30
   >>>print(min(T1))             #10
   ```
5. sum(tuple): Return sum of all elements within tuple
   ```
   >>> print(sum(T1))            #60
   ```
6. all(tuple): it returns True if all the items in the tuple are true, otherwise it returns false. If the tuple is empty, the function also returns true.
   ```
   >>>T1=(10,20,30)              T2=(10,0,20)
   >>>print(all(T1))             #True
   >>>print(all(T2))             #False
   ```
7. any(): Python any() function accepts iterable (list, tuple, dictionary etc.) as an argument and return true if any of the element in iterable is true, else it returns false. If iterable is empty then any() method returns false.
   ```
   >>>print(any(T1))      #True
   ```
8. tuple(seq): It converts any sequence to the tuple.
   ```
   >>>str="Python"
   >>>T1=tuple(str)
   >>>print(T1)  #('P','y','t','h','o','n')
   ```
9. tuple.count(obj.): It returns the number of occurrences of the specified object in the tuple.
   ```
    >>>L1=(10,20,40,10,20)
   >>>print(L1.count(20))        #2
   ```
10. tuple.index(obj): It retuurns the lowest index in the tuple that object appears.
    ```
    >>>T1=(10,20,30,20)
    >>> print(T1.index(20))        # 1
    ```

Packing and Unpacking:

In tuple packing, we place value into a new tuple while in tuple unpacking we extract those values back into variables.

```
Ex >>> t=(101,"Nilesh",80.78)          #tuple packing
>>> (rollno, name, marks)=t            #tuple unpacking
>>> print(rollno)                      # 101
>>>print(name, marks)                  #Nilesh  80.78
```

Python Set:

The set in python can be defined as the unordered collection of various items enclosed within the curly braces. The elements of the set can not be duplicate. The elements of the python set can not be changed.There is no index attached to the elements of the set, i.e., we cannot directly access any element of the set by the index. However, we can print them all together or we can get the list of elements by looping through the set.

Creating a set:

The set can be created by enclosing the comma separated items with the curly braces.
Python also provides the set method which can be used to create the set by the passed sequence.

Creating set using curly brackets:

```
>>> city={"Pune", "Mumbai", "Nashik"}
>>> print(city)                        # {'Pune', 'Nashik', 'Mumbai'}
>>> for i in city:
        print(i, end=" ")              # Pune  Nashik  Mumbai
```

Creating set using set() method:

```
>>> names=set(["Shubham", "Nilesh", "Pranav"])
>>>print(names)                        #{'Pranav', 'Shubham', 'Nilesh'}
```

Adding items to the set:

Python provides the add() method which can be used to add some particular item to the set.

```
>>> names.add("Rajesh")
>>> print(names)                  #{'Pranav', 'Shubham', 'Rajesh', 'Nilesh'}
```

To add more than one item in the set, Python provides the update() method.

```
 >>> print(city)                  #{'Pune', 'Nashik', 'Mumbai'}
 >>> city.update(["Jalgaon","Nagpur","Satara"])
>>> print(city)
                    #   {'Satara', 'Jalgaon', 'Pune', 'Mumbai', 'Nagpur', 'Nashik'}
```

Removing items from the set:

Python provides discard() method which can be used to remove the items from the set.

```
>>> city.discard("Mumbai")
>>> print(city)              #   {'Satara', 'Jalgaon', 'Pune', 'Nagpur', 'Nashik'}
```

Python also provide the remove() method to remove the items from the set.

```
>>> print(city)              #{'Satara', 'Jalgaon', 'Pune', 'Nagpur', 'Nashik'}
>>> city.remove("Satara")
>>> print(city)              #{'Jalgaon', 'Pune', 'Nagpur', 'Nashik'}
```

We can also use the pop() method to remove the item. However, this method will always remove the first item.

```
        >>> print(city)         #{'Jalgaon', 'Pune', 'Nagpur', 'Nashik'}
        >>> city.pop()                  #'Jalgaon'
```
Python provides the clear() method to remove all the items from the set.
```
        >>> print(city)                 #{'Nashik', 'Dhule'}
        >>> city.clear()
        >>> print(city)                 #set()
```

Difference between discard() and remove():
If the item to be deleted from the set using discard() doesn't exist in the set, the python will not give the error.
On the other hand, if the item to be deleted from the set using remove() doesn't exist in the set, the python will give the error.

Union of two Sets:
The union of two sets are calculated by using the or (|) operator. The union of the two sets contains the all the items that are present in both the sets.
```
Ex.             >>>s1={1,2,3};         s2={3,4,5}
                >>>s3=s1|s3
                >>>print(s3)                    #{1,2,3,4,5}
```
Python also provides the union() method which can also be used to calculate the union of two sets.
```
                >>> print(s1.union(s2))        #{1,2,3,4,5}
```

Intersection of two sets:
The & (intersection) operator is used to calculate the intersection of the two sets in python.
The intersection of the two sets are given as the set of the elements that common in both sets.
```
Ex      >>> s1={"Pune","Mumbai","Jalgaon"}
        >>> s2={"Nahik","Pune","Nagpur","Jalgaon"}
        >>> s3=s1&s2
        >>> print(s3)                           #{'Jalgaon', 'Pune'}
```
➢ using intersection() method
```
        >>>s3=s1.intersection(s2)
        >>>print(s3)                            #{'Jalgaon', 'Pune'}
```

The intersection_update() method:
The intersection_update() method removes the items from the original set that are not present in both the sets (all the sets if more than one are specified).
The Intersection_update() method is different from intersection() method since it modifies the original set by removing the unwanted items, on the other hand, intersection() method returns a new set.
```
Ex                      >>> a={1,2,3,4,5}
                        >>> b={3,5,7,8,9}
                        >>> a.intersection_update(b)
                        >>> print(a)            #{3, 5}
```

Difference of two sets:
The difference of two sets can be calculated by using the subtraction (-) operator.
The resulting set consists of all the elements form set 1 which are not available in set2
```
Ex      >>> a={1,2,3,4}
        >>> b={3,5,4,8}
        >>> c=a-b
        >>> print(c)                        #{1, 2}
        >>>print(b-a)                       #{5,8}
```
using difference() method
```
>>>print(a.difference(b))           #{1,2}
```

The difference_update():
The set difference_update() method modifies the existing set.
If (A – B) is performed, then A gets modified into (A – B), and if (B – A) is performed, then B gets modified into (B – A).
Ex.
```
>>>a={1,2,3,4,5};  b={3,4,5,6}
>>>a.difference_update(b)
>>>print(a)             #{1,2}
```

The symmetric_difference():
This in-built function of Python Set helps us to get the symmetric difference between two sets, which is equal to the elements present in either of the two sets, but not common to both the sets.
```
>>>print(a.symmetirc_difference(b))#{1,2,6}
```

The symmetric_difference_update method:
symmetric_difference() method returns a new set which contains symmetric difference of two sets.
The symmetric_difference_update() method updates the set calling symmetric_difference_update() with the symmetric difference of sets.
```
>>>a={1,2,3,4};       b={2,3,6,7}
>>>a.symmetric_difference_update(b)
>>>print(a)             #{1,4,6,7}
```

issuperset() in Python:
The issuperset() method returns True if all elements of a set A occupies set B which is passed as an argument and returns false if all elements of B not present in A.
This means if A is a superset of B then it returns true; else False
Syntax: A.issuperset(B) checks whether A is a superset of B or not. True if A is a superset of B; otherwise false.
Ex
```
>>>A={1,2,3,4,5};          B={2,3,4}
>>>A.issuperset(B)         #True
>>>B.issuperset(A)         #False
```
issubset() in python:
returns true if first set is a subset of seconds set otherwise false
```
>>> A.issubset(B)          #False
>>>B.issubset(A)           #True
```

isdisjoint() function in Python:
Two sets are said to be disjoint when their intersection is null.
In simple words they do not have any common element in between them.
Syntax: seta.isdisjoint(setb)
```
>>>a={1,2,3};b={3,4,5};      c={7,8,9}
>>>a.isdisjoint(b)      #False
>>>a.isdisjoint(c)      #True
```

Set comparisons:

Python allows us to use the comparison operators i.e., <, >, <=, >= , == with the sets by using which we can check whether a set is subset, superset, or equivalent to other set.

The boolean True or False is returned depending upon the items present inside the sets.

```
Ex      >>>a={1,2,3,4};        b={1,2,3};      c={1,2,3};
        >>>print(a>b) #True
        >>>print(a<b) #False
        >>>print(b>a) #False
        >>>print(a==b)        #False
        >>>print(b==c)        #True
        >>>print(a>=b)        #True
```

Python Dictionary:

Dictionary in Python is an unordered collection of items in the form of key-value pair.

Dictionary holds key : value pair.

Each key-value pair in a Dictionary is separated by a colon :, whereas each item is separated by a 'comma'.

Keys of a Dictionary must be unique and of immutable data type such as Strings, Integers and tuples, but the values associated with key can be repeated and be of any type.

In Python, a Dictionary can be created by placing sequence of elements within curly {} braces, separated by 'comma'.

```
        Ex              >>> d={1 : 'Hi', 2 : 'Hello', 3: 'Hello'}
                        >>>print(d)    #{1 : 'Hi', 2 : 'Hello', 3: 'Hello'}
```

Different ways of accessing dictionary

```
>>> d={"Rollno":101, "Name":"Nilesh", "Marks":80.75}
```

1. Printing whole dictionary using print() method :

```
>>>print(d)
      {"Rollno":101,"Name":"Nilesh", "Marks": 80.75}
```

2. Accessing Individual value using index:

```
>>>print(d["Name"])                     #Nilesh
```

3. for loop to print all the keys of a dictionary

```
>>>for x in d:
        print(x, end="  ")              # Name Rollno Marks
```

4. for loop to print all the values of the dictionary

```
>>>for x in d:
        print(d[x],end= "  ")    # Nilesh 101 80.75
```

5. for loop to print the values of the dictionary by using values() method

```
>>> for i in d.values()
            print(i,  end="    ")     # Nilesh 101 80.75
```

6. for loop to print the items of the dictionary by using items() method:

```
>>> for in d.items()
            print(i)
                        #output:          ('Name',  'Nilesh')
                                          ('RollNo', 101)
                                          ('Marks', 80.75)
```

7. For loop to print key value pair:

```
 >>> for k,v in d.items():
                print(k,v)
                                        #output:            Name  Nilesh
                                                             Rollno  101
                                                            Marks  80.75
```

8. Printing individual values of dictionary using format specifier:

```
  >>> d={“Rollno”:101, “Name”:“Nilesh”, “Marks”:80.75}
  >>> print("Roll No=%d"%d["Rollno"])
        Roll No=101
   >>> print("Name of student=%s"%d["Name"])
         Name of student=Nilesh
   >>> print("Marks Obtained=%f"%d["Marks"])
        Marks Obtained=80.750000
```

Dictionary can also be created by the built-in function dict(). An empty dictionary can be created by just placing to curly braces{}.

```
>>>D={ }                          # empty dictionary
>>>D=dict({1:”Hi”,2:”Hello”})
>>>print(D)                 #{1:”Hi”,2:”Hello”}
```

Note – Dictionary keys are case sensitive, same name but different cases of Key will be treated distinctly.

Updating Dictionary:
You can update a dictionary by adding a new entry or a key-value pair, modifying an existing entry, or deleting an existing entry

Ex.
```
>>> d={1: “One”, 2: “Two”}
>>>print(d)                 #{1: 'One', 2: 'Two‘}
>>>d[2]=“Twelve”
>>>print(d)                 #{1: 'One', 2: 'Tweleve'}
>>>d[3]=“Three”
>>>print(d)          #{1: 'One', 2: 'Tweleve', 3: 'Three'}
```

Delete Dictionary Elements:
You can either remove individual dictionary elements or clear the entire contents of a dictionary.
You can also delete entire dictionary in a single operation.

Ex.
```
        >>> del d[2]
        >>> print(d)        #{1: 'One', 3: 'Three'}
```

Removing all elements for dictionary

```
>>> print(d)   #{1: 'One', 3: 'Three'}
>>>d.clear()   # remove all entries in dict
>>>print(d)    #{ }
>>>del  d      # delete entire dictionary
>>>print(d)    #Error
```

Properties of Dictionary:

1. In the dictionary, we can not store multiple values for the same keys.
   If we pass more than one values for a single key, then the value which is last assigned is considered as the value of the key.

