Savitribai Phule Pune University

T. Y. B. B. A. (C. A.) Semester-V

(NEP 2024 Pattern)

Laboratory Book

MJ-303-PJP : Computer Laboratory based on Java and Python Programming

Student Name: _____

College Name:_____

Roll No.: _____ Division: _____ Seat No:_____

Academic Year: 20____ - 20_____

Savitribai Phule Pune University

T. Y. B. B. A. (C. A.) Semester-V

(NEP 2024 Pattern)

CERTIFICATE

This is to certify that Mr./Ms._____

Seat Number _____of T.Y.B.B.A. (C.A) Sem-V has successfully completed Computer Laboratory based on Java and Python Programmingin the year 20___- 20___ .

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 javaand 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 Python assignments.

   Section-I: Core java is divided into five assignments.
   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
- JDK

Assignment Completion Sheet

Section-I: Java Programming			
Sr. No.	**Assignment Name**	**Marks (out of 5)**	**Teacher's Sign**
1	Introduction to Java		
2	Object-Oriented Programming Concepts		
3	Packages & Collection		
4	File and Exception Handling		
5	Multithreading & Networking		
6	User Interface with AWT and Swing		
7	JDBC		
Total (Out of 25)			
Total (Out of 5)			

Section-II: Python Programming			
Sr. No.	**Assignment Name**	**Marks (out of 5)**	**Teacher's Sign**
1	Fundamentals of Python		
2	Modules and Packages		
3	File & Exception Handling		
4	Introduction to Data Analytics and Data Visualization		
5	Introduction to Flask		
6	Database Connectivity		
Total (Out of 40)			
Total (Out of 5)			

Instructor Signature:

Section – I
Java Programming

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.**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.

Evaluation

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

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 and returning objects

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 :

Example: Class_name [] objArray;
or
Class_name objArray[];

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();
}
}
}
```

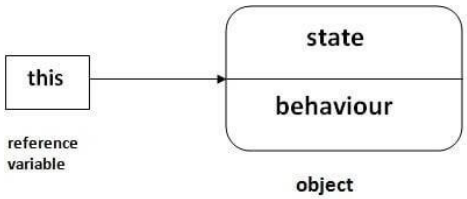
this keyword:

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 and inner classes

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();
}}
```

String handling and wrapper classes

String handling:

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.	Method	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	**booleancontentEquals(StringBuffersb)**	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	**booleanendsWith(String suffix)**	Tests if this string ends with the specified suffix.
10	**booleanequals(Object anObject)**	Compares this string to the specified object.
11	**booleanequalsIgnoreCase(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	**booleanmatches(String regex)**	Tells whether or not this string matches the given regular expression.
27	**booleanregionMatches(booleanignoreCase, int toffset, String other, int ooffset, int len)**	Tests if two string regions are equal.
28	**booleanregionMatches(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	**booleanstartsWith(String prefix)**	Tests if this string starts with the specified prefix.
35	**booleanstartsWith(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 show use of substring.

```
Class stringop1
{
      public static void main(String args[])
      {
            String st1= "Java is platform independent";
            // to construct a substring we write
            String sub=st1.substring(9,16);
            // print these values
            System.out.println("Original String is:+st1);
            System.out.println("Substring is:+sub);
      }
}
```

Example: Java program to show use of Concatenating string.

```
Class stringop2
{
      public static void main (String args[])
      {
            String st1= "Java is platform independent";
            String st2= "and more secure"
            String con=st1.concat(st2);

        // print these values
            System.out.println("Original String is:" +st1);
            System.out.println("Orignial String is:" +st2)
            System.out.println("Concatenating string is:" +con);
      }
}
```

Wrapper classes:

In Java, wrapper classes allow primitive data types to be represented as objects. This enables primitives to be used in object-oriented features such as collections, generics, and APIs that require objects.Each wrapper class encapsulates a corresponding primitive value inside an object (e.g., Integer for int, Double for double).

- Java provides wrapper classes for all eight primitive data types to support object-based operations.

Autoboxing:

The automatic conversion of primitive types to the object of their corresponding wrapper classes is known as autoboxing. For example: conversion of int to Integer, long to Long, double to Double, etc.

Unboxing:

Unboxing is the automatic conversion of a wrapper class object back into its corresponding primitive type.

Example:

```
class WrapperExample
{
	public static void main(String[] args)
	  {
	  int b = 357;
	// Autoboxing: primitive int -> Integer object
		Integer a = b;
		System.out.println("The primitive int b is: " + b);
		System.out.println("The Integer object a is: " + a);
	}
}
```

Common Methods of Wrapper Classes

Sr. No.	Methods	Description
1	**parseInt(String s)**	Converts a String into its corresponding primitive type
2	**valueOf(String s)**	Converts a String into a wrapper object
3	**valueOf(primitive)**	Converts a primitive value into a wrapper object
4	**xxxValue()**	Converts a wrapper object into its primitive type
5	**toString()**	Converts wrapper object into a String
6	**compareTo(Xxxobj)**	Compares two wrapper objects
7	**equals(Object obj)**	Compares values, not references
8	**hashCode()**	Returns hash code of the object
9	**min(x, y)**	Returns minimum of two values
10	**max(x, y)**	Returns maximum of two values
11	**sum(x, y)**	Returns sum of two values
12	**compare(x, y)**	Compares two primitive values
13	**isNaN()**	Checks if value is Not a Number (Float/Double only)
14	**isInfinite()**	Checks infinite value (Float/Double only)
15	**decode(String s)**	Decodes decimal, hex, or octal string
16	**parseBoolean(String s)**	Converts string to boolean

Example:

```
class WrapperExmple2
{
	public static void main(String[] args)
	  {
	   byte b = 1;
		Byte byteObj = Byte.valueOf(b);

		int i = 10;
		Integer intObj = Integer.valueOf(i);

		float f = 18.6f;
		Float floatObj = Float.valueOf(f);

		double d = 250.5;
		Double doubleObj = Double.valueOf(d);

		char c = 'a';
		Character charObj = c; // autoboxing

		System.out.println("Wrapper Objects:");
		System.out.println(byteObj);
		System.out.println(intObj);
		System.out.println(floatObj);
		System.out.println(doubleObj);
		System.out.println(charObj);

		// Unboxing
		byte bv = byteObj;
		int iv = intObj;
		float fv = floatObj;
		double dv = doubleObj;
		char cv = charObj;

		System.out.println("\nUnwrapped values:");
		System.out.println(bv);
		System.out.println(iv);
		System.out.println(fv);
		System.out.println(dv);
		System.out.println(cv);
	}
}
```

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.

Evaluation:

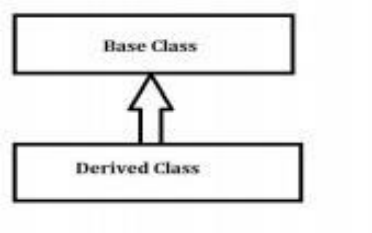
0: Not Done [ ]	**1: Incomplete [ ]**	**2: Late Complete [ ]**
3: Needs Improvement [ ]	**4: Complete [ ]**	**5: WellDone [ ]**

Signature of Instructor

Assignment No. 3: Inheritance, Package and Collection

Inheritance:

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.



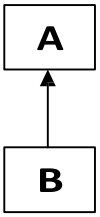
Syntax:

```
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.

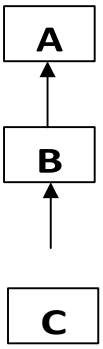


Syntax:

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

B) Multilevel Inheritance:

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

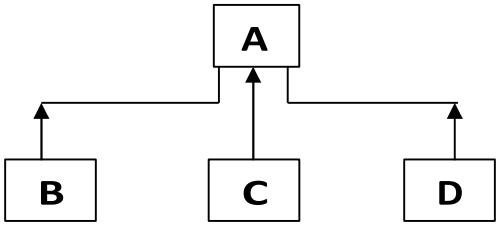


Syntax:

```
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 usingsuper 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();
}
}
```

Abstract Classes and Interfaces

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

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

                     ref = c; ref.Display1();    // now ref refers to a C object
                                                 // 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.

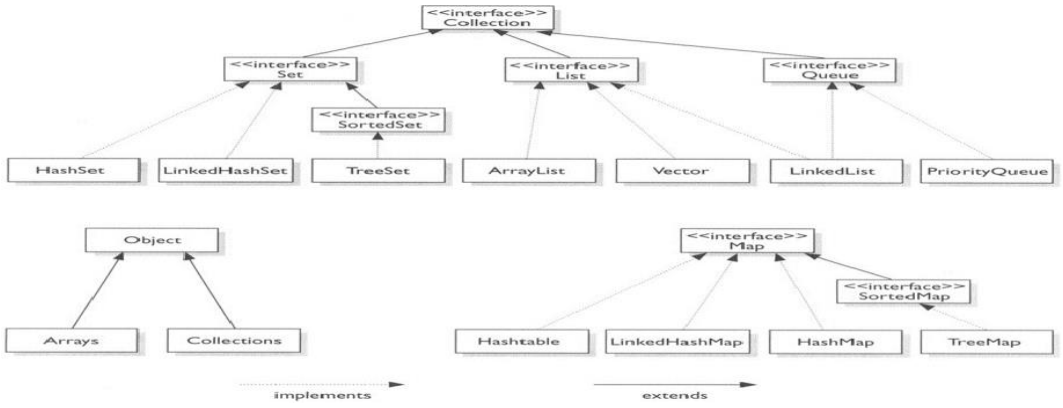
Collection: A collection is an object that groups multiple elements into a single unit i.e. single unit of objects. Collections are used to store, retrieve, manipulate, and communicate aggregate data.

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

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:

- **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());
}
}
}
```

- **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.Entrym: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:

- 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)
- 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)
- 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.
- 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'.
- 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:
- Find Maximum of two numbers.
- 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:
   - Display the contents of the List using an iterator
   - 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:
    - Fibonacci series
    - Cube of numbers
    - Square of numbers
6. Write a java program to generate 'n' terms of the above series.

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 f 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

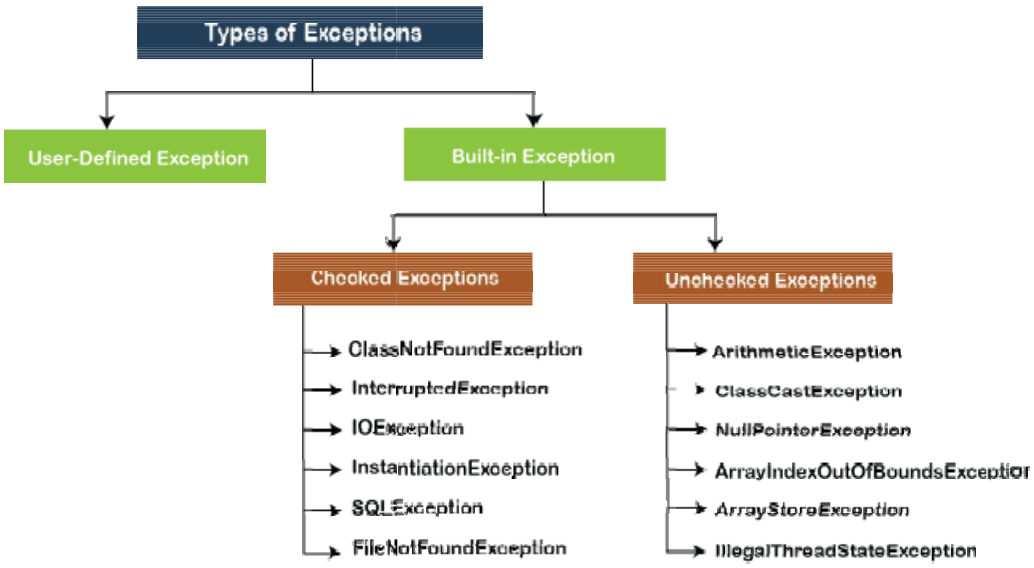
Abnormal condition of a program that terminates its execution is called Exception.

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**


Types of Exceptions
User-Defined Exception
Built-in Exception
Checked Exceptions
Unchecked Exceptions
ClassNotFoundException
InterruptedException
IOException
InstantiationException
SQLException
FileNotFoundException
ArithmeticException
ClassCastException
NullPointerException
ArrayIndexOutOfBoundsException
ArrayStoreException
IllegalThreadStateException

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)
{

}
finally
{
}
}
}
System.out.println ("Error Is " + obj);
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)))
{
}
else
}throw new NameValExc(); flag=0;
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)
{
```

```
}
else
{ throw new ZeroNumExc();

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 >]

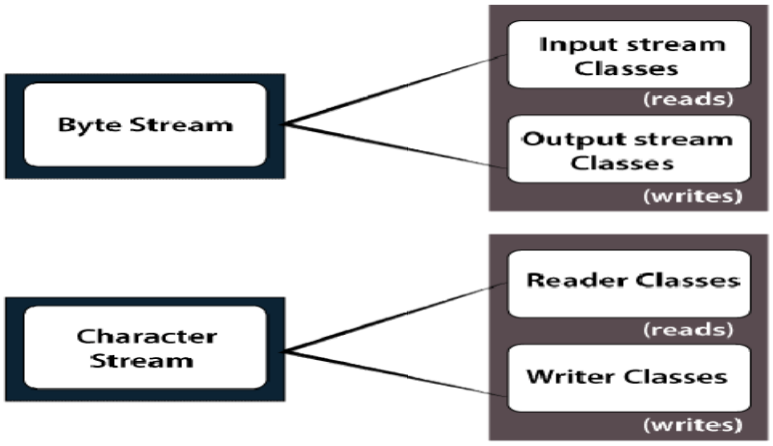
File Handling:

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,wecan 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 St ream 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)
{
}
else
{
}}}
System.out.println("The name of the file is changed.");
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.

Evaluation:

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

Signature of Instructor

Assignment No.-5 Multithreading and Networking

Multithreading in Java

It is a process of executing multiple threads simultaneously to maximize CPU utilization. A **thread** is the smallest, lightweight unit of execution within a program. **By using multithreading, you can perform multiple background operations concurrently without blocking the main application flow.**

Thread Creation:

Java provides two primary built-in mechanisms to create and run threads:

1. **Implementing the Runnable Interface**:

   Create a thread is to implement the **Runnable** interface. The thread can be run by passing an instance of the class to a Thread object's constructor and then calling the thread's start() method.

   Example :

```
public class RunnableDemo implements Runnable
{
public void run(){
System.out.println("This code is running in a thread");
}
public static void main(String[] args){
RunnableDemo obj=new RunnableDemo();
Thread thread=new Thread(obj);
thread.start();
System.out.println("This code is outside of the thread");
}
}
```

2. **Extending the Thread Class**:

   A simpler approach where your class inherits directly from java.lang.Thread. It can be created by extending the Thread class and overriding its run() method.

   Example:

```
public class TDemo extends Thread {
public void run(){
System.out.println("This code is running in a thread");
}
public static void main(String[] args){
TDemo thread =new TDemo();
thread.start();
System.out.println("This code is outside of the thread");
}
}
```

Life cycle of Thread

- **Newborn** : When a thread is created (by new statement) but not yet to run, it is called in Newborn state. In this state, the local data members are allocated and initialized.