```
        >>>d={"RN":101,"Name":"Suresh","Marks":80,"Name":"Rajesh"}
```

```
>>> print(d)    #{'Name': 'Rajesh', 'RN': 101, 'Marks': 80}
```

2. In python, the key cannot be any mutable object.
   We can use numbers, strings, or tuple as the key but we can not use any mutable object like the list as the key in the dictionary.
3. Dictionary keys are case sensitive- Same key name but with the different case are treated as different keys in Python dictionaries.

Built-in Dictionary functions:

1. len(): It is used to calculate the length of the dictionary.

Ex
```
>>>d={1: “One”, 2: “Two”}
>>>print(len(d))   #2
```

2. str(dict.): It converts the dictionary into the printable string
```
>>>s=print(str(d))
>>>print(s)             #{1: ‘One', 2: 'Two'}
>>>type(s)              # <class 'str'>
```
3. copy(): It returns a shallow copy of the dictionary.
   Ex
```
>>>d1={1: “One”, 2: “Two”}
>>> d2=d1.copy()
>>>print(d2)            #{1: “One”, 2: “Two”}
```
4. keys(): It returns all the keys of the dictionary.
```
>>> L=list(d1.keys())
>>>print(L)             #[1, 2]
```
5. values(): It returns all the values of the dictionary.
```
>>>L=list(d1.values())
>>>print(L)             #['One', 'Two']
```
6. popitem(): It remove and returns first item of the dictionary
```
>>> print(d1.popitem())              #(1: “One”)
>>> print(d1)                #{2: “Two”}
```
7. pop(key): This method removes the specified item from the dictionary and return the value of specified key.
```
>>>x=d1.pop(1)
>>>print(x)                  # One
```
8. Update(): The update() method inserts the specified items to the dictionary.

Ex
```
>>>print(d)
{'Name': 'Shubham', 'RollNo': 101, 'Marks': 80}
>>> d.update({"Address":"Pimpri"})
>>> print(d)
{'Address': 'Pimpri', 'Name': 'Shubham', 'RollNo': 101,          'Marks': 80}
```

Assignments:

Practice Set:

1. Write a Python program to add and remove operation on set.
2. Write a Python program to do iteration over sets.
3. Write a Python program to find the length of a set.
4. Write a Python program to create a tuple with numbers and print one item.
5. Write a Python script to add a key to a dictionary.
   Sample Dictionary : {0: 10, 1: 20}
   Expected Result : {0: 10, 1: 20, 2: 30}

Set A:

1. Write a Python program to find maximum and the minimum value in a set.
2. Write a Python program to add an item in a tuple.
3. Write a Python program to convert a tuple to a string.
4. Write a Python program to create an intersection of sets.
5. Write a Python program to create a union of sets.
6. Write a Python script to check if a given key already exists in a dictionary.
7. Write a Python script to sort (ascending and descending) a dictionary by value.

Set B:

1. Write a Python program to create set difference and a symmetric difference.
2. Write a Python program to create a list of tuples with the first element as the number and second element as the square of the number.
3. Write a Python program to unpack a tuple in several variables.
4. Write a Python program to get the 4th element from front and 4th element from last of a tuple.
5. Write a Python program to find the repeated items of a tuple.
6. Write a Python program to check whether an element exists within a tuple.
7. Write a Python script to concatenate following dictionaries to create a new one. Sample Dictionary : dic1={1:10, 2:20} dic2={3:30, 4:40} dic3={5:50,6:60}

   Expected Result : {1: 10, 2: 20, 3: 30, 4: 40, 5: 50, 6: 60}

Set C:

1. Write a Python program to create a shallow copy of sets.
2. Write a Python program to combine two dictionary adding values for common keys.

```
        d1 = {'a': 100, 'b': 200, 'c':300}
        d2 = {'a': 300, 'b': 200, 'd':400}
    Sample output: Counter({'a': 400, 'b': 400, 'd': 400, 'c': 300})
```

Evaluation

0: Not Done [ ]	**1: Incomplete [ ]**	**2: Late Complete [ ]**
3: Need Improvement [ ]	**4: Complete [ ]**	**5: Well Done [ ]**

Signature of the Instructor

Assignment 4: Working with Functions, Modules and Packages

Functions in Python:
Function is named, independent block of statements that perform a specific task and may return a value to the calling program.
Function is block of reusable code which can be called whenever required

Creating a function:
In python, we can use def keyword to define the function. Syntax is as follows

```
        def  my_function():
                function-code
                return <expression>
```

The function block is started with the colon (:) and all the same level block statements remain at the same indentation.
A function can accept any number of parameters that must be the same in the definition and function calling.

Function calling
In python, a function must be defined before the function calling otherwise the python interpreter gives an error.
Once the function is defined, we can call it from another function or the python prompt.
To call the function, use the function name followed by the parentheses.

```
def  my_function():                      #function code
        print("Python is very esasy")
my_function()                            #calling function
                                         #output: Python is very easy
```

Function Arguments:
You can call a function by using the following types of arguments −

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

Python Required Arguments:
When we call a function with some values, these values get assigned to the arguments according to their position.
These are the arguments which are required to be passed at the time of function calling with the exact match of their positions in the function call and function definition.
If either of the arguments is not provided in the function call, or the position of the arguments is changed, then the python interpreter will show the error.

Python Keyword Argument:
Python allows functions to be called using keyword arguments. When we call functions in this way, the order (position) of the arguments can be changed. This kind of function call will enable us to pass the arguments in the random order.
Following calls to the above function are all valid and produce the same result.

```
def sum(a,b):
        print(a,b,a+b)
sum(2,3)                 # Required argument  output : 2 3 5
sum(b=2,a=5)             # Keyword argument output: 5 2 7
```

Python Default Arguments:

Function arguments can have default values in Python.
We can provide a default value to an argument by using the operator (=).
Any number of arguments in a function can have a default value. But once we have a default argument, all the arguments to its right must also have default values.
This means to say, non-default arguments cannot follow default arguments.

```
def greet(name,msg="Good Morning"):
        print(msg,name)
greet("Nilesh", "HI")              #HI Nilesh
greet("Suresh", "Hello")           #Hello Suresh
greet("Ramesh")                    #Good Morning Ramesh
```

Variable length Arguments: (Packing Argument)

Sometimes we may not know the number of arguments to be passed in advance.
In such cases, Python provides us the flexibility to pass any number of arguments which are treated as tuples at the function call.
However, at the function definition, we have to define the name of argument preceded with * def greet(*names):

```
for i in names:                        #output:
        print("Hi",i)                  Hi Nilesh
                                       Hi Ramesh
greet("Nilesh", "Ramesh", "Suresh")    Hi Suresh
```

Python Anonymous function lambda:

In Python, anonymous function means that a function is without a name.
The lambda keyword is used to create anonymous functions. It has the following syntax:

lambda arguments : expression

This function can have any number of arguments but only one expression, which is evaluated and returned.
You need to keep in your knowledge that lambda functions are syntactically restricted to a single expression.

```
Ex.     >>>a=lambda x, y : x+y
        >>>print(a(3,4))          # 7
```

Recursion:

Recursion is the process of defining something in terms of itself.
The function which calls itself is called as recursion.
A function can call other functions. It is even possible for the function to call itself. These type of construct are termed as recursive functions.
following is recursive function to calculate sum of digits of a input number

```
def  sumofd(n):
        if n==0:
                return 0
        else
        return n%10+sumofd(n//10)
N=int(input("Enter any Number ="))              #123
print("Sum of digits= ",sumofd(N))              #Sum of Digits=7
```

Unpacking argument lists:
We can use * to unpack the list elements so that all elements of list can be passed as different parameters.
Consider a situation where we have a function that receives four arguments. We want to make call to this function and we have a list of size 4.
If we simply pass list to the function, the call doesn't work.
Following program illustrate unpacking of arguments

```
    def  myfun(a,b,c,d):
        print(a,b,c,d)

    L=[1,2,3,4]
    myfun(L)                #error
    myfun(*L)               #unpacking of argument using *
                            output  : 1 2 3 4
```

We use two operators * (for list, string and tuple) and ** (for dictionaries) for unpacking of argument

```
    def Myfun(a,b,c):
        print(a,b,c)
    T=(10,20,30)
    D={'a':10, 'b':20, 'c':30}
    Myfun(*T)               #10 20 30
    Myfun(**D)              #10 20 30
```

Packing Arguments:
When we don't know how many arguments need to be passed to a python function, we can use Packing to pack all arguments in a tuple.

```
    def myfun(*a):
        print(a)
    def fun(**d):
        print(d)
    myfun(10,20,30)         #packing to tuple          Output: (10, 20, 30)
    fun(x=10, y=20)         #packing to dictionary     Output:{'x':10, 'y':20}
```

Generator Function in Python:
If a function contains at least one yield statement, it becomes a generator function.
Both yield and return statement will return some value from a function.
The difference is that, while a return statement terminates a function entirely, yield statement pauses the function saving all its states and later continues from there on successive calls.

```
    >>> def gen():
            n=1
            yield n
            n=n+1
            yield n
            n=n+1
            yield n
```

```
>>>a=gen()
>>>next(a)          #1
>>>next(a)          #2
>>>next(a)          #3
```

Python Modules:
A python module can be defined as a python program file which contains a python code including python functions, class, or variables. In other words, we can say that our python code file saved with the extension (.py) is treated as the module. We may have a runnable code inside the python module. Modules in Python provides us the flexibility to organize the code in a logical way. To use the functionality of one module into another, we must have to import the specific module.
Example

```
#displayMsg prints a message to the name being passed.
def displayMsg(name):
   print("Hi "+name)
```

Loading the module in our python code. We need to load the module in our python code to use its functionality. Python provides two types of statements as defined below.

1. The import statement
2. The from-import statement

Example:

```
import file;
name = input("Enter the name?")
file.displayMsg(name)
```

The from-import statement:
from < module-name> import <name 1>, <name 2>..,<name n>

calculation.py:

```
  #place the below code in the calculation.py
  def summation(a,b):
     return a+b
  def multiplication(a,b):
     return a*b;
  def divide(a,b):
     return a/b;
```

Main.py:

```
  from calculation import summation
  #it will import only the summation() from calculation.py
  a = int(input("Enter the first number"))
  b = int(input("Enter the second number"))
print("Sum=",summation(a,b))                    #we do not need to specify the module name while accessing
summation()
```

The datetime Module:
The datetime module enables us to create the custom date objects, perform various operations on dates like the comparison, etc. To work with dates as date objects, we have to import the datetime module into the python source code.

Example
```
import datetime
#returns the current datetime object
print(datetime.datetime.now())
```

The calendar module:
Python provides a calendar object that contains various methods to work with the calendars. Consider the following example to print the calendar for the last month of 2018.

Example
```
import calendar;
cal = calendar.month(2020,3)
#printing the calendar of December 2018
print(cal)
```

Packages:
The packages in python facilitate the developer with the application development environment by providing a hierarchical directory structure where a package contains sub-packages, modules, and sub-modules. The packages are used to categorize the application level code efficiently.
Let's create a package named Employees in your home directory.
Consider the following steps.
1. Create a directory with name Employees on path /home.
2. Create a python source file with name ITEmployees.py on the path /home/Employees**.**
```
ITEmployees.py
def getITNames():

        List = ["John", "David", "Nick",    "Martin"]

        return List;
```
3. Similarly, create one more python file with name BPOEmployees.py and create a function getBPONames().

4. Now, the directory Employees which we have created in the first step contains two python modules. To make this directory a package, we need to include one more file here, that is __init__.py which contains the import statements of the modules defined in this directory.

```
__init__.py
from ITEmployees import getITNames

from BPOEmployees import getBPONames
```

5. Now, the directory Employees has become the package containing two python modules. Here we must notice that we must have to create __init__.py inside a directory to convert this directory to a package.

6. To use the modules defined inside the package Employees, we must have to import this in our python source file. Let's create a simple python source file at our home directory (/home) which uses the modules defined in this package.

```
Test.py
import Employees
print(Employees.getNames())
Output:    ["John", "David", "Nick",    "Martin"]
```

Assignments:

Practice Set:

1. Write a Python program to print Calendar of specific month of input year using calendar module
2. Write a Python script to display datetime in various formats using datetime module
    a. Current date and time
    b. Current year
    c. Month of year
    d. Week number of the year
    e. Weekday of the week
    f. Day of year
    g. Day of the month
    h. Day of week
3. Write an anonymous function to find area of circle.

Set A:

1. Write a recursive function which print string in reverse order.
2. Write a python script using function to calculate X^Y
3. Define a function that accept two strings as input and find union and intersection of them.
4. Write a recursive function to calculate sum of digits of a given input number.
5. Write generator function which generate even numbers up to n

Set B:

1. Write a python script to generate Fibonacci terms using generator function.
2. Write python script using package to calculate area and volume of cylinder and cuboids.
3. Write a python script to accept decimal number and convert it to binary and octal number using function.

4. Write a function which print a dictionary where the keys are numbers between 1 and 20
5. (both included) and the values are square of keys
6. Write a generator function which generates prime numbers up to n.

Set C:

1. Write a program to illustrate function duck typing.

Evaluation

0: Not Done [ ]	**1: Incomplete [ ]**	**2: Late Complete [ ]**
3: Needs Improvement [ ]	**4: Complete [ ]**	**5: Well Done [ ]**

Signature of the Instructor

Assignment 5: Python Classes and Objects

Python OOPs Concepts:

Like other object oriented programming languages, python is also an object-oriented language allows us to develop applications using an Object Oriented approach.
In Python, we can easily create and use classes and objects.

Creating classes in python:

In python, a class can be created by using the keyword class followed by the class name and colon.syntax is as follows.

```
        class  class_name:
                " " " Optional Doc String" " "
                #data members
                #member function
```

Documentation string can be accessed using class-name.__doc__ method.

```
                >>>class myclass:
                        " " " This is example of doc string. """
                        rollno=101
                        name="Nilesh"
                        def display(self):
                                print(self.rollno, self.name)
                >>>s=myclass()                          #creating object
                >>>print(myclass.__doc__)               #accessing doc string
                >>>s.display()                          #101 Nilesh
```

self is used to refers to the current class object. It is always the first argument in the function.

Creating an instance of the class:

The syntax to create the instance of the class is given below

```
                        object-name = class-name(arguments)
        Ex.     class computer:
                        cpu=None
                        ram=None
                        def config(self,a,b):
                                 self.cpu=a
                                self.ram=b
                        def display(self):
                                print(self.cpu,self.ram)
                c1=computer()
                c2=computer()
                c1.config("i3",4)
                c1.display()                    #i3  4
                c2.config("i5",6)
                c2.display()                    #i5  6
```

Data abstraction in python:

In python, we can also perform data hiding by adding the double underscore (___) as a prefix to the attribute which is to be hidden.

After this, the attribute will not be visible outside of the class through the object.

```
>>> class myclass:
        a=10
        __b=20
        def display(self):
                print(self.a, self.__b)
>>> m=myclass()
>>> m.a                         #10
>>> m.__b                       #Error
>>> m.display()                 #10 20
>>> myclass.a                   #10
>>> myclass.__b                 #Error
```

Python Constructor:

A constructor is a special type of method (function) which is used to initialize the instance members of the class. Constructors can be of two types.

1. Parameterized Constructor
2. Non-parameterized Constructor

Constructor definition is executed when we create the object of this class.

Creating the constructor in python:

In python, the method __init__ simulates the constructor of the class.

This method is called when the class is instantiated.

We can pass any number of arguments at the time of creating the class object, depending upon __init__ definition.

It is mostly used to initialize the class attributes.

Example of parameterized constructor:

```
>>> class computer:
        def __init__(self,a,b):
                self.cpu=a
                self.ram=b
        def display(self):
                print(self.cpu,self.ram)

>>> c1=computer("i5",6)
>>> c1.display()
        i5 6
>>> c2=computer("i7",8)
>>> c2.display()
        i7 8
```

Example of non parameterized constructor

```
>>> class myclass:
        def __init__(self):
                print("Object is created")

>>> obj=myclass()
Object is created
```

Operator Overloading in Python:

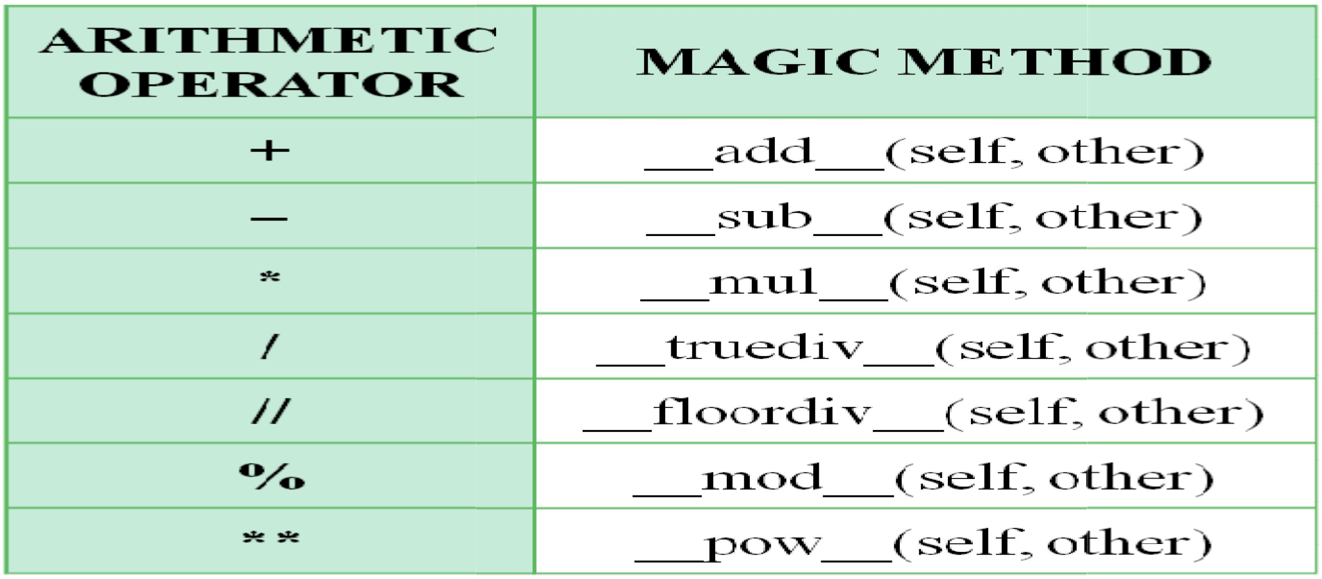
The same built-in operator shows different behavior for objects of different classes, this is called Operator Overloading.

Operator Overloading means giving extended meaning beyond their predefined operational meaning.

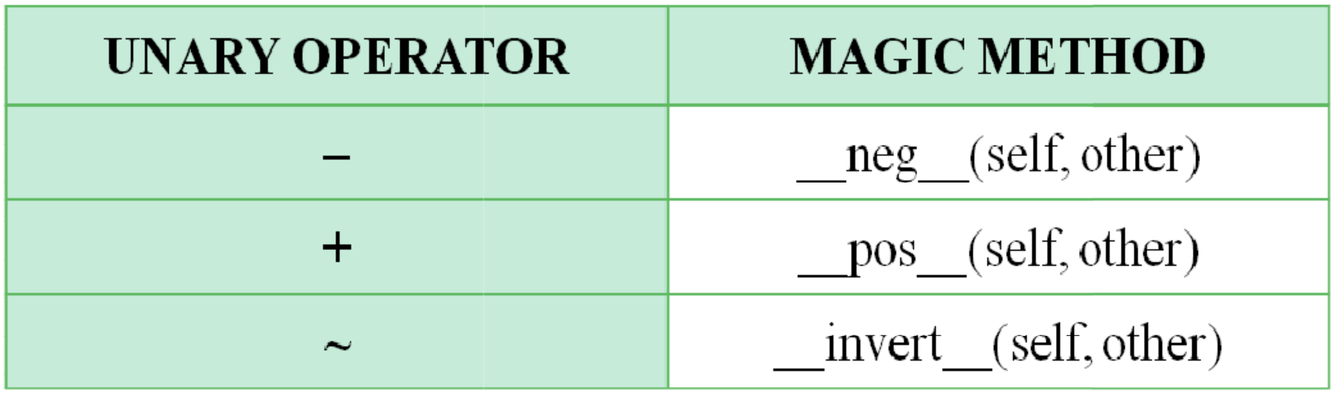
For example operator + is used to add two integers as well as join two strings and merge two lists.

To perform operator overloading, Python provides some special function or **magic function** that is automatically invoked when it is associated with that particular operator.

For example, when we use + operator, the magic method **__add__** is automatically invoked in which the operation for + operator is defined.



ARITHMETIC OPERATOR	MAGIC METHOD
+	___add___(self, other)
–	___sub___(self, other)
*	___mul___(self, other)
/	___truediv___(self, other)
//	___floordiv___(self, other)
%	___mod___(self, other)
**	___pow___(self, other)



UNARY OPERATOR	MAGIC METHOD
–	__neg__(self, other)
+	__pos__(self, other)
~	__invert__(self, other)

# Python Program illustrate how to overload an binary + operator

```
 class A:
        def __init__(self, a):
                self.a = a
        def __add__(self, o):
                return self.a + o.a
    ob1 = A(10)
    ob2 = A(20)
    ob3 = A(“DYP")
    ob4 = A(“College")
    print(ob1 + ob2)                #30
    print(ob3 + ob4)                #DYPCollege
```

# Python program to overload a comparison operators

```
    class A:
        def __init__(self, a):
                self.a = a
        def __gt__(self, other):
                if(self.a>other.a):
                                return True
                else:
                                return False
    ob1 = A(2)
    ob2 = A(3)
    if(ob1>ob2):
        print("ob1 is greater than ob2")
    else:
        print("ob2 is greater than ob1")
```

Python Method Overloading:

Like other languages (for example method overloading in C++) do, python does not supports method overloading.

We may overload the methods but can only use the latest defined method.

```
    def product(a, b):
        p = a * b
        print(p)
    def product(a, b, c):
        p = a * b*c
        print(p)
    product(4, 5)                   #Error
    product(4, 5, 5)                #14
```

Assignments:

Practice Set :

1) Write a python program using simple class having class name as Student.
2) Write a python program for Counting the number of students using more objects of a class.

3) Write a python program for parameterized constructor has multiple parameters along with the self Keyword.

Set A:

1) Write a Python Program to Accept, Delete and Display students details such as Roll.No, Name, Marks in three subject, using Classes. Also display percentage of each student.
2) Write a Python program that defines a class named circle with attributes radius and center, where center is a point object and radius is number. Accept center and radius from user. Instantiate a circle object that represents a circle with its center and radius as accepted input.
3) Write a Python class which has two methods get_String and print_String. get_String accept a string from the user and print_String print the string in upper case. Further modify the program to reverse a string word by word and print it in lower case.
4) Write Python class to perform addition of two complex numbers using binary + operator overloading.

Set B:

1) Define a class named Rectangle which can be constructed by a length and width. The Rectangle class has a method which can compute the area and volume.
2) Write a function named pt_in_circle that takes a circle and a point and returns true if point lies on the boundry of circle.
3) Write a Python Program to Create a Class Set and Get All Possible Subsets from a Set of Distinct Integers.
4) Write a python class to accept a string and number n from user and display n repetition of strings using by overloading * operator.

Set C:

1) Python Program to Create a Class which Performs Basic Calculator Operations.
2) Define datetime module that provides time object. Using this module write a program that gets current date and time and print day of the week.

Evaluation :

0: Not Done [ ] **1: Incomplete [ ]** **2: Late Complete [ ]**

3: Needs Improvement [ ] **4: Complete [ ]** **5: Well Done [ ]**

Signature of the Instructor

Assignment 6: Inheritance

Introduction:
A language feature would not be worthy of the name “class” without supporting inheritance. The syntax for a derived class definition looks like this:

```
class Derived ClassName(BaseClassName):
         <statement-1>
        .
         .
         .
        <statement-N>
```

The name BaseClassName must be defined in a scope containing the derived class definition. In place of a base class name, other arbitrary expressions are also allowed. This can be useful, for example, when the base class is defined in another module:

```
class  Derived_Class_Name (modname.BaseClassName):
```

Execution of a derived class definition proceeds the same as for a base class. When the class object is constructed, the base class is remembered. This is used for resolving attribute references: if a requested attribute is not found in the class, the search proceeds to look in the base class. This rule is applied recursively if the base class itself is derived from some other class.

There’s nothing special about instantiation of derived classes: Derived_Class_Name() creates a new instance of the class. Method references are resolved as follows: the corresponding class attribute is searched, descending down the chain of base classes if necessary, and the method reference is valid if this yields a function object.

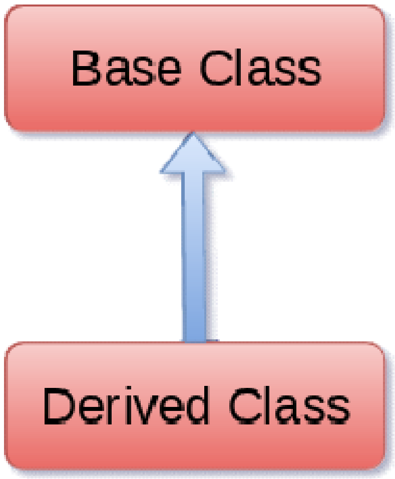
Derived classes may override methods of their base classes. Because methods have no special privileges when calling other methods of the same object, a method of a base class that calls another method defined in the same base class may end up calling a method of a derived class that overrides it. (For C++ programmers: all methods in Python are effectively virtual.)

An overriding method in a derived class may in fact want to extend rather than simply replace the base class method of the same name. There is a simple way to call the base class method directly: just call BaseClassName.methodname(self, arguments). This is occasionally useful to clients as well. (Note that this only works if the base class is accessible as BaseClassName in the global scope.)

Python has two built-in functions that work with inheritance:

- Use isinstance() to check an instance's type: isinstance(obj, int) will be True only if obj.__class__ is int or some class derived from int.
- Use issubclass() to check class inheritance: issubclass(bool, int) is True since bool is a subclass of int. However, issubclass(float, int) is False since float is not a subclass of int.

Inheritance is an important aspect of the object-oriented paradigm. Inheritance provides code reusability to the program because we can use an existing class to create a new class instead of creating it from scratch. In python, a derived class can inherit base class by just mentioning the base in the bracket after the derived class name. Consider the followingsyntax to inherit a base class into the derived class.



Syntax is ,

```
class derived-class (base class):
        <class-suite>
```

A class can inherit multiple classes by mentioning all of them inside the bracket. Consider the following

Syntax**:**

```
class derive-class(<base class 1>, <base class 2>, ..... <base class n>):
<class - suite>
```

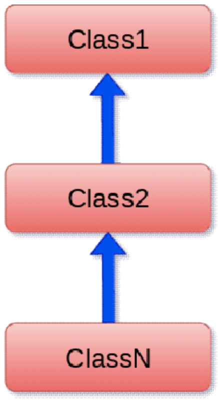
Example :

```
class Human:
        def speak(self):
                print("Human Speaking")
#child class Mohan inherits the base class Human
```

```
class Mohan(Human):
        def sleep(self):
                        print("Mohan sleeping")
m = Mohan()
m.sleep()
m.speak()
```

Multi-Level inheritance:

Multi-Level inheritance is possible in python like other object-oriented languages. Multi-level inheritance is archived when a derived class inherits another derived class. There is no limit on the number of levels up to which, the multi-level inheritance is archived in python.



The syntax of multi-level inheritance is given below.

Syntax

```
class class1:
         <class-suite>
class class2(class1):
         <class suite>
class class3(class2):
        <class suite>
```

Example:

```
class Human:
        def speak(self):
                print("Human Speaking")
#The child class Mohan inherits the base class Human
class Mohan(Human):
```

```
                    def sleep(self):
                            print("Mohan sleeping")
    #The child class Kids inherits another child class Dog
            class Kids(Mohan):
                    def eat(self):
                            print("Eating Rice ...")
            k = Kids()
            k.sleep()
             k.speak()
            k.eat()
```

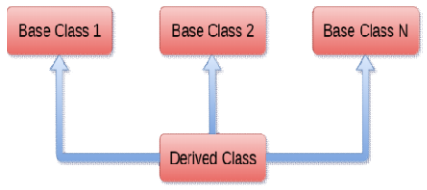
Multiple inheritance:
Python provides us the flexibility to inherit multiple base classes in the child class.
Python supports a form of multiple inheritance as well. A class definition with multiple base classes looks like this:

```
class DerivedClassName(Base1, Base2, Base3):
        <statement-1>
        .
        .
        .
        <statement-N>
```

For most purposes, in the simplest cases, you can think of the search for attributes inherited from a parent class as depth-first, left-to-right, not searching twice in the same class where there is an overlap in the hierarchy. Thus, if an attribute is not found in DerivedClassName, it is searched for in Base1, then (recursively) in the base classes of Base1, and if it was not found there, it was searched for in Base2, and so on.

In fact, it is slightly more complex than that; the method resolution order changes dynamically to support cooperative calls to super(). This approach is known in some other multiple-inheritance languages as call-next-method and is more powerful than the super call found in single-inheritance languages.

Dynamic ordering is necessary because all cases of multiple inheritance exhibit one or more diamond relationships (where at least one of the parent classes can be accessed through multiple paths from the bottommost class). For example, all classes inherit from object, so any case of multiple inheritance provides more than one path to reach object. To keep the base classes from being accessed more than once, the dynamic algorithm linearizes the search order in a way that preserves the left-to-right ordering specified in each class, that calls each parent only once, and that is monotonic (meaning that a class can be subclassed without affecting the precedence order of its parents). Taken together, these properties make it possible to design reliable and extensible classes with multiple inheritances.



The syntax to perform multiple inheritance is given below.

Syntax:

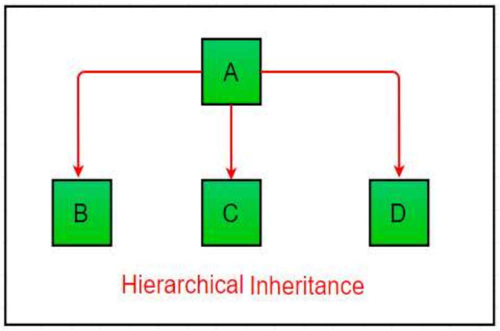
```
class Base1:
        <class-suite>

class Base2:
         <class-suite>
                          .
class BaseN:
        <class-suite>

class Derived(Base1, Base2, ...... BaseN):
        <class-suite>
```

Example:

```
class Calculation1:
        def Summation(self,a,b):
                return a+b;
class Calculation2:
        def Multiplication(self,a,b):
                return a*b;
class Derived(Calculation1,Calculation2):
        def Divide(self,a,b):
                return a/b;
d = Derived()
print(d.Summation(10,20))
print(d.Multiplication(10,20))
print(d.Divide(10,20))
```

Hierarchical Inheritance:

When more than one derived classes are created from a single base this type of inheritance is called hierarchical inheritance. In this program, we have a parent (base) class and two child (derived) classes.



Example:

```
 # Base class
 class Parent:
     def func1(self):
           print("This function is in parent class.")

 # Derived class1
class Child1(Parent):
     def func2(self):
           print("This function is in child 1.")

# Derived class2
class Child2(Parent):
     def func3(self):
           print("This function is in child 2.")

# Driver's code
 object1 = Child1()
 object2 = Child2()
 object1.func1()
 object1.func2()
 object2.func1()
 object2.func3()
```

Hybrid Inheritance:

Inheritance consisting of multiple types of inheritance is called hybrid inheritance.

Example:

```
# Python program to demonstrate
# hybrid inheritance

class School:
    def func1(self):
        print("This function is in school.")

class Student1(School):
    def func2(self):
        print("This function is in student 1. ")

class Student2(School):
    def func3(self):
        print("This function is in student 2.")

class Student3(Student1, School):
    def func4(self):
        print("This function is in student 3.")

# Driver's code
object = Student3()
object.func1()
object.func2()
```

IS-A Relationship and HAS-A Relationship:

One of the advantages of an Object-Oriented programming language is code reuse. There are two ways we can do code reuse either by the implementation of inheritance (IS-A relationship), or object composition (HAS-A relationship). Although the compiler and Java virtual machine (JVM) will do a lot of work for you when you use inheritance, you can also get at the functionality of inheritance when you use composition.

1) **IS-A Relationship :**

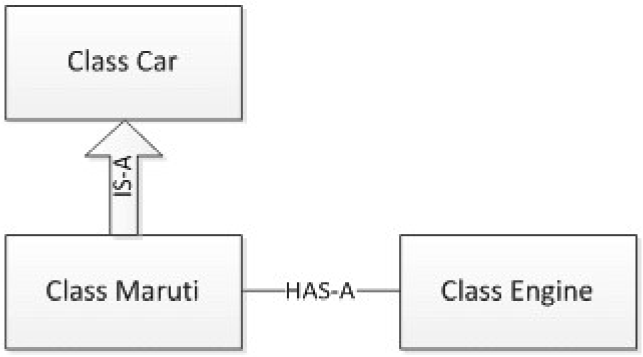
In object-oriented programming, the concept of IS-A is a totally based on Inheritance, which can be of two types Class Inheritance or Interface Inheritance. It is just like saying "A is a B type of thing". For example, Apple is a Fruit, Car is a Vehicle etc. Inheritance is uni-directional. For example, House is a Building. But Building is not a House.

It is a key point to note that you can easily identify the IS-A relationship. Wherever you see an extends keyword or implements keyword in a class declaration, then this class is said to have IS-A relationship.

- IS-A relationship based on Inheritance, which can be of two types Class Inheritance or Interface Inheritance.

2) **HAS-A Relationship**

Composition (HAS-A) simply mean the use of instance variables that are references to other objects. For example Maruti has Engine, or House has Bathroom.
Let's understand these concepts with an example of Car class.



- Has-a relationship is composition relationship which is a productive way of code reuse.

Assignments:

Practice Set :

1) Write a python program to demonstrate single inheritance using findArea() function.
2) Write a python program that will show the simplest form of inheritance using info() function.

SET A:

1) Write a python program to demonstrate multilevel inheritance by using Base class name as "Team" which inherits Derived class name as "Dev".
2) Write a python program by considering Baseclass as TeamMember and Derived class as TeamLeader use multiple inheritance concept to demonstrate the code.
3) Write a python program to make use of issubclass () or isinstance() functions to check the relationships of two classes and instances.

SET B :

1) Write a python program to inherit (Derived class) "course" from (base class) "University" Using hybrid inheritance concept.
2) Write a python program to show the Hierarchical inheritance of two or more classes named as "Square " & " Triangle" inherit from a single Base class as "Area " .
3) Define a class named Shape and its subclass (Square/Circle). The subclass has an init function which takes an argument (length/radius). Both classes have an area and volume

function which can print the area and volume of the shape where Shape's area is 0 by default.

4) Python Program to Create a Class in which One Method Accepts a String from the User and Another method Prints it. Define a class named Country which has a method called print Nationality. Define subclass named state from Country which has a method called print State . Write a method to print state, country and nationality.

SET C :

1) Write a Python Program to depict multiple inheritance when method is overridden in both classes and check the output accordingly.
2) Write a Python Program to describe a HAS-A Relationship(Composition).

Evaluation

0: Not Done [ ]	**1: Incomplete [ ]**	**2: Late Complete [ ]**
3: Needs Improvement [ ]	**4: Complete [ ]**	**5: Well Done [ ]**

Signature of the Instructor

Assignment 7: Exception Handling

What is exception?

The exception is an abnormal condition that halts the execution of the program.

An exception can be defined as an abnormal condition in a program resulting in halting of program execution and thus the further code is not executed.

Python provides us with the way to handle the Exception so that the other part of the code can be executed without any disruption.

However, if we do not handle the exception, the interpreter doesn't execute all the code that exists after that.

Common Exceptions:

A list of common exceptions that can be thrown from a normal python program is given below.

1. ZeroDivisionError: Occurs when a number is divided by zero.
2. NameError: It occurs when a name is not found. It may be local or global.
3. IOError: It occurs when Input Output operation fails.
4. EOFError: It occurs when the end of the file is reached, and yet operations are being performed.
5. ArithmeticError: Base class for all errors that occur for numeric calculation.
6. OverflowError: Raised when a calculation exceeds maximum limit for a numeric type.
7. KeyboardInterrupt: Raised when the user interrupts program execution, usually by pressing Ctrl+c.
8. IndexError: Raised when an index is not found in a sequence.
9. KeyError: Raised when the specified key is not found in the dictionary.
10. 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.

Exception handling in python:

If the python program contains suspicious code that may throw the exception, we must place that code in the try block. The try block must be followed with the except statement which contains a block of code that will be executed if there is some exception in the try block.

Syntax:

```
try:
    #block of code
except Exception1:
    #block of code
except Exception2:
    #block of code
```

We can also use the else statement with the try-except statement in which, we can place the code which will be executed in the scenario if no exception occurs in the try block.
The syntax to use the else statement with the try-except statement is given below.

```
try:
        #block of code
except Exception1:
        #block of code
else:
        #this code executes if no except block is executed
```

The except statement with no exception:

Python provides the flexibility not to specify the name of exception with the except statement.