- R**unnable** : The Runnable state means that a thread is ready to run and is awaiting for the control of the processor, or in other words, threads are in this state in a queue and wait their turns to be executed.
- **Running** : Running means that the thread has control of the processor, its code is currently being executed and thread will continue in this state until it get preempted by a higher priority thread, or until it relinquishes control.
- **Blocked** : A thread is Blocked means that it is being prevented from the Runnable (or Running) state and is waiting for some event in order for it to reenter the scheduling queue.
- **Dead** : A thread is Dead when it finishes its execution or is stopped (killed) by another thread.

Methods for controlling a thread's state
start ( ) : A newborn thread with this method enter into Runnable state and Java run time create a system thread context and starts it running. This method for a thread object can be called once only
stop( ) : This method causes a thread to stop immediately. This is often an abrupt way to end a thread.
suspend( ) : This method is different from stop( ) method. It takes the thread and causes it to stop running and later on can be restored by calling it again.
resume ( ) : This method is used to revive a suspended thread. There is no guarantee that the thread will start running right way, since there might be a higher priority thread running already, but, resume()causes the thread to become eligible for running.
sleep (int n) : This method causes the run time to put the current thread to sleep for n milliseconds. After n milliseconds have expired, this thread will become elligible to run again.
yield( ) : The yield() method causes the run time to switch the context from the current thread to the next available runnable thread. This is one way to ensure that the threads at lower priority do not get started.

Example:

```
class A extends Thread {
public void run() {
for(int  i=1;i<=5;i++){
if(i==1)
yield();
System.out.println("A="+i);
}
System.out.println("Exit from A");
}
}

class B extends Thread {
public void run() {
for(int  i=1;i<=5;i++){
System.out.println("B="+i);
if(i==3)
stop();
}
System.out.println("Exit from B");
}
}

class C extends Thread {
```

```
public void run() {
for(int  i=1;i<=5;i++){
System.out.println("C="+i);
if(i==1) {
try{
sleep(1000);
}catch(Exception e){}
}
}
System.out.println("Exit from C");
}
}
Class ThreadDemo{
public static void main(String args[]){
A  a1=new A();
B  b1=new B();
C  c1=new C();
a1.start();
b1.start();
c1.start();
}}
```

Thread Priority

Every Java thread has a priority that helps the operating system determine the order in which threads are scheduled

Built-in Property Constants of Thread Class

Java thread priorities are in the range between MIN_PRIORITY (a constant of 1) and MAX_PRIORITY (a constant of 10). By default, every thread is given priority NORM_PRIORITY (a constant of 5).

Thread Priority Setter and Getter Methods

- **Thread.getPriority() Method**: This method is used to get the priority of a thread.
- **Thread.setPriority() Method**: This method is used to set the priority of a thread, it accepts the priority value and updates an existing priority with the given priority.

Example

```
public class ThreadA{
public void display(){
System.out.println("Name: "+Thread.currentThread().getName());
System.out.println("Thread Priority: +Thread.currentThread().getPriority());
}
}
class Demo
```

```
{
public static void main(String args[]){
ThreadA thread =new ThreadA();
thread.display();
}
}
```

Thread Synchronization

Thread synchronization in Java is a mechanism that controls the access of multiple threads to shared resources to prevent data inconsistency and race conditions.

Core Ways to Implement Synchronization

You can implement synchronization in Java using the synchronized keyword in three major ways

1. **Synchronized Method**

Locks the entire method. A thread must acquire the **object-level lock** of the specific instance to execute the method.

Example

```
class Counter {
   private int count = 0;
   public synchronized void increment() {
      count++;
   }
   public int getCount() {
      return count;
   }
}
class ThreadDemo extends Thread {
   Counter counter;

ThreadDemo(Counter counter) {
this.counter = counter;
   }

   public void run() {
      for (int i = 0; i < 1000; i++) {
counter.increment();
      }
   }
}

public class MDemo {
   public static void main(String[] args) {
   Counter counter = new Counter();
   ThreadDemo t1 = new ThreadDemo(counter);
   ThreadDemo t2 = new ThreadDemo(counter);
      t1.start();
      t2.start();
      try {
         t1.join();
```

```
            t2.join();
        } catch (InterruptedException e) {

    e.printStackTrace();
        }
      System.out.println("Final count: " + counter.getCount());
    }
}
```

2. Synchronized Block

Locks only a specific section of code inside a method. It provides better performance than a synchronized method because it minimizes the time the lock is held.

Example

```
class Counter {
    private int count = 0;
    public void increment() {
        synchronized (this) {
            count++;
        }
    }

    public int getCount() {
        return count;
    }
}

class ThreadDemo extends Thread {
    Counter counter;
    ThreadDemo(Counter counter) {
    this.counter = counter;
    }
    public void run() {
        for (int i = 0; i < 1000; i++) {
          counter.increment();
        }
    }
}

public class MainDemo {
    public static void main(String[] args) {
        Counter counter = new Counter();
ThreadDemo t1 = new ThreadDemo(counter);
ThreadDemo t2 = new ThreadDemo(counter);

        t1.start();
        t2.start();
```

```
        try {
            t1.join();
            t2.join();
        } catch (InterruptedException e)
{
e.printStackTrace();
        }

System.out.println("Final count: " + counter.getCount());
    }
}
```

3. Static Synchronization

Locks at the **class level** instead of the object level. This is used when you need to protect static data or when multiple threads are accessing separate object instances but need a global lock

Example

```
class Counter {
    private static int count = 0;

    public static synchronized void increment() {
        count++;
    }

    public static int getCount() {
        return count;
    }
}
```

Inter-thread Communication in Java

Inter-thread Communication in Java allows synchronized threads to communicate with each other. Java provides wait(), notify(), and notifyAll() methods to facilitate this communication.

Example

```
class SharedResource {
    private boolean ready = false;
    synchronized void produce() {
        try {
            while (ready) {
                wait();
            }
System.out.println("Producing...");
            ready = true;
            notify();
        } catch (InterruptedException e) {
          e.printStackTrace();
        }
    }
```

```
    synchronized void consume() {
        try {
            while (!ready) {
                wait();
            }
System.out.println("Consuming...");
            ready = false;
            notify();
        } catch (InterruptedException e) {
e.printStackTrace();
        }
    }
}
public class InterThreadDemo {
    public static void main(String[] args) {
SharedResource resource = new SharedResource();

        Thread producer = new Thread(resource::produce);
        Thread consumer = new Thread(resource::consume);

producer.start();
consumer.start();
    }
}
```

Networking

Java Networking is the concept of connecting two or more computing devices to exchange data and resources. It allows Java applications to communicate over the internet or local networks.

Common Network Protocols

1. TCP (Transmission Control Protocol)

TCP is a connection-oriented protocol that ensures data is delivered accurately and in the correct order. It establishes a connection before communication and verifies successful data transfer.

2. UDP (User Datagram Protocol)

UDP is a connectionless protocol designed for fast data transmission without delivery guarantees. It sends data packets independently, making it suitable for real-time communication.

Java Networking Terminology

- **IP Address:** Logical address that identifies a device on a network (e.g., 192.168.0.1). Range: 0.0.0.0 – 255.255.255.255.
- **Port Number:** Identifies applications uniquely on a device. Range: 0 – 65,535.
- **Protocol:** Set of rules for communication (e.g., TCP, FTP, HTTP).

- **MAC Address:** Unique identifier assigned to a network interface card, written in hexadecimal.
- **Socket: Endpoint for two-way communication between devices.**
- **Connection-Oriented Protocol: Requires connection setup (e.g., TCP).**
- **Connectionless Protocol:** No prior connection setup, data sent directly (e.g., UDP).

Java Networking classes

The java.net package provides classes to handle networking operations in Java. Commonly used classes are:

DatagramPacket: Represents a data packet used in UDP-based, connectionless communication.
InetAddress: Represents IP addresses and hostnames, supporting both IPv4 and IPv6.
ServerSocket: Implements the server side of a TCP connection and listens for client requests on a port.
Socket: Represents a client-side TCP connection used to send and receive data.
DatagramSocket: Used for sending and receiving UDP packets and supports broadcast communication.
Proxy: Represents proxy settings for a connection. Useful for network security and anonymity.
URL: Represents a Uniform Resource Locator pointing to a web resource such as a file or web page.
URLConnection: Represents a communication link between a Java application and a URL resource.
HttpURLConnection: A subclass of URLConnection that supports HTTP-specific operations like GET and POST.
URLEncoder/ URLDecoder: Encode and decode strings to make them safe for transmission in URLs.

Socket Programming

Socket programming enables communication between client and server applications over TCP or UDP.

Steps for TCP Communication

1. Server creates a ServerSocket and listens on a port.
2. Client creates a Socket and requests a connection.
3. Server accepts the connection and returns a Socket.
4. Data is exchanged via I/O streams (InputStream, OutputStream).
5. Connection is closed when communication ends.

Socket Class

The Socket class represents a TCP client-side connection. It allows sending, receiving data, and closing connections. Each socket is associated with one remote host, for another host a new socket must be created.

Constructor of Socket class

public Socket(String host, int port) throws UnknownHostException,IOException
public Socket(InetAddress address, int port) throws IOException
public Socket(String host, int port, InetAddress address, int localport) throws IOException
public Socket()

Common Methods:

- **connect(SocketAddress host, int timeout):** Connects the socket to a given host.

- **getPort():** Returns the remote port of the connection.
- **getInetAddress():** Returns the remote IP address.
- **getLocalPort():** Returns the local port of the socket.
- **getRemoteSocketAddress():** Returns the remote socket address.
- **getInputStream():** Returns input stream of the socket.
- **getOutputStream():** Returns output stream of the socket.
- **close():** Closes the socket connection.

ServerSocket Class

The ServerSocket class implements the server side of a TCP connection. It listens for client requests on a given port. If the port is unavailable, the constructor throws an exception.

Constructor of ServerSocket Class

public ServerSocket(int port)throws IOException
public ServerSocket(int port,int backlog)throws IOException
public ServerSocket(int port,int backlog,InetAddress address)throws IOException
public ServerSocket()throws IOException

Common Methods:

- **getLocalPort():** Returns the port the server is listening on.
- **setSoTimeout(int timeout):** Sets timeout while waiting for a client.
- **accept():** Waits and accepts a client connection.
- **bind(SocketAddress host, int backlog):** Binds server to a specific address and port.

Example: Basic Client-Server Communication

Client side

```
import java.io.*;
import java.net.*;
public class ClientSide {
public static void main(String[] args) {
try {
Socket socket = new Socket("127.0.0.1", 5000);
DataOutputStream out = new DataOutputStream(socket.getOutputStream());
BufferedReader br = new BufferedReader(new InputStreamReader(System.in)));
System.out.println("Connected to Server");
 String  str;
while (!(str = br.readLine()).equals("End")) {
out.writeUTF(str);
          }
       } catch (IOException e) {
e.printStackTrace();
       }
   }
}
```

Server Side

```
import java.io.*;
import java.net.*;
public class ServerSide {
public static void main(String[] args) {
try {
ServerSocket server = new ServerSocket(5000);
Socket socket = server.accept();
DataInputStream in = new DataInputStream(socket.getInputStream()));
System.out.println("Server started. Waiting for client...");
String str;
while (!(str = in.readUTF()).equals("End")) {
System.out.println("Client: " + str);
          }
      } catch (IOException e) {
e.printStackTrace();
      }
   }
}
```

Execution Steps

1. Run ServerSide. It waits for a client.
2. Run ClientSide. It connects to the server.
3. Type messages in the client window, server displays them.

Steps for UDP Communication

In Java, the java.net.DatagramSocket class represents a socket for sending and receiving datagram packets and the java.net.DatagramPacket class represent the data container.

Creating a DatagramSocket

To send or receive a datagram packet in Java, you must first create a DatagramSocket.

Constructors of DatagramSocket class

- **DatagramSocket( ) throws SocketException :** It creates a DatagramSocket bound to any unused port on the local computer.
- **DatagramSocket(int port) throws SocketException :** It creates a DatagramSocket bound to the port specified by port.
- **DatagramSocket(int port, InetAddressipAddress) throws SocketException :** It constructs a DatagramSocket bound to the specified port and InetAddress.
- **DatagramSocket(SocketAddress address) throws SocketException :** It constructs a DatagramSocket bound to the specified SocketAddress.

Methods:

- void send(DatagramPacket packet) throws IOException

- void receive(DatagramPacket packet) throws IOException

  The send( ) method sends packet to the port specified by packet. The receive method waits for a packet to be received from the port specified by packet and returns the result.

  Creating a DatagramPacket
  To send or receive data, you need to create a DatagramPacket.
  Constructor of DatagramPacket Class.

- **DatagramPacket(byte data[ ], int size) :** It specifies a buffer that will receive data and the size of a packet. It is used for receiving data over a DatagramSocket
- **DatagramPacket(byte data[ ], int offset, int size) :** It allows you to specify an offset into the buffer at which data will be stored.
- **DatagramPacket(byte data[ ], int size, InetAddress ipAddress, int port) :** It specifies a target address and port, which are used by a DatagramSocket to determine where the data in the packet will be sent.
- **DatagramPacket(byte data[ ], int offset, int size, InetAddress ipAddress, int port) :** It transmits packets beginning at the specified offset into the data.

  Server Side

```
public class UDPServer {
public static void main(String[] args) throws IOException {
DatagramSocket socket = new DatagramSocket(3000);
byte[] buffer = new byte[1024];
DatagramPacket packet = new DatagramPacket(buffer, buffer.length);
while (true) {
socket.receive(packet);
System.out.println("Received: " + new String(packet.getData()));
socket.send(packet); // echo back
    }
  }
}
```

  Client Side

```
public class UDPClient {
public static void main(String[] args) throws IOException {
DatagramSocket socket = new DatagramSocket();
    String message = "Hello, Friend!";
byte[] buffer = message.getBytes();
InetAddress address = InetAddress.getByName("localhost");
DatagramPacket packet = new DatagramPacket(buffer, buffer.length, address, 3000);
socket.send(packet);
socket.receive(packet);
System.out.println("Received: " + new String(packet.getData()));
socket.close();
  }
}
```

InetAddress Class

The InetAddress class represents IP addresses and hostnames. It supports both IPv4 and IPv6.

Types of Addresses

- **Unicast:** Identifies a single interface.
- **Multicast:** Identifies a group of interfaces.
  Common Methods:
- **getByAddress(byte[] addr):** Returns an InetAddress from raw IP.
- **getByName(String host):** Returns IP of a given hostname.
- **getLocalHost():** Returns local host IP.
- **getHostName():** Returns hostname.
- **getHostAddress():** Returns IP address as string.

Example:

```
import java.net.*;
public class InetAddressExample {
public static void main(String[] args) throws Exception {
InetAddress local = InetAddress.getLocalHost();
System.out.println("Local Host: " + local);
System.out.println("Host Name: " + local.getHostName());
InetAddress adr = InetAddress.getByName("www.google.org");
System.out.println("Google IP: " + adr);
   }
}
```

URL Class

The URL class represents a Uniform Resource Locator that points to a resource on the web.

Components of a URL

- **Protocol:** How communication happens (http, https, ftp).
- **Hostname:** The server where the resource exists.
- **File/Path:** The location of the resource.
- **Port (optional):** Communication port number.
  Common Methods:
- **getProtocol():** Returns protocol of URL.
- **getHost():** Returns hostname.
- **getPort():** Returns port of the URL.
- **getFile():** Returns filename.
- **getPath():** Returns path of the URL.
- **toString():** Returns URL as a string.
- **getDefaultPort():** Returns default port of the protocol.