```
try:
        a = int(input("Enter a:"))
        b = int(input("Enter b:"))
        c = a/b;
        print("a/b = %d"%c)
except:
        print("can't divide by zero")
else:
        print("Hi I am else block")
```

Points to remember:

Python facilitates us not to specify the exception with the except statement.

We can declare multiple exceptions in the except statement since the try block may contain the statements which throw the different type of exceptions.

We can also specify an else block along with the try-except statement which will be executed if no exception is raised in the try block.

Declaring multiple exceptions:

The python allows us to declare the multiple exceptions with the except clause.

Declaring multiple exceptions is useful in the cases where a try block throws multiple exceptions.

Syntax:

```
try:
   #block of code
except (Exception 1, Exception 2,...,Exception n)
   #block of code
else:
   #block of code
```

Example Declaring Multiple Exception:

```
try:
        a=10/0
        fp=open("e:\sms2.txt","r")
except  (ArithmeticError, IOError):
        print "Arithmetic Exception"
else:
        print "Successfully Done"
```

The try-finally Clause:

You can use a finally: block along with a try: block.

The finally block is a place to put any code that must execute, whether the try-block raised an exception or not.

The syntax of the try-finally statement is this

```
try:
   # block of code
   # this may throw an exception
except Exception:
        #Exception code
finally:
   # block of code
   # this will always be executed
```

Custom Exception:

The python allows us to create our exceptions that can be raised from the program and caught using the except clause.

Raising exceptions:

An exception can be raised by using the raise clause in python. The syntax to use the raise statement is given below.

Syntax: raise Exception

To raise an exception, raise statement is used. The exception class name follows it.

#Example User defined exception

```
        a=10
        b=12
        try:
        if b>10:
                raise Exception
        c=a/b
        print("c=",c)
    except Exception:
        print("Error occur")
```

The assert Statement:

When it encounters an assert statement, Python evaluates the accompanying expression, which is hopefully true.

If the expression is false, Python raises an Assertion Error exception.

The syntax for assert is −

```
assert Expression , "Error Msg"
```

If the assertion fails, Python uses Argument Expression as the argument for the Assertion Error.

Assertion Error exceptions can be caught and handled like any other exception using the try-except statement, but if not handled, they will terminate the program and produce a traceback.

Assignments:

Practice Set :

1) write a python program that try to access the array element whose index is out of bound and handle the corresponding exception.
2) Write a Python program to input a positive integer. Display correct message for correct and incorrect input.

SET A :

1) Define a custom exception class which takes a string message as attribute.
2) Write a function called oops that explicitly raises a IndexError exception when called. Then write another function that calls oops inside a try/except statement to catch the error.

3) Change the oops function you just wrote to raise an exception you define yourself, called MyError, and pass an extra data item along with the exception. Then, extend the try statement in the catcher function to catch this exception and its data in addition to IndexError, and print the extra data item.

SET B :

1) Define a class Date(Day, Month, Year) with functions to accept and display it. Accept date from user. Throw user defined exception "invalidDateException" if the date is invalid.
2) Write text file named test.txt that contains integers, characters and float numbers. Write a Python program to read the test.txt file. And print appropriate message using exception

SET C:

1) Write a function called safe (func, *args) that runs any function using apply, catches any exception raised while the function runs, and prints the exception using the exc_type and exc_value attributes in the sys module. Then, use your safe function to run the oops function you wrote in Exercises 3. Put safe in a module file called tools.py, and pass it the oops function interactively. Finally, expand safe to also print a Python stack trace when an error occurs by calling the built-in print_exc() function in the standard traceback module (see the Python library reference manual or other Python books for details)

2) Change the oops function in question 4 from SET A to raise an exception you define yourself, called MyError, and pass an extra data item along with the exception. You may identify your exception with either a string or a class. Then, extend the try statement in the catcher function to catch this exception and its data in addition to IndexError, and print the extra data item. Finally, if you used a string for your exception, go back and change it be a class instance.

Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: Well Done [ ]

Signature of Instructor

Assignment 8: Python GUI Programming using Tkinter

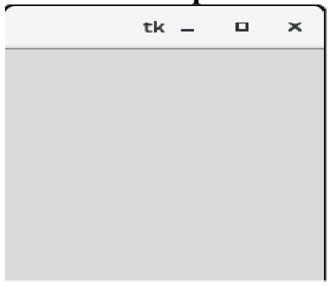
Introduction:
Python provides the standard library Tkinter for creating the graphical user interface for desktop based applications. Developing desktop based applications with python Tkinter is not a complex task. An empty Tkinter top-level window can be created by using the following steps.

1. import the Tkinter module.
2. Create the main application window.
3. Add the widgets like labels, buttons, frames, etc. to the window.
4. Call the main event loop so that the actions can take place on the user's computer screen.

Example

```
# !/usr/bin/python3
from tkinter import *
#creating the application main window.
top = Tk()
#Entering the event main loop
top.mainloop()
```

Output:



Tkinter widgets:
There are various widgets like button, label,entry, canvas, checkbutton,Radio button, frame etc. that are used to build the python GUI applications.

Python Tkinter Geometry:
The Tkinter geometry specifies the method by using which, the widgets are represented on display. The python Tkinter provides the following geometry methods.

1. The pack() method
2. The grid() method
3. The place() method

Python Tkinter pack() method:
The pack() widget is used to organize widget in the block. The positions widgets added to the python application using the pack() method can be controlled by using the various options specified in the method call.
However, the controls are less and widgets are generally added in the less organized manner. The syntax to use the pack() is given below.

Syntax: widget.pack(options)

A list of possible options that can be passed in pack() is given below.

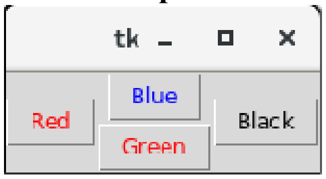
- ➢ expand: If the expand is set to true, the widget expands to fill any space.

- Fill: By default, the fill is set to NONE. However, we can set it to X or Y to determine whether the widget contains any extra space.
- size: it represents the side of the parent to which the widget is to be placed on the window.

Example:

```
# !/usr/bin/python3
from tkinter import *
parent = Tk()
redbutton = Button(parent, text = "Red", fg = "red")
redbutton.pack( side = LEFT)
greenbutton = Button(parent, text = "Black", fg = "black")
greenbutton.pack( side = RIGHT )
bluebutton = Button(parent, text = "Blue", fg = "blue")
bluebutton.pack( side = TOP )
blackbutton = Button(parent, text = "Green", fg = "red")
blackbutton.pack( side = BOTTOM)
parent.mainloop()
```

Output:



Python Tkinter grid() method:

The grid() geometry manager organizes the widgets in the tabular form. We can specify the rows and columns as the options in the method call. We can also specify the column span (width) or rowspan(height) of a widget.

This is a more organized way to place the widgets to the python application. The syntax to use the grid() is given below.

Syntax: widget.grid(options)

A list of possible options that can be passed inside the grid() method is given below.

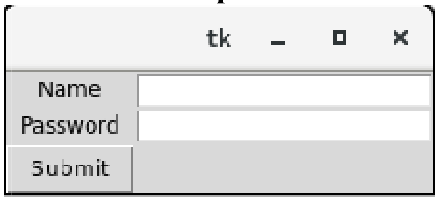
- Column- The column number in which the widget is to be placed. The leftmost column is represented by 0.
- Columnspan- The width of the widget. It represents the number of columns up to which, the column is expanded.
- ipadx, ipady- It represents the number of pixels to pad the widget inside the widget's border.
- padx, pady- It represents the number of pixels to pad the widget outside the widget's border.
- Row- The row number in which the widget is to be placed. The topmost row is represented by 0.
- Rowspan- The height of the widget, i.e. the number of the row up to which the widget is expanded.

- Sticky- If the cell is larger than a widget, then sticky is used to specify the position of the widget inside the cell. It may be the concatenation of the sticky letters representing the position of the widget. It may be N, E, W, S, NE, NW, NS, EW, ES.

Example

```
# !/usr/bin/python3
from tkinter import *
parent = Tk()
name = Label(parent,text = "Name").grid(row = 0, column = 0)
e1 = Entry(parent).grid(row = 0, column = 1)
password = Label(parent,text = "Password").grid(row = 1, column = 0)
e2 = Entry(parent).grid(row = 1, column = 1)
submit = Button(parent, text = "Submit").grid(row = 4, column = 0)
parent.mainloop()
```

Output:



Python Tkinter place() method:

The place() geometry manager organizes the widgets to the specific x and y coordinates.

Syntax: widget.place(options)

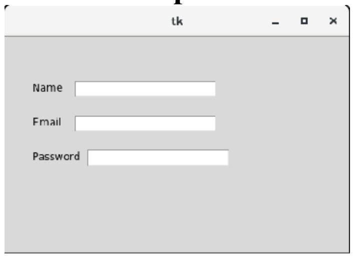
A list of possible options is given below.

- Anchor: It represents the exact position of the widget within the container. The default value (direction) is NW (the upper left corner)
- bordermode: The default value of the border type is INSIDE that refers to ignore the parent's inside the border. The other option is OUTSIDE.
- height, width: It refers to the height and width in pixels.
- relheight, relwidth: It is represented as the float between 0.0 and 1.0 indicating the fraction of the parent's height and width.
- relx, rely: It is represented as the float between 0.0 and 1.0 that is the offset in the horizontal and vertical direction.
- x, y: It refers to the horizontal and vertical offset in the pixels.

Example

```
# !/usr/bin/python3
from tkinter import *
top = Tk()
top.geometry("400x250")
name = Label(top, text = "Name").place(x = 30,y = 50)
email = Label(top, text = "Email").place(x = 30, y = 90)
password = Label(top, text = "Password").place(x = 30, y = 130)
e1 = Entry(top).place(x = 80, y = 50)
e2 = Entry(top).place(x = 80, y = 90)
e3 = Entry(top).place(x = 95, y = 130)
top.mainloop()
```

Output:



Tkinter widgets:

1. **Python Tkinter Button:**
   The button widget is used to add various types of buttons to the python application. Python allows us to configure the look of the button according to our requirements. Various options can be set or reset depending upon the requirements. We can also associate a method or function with a button which is called when the button is pressed. The syntax to use the button widget is given below**.**

   Syntax: W = Button(parent, options)

A list of possible options is given below.

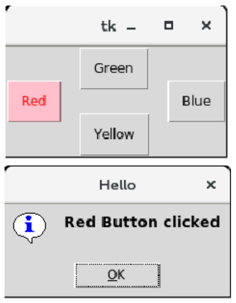
SN	Option	Description
1	activebackground	It represents the background of the button when the mouse hover the button.
2	activeforeground	It represents the font color of the button when the mouse hover the button.
3	Bd	It represents the border width in pixels.
4	Bg	It represents the background color of the button.
5	Command	It is set to the function call which is scheduled when the function is called.
6	Fg	Foreground color of the button.
7	Font	The font of the button text.
8	Height	The height of the button. The height is represented in the number of text lines for the textual lines or the number of pixels for the images.
10	Highlightcolor	The color of the highlight when the button has the focus.
11	Image	It is set to the image displayed on the button.
12	justify	It illustrates the way by which the multiple text lines are represented. It is set to LEFT for left justification, RIGHT for the right justification, and CENTER for the center.
13	Padx	Additional padding to the button in the horizontal direction.
14	pady	Additional padding to the button in the vertical direction.
15	Relief	It represents the type of the border. It can be SUNKEN, RAISED, GROOVE, and RIDGE.
17	State	This option is set to DISABLED to make the button unresponsive. The ACTIVE represents the active state of the button.
18	Underline	Set this option to make the button text underlined.
19	Width	The width of the button. It exists as a number of letters for textual

		buttons or pixels for image buttons.
20	Wraplength	If the value is set to a positive number, the text lines will be wrapped to fit within this length.

Example:

```
from tkinter import *
top = Tk()
top.geometry("200x100")

def fun():
   messagebox.showinfo("Hello", "Red Button clicked")

b1 = Button(top,text = "Red",command = fun,activeforeground = "red",activebackground = "pink",p
ady=10)
b2 = Button(top, text = "Blue",activeforeground = "blue",activebackground = "pink",pady=10)
b3 = Button(top, text = "Green",activeforeground = "green",activebackground = "pink",pady = 10)
b4 = Button(top, text = "Yellow",activeforeground = "yellow",activebackground = "pink",pady = 10
)
b1.pack(side = LEFT)
b2.pack(side = RIGHT)
b3.pack(side = TOP)
b4.pack(side = BOTTOM)
top.mainloop()
```

Output:



Red Button clicked

2. **Python Tkinter Entry:**
The Entry widget is used to provde the single line text-box to the user to accept a value from the user. We can use the Entry widget to accept the text strings from the user. It can only be used for one line of text from the user. For multiple lines of text, we must use the text widget.The syntax to use the Entry widget is given below.
Syntax: w = Entry (parent, options)
A list of possible options is given below.

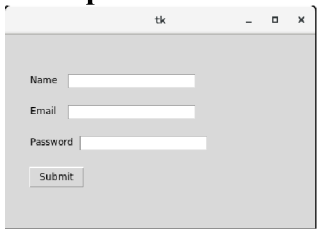
SN	Option	Description
1	bg	The background color of the widget.
2	bd	The border width of the widget in pixels.
3	cursor	The mouse pointer will be changed to the cursor type set to the

		arrow, dot, etc.
4	exportselection	The text written inside the entry box will be automatically copied to the clipboard by default. We can set the exportselection to 0 to not copy this.
5	fg	It represents the color of the text.
6	font	It represents the font type of the text.
7	highlightbackground	It represents the color to display in the traversal highlight region when the widget does not have the input focus.
8	highlightcolor	It represents the color to use for the traversal highlight rectangle that is drawn around the widget when it has the input focus.
9	highlightthickness	It represents a non-negative value indicating the width of the highlight rectangle to draw around the outside of the widget when it has the input focus.
10	insertbackground	It represents the color to use as background in the area covered by the insertion cursor. This color will normally override either the normal background for the widget.
11	insertborderwidth	It represents a non-negative value indicating the width of the 3-D border to draw around the insertion cursor. The value may have any of the forms acceptable to Tk_GetPixels.
12	insertofftime	It represents a non-negative integer value indicating the number of milliseconds the insertion cursor should remain "off" in each blink cycle. If this option is zero, then the cursor doesn't blink: it is on all the time.
13	insertontime	Specifies a non-negative integer value indicating the number of milliseconds the insertion cursor should remain "on" in each blink cycle.
14	insertwidth	It represents the value indicating the total width of the insertion cursor. The value may have any of the forms acceptable to Tk_GetPixels.
15	justify	It specifies how the text is organized if the text contains multiple lines.
16	relief	It specifies the type of the border. Its default value is FLAT.
17	selectbackground	The background color of the selected text.
18	selectborderwidth	The width of the border to display around the selected task.
19	selectforeground	The font color of the selected task.
20	show	It is used to show the entry text of some other type instead of the string. For example, the password is typed using stars (*).
21	textvariable	It is set to the instance of the StringVar to retrieve the text from the entry.
22	width	The width of the displayed text or image.
23	xscrollcommand	The entry widget can be linked to the horizontal scrollbar if we want the user to enter more text then the actual width of the widget.

Example:

```
# !/usr/bin/python3
from tkinter import *
top = Tk()
top.geometry("400x250")
name = Label(top, text = "Name").place(x = 30,y = 50)
email = Label(top, text = "Email").place(x = 30, y = 90)
password = Label(top, text = "Password").place(x = 30, y = 130)
sbmitbtn = Button(top, text = "Submit",activebackground = "pink", activeforeground = "blue").place
(x = 30, y = 170)
e1 = Entry(top).place(x = 80, y = 50)
e2 = Entry(top).place(x = 80, y = 90)
e3 = Entry(top).place(x = 95, y = 130)
top.mainloop()
```

Output:



Entry widget methods:

Python provides various methods to configure the data written inside the widget. There are the following methods provided by the Entry widget.

SN	Method	Description
1	delete(first, last = none)	It is used to delete the specified characters inside the widget.
2	get()	It is used to get the text written inside the widget.
3	icursor(index)	It is used to change the insertion cursor position. We can specify the index of the character before which, the cursor to be placed.
4	index(index)	It is used to place the cursor to the left of the character written at the specified index.
5	insert(index,s)	It is used to insert the specified string before the character placed at the specified index.
6	select_adjust(index)	It includes the selection of the character present at the specified index.
7	select_clear()	It clears the selection if some selection has been done.
8	select_form(index)	It sets the anchor index position to the character specified by the index.
9	select_present()	It returns true if some text in the Entry is selected otherwise returns false.
10	select_range(start,end)	It selects the characters to exist between the specified range.
11	select_to(index)	It selects all the characters from the beginning to the specified index.

12	xview(index)	It is used to link the entry widget to a horizontal scrollbar.
13	xview_scroll(number,what)	It is used to make the entry scrollable horizontally.

3. Python Tkinter Label:

The Label is used to specify the container box where we can place the text or images. This widget is used to provide the message to the user about other widgets used in the python application. There are the various options which can be specified to configure the text or the part of the text shown in the Label. The syntax to use the Label is given below.

Syntax: w = Label (master, options)

A list of possible options is given below.

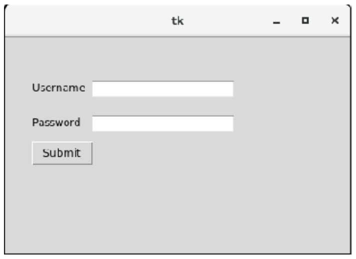
SN	Option	Description
1	anchor	It specifies the exact position of the text within the size provided to the widget. The default value is CENTER, which is used to center the text within the specified space.
2	bg	The background color displayed behind the widget.
3	bitmap	It is used to set the bitmap to the graphical object specified so that, the label can represent the graphics instead of text.
4	bd	It represents the width of the border. The default is 2 pixels.
5	cursor	The mouse pointer will be changed to the type of the cursor specified, i.e., arrow, dot, etc.
6	font	The font type of the text written inside the widget.
7	fg	The foreground color of the text written inside the widget.
8	height	The height of the widget.
9	image	The image that is to be shown as the label.
10	justify	It is used to represent the orientation of the text if the text contains multiple lines. It can be set to LEFT for left justification, RIGHT for right justification, and CENTER for center justification.
11	padx	The horizontal padding of the text. The default value is 1.
12	pady	The vertical padding of the text. The default value is 1.
13	relief	The type of the border. The default value is FLAT.
14	text	This is set to the string variable which may contain one or more line of text.
15	textvariable	The text written inside the widget is set to the control variable StringVar so that it can be accessed and changed accordingly.
16	underline	We can display a line under the specified letter of the text. Set this option to the number of the letter under which the line will be displayed.
17	width	The width of the widget. It is specified as the number of characters.
18	wraplength	Instead of having only one line as the label text, we can break it to the number of lines where each line has the number of characters specified to this option.

Example:

```
# !/usr/bin/python3
from tkinter import *
top = Tk()
```

```
top.geometry("400x250")
#creating label
uname = Label(top, text = "Username").place(x = 30,y = 50)
#creating label
password = Label(top, text = "Password").place(x = 30, y = 90)
sbmitbtn = Button(top, text = "Submit",activebackground = "pink", activeforeground = "blue").place
(x = 30, y = 120)
e1 = Entry(top,width = 20).place(x = 100, y = 50)
e2 = Entry(top, width = 20).place(x = 100, y = 90)
top.mainloop()
```

Output:



4. Python Tkinter Checkbutton:

The Checkbutton is used to track the user's choices provided to the application. In other words, we can say that Checkbutton is used to implement the on/off selections.

The Checkbutton can contain the text or images. The Checkbutton is mostly used to provide many choices to the user among which, the user needs to choose the one. It generally implements many of many selections. The syntax to use the checkbutton is given below.

Syntax: w = checkbutton(master, options)

A list of possible options is given below.

SN	Option	Description
1	activebackground	It represents the background color when the checkbutton is under the cursor.
2	activeforeground	It represents the foreground color of the checkbutton when the checkbutton is under the cursor.
3	bg	The background color of the button.
4	bitmap	It displays an image (monochrome) on the button.
5	bd	The size of the border around the corner.
6	command	It is associated with a function to be called when the state of the checkbutton is changed.
7	cursor	The mouse pointer will be changed to the cursor name when it is over the checkbutton.
8	disableforeground	It is the color which is used to represent the text of a disabled checkbutton.
9	font	It represents the font of the checkbutton.
10	fg	The foreground color (text color) of the checkbutton.
11	height	It represents the height of the checkbutton (number of lines). The default height is 1.
12	highlightcolor	The color of the focus highlight when the checkbutton is under focus.

13	image	The image used to represent the checkbutton.
14	justify	This specifies the justification of the text if the text contains multiple lines.
15	offvalue	The associated control variable is set to 0 by default if the button is unchecked. We can change the state of an unchecked variable to some other one.
16	onvalue	The associated control variable is set to 1 by default if the button is checked. We can change the state of the checked variable to some other one.
17	padx	The horizontal padding of the checkbutton
18	pady	The vertical padding of the checkbutton.
19	relief	The type of the border of the checkbutton. By default, it is set to FLAT.
20	selectcolor	The color of the checkbutton when it is set. By default, it is red.
21	selectimage	The image is shown on the checkbutton when it is set.
22	state	It represents the state of the checkbutton. By default, it is set to normal. We can change it to DISABLED to make the checkbutton unresponsive. The state of the checkbutton is ACTIVE when it is under focus.
24	underline	It represents the index of the character in the text which is to be underlined. The indexing starts with zero in the text.
25	variable	It represents the associated variable that tracks the state of the checkbutton.
26	width	It represents the width of the checkbutton. It is represented in the number of characters that are represented in the form of texts.
27	wraplength	If this option is set to an integer number, the text will be broken into the number of pieces.

Methods:

The methods that can be called with the Checkbuttons are described in the following table.

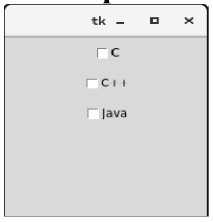
SN	Method	Description
1	deselect()	It is called to turn off the checkbutton.
2	flash()	The checkbutton is flashed between the active and normal colors.
3	invoke()	This will invoke the method associated with the checkbutton.
4	select()	It is called to turn on the checkbutton.
5	toggle()	It is used to toggle between the different Checkbuttons.

Example:

```
from tkinter import *
top = Tk()
top.geometry("200x200")
  checkvar1 = IntVar()
  checkvar2 = IntVar()
  checkvar3 = IntVar()
  chkbtn1 = Checkbutton(top, text = "C", variable = checkvar1, onvalue = 1, offvalue = 0, height = 2,
 width = 10)
```

```
  chkbtn2 = Checkbutton(top, text = "C++", variable = checkvar2, onvalue = 1, offvalue = 0, height
= 2, width = 10)
  chkbtn3 = Checkbutton(top, text = "Java", variable = checkvar3, onvalue = 1, offvalue = 0, height =
 2, width = 10)
  chkbtn1.pack()
  chkbtn2.pack()
  chkbtn3.pack()
top.mainloop()
```

Output:



5. Python Tkinter Radiobutton:

The Radiobutton widget is used to implement one-of-many selection in the python application. It shows multiple choices to the user out of which, the user can select only one out of them. We can associate different methods with each of the radiobutton. We can display the multiple line text or images on the radiobuttons. To keep track the user's selection the radiobutton, it is associated with a single variable. Each button displays a single value for that particular variable. The syntax to use the Radiobutton is given below.

Syntax: w = Radiobutton(top, options)

SN	Option	Description
1	activebackground	The background color of the widget when it has the focus.
2	activeforeground	The font color of the widget text when it has the focus.
3	anchor	It represents the exact position of the text within the widget if the widget contains more space than the requirement of the text. The default value is CENTER.
4	bg	The background color of the widget.
5	bitmap	It is used to display the graphics on the widget. It can be set to any graphical or image object.
6	borderwidth	It represents the size of the border.
7	command	This option is set to the procedure which must be called every-time when the state of the radiobutton is changed.
8	cursor	The mouse pointer is changed to the specified cursor type. It can be set to the arrow, dot, etc.
9	font	It represents the font type of the widget text.
10	fg	The normal foreground color of the widget text.
11	height	The vertical dimension of the widget. It is specified as the number of lines (not pixel).
12	highlightcolor	It represents the color of the focus highlight when the widget has the focus.

13	highlightbackground	The color of the focus highlight when the widget is not having the focus.
14	image	It can be set to an image object if we want to display an image on the radiobutton instead the text.
15	justify	It represents the justification of the multi-line text. It can be set to CENTER(default), LEFT, or RIGHT.
16	padx	The horizontal padding of the widget.
17	pady	The vertical padding of the widget.
18	relief	The type of the border. The default value is FLAT.
19	selectcolor	The color of the radio button when it is selected.
20	selectimage	The image to be displayed on the radiobutton when it is selected.
21	state	It represents the state of the radio button. The default state of the Radiobutton is NORMAL. However, we can set this to DISABLED to make the radiobutton unresponsive.
22	text	The text to be displayed on the radiobutton.
23	textvariable	It is of String type that represents the text displayed by the widget.
24	underline	The default value of this option is -1, however, we can set this option to the number of character which is to be underlined.
25	value	The value of each radiobutton is assigned to the control variable when it is turned on by the user.
26	variable	It is the control variable which is used to keep track of the user's choices. It is shared among all the radiobuttons.
27	width	The horizontal dimension of the widget. It is represented as the number of characters.
28	wraplength	We can wrap the text to the number of lines by setting this option to the desired number so that each line contains only that number of characters.

Methods: The radiobutton widget provides the following methods.

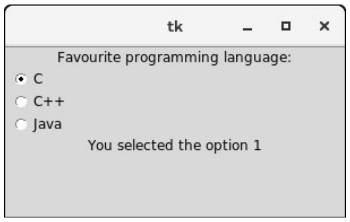
SN	Method	Description
1	deselect()	It is used to turn of the radiobutton.
2	flash()	It is used to flash the radiobutton between its active and normal colors few times.
3	invoke()	It is used to call any procedure associated when the state of a Radiobutton is changed.
4	select()	It is used to select the radiobutton.

Example:

```
from tkinter import *
def selection():
  selection = "You selected the option " + str(radio.get())
  label.config(text = selection)

top = Tk()
```

```
top.geometry("300x150")
radio = IntVar()
lbl = Label(text = "Favourite programming language:")
lbl.pack()
R1 = Radiobutton(top, text="C", variable=radio, value=1,command=selection)
R1.pack( anchor = W )

R2 = Radiobutton(top, text="C++", variable=radio, value=2,command=selection)
R2.pack( anchor = W )

R3 = Radiobutton(top, text="Java", variable=radio, value=3,command=selection)
R3.pack( anchor = W)

label = Label(top)
label.pack()
top.mainloop()
```

Output:



```
You selected the option 1
```

6. **Python Tkinter Frame:**

Python Tkinter Frame widget is used to organize the group of widgets. It acts like a container which can be used to hold the other widgets. The rectangular areas of the screen are used to organize the widgets to the python application. The syntax to use the Frame widget is given below.

Syntax: w = Frame(parent, options)

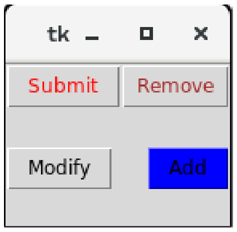
A list of possible options is given below.

SN	Option	Description
1	bd	It represents the border width.
2	bg	The background color of the widget.
3	cursor	The mouse pointer is changed to the cursor type set to different values like an arrow, dot, etc.
4	height	The height of the frame.
5	highlightbackground	The color of the background color when it is under focus.
6	highlightcolor	The text color when the widget is under focus.
7	highlightthickness	It specifies the thickness around the border when the widget is under the focus.
8	relief	It specifies the type of the border. The default value if FLAT.
9	width	It represents the width of the widget.

Example:

```
from tkinter import *
top = Tk()
top.geometry("140x100")
frame = Frame(top)
frame.pack()
leftframe = Frame(top)
leftframe.pack(side = LEFT)
rightframe = Frame(top)
rightframe.pack(side = RIGHT)
btn1 = Button(frame, text="Submit", fg="red",activebackground = "red")
btn1.pack(side = LEFT)
btn2 = Button(frame, text="Remove", fg="brown", activebackground = "brown")
btn2.pack(side = RIGHT)
btn3 = Button(rightframe, text="Add", fg="blue", activebackground = "blue")
btn3.pack(side = LEFT)
btn4 = Button(leftframe, text="Modify", fg="black", activebackground = "white")
btn4.pack(side = RIGHT)
top.mainloop()
```

Output:



7. Python Tkinter Listbox:

The Listbox widget is used to display the list items to the user. We can place only text items in the Listbox and all text items contain the same font and color.

The user can choose one or more items from the list depending upon the configuration. The syntax to use the Listbox is given below.

w = Listbox(parent, options)

A list of possible options is given below.

SN	Option	Description
1	bg	The background color of the widget.
2	bd	It represents the size of the border. Default value is 2 pixel.
3	cursor	The mouse pointer will look like the cursor type like dot, arrow, etc.
4	font	The font type of the Listbox items.
5	fg	The color of the text.
6	height	It represents the count of the lines shown in the Listbox. The default value is 10.
7	highlightcolor	The color of the Listbox items when the widget is under focus.
8	highlightthickness	The thickness of the highlight.
9	relief	The type of the border. The default is SUNKEN.
10	selectbackground	The background color that is used to display the selected text.

11	selectmode	It is used to determine the number of items that can be selected from the list. It can set to BROWSE, SINGLE, MULTIPLE, EXTENDED.
12	width	It represents the width of the widget in characters.
13	xscrollcommand	It is used to let the user scroll the Listbox horizontally.
14	yscrollcommand	It is used to let the user scroll the Listbox vertically.

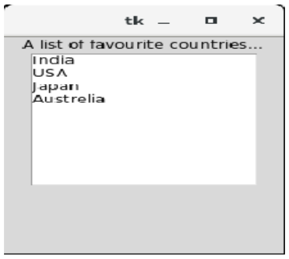
Methods: There are the following methods associated with the Listbox.

SN	Method	Description
1	activate(index)	It is used to select the lines at the specified index.
2	curselection()	It returns a tuple containing the line numbers of the selected element or elements, counting from 0. If nothing is selected, returns an empty tuple.
3	delete(first, last = None)	It is used to delete the lines which exist in the given range.
4	get(first, last = None)	It is used to get the list items that exist in the given range.
5	index(i)	It is used to place the line with the specified index at the top of the widget.
6	insert(index, *elements)	It is used to insert the new lines with the specified number of elements before the specified index.
7	nearest(y)	It returns the index of the nearest line to the y coordinate of the Listbox widget.
8	see(index)	It is used to adjust the position of the listbox to make the lines specified by the index visible.
9	size()	It returns the number of lines that are present in the Listbox widget.
10	xview()	This is used to make the widget horizontally scrollable.
11	xview_moveto(fraction)	It is used to make the listbox horizontally scrollable by the fraction of width of the longest line present in the listbox.
12	xview_scroll(number, what)	It is used to make the listbox horizontally scrollable by the number of characters specified.
13	yview()	It allows the Listbox to be vertically scrollable.
14	yview_moveto(fraction)	It is used to make the listbox vertically scrollable by the fraction of width of the longest line present in the listbox.
15	yview_scroll (number, what)	It is used to make the listbox vertically scrollable by the number of characters specified.

Example:

```
# !/usr/bin/python3
  from tkinter import *
  top = Tk()
  top.geometry("200x250")
  lbl = Label(top,text = "A list of favourite countries...")
 listbox = Listbox(top)
 listbox.insert(1,"India")
 listbox.insert(2, "USA")
 listbox.insert(3, "Japan")
 listbox.insert(4, "Austrelia")
 lbl.pack()
listbox.pack()
 top.mainloop()
```

Output:



8. Python Tkinter Message:

The Message widget is used to show the message to the user regarding the behaviour of the python application. The message widget shows the text messages to the user which can not be edited. The message text contains more than one line. However, the message can only be shown in the single font. The syntax to use the Message widget is given below.

Syntax: w = Message(parent, options)

A list of possible options is given below.

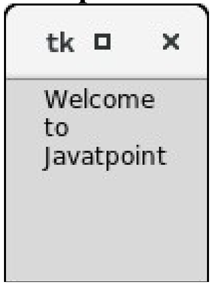
SN	Option	Description
1	anchor	It is used to decide the exact position of the text within the space provided to the widget if the widget contains more space than the need of the text. The default is CENTER.
2	bg	The background color of the widget.
3	bitmap	It is used to display the graphics on the widget. It can be set to any graphical or image object.
4	bd	It represents the size of the border in the pixel. The default size is 2 pixel.
5	cursor	The mouse pointer is changed to the specified cursor type. The cursor type can be an arrow, dot, etc.
6	font	The font type of the widget text.
7	fg	The font color of the widget text.
8	height	The vertical dimension of the message.
9	image	We can set this option to a static image to show that onto the widget.
10	justify	This option is used to specify the alignment of multiple line of code with respect to each other. The possible values can be LEFT (left alignment), CENTER (default), and RIGHT (right alignment).
11	padx	The horizontal padding of the widget.
12	pady	The vertical padding of the widget.
13	relief	It represents the type of the border. The default type is FLAT.
14	text	We can set this option to the string so that the widget can represent the specified text.
15	textvariable	This is used to control the text represented by the widget. The textvariable can be set to the text that is shown in the widget.
16	underline	The default value of this option is -1 that represents no underline. We can

		set this option to an existing number to specify that nth letter of the string will be underlined.
17	width	It specifies the horizontal dimension of the widget in the number of characters (not pixel).
18	wraplength	We can wrap the text to the number of lines by setting this option to the desired number so that each line contains only that number of characters.

Example:

```
from tkinter import *
  top = Tk()
top.geometry("100x100")
var = StringVar()
msg = Message( top, text = "Welcome to Javatpoint")
  msg.pack()
top.mainloop()
```

Output:



9. Python Tkinter Menubutton:

The Menubutton widget can be defined as the drop-down menu that is shown to the user all the time. It is used to provide the user a option to select the appropriate choice exist within the application. The Menubutton is used to implement various types of menus in the python application. A Menu is associated with the Menubutton that can display the choices of the Menubutton when clicked by the user. The syntax to use the python tkinter Menubutton is given below.

Syntax : w = Menubutton(Top, options)

A list of various options is given below.

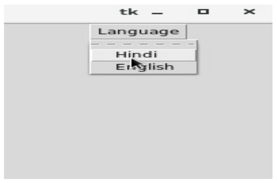
SN	Option	Description
1	activebackground	The background color of the widget when the widget is under focus.
2	activeforeground	The font color of the widget text when the widget is under focus.
3	anchor	It specifies the exact position of the widget content when the widget is assigned more space than needed.
4	bg	It specifies the background color of the widget.
5	bitmap	It is set to the graphical content which is to be displayed to the widget.
6	bd	It represents the size of the border. The default value is 2 pixels.
7	cursor	The mouse pointer will be changed to the cursor type specified when the widget is under the focus. The possible value of the cursor type is arrow, or dot etc.
8	direction	It direction can be specified so that menu can be displayed to the specified direction of the button. Use LEFT, RIGHT, or ABOVE to

		place the widget accordingly.
9	disabledforeground	The text color of the widget when the widget is disabled.
10	fg	The normal foreground color of the widget.
11	height	The vertical dimension of the Menubutton. It is specified as the number of lines.
12	highlightcolor	The highlight color shown to the widget under focus.
13	image	The image displayed on the widget.
14	justify	This specified the exact position of the text under the widget when the text is unable to fill the width of the widget. We can use the LEFT for the left justification, RIGHT for the right justification, CENTER for the centre justification.
15	menu	It represents the menu specified with the Menubutton.
16	padx	The horizontal padding of the widget.
17	pady	The vertical padding of the widget.
18	relief	This option specifies the type of the border. The default value is RAISED.
19	state	The normal state of the Mousebutton is enabled. We can set it to DISABLED to make it unresponsive.
20	text	The text shown with the widget.
21	textvariable	We can set the control variable of string type to the text variable so that we can control the text of the widget at runtime.
22	underline	The text of the widget is not underlined by default but we can set this option to make the text of the widget underlined.
23	width	It represents the width of the widget in characters. The default value is 20.
24	wraplength	We can break the text of the widget in the number of lines so that the text contains the number of lines not greater than the specified value.

Example

```
# !/usr/bin/python3
  from tkinter import *
  top = Tk()
  top.geometry("200x250")
  menubutton = Menubutton(top, text = "Language", relief = FLAT)
  menubutton.grid()
  menubutton.menu = Menu(menubutton)
  menubutton["menu"]=menubutton.menu
  menubutton.menu.add_checkbutton(label = "Hindi", variable=IntVar())
  menubutton.menu.add_checkbutton(label = "English", variable = IntVar())
  menubutton.pack()
  top.mainloop()
```

Output:



10. Python Tkinter Menu:
The Menu widget is used to create various types of menus (top level, pull down, and pop up) in the python application. The top-level menus are the one which is displayed just under the title bar of the parent window. We need to create a new instance of the Menu widget and add various commands to it by using the add() method. The syntax to use the Menu widget is given below.
Syntax: w = Menu(top, options)
A list of possible options is given below.

SN	Option	Description
1	activebackground	The background color of the widget when the widget is under the focus.
2	activeborderwidth	The width of the border of the widget when it is under the mouse. The default is 1 pixel.
3	activeforeground	The font color of the widget when the widget has the focus.
4	bg	The background color of the widget.
5	bd	The border width of the widget.
6	cursor	The mouse pointer is changed to the cursor type when it hovers the widget. The cursor type can be set to arrow or dot.
7	disabledforeground	The font color of the widget when it is disabled.
8	font	The font type of the text of the widget.
9	fg	The foreground color of the widget.
10	postcommand	The postcommand can be set to any of the function which is called when the mourse hovers the menu.
11	relief	The type of the border of the widget. The default type is RAISED.
12	image	It is used to display an image on the menu.
13	selectcolor	The color used to display the checkbutton or radiobutton when they are selected.
14	tearoff	By default, the choices in the menu start taking place from position 1. If we set the tearoff = 1, then it will start taking place from 0th position.
15	title	Set this option to the title of the window if you want to change the title of the window.

Methods: The Menu widget contains the following methods.

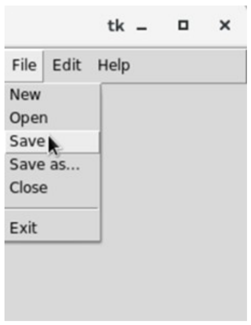
SN	Option	Description
1	add_command(options)	It is used to add the Menu items to the menu.
2	add_radiobutton(options)	This method adds the radiobutton to the menu.
3	add_checkbutton(options)	This method is used to add the checkbuttons to the menu.
4	add_cascade(options)	It is used to create a hierarchical menu to the parent menu by

		associating the given menu to the parent menu.
5	add_seperator()	It is used to add the seperator line to the menu.
6	add(type, options)	It is used to add the specific menu item to the menu.
7	delete(startindex, endindex)	It is used to delete the menu items exist in the specified range.
8	entryconfig(index, options)	It is used to configure a menu item identified by the given index.
9	index(item)	It is used to get the index of the specified menu item.
10	insert_seperator(index)	It is used to insert a seperator at the specified index.
11	invoke(index)	It is used to invoke the associated with the choice given at the specified index.
12	type(index)	It is used to get the type of the choice specified by the index.

Example:

```
from tkinter import Toplevel, Button, Tk, Menu

top = Tk()
menubar = Menu(top)
file = Menu(menubar, tearoff=0)
file.add_command(label="New")
file.add_command(label="Open")
file.add_command(label="Save")
file.add_command(label="Save as...")
file.add_command(label="Close")
  file.add_separator()
  file.add_command(label="Exit", command=top.quit)
  menubar.add_cascade(label="File", menu=file)
edit = Menu(menubar, tearoff=0)
edit.add_command(label="Undo")
  edit.add_separator()
  edit.add_command(label="Cut")
edit.add_command(label="Copy")
edit.add_command(label="Paste")
edit.add_command(label="Delete")
edit.add_command(label="Select All")
  menubar.add_cascade(label="Edit", menu=edit)
help = Menu(menubar, tearoff=0)
help.add_command(label="About")
menubar.add_cascade(label="Help", menu=help)
  top.config(menu=menubar)
top.mainloop()
```

Output:



11. Python Tkinter Text:

The Text widget is used to show the text data on the Python application. However, Tkinter provides us the Entry widget which is used to implement the single line text box. The Text widget is used to display the multi-line formatted text with various styles and attributes. The Text widget is mostly used to provide the text editor to the user. The Text widget also facilitates us to use the marks and tabs to locate the specific sections of the Text. We can also use the windows and images with the Text as it can also be used to display the formatted text. The syntax to use the Text widget is given below.

Syntax: w = Text(top, options)

A list of possible options that can be used with the Text widget is given below.

SN	Option	Description
1	bg	The background color of the widget.
2	bd	It represents the border width of the widget.
3	cursor	The mouse pointer is changed to the specified cursor type, i.e. arrow, dot, etc.
4	exportselection	The selected text is exported to the selection in the window manager. We can set this to 0 if we don't want the text to be exported.
5	font	The font type of the text.
6	fg	The text color of the widget.
7	height	The vertical dimension of the widget in lines.
8	highlightbackground	The highlightcolor when the widget doesn't has the focus.
9	highlightthickness	The thickness of the focus highlight. The default value is 1.
10	highlighcolor	The color of the focus highlight when the widget has the focus.
11	insertbackground	It represents the color of the insertion cursor.
12	insertborderwidth	It represents the width of the border around the cursor. The default is 0.
13	insertofftime	The time amount in Milliseconds during which the insertion cursor is off in the blink cycle.
14	insertontime	The time amount in Milliseconds during which the insertion cursor is on in the blink cycle.
15	insertwidth	It represents the width of the insertion cursor.
16	padx	The horizontal padding of the widget.
17	pady	The vertical padding of the widget.

18	relief	The type of the border. The default is SUNKEN.
19	selectbackground	The background color of the selected text.
20	selectborderwidth	The width of the border around the selected text.
21	spacing1	It specifies the amount of vertical space given above each line of the text. The default is 0.
22	spacing2	This option specifies how much extra vertical space to add between displayed lines of text when a logical line wraps. The default is 0.
23	spacing3	It specifies the amount of vertical space to insert below each line of the text.
24	state	It the state is set to DISABLED, the widget becomes unresponsive to the mouse and keyboard unresponsive.
25	tabs	This option controls how the tab character is used to position the text.
26	width	It represents the width of the widget in characters.
27	wrap	This option is used to wrap the wider lines into multiple lines. Set this option to the WORD to wrap the lines after the word that fit into the available space. The default value is CHAR which breaks the line which gets too wider at any character.
28	xscrollcommand	To make the Text widget horizontally scrollable, we can set this option to the set() method of Scrollbar widget.
29	yscrollcommand	To make the Text widget vertically scrollable, we can set this option to the set() method of Scrollbar widget.

Methods: We can use the following methods with the Text widget.

SN	**Method**	**Description**
1	delete(startindex, endindex)	This method is used to delete the characters of the specified range.
2	get(startindex, endindex)	It returns the characters present in the specified range.
3	index(index)	It is used to get the absolute index of the specified index.
4	insert(index, string)	It is used to insert the specified string at the given index.
5	see(index)	It returns a boolean value true or false depending upon whether the text at the specified index is visible or not.

12. Tkinter messagebox:

The messagebox module is used to display the message boxes in the python applications. There are the various functions which are used to display the relevant messages depending upon the application requirements.

The syntax to use the messagebox is given below.

Syntax: messagebox.function_name(title, message [, options])

Parameters:

- **function_name:** It represents an appropriate message box function.

- **title:** It is a string which is shown as a title of a message box.
- **message:** It is the string to be displayed as a message on the message box.
- **options:** There are various options which can be used to configure the message dialog box.

The two options that can be used are default and parent.

1. default

The default option is used to mention the types of the default button, i.e. ABORT, RETRY, or IGNORE in the message box.

2. parent

The parent option specifies the parent window on top of which, the message box is to be displayed.

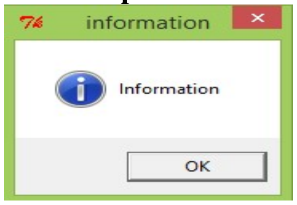
There is one of the following functions used to show the appropriate message boxes. All the functions are used with the same syntax but have the specific functionalities.

1. **showinfo():** The showinfo() messagebox is used where we need to show some relevant information to the user.

Example:

```
# !/usr/bin/python3
  from tkinter import *
  from tkinter import messagebox
  top = Tk()
  top.geometry("100x100")
  messagebox.showinfo("information","Information")
  top.mainloop()
```

Output:

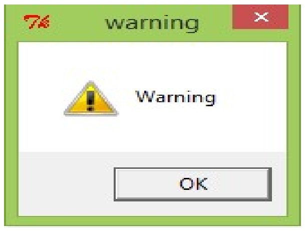


2. **showwarning():** This method is used to display the warning to the user. Consider the following example**.**

Example:

```
# !/usr/bin/python3
from tkinter import *
  from tkinter import messagebox
  top = Tk()
top.geometry("100x100")
messagebox.showwarning("warning","Warning")
  top.mainloop()
```

Output:

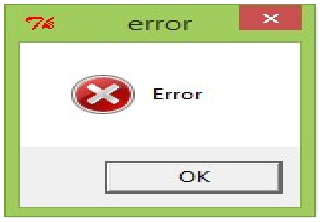


3. **showerror():** This method is used to display the error message to the user. Consider the following example.

Example:

```
# !/usr/bin/python3
from tkinter import *
from tkinter import messagebox

top = Tk()
top.geometry("100x100")
messagebox.showerror("error","Error")
top.mainloop()
```

Output:

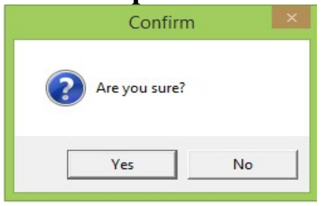


4. **askquestion():** This method is used to ask some question to the user which can be answered in yes or no. Consider the following example.

Example

```
# !/usr/bin/python3
from tkinter import *
from tkinter import messagebox
  top = Tk()
top.geometry("100x100")
messagebox.askquestion("Confirm","Are you sure?")
top.mainloop()
```

Output:

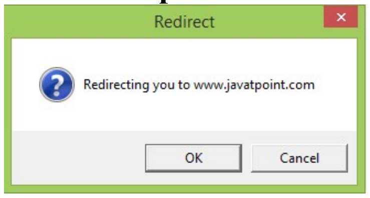


5. **askokcancel():** This method is used to confirm the user's action regarding some application activity. Consider the following example.

Example:

```
# !/usr/bin/python3
from tkinter import *
from tkinter import messagebox
 top = Tk()
top.geometry("100x100")
messagebox.askokcancel("Redirect","Redirecting you to www.javatpoint.com")
top.mainloop()
```

Output:

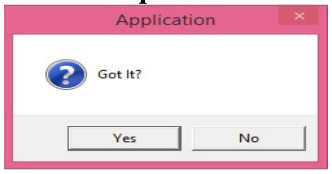

Redirect
Redirecting you to www.javatpoint.com

6. **askyesno():** This method is used to ask the user about some action to which, the user can answer in yes or no. Consider the following example.

Example

```
# !/usr/bin/python3
from tkinter import *
from tkinter import messagebox
 top = Tk()
top.geometry("100x100")
messagebox.askyesno("Application","Got It?")
top.mainloop()
```

Output:

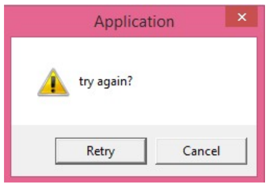

Application
Got It?

7. **askretrycancel():** This method is used to ask the user about doing a particular task again or not. Consider the following example.

Example

```
# !/usr/bin/python3
from tkinter import *
from tkinter import messagebox
top = Tk()
top.geometry("100x100")
messagebox.askretrycancel("Application","try again?")
 top.mainloop()
```

Output:


Application

try again?

Assignments:

Practice Set:

1. Write a Python GUI program to import Tkinter package and create a window and set its title.
2. Write a Python GUI program to create two buttons exit and hello using tkinter module.
3. Write a Python GUI program to create a Checkbutton widget using tkinter module. Write a Python GUI program to create three single line text-box to accept a value from the user using tkinter module.
4. Write a Python GUI program to create three radio buttons widgets using tkinter module.
5. Write a Python GUI program to create a Listbox bar widgets using tkinter module.

Set A:

1. Write Python GUI program to display an alert message when a button is pressed.
2. Write Python GUI program to Create background with changing colors
3. Write a Python GUI program to create a label and change the label font style (font name, bold, size) using tkinter module.
4. Write a Python GUI program to create a Text widget using tkinter module. Insert a string at the beginning then insert a string into the current text. Delete the first and last character of the text.
5. Write a Python GUI program to accept dimensions of a cylinder and display the surface area and volume of cylinder.
6. Write Python GUI program that takes input string and change letter to upper case when a button is pressed.

Set B:

1. Write Python GUI program to take input of your date of birth and output your age when a button is pressed.
2. Write Python GUI program which accepts a sentence from the user and alters it when a button is pressed. Every space should be replaced by *, case of all alphabets should be reversed, digits are replaced by ?.
3. Write Python GUI A program to create a digital clock with Tkinter to display the time.
4. Create a program to generate a random password with upper and lower case letters.
5. Write Python GUI program which accepts a number n to displays each digit of number in words.
6. Write Python GUI program to accept a decimal number and convert and display it to binary, octal and hexadecimal number.
7. Write Python GUI program to add items in listbox widget to print and delete the selected items from listbox on button click. Provide two separate button for print and delete.

8. Write Python GUI program to add menu bar with name of colors as options to change the background color as per selection from menu option.
9. Write Python GUI program to accept a number n and check whether it is Prime, Perfect or Armstrong number or not. Specify three radio buttons.
10. Write a Python GUI program to create a label and change the label font style (font name, bold, size). Specify separate checkbuttton for each style.

Set C:

1. Write a Python GUI program to implement simple calculator.

Evaluation

0: Not Done [ ]	**1: Incomplete [ ]**	**2: Late Complete [ ]**
3: Need Improvement [ ]	**4: Complete [ ]**	**5: Well Done [ ]**

Signature of Instructor