Example:

```
import java.net.*;
public class URLExample {
public static void main(String[] args) throws Exception {
 URL url = new URL("https://gmail.com");
System.out.println("Protocol : " + url.getProtocol());
System.out.println("Host : " + url.getHost());
System.out.println("File : " + url.getFile());
System.out.println("Path : " + url.getPath());
System.out.println("Default Port : " + url.getDefaultPort());
   }
}
```

URLConnection class

If you want to retrieve the data associated with URL then the openConnection() method of URL class is used. The openConnection() method makes the connection with actual URL .

Common methods:

getDate() : return date .
getExpiration(): return URL expiration time.
getLastModified(): return last modified date.
getContentLength(): return length of content.

```
public class URLConExample {
public static void main(String[] args) throws Exception
{
      URL url = new URL("https://gmail.com");

System.out.println("Protocol : " + url.getProtocol());
URLConnection cn= u1.openConnection();

System.out.println("Date : " + cn.getDate());
System.out.println("Expired at : " + cn.getExpiration());
System.out.println("length : " + cn.getContentLength());
System.out.println("Last Modified at : " + cn.getLastModified());
   }
}
```

Practice set:

- Program to find port no running on server.
- Program to get remote and local socket address.
- Write a multithreading program to display entered college name 50 times.
- Program to access daytime service from server using socket.
- Write a multithreading program to display given number multiplication table after 3 seconds.

- Write a program to display expiration date & last modified date for the http://www.sal.edu.in.

SET A:

1. Program to read the source code of the Web Page.
2. Write a program to download web page using URL Class.
3. Write a program to demonstrate life cycle of thread.
4. Write a TCP/IP client server program in which client send any string and server responds with reverse string.
5. Write a multi-threading program to print odd number using one thread and even number using other.

SET B:

1. Write a multithreading program for car race.
2. Write a TCP/IP client server program in which client send two strings and server responds with concatenate string.
3. Write a program to create two thread class ,one that display Armstrong numbers between 1 to 500 and another thread display first 20 Fibonacci numbers.
4. Write a client server program using TCP/IP where the client sends n numbers and server responds the sum of n number.

SET C:

1. Write a client server program using TCP/IP where the client sends 10 numbers and server responds with the number in sorted order.
2. Write a Java program that implements a multi-thread application that has three threads. First thread generates random integer every 1 second and if the value is even, second thread computes the square of the number and prints. If the value is odd, the third thread will print the value of cube of the number.
3. Write a Java program that simulates a traffic light. The program lets the user selects one of three lights: red, yellow, or green with radio buttons. On selecting a button, an appropriate message with Stop, Ready, or Go should appear above the buttons in the selected color. Initially, there is no message shown .

Evaluation:

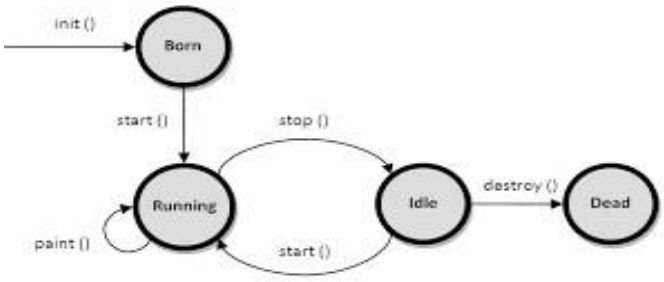
0: Not Done [ ]	**1: Incomplete [ ]**	**2: Late Complete [ ]**
3: Needs Improvement [ ]	**4: Complete [ ]**	**5: WellDone [ ]**

Signature of Instructor

Assignment No. 6: User interface with AWT 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.

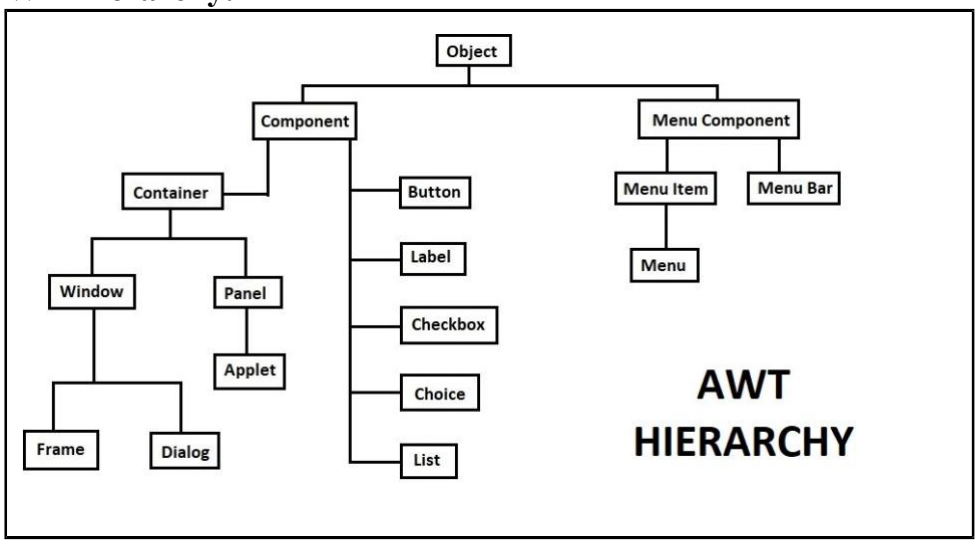
```
import    java.applet.*;
import java.awt.*;
public class MyApplet extends Applet
{
int height, width; public void
init()
{
height = getSize().height; width = getSize().width; setName("MyApplet");
}
public void paint(Graphics g)
{
g.drawRoundRect(10, 30, 120, 120, 2, 3);
}
}
```

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 '+' :
case '-' :
case '*' :
case '/' :
}
ans=no1 + no2; break;
ans=no1 - no2; break;
ans=no1 * no2; break;
ans=no1 / no2; break;
input.setText(Integer.toString(ans));
}
if(btn==clear)
{
input.setText("");
```

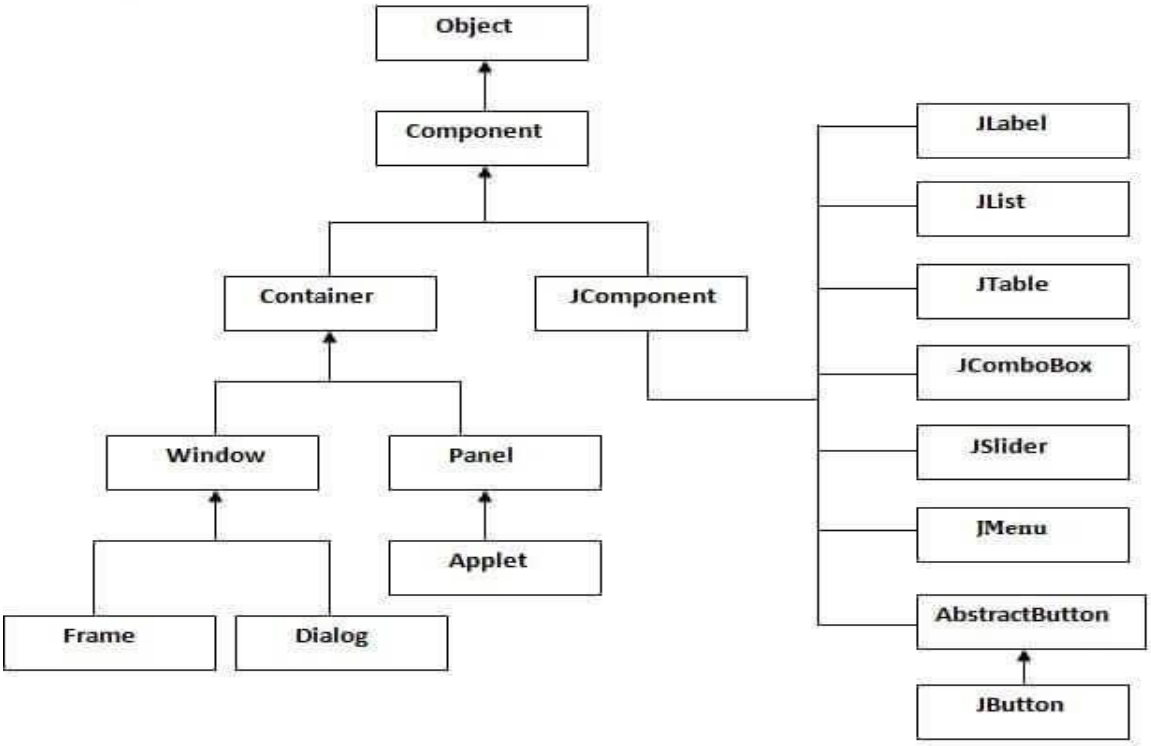
```
input.requestFocus();
}
if(btn==on)
{
input.setEnabled(true);
addn.setEnabled(true);
subn.setEnabled(true);
muln.setEnabled(true);
divn.setEnabled(true);
}
}
}
```

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:

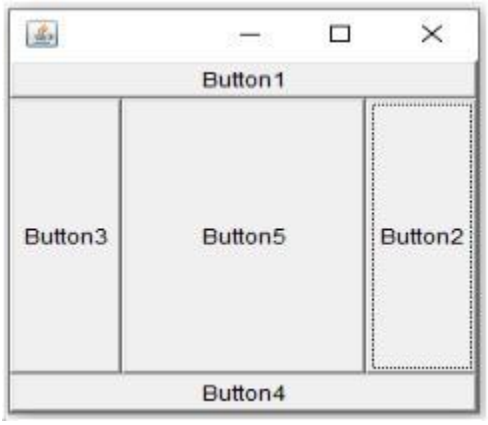
BorderLayout bl = new BorderLayout();

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);

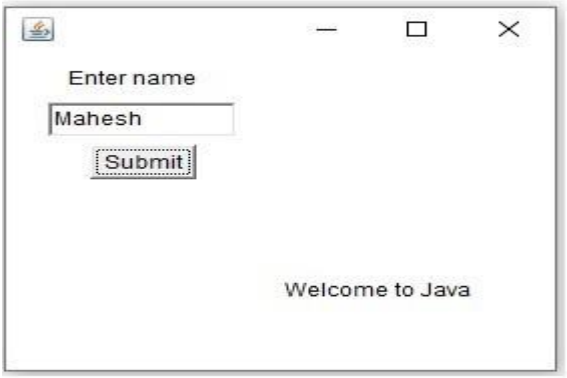
GridLayout gl = new GridLayout(int rows, int cols, int vgap, int hgap);

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

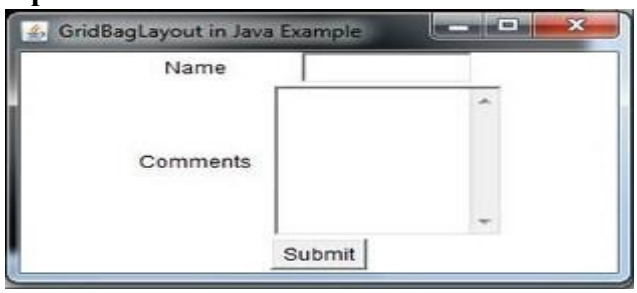
gridwidth, gridheight	Define how many cells will occupy component (height and width). by The default is 1. The indication is relative to the other components of the line or the column. The GridBagConstraints.REMAINDER value specifies that the next component inserted will be the last of the line or the 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. **JPanel** – 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();

Component used in Swing:

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.

Creating a ButtonGroup:

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:

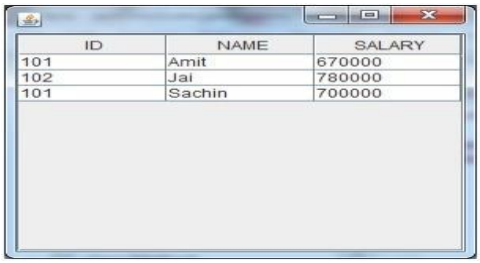
JTable():Creates a table with empty cells.

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:



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 e)	Invoked when the value of the adjustable has changed.
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:

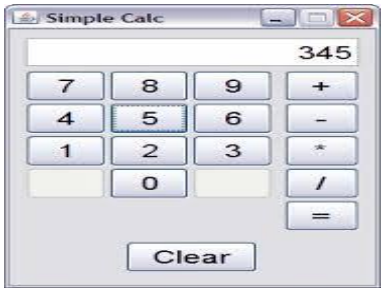
1. Develop an applet that draws a circle. The dimension of the applet should be 500 * 300 pixels the circle should be centered the applet and have a radius of 100 pixels. Display your name centered in a circle(using drawOval() method)
2. Write a program to create a frame using AWT. Implement mouseClicked(), 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 .

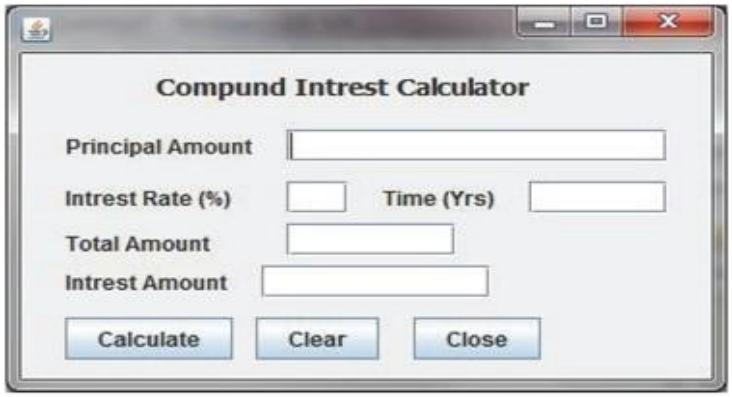


2. Write a java program to implement following. Program should handle appropriate events.

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.

Evaluation:

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

Signature of Instructor

Assignment No. 7: JDBC

Introduction to JDBC :

JDBC (Java Database Connectivity) is a technology used to connect a Java application to a database, by using a JDBC driver. It enables Java programs to execute SQL statements and interact with relational databases.

To develop a JDBC application, the java.sql package must be imported, as it contains the essential classes and interfaces required for database connectivity, SQL statement execution, and result processing.

Jdbc Process:

For connecting java program JDBC Process is used. It consists of following steps:

1. **To import JDBC-API.**
   The name of jdbc-api is java.sql. It consists of following classes and interfaces which are require for the JDBC application. The List is as follow:
   Interfaces:
   Connection
   Statement
   PreparedStatement
   CollableStatement
   ResultSet
   ResultSetMetaData
   DatabaseMetaData
   Classes:
   DriverManager
   Class
2. **To load the driver.**
   Loading is a process used to bring the drivers from operating system to JVM. For loading drivers following statement is used:

   Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
   The forName() is a static method of class ***Class*** used to load given driver into the JVM, If driver is not available then it throws ClassNotFoundException.

3. **To establish the connection between java program and Database or DSN (Data Source Name).**

   Connection cn=DriverManager.getConnection("jdbc:odbc:DSN", "username", "password");

The getConnection () is a static method of DriverManager class used to set connection between java program and database or DSN.

4. **To create a Statement.**
   Statement st=cn.createStatement ();
   Statement interface is used to execute database queries after making the connection with database. It is used to execute static query. The Statement interface has following methods:
   - boolean execute()
   - int executeUpdate()
   - ResultSet executeQuery()
5. **To generate the result.**
   Consider Emp (Eno, EName, Salary) table.
   ResultSet rs = st . executeQuery("select * from emp");
   After execution of database query, data from Emp table will get store into ResultSet object rs.

Emp Table

ENo	EName	Salary
101	Advait	70,000
102	Ajay	60,000
103	Vikas	80,000

ResultSet rs

ENo	EName	Salary
101	Advait	70,000
102	Ajay	60,000
103	Vikas	80,000

For reading data record wise from ResultSet interface next() method is used. rs.next()
rs.getString(2); => ***Advait***

6. **To close the Connection.**
   cn.close();

JDBC Components:

1. **DriverManager Class:**

   - The DriverManager class acts as an interface between user and drivers.
   - It keeps track of the drivers that are available and handles establishing a connection between a database and the appropriate driver.
   - The DriverManager class maintains a list of Driver classes that have registered themselves by calling the method **DriverManager.registerDriver ().**

   Methods used in DriverManager class

Method	Description
public static void registerDriver(Driver driver)	It is used to register the given driver with DriverManager.
public static void deregisterDriver(Driver driver)	It is used to deregister the given driver (drop the driver from the list) with DriverManager.
public static Connection getConnection(String url)	It is used to establish the connection with the specified url.
public static Connection getConnection(String url, String userName, String password)	It is used to establish the connection with the specified url, username and password.

Syntax:

Connection cn= DriverManager.getConnection(ConnectionString,userID,Password);

2. Connection Interface:

- A Connection is the session between java application and database
- The Connection interface is a factory of Statement, PreparedStatement, and DatabaseMetaData i.e. object of Connection can be used to get the object of Statement and DatabaseMetaData
- The Connection interface provide many methods for transaction management like commit(), rollback() etc.
- By default, connection commits the changes after executing queries.

Methods used in Connection Interface:

Method	Description
public Statement createStatement()	Creates a statement object that can be used to execute SQL queries.
public Statement createStatement(int resultSetType,int resultSetConcurrency)	Creates a Statement object that will generate ResultSet objects with the given type and concurrency.
public void setAutoCommit(boolean status)	It is used to set the commit status. By default it is true.
public void commit()	It saves the changes made since the previous commit/rollback permanent.

public void rollback()	Drops all changes made since the previous commit/rollback.
public void close()	closes the connection and Releases a JDBC resources immediately.

Syntax:

Connection cn; //creating a Connection object

3. Statement Interface:

Statement st = cn.createStatement (); **//creating**
Statement object There are three types of Statements

a. Statement
b. PreparedStatement
c. CallableStatement

a. Statement Interface:

- The Statement interface provides methods to execute queries with the database.
- The statement interface is a factory of ResultSet i.e. it provides factory method to get the object of ResultSet
- The important methods of Statement interface are as follows:

Method	Description
public ResultSet executeQuery(String sql)	It is used to execute SELECT query. It returns the object of ResultSet.
public int executeUpdate(String sql)	It is used to execute specified query, it may be create, drop, insert, update, delete etc.
public boolean execute(String sql)	It is used to execute queries that may return multiple results.
public int[] executeBatch()	It is used to execute batch of commands.

b. PreparedStatement:

- The PreparedStatement interface is a sub interface of Statement.
- It is used to execute parameterized precompiled query.
- Let's see the example of parameterized query:
  String sql="insert into emp values(?,?,?)";
- Here, we are passing parameter (?) as a placeholder for the values. Its value will be set by calling the setter methods of PreparedStatement.
- The performance of the application will be faster if you use PreparedStatement interface because query is compiled only once.

- The prepareStatement () method of Connection interface is used to return the object of PreparedStatement.
- **Syntax**

Method	Description
public void setInt(int paramIndex, int value)	Sets the integer value to the given parameter index.
public void setString(int paramIndex, String value)	Sets the String value to the given parameter index.
public void setFloat(int paramIndex, float value)	Sets the float value to the given parameter index.
public void setDouble(int	Sets the double value to the given parameter
paramIndex, double value)	index.
public int executeUpdate()	Executes the query. It is used for create, drop, insert, update, delete etc.
public ResultSet executeQuery()	Executes the select query. It returns an instance of ResultSet.

- **CallableStatement Interface:**
  - CallableStatement interface is used to call the stored procedures and functions
  - For Example: Suppose you need the get the age of the employee based on the date of birth, you may create a function that receives date as the input and returns age of the employee as the output.
  - The prepareCall() method of Connection interface returns the instance of CallableStatement.
  - **Syntax:**

```
public CallableStatement prepareCall("{ call procedurename(?,?...?)}");
```

- **Syntax:**

```
CallableStatement stmt=cn.prepareCall("{call

Disp(?,?)}"); Here it calls the procedure Disp

that receives 2 arguments.
```

4. ResultSet Interface

- The object of ResultSet maintains a cursor pointing to a row of a table. Initially, cursor points to before the first row.
- ResultSet object can be moved forward only and it is not updatable.
- But we can make this object to move forward and backward direction by passing

either TYPE_SCROLL_INSENSITIVE or TYPE_SCROLL_SENSITIVE in **createStatement(int, int)** method as well as we can make this object as updatable.

- **There are three Types of ResultSet:**

Type	Description
ResultSet.TYPE_FORWARD_ONLY (Default)	The cursor can only move forward in the result set.
ResultSet.TYPE_SCROLL_INSENSITIVE	The cursor can scroll forward and backward, and the result set is not sensitive to changes made by others to the database that occur after the result set was created.
ResultSet.TYPE_SCROLL_SENSITIVE.	The cursor can scroll forward and backward, and the result set is sensitive to changes made by others to the database that occur after the result set was created.

For Example

Statement st = cn.createStatement (ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE);

- Syntax: *Using Statement Interface*

```
Statement st = cn.createStatement();

ResultSet rs = st.executeQuery("select * from Emp");

String sql = "select * from Emp";

PreparedStatement ps =

cn.prepareStatement(sql); ResultSet

rs = st.executeQuery();
```

- Methods used in ResultSet interface:

Method	Description
public boolean next()	It is used to move the cursor to the one row next from the current position.
public boolean previous()	It is used to move the cursor to the one row previous from the current position.
public boolean first()	It is used to move the cursor to the first row in result set object.

public boolean last()	It is used to move the cursor to the last row in result set object.
public boolean absolute(int row)	It is used to move the cursor to the specified row number in the ResultSet object.
public boolean relative(int row)	It is used to move the cursor to the relative row number in the ResultSet object, it may be positive or negative.
public int getInt(int columnIndex)	It is used to return the data of specified column index of the current row as int.
public int getInt(String columnName)	It is used to return the data of specified column name of the current row as int.
public String getString(int columnIndex)	It is used to return the data of specified column index of the current row as String.
public String getString(String columnName)	It is used to return the data of specified column name of the current row as String.

5. **Closing the resources:**

- **Once all the data is retrieved and all the operations are completed. We must close all the resources**

```
st.close();
//Statement
closed rs.close();
//ResultSet close
cn.close();
//Connection
closed
```

Examples:

1. **Write a Menu Driven Java program for the following:**
   1. Create
   2. Insert
   3. Update
   4. Delete
   5. Display
   6. Exit

Solution:

```
import java.sql.*;
import java.io.*;
class DBDemo
        {
          public static void main(String args[]) throws Exception
          {
```

```
String sql,sn;
int r,p,k,ch;
BufferedReader br=new BufferedReader (new InputStreamReader(System.in));
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection
cn=DriverManager.getConnection("jdbc:odbc:DSN","",
""); Statement st=cn.createStatement();
do
{
   System.out.println("1. Create");
   System.out.println("2. Insert");
   System.out.println("3. Update");
   System.out.println("4. Delete");
   System.out.println("5. Display");
   System.out.println("6. Exit");
   System.out.println("Enter Your Choice");
   ch=Integer.parseInt(br.readLine());
   switch(ch)
   {
     case 1:
          sql="create table stud (rno number, sname text, per
          number)"; st.execute(sql);
          System.out.println("Table Is
     Created...!"); break;
     case 2:
           System.out.println("RNo
           Sname Per");
           r=Integer.parseInt(br.read
           Line()); sn=br.readLine();
           p=Integer.parseInt(br.readLine());
           sql="insert into stud values("+ r + ",'"+ sn +"'," + p +")";
           k=st.executeUpdate(sql);
           if(k>0)
              System.out.println("Record Is Added...!");
              break;
  case 3:

           System.out.println("R
           No And Per");
           r=Integer.parseInt(br.r
           eadLine());
           p=Integer.parseInt(br.
           readLine());
           sql="update stud set per=" + p + " where rno=" + r;
           k=st.executeUpdate(sql);
           if(k>0)
              System.out.println("Record Is Updated...!");
break;
```

```
case 4:
            System.out.println("RNo");
            r=Integer.parseInt(br.readLine());
            sql="delete from stud where rno=" + r;
            k=st.executeUpdate(sql);
            if(k>0)
               System.out.println("Record Is Deleted...!");
            break;
 case 5:
            System.out.println("Records from Table are...");
            ResultSet rs=st.executeQuery("select * from stud");
            while(rs.next())
 {
             System.out.println(rs.getString(1) + " " + rs.getString(2) + " " +
            rs.getString(3));
   }
   break;
   case 6:
          cn.close();
          System.exit(0);
   }
  }
  while(ch!=6);
 }
}
```

2. Write a java program for the implementation of DDL Commands.(Use Swing) Solution:

```
import java.awt.*;
import java.awt.event.*;
import java.sql.*;
import javax.swing.*;
class DDLDemo extends Frame implements ActionListener
{
 Button b1,b2,b3;
 TextField t1;
 String sql;
 Label l1;
 Connection cn;
 Statement st;
 public DDLDemo()
 {
   setLayout(null);
   l1=new Label("Type Your Query Here");
   t1=new TextField();
   b1=new Button("Create");
   b2=new Button("Alter");
   b3=new Button("Drop");
   l1.setBounds(100,100,100,30);
```

```
t1.setBounds(220,100,300,30);
b1.setBounds(100,140,100,30);
b2.setBounds(220,140,100,30);
b3.setBounds(340,140,100,30);

add(l1);
add(t1);
add(b1);
add(b2);
add(b3);

 setSize(500,500);
 setVisible(true);
 setTitle("DDL Commands");
 b1.addActionListener(this);
b2.addActionListener(this);
b3.addActionListener(this);
try
{
  Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
  cn=DriverManager.getConnection("jdbc:odbc:DSN","","");
  st=cn.createStatement();
}
catch(Exception obj)
{
}
}
public void actionPerformed(ActionEvent ae)
{
  try
  {
    Button
    b=(Button)ae.getS
    ource(); if(b==b1)
    {
      sql=t1.
      getText
      ();
      st.exec
      ute(sql)
      ;
      JOptionPane.showMessageDialog(null," Table Is Created");
    }
   if(b==b2)
   {
     sql=t1.g
     etText()
     ;
```

```
            st.execu
            te(sql);
            JOptionPane.showMessageDialog(null, "Table Is Altered");
          }
          if(b==b3)
          {
            sql=t1.g
            etText()
            ;
            st.execu
            te(sql);
           JOptionPane.showMessageDialog(null, "Table Is Deleted");
          }
        }
        catch(Exception obj)
        {
        }
    }
    public static void main(String args[]) throws Exception
    {
      new DDLDemo();
    }
  }
```

3. Write a java program for the implementation of Scrollable ResultSet. Solution:

```
import java.sql.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class ScrImp extends JFrame implements ActionListener
{
  JButton b1,b2,b3,b4;
  JTextField t1,t2,t3;
  JLabel l1,l2,l3;
  String sql,en;
  int e,s,i;
 Connection cn;
 ResultSet rs;
 Statement st;
 public ScrImp()
   {
     setLayout(null); l1=new
     JLabel("ENo");
     l2=new JLabel("EName");
     l3=new JLabel("Salary");
     t1=new JTextField();
     t2=new JTextField();
```

```
t3=new JTextField();
b1=new JButton("First");
b2=new JButton("Next");
b3=new JButton("Prev");
b4=new JButton("Last");

l1.setBounds(100,100,100,30);
t1.setBounds(220,100,100,30);
l2.setBounds(100,140,100,30);
t2.setBounds(220,140,100,30);
l3.setBounds(100,180,100,30);
t3.setBounds(220,180,100,30);
b1.setBounds(100,220,100,30);
b2.setBounds(220,220,100,30);
b3.setBounds(100,260,100,30);
b4.setBounds(220,260,100,30);

add(l1);
add(t1);
add(l2);
add(t2);
add(l3);
add(t3);
add(b1);
add(b2);
add(b3);
add(b4);
setSize(500,500);
setVisible(true);
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

b1.addActionListener(this);
b2.addActionListener(this);
b3.addActionListener(this);
b4.addActionListener(this);
try
{
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
cn=DriverManager.getConnection("jdbc:odbc:TYBBACA","","");
 st=cn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                       ResultSet.CONCUR_UPDATABLE);
  rs=st.executeQuery("select * from emp"); rs.next();
  t1.setText(rs.getString(1));
  t2.setText(rs.getString(2));
  t3.setText(rs.getString(3));
}
catch(Exception obj)
{
```

```
    }
  }
   public void actionPerformed (ActionEvent ae)
   {
     try
     {
       JButton b=(JButton)ae.getSource();
       if(b==b1)
       {
    rs.first();
         t1.setText(rs.getString(1));
         t2.setText(rs.getString(2));
         t3.setText(rs.getString(3));
       }
       if(b==b2)
       {
         rs.next();
         t1.setText(rs.getString(1));
         t2.setText(rs.getString(2));
         t3.setText(rs.getString(3));
       }
       if(b==b3)
       {
         rs.previous();
         t1.setText(rs.getString(1));
         t2.setText(rs.getString(2));
         t3.setText(rs.getString(3));
       }
       if(b==b4)
       {
         rs.last();
         t1.setText(rs.getString(1));
         t2.setText(rs.getString(2));
         t3.setText(rs.getString(3));
       }
     }
     catch(Exception obj)
     {
     }
   }
   public static void main(String args[])
   {
     new ScrImp();
   }
  }
```

4. **Write a java program for the implementation of JTable. Solution:**

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

```
class JTab extends JFrame
{
String head[]={"ENo", "EName","Salary"};
String data[][];
int i;
public JTab()
{
i=0;
setSize(500,500); setVisible(true); try
{
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection cn = DriverManager.getConnection ("jdbc:odbc:TYBBACA", "","");
Statement st=cn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_UPDATABLE);
ResultSet rs=st.executeQuery("select * from emp"); rs.last();
int r=rs.getRow(); data=new
String[r][3]; rs.first();
while(rs.next())
{
data[i][0]=rs.getString(1);
data[i][1]=rs.getString(2);
data[i][2]=rs.getString(3); i++;
}
//rs.refreshRow();
JTable jt=new JTable(data,head);
JScrollPane jp=new JScrollPane(jt);
add(jp);
}
catch(Exception obj)
System.out.println(obj);
}
}
public static void main(String args[])
{
new JTab();
}
}
```

Section – II
Python Programming

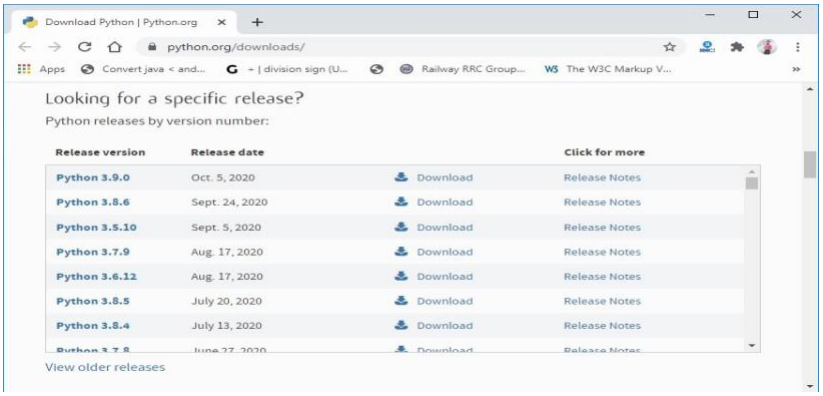
Assignment 1: Introduction to Python

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.



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.

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"
```

1. 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!
```

2. 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': printx
```

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"))

1. Variables and Data Types

Practice Programs

1.1 Display a Message Using a Variable
>>>message = "Hello Python"
>>>print(message)

1.2 Store and Display Student Information

```
name = "Rahul"
age = 20
print("Name:", name)
print("Age:", age)
```

1.3 Find Sum of Two Numbers

```
a = 10
b = 20
sum = a + b
print("Sum =", sum)
```

1.4 Accept and Display User Name

```
name = input("Enter your name: ")
print("Welcome", name)
```

1.5 Display Different Data Types

```
a = 10
b = 3.14
c = "Python"
d = True
print(type(a))
print(type(b))
print(type(c))
print(type(d))
```

1.6 Type Conversion

```
num = 25
fnum = float(num)
        print(fnum)
```

1.7 Convert Float to Integer

```
        num = 15.75
        inum = int(num)
        print(inum)
```

1.8 Swap Two Variables

```
a = 5
b = 10
a, b = b, a
print("a =", a)
print("b =", b)
```

1.9 Store Boolean Values

```
is_student = True
```

```
print(is_student)
print(type(is_student))
```

1.10 Use Multiple Assignment

```
x = y = z = 100
print(x)
print(y)
print(z)
```

1.11 Assign Multiple Values in One Line

```
a, b, c = 10, 20, 30
print(a, b, c)
```

2. Operators and Expressions

Practice Programs

2.1. Arithmetic Operators

```
a = 15
b = 4
print("Addition =", a + b)
print("Subtraction =", a - b)
print("Multiplication =", a * b)
print("Division =", a / b)
print("Floor Division =", a // b)
print("Modulus =", a % b)
print("Power =", a ** b)
```

2,2 Relational Operators

```
 a = int(input("Enter First Number: "))
b = int(input("Enter Second Number: "))

print("a > b :", a > b)
print("a < b :", a < b)
print("a == b :", a == b)
print("a != b :", a != b)
print("a >= b :", a >= b)
print("a <= b :", a <= b)
```

2.3 Assignment Operators

```
x = 20
```

```
x += 5
print("After += :", x)
x -= 3
print("After -= :", x)
x *= 2
print("After *= :", x)
x /= 4
print("After /= :", x)
```

2.4 Bitwise Operators

```
a = 12
b = 10
print("AND =", a & b)
print("OR =", a | b)
print("XOR =", a ^ b)
print("NOT a =", ~a)
print("Left Shift =", a << 2)
print("Right Shift =", a >> 2)
```

2.5 Python program for Scholarship Eligibility

```
percentage = float(input("Enter Percentage: "))
income = float(input("Enter Family Income: "))
eligible = percentage >= 75 and income <= 300000
print("Scholarship Eligible:", eligible)
```

2.6 Python program for percentage = float(input("Enter Percentage: "))

```
cgpa = float(input("Enter CGPA: "))
backlogs = int(input("Enter Number of Backlogs: "))
eligible = cgpa >= 7.5 and backlogs == 0
print("Placement Eligible:", eligible)
```

2.7 Python Program for Permission Management System

```
READ = 4
WRITE = 2
EXECUTE = 1
permission = READ | WRITE
print("Permission Value =", permission)
print("Can Read:", bool(permission & READ))
print("Can Write:", bool(permission & WRITE))
print("Can Execute:", bool(permission & EXECUTE))
```

2.8 Python Program for Encryption (using XOR operator)

```
key = 25
```

```
data = int(input("Enter Number: "))
encrypted = data ^ key
decrypted = encrypted ^ key
print("Encrypted =", encrypted)
print("Decrypted =", decrypted)
```

2.9 Python program for Scientific Calculator

```
import math
num = float(input("Enter Number: "))
print("Square Root =", math.sqrt(num))
print("Square =", num ** 2)
print("Cube =", num ** 3)
print("Percentage of Number =", num / 100)
```

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. A university follows a 10-point grading system. Write a Python program that accepts marks obtained by a student in five subjects (out of 100), converts the marks into grade points according to the grading table below, and calculates the Grade Point Average (GPA).

   Grade Point Rules

Marks (%)	Grade	Grade Point
90 – 100	O (Outstanding)	10
80 – 89	A+	9
70 – 79	A	8
60 – 69	B+	7
50 – 59	B	6
45 – 49	C	5
40 – 44	P (Pass)	4
Below 40	F (Fail)	0

   GPA Formula

$$GPA = \frac{GP_1 + GP_2 + GP_3 + GP_4 + GP_5}{5}$$

   Where:

- GP_1 = Grade Point of Subject 1
- GP_2 = Grade Point of Subject 2
- GP_3 = Grade Point of Subject 3
- GP_4 = Grade Point of Subject 4
- GP_5 = Grade Point of Subject 5

2. An online shopping portal offers a 10% discount on products and charges 18% GST on the discounted amount. Write a Python program to accept the product price and calculate the discount amount, GST amount, and final bill payable by the customer.

   Hint:
   Given:

- Product Price = P
- Discount Rate = 10%
- GST Rate = 18%

   Step 1: Calculate Discount Amount

$$Discount = P \times \frac{10}{100}$$

   Step 2: Calculate Discounted Price

$$Discounted\ Price = P - Discount$$

   Step 3: Calculate GST Amount

$$GST = Discounted\ Price \times \frac{18}{100}$$

Step 4: Calculate Final Bill Amount

$$Final\ Bill = Discounted\ Price + GST$$

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

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:

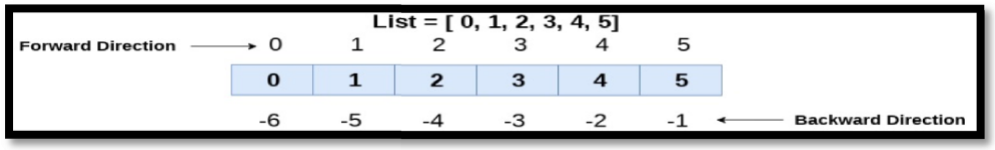
The elements of the list can be accessed using the slice operator [].
The index starts from 0 and goes to length - 1. The first element of the list is stored at the 0th index, the second element of the list is stored at the 1st index, and so on.
Consider the following example.

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)
```

```
>>>list[2]=45                          #: [1,2,3,4]
>>>print(list)
>>>list[1:3]=[55,56]                   #:[1,2,45,4]
>>>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];                     #[10,20,30,10,20,30]
>>>L2=[40,50]                         #[10,20,30,40,50]
>>>print(L1*2)
>>>print(L1+L2)
```

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
```

1. list(seq): It converts any sequence to the list.

Ex. >>>str="Python"
>>>L1=list(str)
>>>print(L1) #['P','y','t','h','o','n']

2. 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

3. 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']

4. 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

5. 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]

6. list.pop(): It removes and returns the last object of the list. Ex >>>L1=[10,20,30]
>>print(L1.pop()) #30

7. list.reverse(): It reverses the list.
Ex. >>>print(L1.reverse()) # [30 20 10]

8. 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]
```

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, Dictionaries and Sets

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)
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}
```

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.
   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: Modules and Packages

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()
```

Working with Built-in Modules

Python built-in modules are a set of libraries that come pre-installed with the Python installation. It will become a tedious task to install any required modules in between the process of development that's why Python comes with some most used modules required. These modules provide a wide range of

functionalities, from file operations and system-related tasks to mathematical computations and web services.

We can run the below command to get the all available modules in Python:

help('modules')

1) Random Module:

The "random" module in Python is used to generates random numbers and provides the functionality of various random operations such as **'random.randint()', 'random.choice()', 'random.random()', 'random.shuffle()'** and many more.

Example: random number from "1 to 10"

```
import random
num = random.randint(1, 10)
print(f"Random integer between 1 and 10: {num}")
fruits = ["Java", "C", "C++", "Python"]
chosen_fruit = random.choice(fruits)
print(f"Randomly chosen language: {chosen_fruit}")
```

2) Math Module:

The math module offers mathematical functions used for advanced arithmetic operations. This includes trigonometric functions, logarithmic functions, and constants like **pi** and **e**. This module is used to perform complex calculations using Python program.

Example:find the square root of a number using **math.sqrt()** method and the value of PI

```
import math

sqrt_val = math.sqrt(64)

pi_const = math.pi

print(sqrt_val)

print(pi_const)
```

3) The datetime Module:

The datetime module enables us to create the custom date objects, perform various operations on dates likethe comparison, etc. To work with dates as date objects, we have to import the datetime module into the python source code.

Example:

```
import datetime
date_today = datetime.date.today()
time_now = datetime.datetime.now().time()
print(date_today)
print(time_now)
```

4) 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(2026,3)
#printing the calendar of December 2026
print(cal)
```

5)Sys module:

The "sys" module in Python provides functions and variables that interact with the Python runtime environment. It allows developers to access Python interpreter attributes and manipulate them. It offers a range of other functionalities, such as input/output stream access, memory info access, and more.

Example: print the current version of Python programming languageand also print the list of command line arguments passed while running the Python program.

```
import sys
print("Python version:", sys.version)
print("Command line arguments:", sys.argv)
sys.exit(1)
```

The re module in Python provides support for working with regular expressions. Regular expressions (often abbreviated as "regex") are powerful tools for matching strings or sets of strings using a specialized syntax that allows for flexible pattern matching.

Example:

```
import re
pattern = r"[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}"

text = "Contact us at info@example.com and support@example.org for more details."
match = re.search(pattern, text)
if match:
print("First found email:", match.group())

emails = re.findall(pattern, text)
print("All found emails:", emails)
```

User Defined Modules and Functions

Python provides several kinds of modules. Each type plays a different role in application development.

1. Built-in Modules: These come bundled with Python and require no installation - **e.g**., math, random, os.

```
import random
print(random.randint(1, 5))
```

2. User-Defined Modules: These are modules you create yourself, such as **calc.py**.

```
import calc
print(calc.subtract(20, 5))
```

3. External (Third-Party) Modules: These modules are installed using pip - **e.g.**, NumPy, Pandas, Requests.

```
import requests
r = requests.get("https://example.com")
print(r.status_code)
```

4. Package Modules: A package is a directory containing multiple modules, usually with an **__init__.py** file.
Example Directory

mypkg/
__init__.py
calc.py
utils.py

```
from mypkg import utils
print(utils.some_func())
```

Functions in Python:

Function is named, independent block of statements that perform a specific task and may return a value tothe 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 sameindentation.
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 anerror. 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 parameters and return values:

Function Arguments/Parameter:
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
```

Return Values:

Functions can return values using the return statement. This allows a function to send back a result to the caller, enabling further processing or use.

```
def add(a, b):
                                        return a+b
result =add(4, 5)
print (result)                          #9
```

Scope of Variable:

The location where we can find a variable and also access it if required is called the **scope of a variable**

Python local variable:

Local variables are those that are initialized within a function and are unique to that function. It cannot be accessed outside of the function. Let's look at how to make a local variable.

```
def f():

    # local variable
    s = "I love Python Programming"
    print(s)

# Driver code
f()
```

Python Global Variable:

Global variables are the ones that are defined and declared outside any function and are not specified to any function. They can be used by any part of the program.

```
# This function uses global variables
def f():
    print(s)

# Global scope
s = "I love Python Programming"
f()
```

Recursion function:

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 types of constructsare 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
```

Structure of Python Modules

Creating a module:

To create a module, write the desired code and save that in a file with .py extension.

```
# calc.py

def add(x, y):

   return (x+y)

def subtract(x, y):

   return (x-y)
```

Saving module files

save your code in a text file with a **.py** extension

Importing user-defined modules

save your custom functions or classes in a file with a .py extension, and then bring it into your main script using the import keyword followed by the file name without the .py extension.

Using module variables and functions

To use module variables and functions in Python, you define them in a **separate file with a .py extension** and bring them into your main script using the **import keyword**.

Example:

```
# calculator.py

# Module variables (constants)

PI = 3.14159

VERSION = "1.0.0"

# Module functions

def add(a, b):

   """Returns the sum of two numbers."""

   return a + b
```

```
def area_of_circle(radius):
    """Calculates area using the module's PI variable."""
    return PI * (radius ** 2)
```

__name__ == "__main__" concept

In Python, __name__ == "__main__" acts as a conditional guard. It ensures that specific blocks of code are only executed when the Python file is run directly by the user, and not when it is imported into another file as a module.

Python assigns a special built-in variable called __name__ to every Python file (or module):

When you run a file directly (e.g., python my_script.py), Python sets __name__ to the string "__main__".

When you import the file (e.g., import my_script), Python sets __name__ to the actual filename (e.g., "my_script").

Example:

```
# calculator.py
def add(a, b):
    return a + b
def main():
    # Code we only want to run when we execute THIS file directly
print("Running calculator script...")
    result = add(5, 7)
print(f"The result is {result}")
if __name__ == "__main__":
main()
```

Module Documentation:

It can refer to viewing the official standard library docs, reading documentation interactively inside your terminal, or writing documentation for your own code

Docstrings (documentation strings) are literal strings written directly under functions, classes, methods, or modules to explain their behavior. Unlike regular comments, docstrings remain attached to the code object and can be accessed dynamically at runtime

Syntax:

Docstrings are declared using triple double quotes (""") or triple single quotes (''').

Example:

```
def calculate_area(radius):
```

```
    """Calculate the area of a circle given its radius."""
    import math
    return math.pi * (radius ** 2)
```

Comments:

comments are created using the **hash character (#)**, and everything following it on that line is automatically ignored by the interpreter.

Types of Comments

Single-Line & Block Comments

Use a # symbol to start a comment. It can be placed on its own line or right after a piece of code (known as an inline comment).

```
# This is a full-line block comment
print("Hello World")  # This is an inline comment
```

Multi-Line Comments
the standard way to write multi-line comments is to **place a # at the beginning of every single line**
Using help() on custom modules
Python's help() function automatically extracts these multi-line string literals to generate human-readable documentation for your code.

Syntax:

```
help([object])
```

Example:

```
>>>help(collection)
```

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 submodules. 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 mustnotice 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 pythonsource 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"]

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.
2. Write a recursive function to calculate the sum of numbers from 1 to 10.

Evaluation

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

Signature of the Instructor

Assignment 5: 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:

except: else:a = int(input("Enter a:"))
b = int(input("Enter b:"))
c = a/b;
print("a/b = %d"%c) print("can't divide by zero")
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.

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 6:- Introduction of data analytics and data visualization

1. Introduction to Data Analytics

Data Analytics is the process of collecting, organizing, cleaning, transforming and analyzing data to discover meaningful information and support decision-making.

Applications of Data Analytics

- Sales Analysis
- Customer Behaviour Analysis
- Healthcare Analytics
- Educational Analytics
- Financial Forecasting
- Market Research

Python Libraries Used in Data Analytics

Library	Purpose
NumPy	Numerical Computing
Pandas	Data Manipulation
Matplotlib	Data Visualization
Seaborn	Advanced Visualization
Plotly	Interactive Visualization
Scikit-Learn	Machine Learning

2. NUMPY

What is NumPy?

NumPy (Numerical Python) is a Python library used for efficient numerical computation and multidimensional array processing.

np.array()

Purpose: Creates a NumPy array.

Syntax:

```
import numpy as np
arr = np.array([10,20,30,40])
```

Output:

```
[10 20 30 40]
```

Use:

Store numerical data efficiently.

np.zeros()

Purpose:

Creates an array filled with zeros.

Syntax:

```
arr = np.zeros(5)
```

Output:

```
[0. 0. 0. 0. 0.]
```

np.ones()

Purpose:

Creates an array filled with ones.
Syntax:
arr = np.ones(5)
Output:
[1. 1. 1. 1. 1.]

np.arange()
Purpose:
Creates a sequence of evenly spaced values.
Syntax:
arr = np.arange(1,11,2)
Output:
[1 3 5 7 9]

np.linspace()
Purpose:
Creates specified number of equally spaced values.
Syntax:
arr = np.linspace(1,10,5)
Output:
[1.00 3.25 5.50 7.75 10.00]

Array Attributes
shape → Returns rows and columns
Example:
arr.shape
Output:
(2,3)
size → Total number of elements
Output:
6
ndim → Number of dimensions
Output:
2
dtype → Data type
Output:
int64

Array Slicing
Syntax:
arr[start:end]
Example:
arr[1:4]
Output:
[20 30 40]

Mathematical Functions
a=[10,20,30]

b=[1,2,3]
np.add(a,b)
Output:
[11 22 33]
np.subtract(a,b)
Output:
[9 18 27]
np.multiply(a,b)
Output:
[10 40 90]
np.divide(a,b)
Output:
[10. 10. 10.]

Statistical Functions
marks=[60,70,80,90,100]
np.sum(marks)
Output:
400
np.mean(marks)
Output:
80.0
np.median(marks)
Output:
80.0
np.min(marks)
Output:
60
np.max(marks)
Output:
100
np.std(marks)
Output:
14.14
np.var(marks)
Output:
200
np.percentile(marks,25)
Output:
70
np.ptp(marks)
Output:
40

Random Number Functions
np.random.rand(3)
Output:
[0.23 0.45 0.87]

np.random.randint(1,10,5)
Output:
[4 7 1 8 3]
np.random.choice([10,20,30,40],2)
Output:
[20 40]

3. PANDAS
What is Pandas?
Pandas is a Python library used for storing, organizing, cleaning and analyzing structured data.

pd.Series()
Purpose:
Creates one-dimensional labelled data.
Example:
s=pd.Series([10,20,30])
Output:
0 10
1 20
2 30

pd.DataFrame()
Purpose:
Creates tabular data structure.
Example:
data={
'Name':['Amit','Riya'],
'Marks':[80,90]
}
df=pd.DataFrame(data)
Output:
Name Marks
Amit 80
Riya 90

Reading Files
pd.read_csv("students.csv")
Purpose:
Read CSV file
pd.read_excel("students.xlsx")
Purpose:
Read Excel file

Writing Files
df.to_csv()
df.to_excel()
Purpose:
Store processed data.

Data Inspection Functions
df.head()
Purpose:
Display first 5 rows
df.tail()
Purpose:
Display last 5 rows
df.info()
Purpose:
Display structure of dataset
df.describe()
Purpose:
Generate statistical summary
df.shape
Output:
(rows, columns)
df.columns
Output:
Column names
df.dtypes
Output:
Data types

Data Selection
df['Marks']
Purpose:
Select column
df.loc[]
Purpose:
Label-based selection
df.iloc[]
Purpose:
Position-based selection

Data Filtering
df[df['Marks']>60]
Purpose:
Select records satisfying condition
df.query('Marks>60')
Purpose:
Filter records using query

Sorting Data
df.sort_values('Marks')
Ascending order
df.sort_values('Marks',ascending=False)
Descending order

4. DATA CLEANING

Missing Values

df.isnull()
Purpose:
Check missing values
df.dropna()
Purpose:
Remove missing values
df.fillna(0)
Purpose:
Replace missing values

Duplicate Records

df.duplicated()
Purpose:
Identify duplicate records
df.drop_duplicates()
Purpose:
Remove duplicate records

Rename Columns

df.rename(columns={'Marks':'Score'})
Purpose:
Rename column names

Data Type Conversion

df.astype(float)
Purpose:
Convert data type

5. DATA AGGREGATION

df.sum()
Total
df.mean()
Average
df.min()
Minimum
df.max()
Maximum
df.count()
Count
df.value_counts()
Frequency distribution

Group By

df.groupby('Department')
Purpose:
Department-wise analysis
Example:
df.groupby('Department')['Salary'].mean()

Output:
IT 55000
HR 48000
Sales 51000

Aggregation

```
df.groupby('Department').agg(
{'Salary':['mean','max','count']}
)
```

Purpose:
Multiple summaries together

6. PIVOT TABLES

pd.pivot_table()
Purpose:
Generate summary reports
Applications:

- Sales Reports
- Student Performance Reports
- Business Dashboards

7. MATPLOTLIB

Import

```
import matplotlib.pyplot as plt
```

Line Chart

plt.plot(x,y)
Purpose:
Trend Analysis

Bar Chart

plt.bar(x,y)
Purpose:
Comparison Analysis

Horizontal Bar Chart

plt.barh(x,y)
Purpose:
Category Comparison

Pie Chart

plt.pie(values,labels=labels)
Purpose:
Percentage Contribution

Histogram

plt.hist(data)
Purpose:
Distribution Analysis

Scatter Plot
plt.scatter(x,y)
Purpose:
Relationship Analysis

Chart Customization
plt.title()
Add Title
plt.xlabel()
Add X-axis label
plt.ylabel()
Add Y-axis label
plt.legend()
Display Legend
plt.grid()
Display Grid
plt.savefig()
Save chart as image

SET A

Assignment 1: Student Marks Analysis System

A teacher has marks of 20 students stored in a NumPy array. Generate a performance report showing average marks, highest marks, lowest marks, standard deviation, students scoring above average and top 5 scorers.

Tasks:

1. Create a NumPy array of marks of 20 students.
2. Display:
    - Total number of students
    - Highest marks
    - Lowest marks
    - Average marks
3. Count the number of students scoring above the average.
4. Display the marks of the top 5 students.
5. Calculate the standard deviation of marks.

Expected Concepts:

np.array(), indexing, slicing, np.mean(), np.max(), np.min(), np.std(), sorting.

Assignment 2: Sales Performance Report

A shop records monthly sales for one year. Analyze annual sales, average monthly sales, highest and lowest sales month and top three performing months.

Tasks:

1. Create an array containing monthly sales data.
2. Find:
    - Total annual sales
    - Average monthly sales
    - Month with highest sales
    - Month with lowest sales
3. Calculate percentage increase from first month to last month.
4. Display the top 3 performing months.

Expected Concepts:

Arrays, aggregation functions, indexing, sorting, arithmetic operations.

Assignment 3: Employee Information Management

Create an employee DataFrame and generate reports showing average salary, employees earning above average salary, highest paid employee and department-wise employee count.

Tasks:

Create a DataFrame with:

- Employee ID
- Name
- Department
- Salary

Generate a report showing:

1. Total number of employees.
2. Average salary.
3. Employees earning above average salary.
4. Highest paid employee.
5. Department-wise employee count.

Expected Concepts:

DataFrame creation, filtering, aggregation, groupby().

Assignment 4: Student Attendance Analysis

Analyze attendance records and identify students below 75% attendance, average attendance, highest attendance and eligible students.

Tasks:

Create a DataFrame containing:

- Roll No
- Name
- Attendance Percentage

Generate a report showing:

1. Students having attendance below 75%.
2. Average attendance of class.
3. Student with highest attendance.
4. Sort attendance in descending order.
5. Count students eligible for examination (attendance ≥ 75%).

Expected Concepts:

Filtering, sorting, aggregation functions.

Assignment 5: Monthly Expense Dashboard

Create expense data and generate total expenditure, highest expenditure category, bar chart and pie chart.

Tasks:

Create data for:

- Food
- Travel
- Education
- Entertainment
- Utilities

1. Calculate total expenses.
2. Find category with highest expenditure.
3. Create:

- Bar Chart
- Pie Chart

4. Add suitable chart titles and labels.
5. Save the chart as an image file.

Expected Concepts:
DataFrame, aggregation, Matplotlib charts.

SET B

Assignment 1: College Result Analysis System
Calculate total marks, percentage, grades, topper details, class average and grade-wise distribution.

Dataset Fields:

- Roll No
- Name
- Subject1
- Subject2
- Subject3

Tasks:

1. Calculate total marks and percentage.
2. Assign grades:
   - A (≥75)
   - B (60–74)
   - C (50–59)
   - D (40–49)
   - Fail (<40)
3. Display topper details.
4. Calculate class average percentage.
5. Generate grade-wise student count.

Expected Concepts:
NumPy/Pandas, conditional filtering, aggregation.

Assignment 2: Customer Feedback Survey Analysis
Generate customer ratings using random numbers and analyze average rating, median, standard deviation, rating frequency and satisfaction percentage.

Tasks:

1. Generate ratings of 100 customers.
2. Calculate:
   - Mean rating
   - Median rating
   - Standard deviation
3. Count customers for each rating.
4. Display percentage of satisfied customers (rating ≥4).
5. Create a bar chart showing rating distribution.

Expected Concepts:
Random generation, statistical analysis, visualization.

Assignment 3: Employee Salary Analytics
Perform department-wise salary analysis using groupby and aggregation functions. Generate department-wise average salary chart.

Tasks:

1. Create employee dataset containing:
   - Employee Name
   - Department
   - Salary
2. Calculate:
   - Average salary per department
   - Highest salary per department
   - Employee count per department
3. Display employees earning above department average.
4. Create a bar chart of department-wise average salary.

Expected Concepts:
groupby(), agg(), filtering, plotting.

Assignment 4: Sales Data Cleaning and Analysis
Create a dataset with missing values and duplicate records. Clean the data and generate product-wise sales reports and charts.

Tasks:

1. Create a dataset with missing values and duplicates.
2. Identify missing values.
3. Fill or remove missing values.
4. Remove duplicate records.
5. Calculate:
   - Total sales
   - Product-wise sales
   - Average sales
6. Create a bar chart showing product-wise sales.

Expected Concepts:
Data cleaning, aggregation, visualization.

Assignment 5: Business Intelligence Dashboard
Read sales data from a CSV file, generate pivot tables, identify best-performing categories and create line, bar and pie charts for reporting.

Dataset Fields:

- Month
- Product Category
- Sales Amount

Tasks:

1. Import data from CSV.
2. Create a pivot table showing monthly category-wise sales.
3. Identify:
   - Best performing month
   - Best selling category
4. Create:
   - Line chart for monthly sales trend
   - Bar chart for category-wise sales
   - Pie chart for category contribution

5. Save all charts and prepare a summary report.

Expected Concepts:

CSV handling, pivot tables, aggregation, multiple visualizations.

Assignment 1: Retail Store Sales Analytics Dashboard

Problem Statement

A retail chain wants to analyze sales performance across different product categories and regions to improve decision-making.

Dataset Fields

- Invoice No
- Date
- Region
- Product Category
- Quantity Sold
- Unit Price
- Sales Amount

Tasks

1. Import data from a CSV file.
2. Create a new column **Total Revenue = Quantity Sold × Unit Price**.
3. Clean the dataset by removing duplicate records and handling missing values.
4. Generate a pivot table showing:
   - Region-wise sales
   - Category-wise sales
5. Identify:
   - Best performing region
   - Best selling product category
   - Highest revenue transaction
6. Calculate:
   - Total Revenue
   - Average Revenue per Transaction
7. Create:
   - Line chart showing monthly revenue trend
   - Bar chart showing category-wise sales
   - Pie chart showing regional contribution
8. Save all charts and prepare a business summary report.

Expected Concepts

CSV Handling, Data Cleaning, Calculated Columns, GroupBy, Aggregation, Pivot Tables, Multiple Visualizations.

SET C

1. **Retail Store Sales Analytics Dashboard:** A retail chain wants to analyze sales performance across different product categories and regions to improve decision-making.

Dataset Fields

- Invoice No
- Date
- Region
- Product Category
- Quantity Sold
- Unit Price

- Sales Amount

Tasks

1. Import data from a CSV file.
2. Create a new column **Total Revenue = Quantity Sold × Unit Price**.
3. Clean the dataset by removing duplicate records and handling missing values.
4. Generate a pivot table showing:
   - Region-wise sales
   - Category-wise sales
5. Identify:
   - Best performing region
   - Best selling product category
   - Highest revenue transaction
6. Calculate:
   - Total Revenue
   - Average Revenue per Transaction
7. Create:
   - Line chart showing monthly revenue trend
   - Bar chart showing category-wise sales
   - Pie chart showing regional contribution
8. Save all charts and prepare a business summary report.

Expected Concepts

CSV Handling, Data Cleaning, Calculated Columns, GroupBy, Aggregation, Pivot Tables, Multiple Visualizations.

2. **Student Academic Performance Analytics System:** A college wants to analyze student performance and identify academic trends.

Dataset Fields

- Roll Number
- Student Name
- Gender
- Department
- Subject1
- Subject2
- Subject3
- Subject4

Tasks

1. Import student data from a CSV file.
2. Calculate:
   - Total Marks
   - Percentage
   - Grade
3. Generate department-wise performance statistics.
4. Create a pivot table showing average marks by department and gender.
5. Identify:
   - Class topper
   - Department topper
   - Students scoring below 40%
6. Calculate:

- Mean
- Median
- Standard Deviation of percentages

7. Create:
   - Histogram of percentage distribution
   - Bar chart of department-wise average performance
   - Pie chart showing grade distribution
8. Generate a comprehensive academic performance report.

Expected Concepts

CSV Handling, NumPy Statistics, Conditional Filtering, Pivot Tables, Histograms, Aggregation.

3 Customer Purchase Behaviour Analysis: An e-commerce company wants to study customer purchasing patterns and product preferences.

Dataset Fields

- Customer ID
- Age
- Gender
- City
- Product Category
- Purchase Amount
- Purchase Date

Tasks

1. Import the dataset from CSV.
2. Perform data cleaning by handling missing values.
3. Categorize customers into age groups:
   - 18–25
   - 26–35
   - 36–50
   - Above 50
4. Generate:
   - Category-wise purchase analysis
   - City-wise purchase analysis
5. Create pivot tables showing:
   - Age Group vs Product Category
   - City vs Purchase Amount
6. Identify:
   - Highest spending customer
   - Most popular product category
   - City generating highest revenue
7. Create:
   - Scatter plot (Age vs Purchase Amount)
   - Bar chart (Category-wise Revenue)
   - Pie chart (Customer Distribution by Age Group)
8. Prepare a customer insights report.

Expected Concepts

Data Transformation, Pivot Tables, GroupBy Analysis, Scatter Plot, Business Analytics.

4: Employee Workforce Analytics Dashboard: A company wants to analyze workforce distribution, salaries and employee performance.

Dataset Fields

- Employee ID
- Department
- Gender
- Experience
- Salary
- Performance Rating

Tasks

1. Import employee data from CSV.
2. Clean missing and duplicate records.
3. Calculate:
   - Average Salary
   - Department-wise Salary Statistics
   - Performance-wise Salary Statistics
4. Create pivot tables for:
   - Department vs Average Salary
   - Department vs Average Performance Rating
5. Identify:
   - Highest paid employee
   - Best performing department
   - Employees with performance below average
6. Create:
   - Bar chart of department-wise average salary
   - Histogram of salary distribution
   - Scatter plot of experience vs salary
7. Save all visualizations and prepare an HR analytics report.

Expected Concepts

Data Cleaning, Aggregation, Statistical Analysis, Pivot Tables, Scatter Plots, Histograms.

5 Hospital Patient Analytics and Reporting System: A hospital wants to analyze patient records to improve healthcare services and resource allocation.

Dataset Fields

- Patient ID
- Age
- Gender
- Disease Category
- Admission Date
- Treatment Cost
- Length of Stay

Tasks

1. Import patient records from CSV.
2. Clean the dataset by handling missing values and duplicates.
3. Categorize patients into age groups.
4. Calculate:
   - Average Treatment Cost
   - Average Length of Stay
   - Disease-wise Treatment Cost
5. Generate pivot tables showing:
   - Disease Category vs Treatment Cost

- Gender vs Length of Stay

6. Identify:
   - Most common disease category
   - Highest treatment cost case
   - Age group with maximum admissions
7. Create:
   - Line chart showing monthly admissions
   - Bar chart showing disease-wise patient count
   - Pie chart showing gender distribution
   - Histogram showing treatment cost distribution
8. Prepare a hospital analytics dashboard and summary report.

Expected Concepts CSV Handling, Data Cleaning, Statistical Analysis, GroupBy, Pivot Tables, Histograms, Business Reporting

Evaluation

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

Signature of Instructor

Assignment 7:-Introduction to Flask (Web Development using Python)

Getting Started: Installing Software and Running Flask in VS Code

Step 1: Install Python

Flask is a Python framework, so Python must be installed on the computer before anything else.

1. Open a web browser and go to https://www.python.org/downloads/.
2. Click 'Download Python' (the latest version, e.g., Python 3.12).
3. Run the downloaded installer. On the first screen, tick the checkbox 'Add Python to PATH' - this is very important.
4. Click 'Install Now' and wait for the installation to complete, and then close the installer.
5. To confirm Python has been installed correctly, open the Command Prompt and type the commands shown below.

Note:

- Pip is Python's package manager. It is installed automatically along with Python and is used to install Flask and other libraries.
- If the 'python' command is not recognized, Python was not added to PATH during installation - the installer should be run again with the 'Add Python to PATH' option checked.

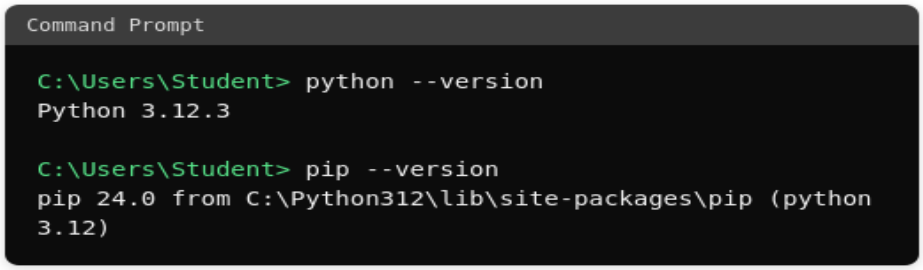


Fig 0.1: Verifying Python and pip installation from the Command Prompt

Practice Questions

Set A:

1. What is the purpose of installing Python before working with Flask?
2. Why is it important to tick the 'Add Python to PATH' option while installing Python?

Set B:

1. Which command is used to check the installed version of Python?
2. What is 'pip', and what is it used for?

Set C:

1. Install Python on your computer and write down the exact version number displayed.
2. Open the Command Prompt and run 'pip --version'. Note down the output you get.

Step 2: Install Visual Studio Code (VS Code)

VS Code is a free code editor that makes it easy to write, run and debug Python/Flask programs.

1. Go to https://code.visualstudio.com/ and click the Download button for Windows.
2. Run the downloaded installer, accept the agreement, and keep clicking 'Next' with the default options, then click 'Install'.

3. Once installed, open VS Code.

4. Click on the Extensions icon in the left sidebar (the square icon) and search for 'Python'.

5. Install the official 'Python' extension provided by Microsoft. This adds IntelliSense (code suggestions), debugging and the ability to run Python files directly from VS Code.

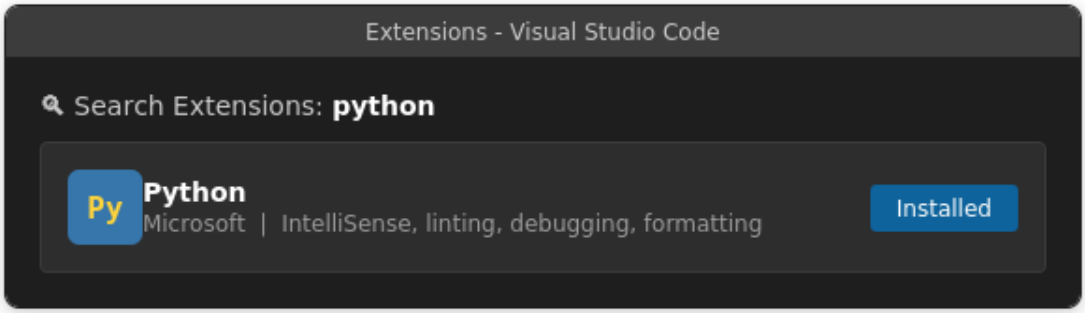


Fig 0.2: Installing the Python extension in VS Code

Practice Questions

Set A:

1. What is VS Code, and why is it useful for Flask development?
2. Which extension should be installed in VS Code for Python development?

Set B:

1. State True or False: 'VS Code is a paid software.'
2. Which icon in the VS Code sidebar is used to install extensions?

Set C:

1. Install VS Code and the Python extension on your computer, and take a screenshot of the Extensions panel showing it installed.
2. List any three features provided by the Python extension in VS Code.

Step 3: Create a Project Folder and Open it in VS Code

Before writing any code, a dedicated folder should be created to keep all the files of the Flask project organized.

1. Create a new folder anywhere on the computer, for example: C:\Users\Student\flask_app

2. Open VS Code, then go to File > Open Folder, and select the 'flask_app' folder created above.

3. Inside VS Code, right-click in the Explorer panel (left sidebar) and choose 'New File' to create a file named app.py - this will contain the Flask code.

4. If the project uses HTML templates, also create a new folder named 'templates' inside 'flask_app' (right-click > New Folder), and create .html files inside it.

5. Always remember: the file app.py must be saved directly inside the main project folder (flask_app), and all HTML template files must be saved inside the 'templates' sub-folder, otherwise Flask will not be able to locate them.

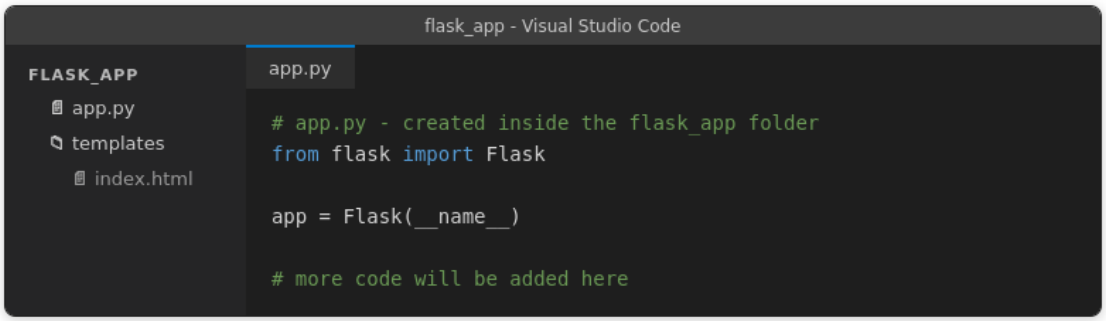


Fig 0.3: Project folder 'flask_app' opened in VS Code with app.py and a templates folder

Practice Questions

Set A:

1. Why should app.py and the templates folder be kept inside the same project folder?
2. What is the purpose of the 'templates' folder in a Flask project?

Set B:

1. Fill in the blank: All HTML template files must be saved inside the _____ folder.
2. State True or False: 'app.py can be saved anywhere on the computer, even outside the project folder.'

Set C:

1. Create a new project folder named 'my_flask_app' and open it in VS Code.
2. Inside 'my_flask_app', create a file app.py and a folder named 'templates'.

Step 4: Install Flask using the VS Code Terminal

Flask is installed using pip, the Python package manager, directly from the terminal built into VS Code.

1. In VS Code, open the terminal using Terminal > New Terminal (or the shortcut Ctrl + `).
2. The terminal opens at the bottom of the screen, already pointing to the project folder (flask_app).
3. Type the following command and press Enter:

```
pip install flask
```

4. Wait for the installation to finish. A message 'Successfully installed flask...' confirms that Flask has been installed correctly.

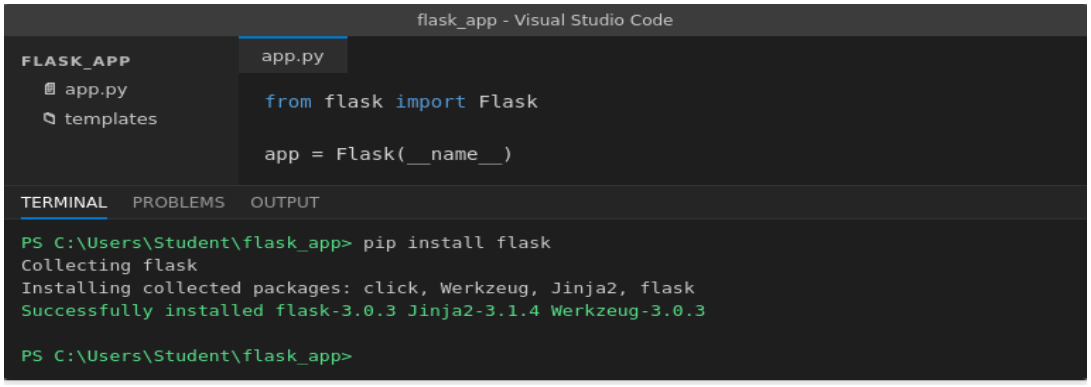


Fig 0.4: Installing Flask using the integrated terminal in VS Code

Practice Questions

Set A:

1. What role does pip play in installing Flask?
2. What message is displayed in the terminal after Flask is installed successfully?

Set B:

1. Write the command used to install Flask using pip.
2. Fill in the blank: The shortcut to open a new terminal in VS Code is Ctrl + _____.

Set C:

1. Open the terminal in VS Code and install Flask. Write down the version of Flask that gets installed.
2. Run the command 'python -m flask --version' and note down the output.

Step 5: Write and Run the First Flask Program

Once Flask is installed, a simple Flask program can be written and run directly from VS Code.

1. Open app.py in VS Code (created in Step 3) and type the program shown below.
2. Save the file using Ctrl + S. Make sure it is saved as app.py inside the flask_app folder.
3. In the VS Code terminal, run the program using the command:
4. The terminal will display a message saying the server is running, along with a URL such as http://127.0.0.1:5000.
5. Hold Ctrl and click on this link in the terminal (or copy it and paste it into any web browser) to open the application.
6. To stop the running server at any time, click inside the terminal and press Ctrl + C.

To run the application:

```
python app.py
```

Important points to remember:

- Every time a change is made to app.py or any HTML template, the file must be saved (Ctrl + S). If debug=True is set, Flask will reload the server automatically; otherwise the server must be stopped (Ctrl + C) and started again using 'python app.py'.
- The address http://127.0.0.1:5000/ refers to 'localhost' - the local computer itself - port 5000 is the default port used by Flask's development server.
- All the practical programs given in the following units should be saved and run using the same step-by-step process described above: write the code in app.py inside the project folder, save HTML files inside the 'templates' folder, run 'python app.py' from the VS Code terminal, and view the result in the browser.

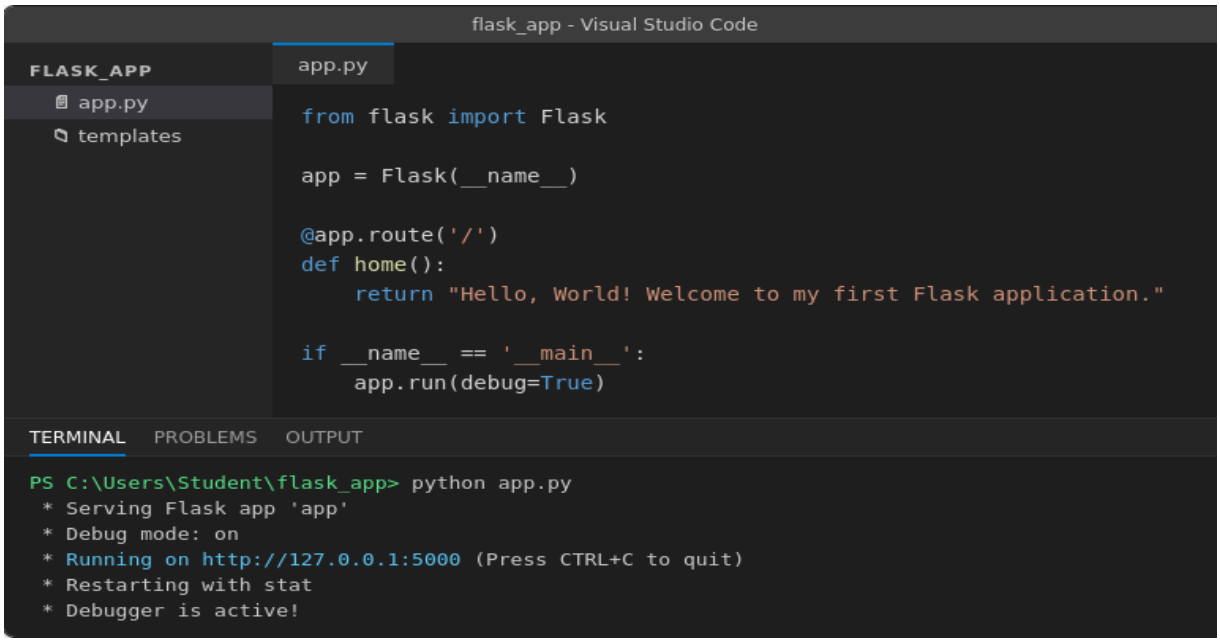


Fig 0.5(a): Running app.py from the VS Code terminal

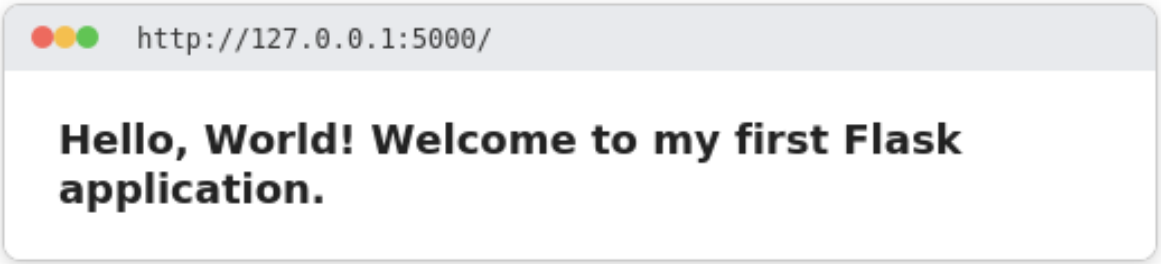


Fig 0.5(b): Output displayed in the browser at http://127.0.0.1:5000/

Practice Questions

Set A:

1. What is the purpose of writing app.run(debug=True) at the end of a Flask program?
2. What does the URL http://127.0.0.1:5000/ refer to?

Set B:

1. Which command is used to run a Flask application from the terminal?
2. State True or False: 'Flask runs on port 8080 by default.'

Set C:

1. Type the first Flask program shown in this unit, save it as app.py, and run it. Write down the message displayed in the browser.
2. Modify the program to display 'My First Flask Program' instead of 'Hello, World!' and run it again.

5.1 Introduction to Flask

1. Introduction to Web Development

Web development refers to the process of building, creating and maintaining websites and web applications that run on the internet or an intranet. It involves a combination of programming, designing and content creation.

Web development is broadly divided into two parts:

- Front-End Development (Client Side): Deals with the part of the website that the user directly interacts with. It is built using HTML, CSS and JavaScript.
- Back-End Development (Server Side): Deals with the server, application logic, database and the overall functioning of the website. It is built using server-side languages such as Python, PHP, Java, etc.

Websites can be classified as static websites, where the content remains the same for every user, and dynamic websites, where the content changes based on user input, database records or other conditions. Dynamic websites are built with the help of web frameworks such as Flask, which make development faster and more organized.

More about Front-End and Back-End:

- The front-end is responsible for everything the user sees and interacts with in the browser - layout, colours, fonts, buttons and forms.
- The back-end is responsible for business logic, processing form input, communicating with the database and sending the final result back to the browser.
- A web framework like Flask sits on the back-end. It receives the request from the browser, runs the required Python code, and returns an HTML page (which may itself be combined with front-end HTML/CSS) as the response.

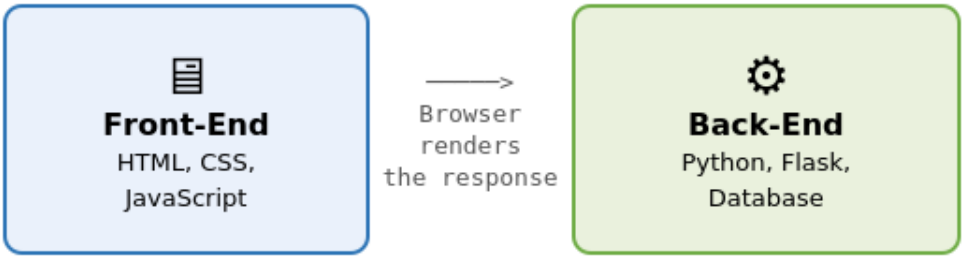


Fig 1.1: Front-end and back-end working together to deliver a web page

Practice Questions

Set A:

1. What is the difference between front-end and back-end development?
2. Give two examples each of front-end and back-end technologies.

Set B:

1. Fill in the blank: A website whose content changes based on user input is called a _____ website.
2. State True or False: 'HTML, CSS and JavaScript are back-end technologies.'

Set C:

1. List any three websites you use daily and classify each as static or dynamic, giving reasons.
2. In your own words, explain why dynamic websites need a web framework such as Flask.

2. Client-Server Architecture

Client-Server architecture is a computing model in which the workload is divided between two types of programs: clients and servers. Both communicate with each other over a network, typically using the HTTP/HTTPS protocol.

- Client: It is the device or application (usually a web browser) that sends a request to the server to access a resource such as a webpage, image or data.
- Server: It is a powerful computer or program that listens for incoming requests, processes them and sends back an appropriate response.
- Request: A message sent by the client to the server asking for some resource or action.
- Response: A message sent back by the server to the client, usually containing the requested data (HTML page, JSON, etc.).

The general flow of client-server communication is as follows:

1. The user enters a URL in the browser (client).
2. The browser sends an HTTP request to the web server.
3. The server processes the request (may interact with a database).
4. The server sends back an HTTP response (HTML page) to the browser.
5. The browser renders the response and displays it to the user.

Additional points about Client-Server Architecture:

- HTTP (HyperText Transfer Protocol) is the set of rules that governs how messages are formatted and transmitted between the client and the server, and how servers and browsers should respond to various commands.
- Every request from the client carries a URL, an HTTP method (GET, POST, etc.) and optionally some data; every response from the server carries a status code (e.g., 200 OK, 404 Not Found) along with the requested content.
- A single server can handle requests from many clients at the same time, which is why this model is also called a one-to-many architecture.

Fig 1.2: Request-response cycle between client and server

Practice Questions

Set A:

1. Define the terms 'Client' and 'Server' with one example each.
2. What is HTTP, and what role does it play in client-server communication?

Set B:

1. Arrange in correct order: (a) Server processes the request (b) Browser sends a request (c) Browser displays the response (d) Server sends a response.
2. State True or False: 'A single server cannot handle requests from multiple clients at the same time.'

Set C:

1. Draw a simple diagram showing the request-response cycle between a client and a server.
2. Explain, in your own words, what happens when you type a website address in your browser and press Enter.

3. Introduction to Flask Framework

Flask is a lightweight and easy-to-learn web framework written in Python. It is classified as a micro-framework because it does not require particular tools or libraries and does not have a default database, form validation tool, or other components that some other frameworks have.

Flask is built on top of two important components:

- Werkzeug: A WSGI (Web Server Gateway Interface) toolkit that handles requests, responses and routing.
- Jinja2: A templating engine used to generate dynamic HTML pages.

Features of Flask:

1. Lightweight and easy to set up.
2. Built-in development server and debugger.
3. Support for URL routing.
4. Integrated support for unit testing.
5. Uses Jinja2 templating engine for dynamic HTML pages.
6. Highly flexible - allows developers to choose their own tools and libraries (ORM, form validation, etc.).

Before creating a Flask application, Flask must be installed using pip (Python's package manager). The following command is used to install Flask:

```
pip install flask
```

Once installed, the installation can be verified by checking the version of Flask installed:

```
python -m flask --version
```

More information:

- Flask follows the WSGI (Web Server Gateway Interface) standard, which defines how a Python web application communicates with a web server. This makes Flask compatible with a wide range of web servers.

- Because Flask is a micro-framework, the developer has full control over which additional libraries to use - for example, SQLAlchemy for database access, Flask-WTF for forms, or Flask-Login for authentication - unlike full-stack frameworks which include all of these by default.
- Flask comes with a built-in development server, which is suitable for testing and learning purposes but should be replaced with a production-grade server (such as Gunicorn or Waitress) when the application is deployed live.

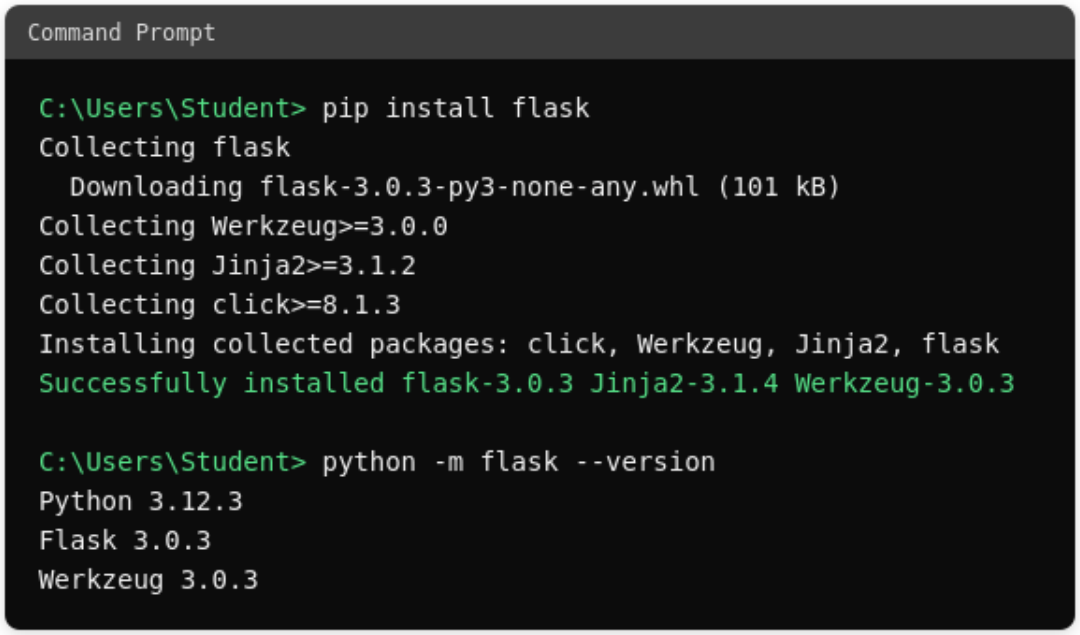


Fig 1.3: Installing Flask using pip and verifying the installed version

Practice Questions

Set A:

1. Why is Flask called a 'micro-framework'?
2. Name the two components on which Flask is built, and state the purpose of each.

Set B:

1. Fill in the blank: Flask follows the _____ standard, which defines how a Python web application communicates with a web server.
2. State True or False: 'Flask comes with a built-in database by default.'

Set C:

1. List any three features of the Flask framework.
2. Find out and write down the name of one other Python web framework apart from Flask.

4. Creating First Flask Application

A basic Flask application can be created by following a few simple steps:

1. Import the Flask class from the flask module.
2. Create an instance of the Flask class - this becomes the WSGI application.
3. Use the @app.route() decorator to bind a URL to a function (view function).

4. The view function returns the response that is to be displayed in the browser.
5. Run the application using app.run() with debug mode enabled for development.

Program: First Flask Application (app.py)

```
# app.py
# First Flask Application
 from flask import Flask
 # creating the Flask application object
app = Flask(__name__)
 # binding URL '/' to the home() function
@app.route('/')
def home():
    return "Hello, World! Welcome to my first Flask application."
 # binding URL '/about' to the about() function
@app.route('/about')
def about():
    return "This is the About page of my Flask website."
 # running the application
if __name__ == '__main__':
    app.run(debug=True)
```

To run the application:

python app.py

Output:

When the program is executed, the Flask development server starts and displays a message similar to:

```
 * Running on http://127.0.0.1:5000 (Press CTRL+C to quit)
```

On opening http://127.0.0.1:5000/ in the browser, the page displays:
Hello, World! Welcome to my first Flask application.

On opening http://127.0.0.1:5000/about in the browser, the page displays:
This is the About page of my Flask website.

More information about the program:

- __name__ is a special Python variable that tells Flask the location of the application, so that it can find related resources such as templates and static files.
- @app.route('/') is a decorator. A decorator adds extra functionality to a function - here it tells Flask which URL should trigger the function written below it (this is called a view function).
- debug=True enables debug mode, which automatically reloads the server whenever the code is changed and shows a detailed error page if an exception occurs - very useful during development.

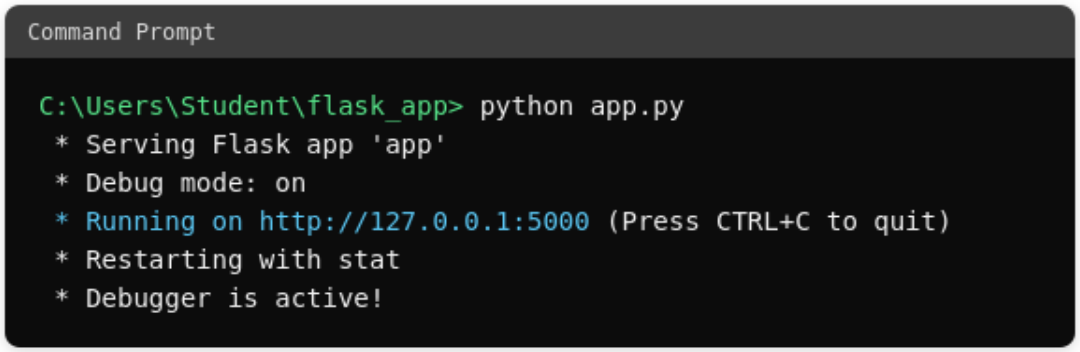


Fig 1.4(a): Output of running 'python app.py' in the command prompt

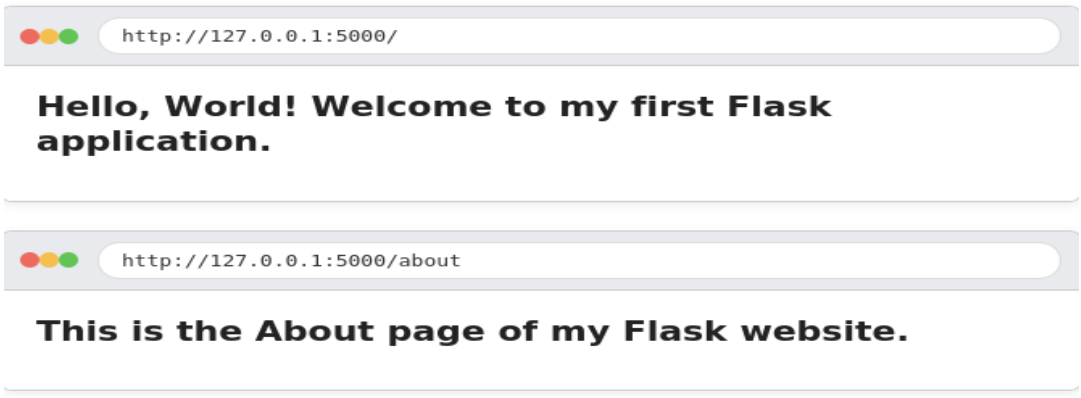


Fig 1.4(b): Browser output for the '/' and '/about' routes

Practice Questions

Set A:

1. What is the use of the @app.route() decorator in a Flask program?
2. What is the purpose of the special variable __name__ in a Flask program?

Set B:

1. Fill in the blank: The function written below @app.route() is called a _____ function.
2. State True or False: 'debug=True automatically restarts the server when the code is changed.'

Set C:

1. Write a Flask program with a route '/' that displays your name, and a route '/college' that displays the name of your college.
2. Modify the first Flask application shown in this unit so that the '/about' page displays 'About Me: <Your Name>, a TYBBA(CA) student.'

5.2 Routing and Templates

1. URL Routing

Routing refers to mapping of URLs to specific functions that handle the logic for that URL. In Flask, routing is performed using the @app.route() decorator placed above a view function.

When a client requests a particular URL, Flask matches it with the routes defined in the application and executes the corresponding view function, returning its result as the response.

Syntax:

```
@app.route('/url_path')
def function_name():
    return "Response content"
```

```
# app.py
# Demonstrating URL Routing

from flask import Flask

app = Flask(__name__)

@app.route('/')
def home():
    return "Welcome to the Home Page"

@app.route('/contact')
def contact ():
    return "This is the Contact Page"

@app.route('/services')
def services():
    return "List of Services offered by us"

if __name__ == '__main__':
    app.run(debug=True)
```

Output:

```
http://127.0.0.1:5000/          -> Welcome to the Home Page
http://127.0.0.1:5000/contact   ->  This is the Contact Page
http://127.0.0.1:5000/services  ->  List of Services offered by us
```

More information:

- Each @app.route () decorator can be applied to only one function, but the same function can be reused for multiple routes by stacking multiple @app.route() decorators above it.
- If a user tries to access a URL that has not been defined in any route, Flask automatically returns a '404 Not Found' error page.
- Routes are matched in the order they appear, and Flask is case-sensitive and exact when matching URL paths (e.g., '/Contact' and '/contact' are treated as different routes).

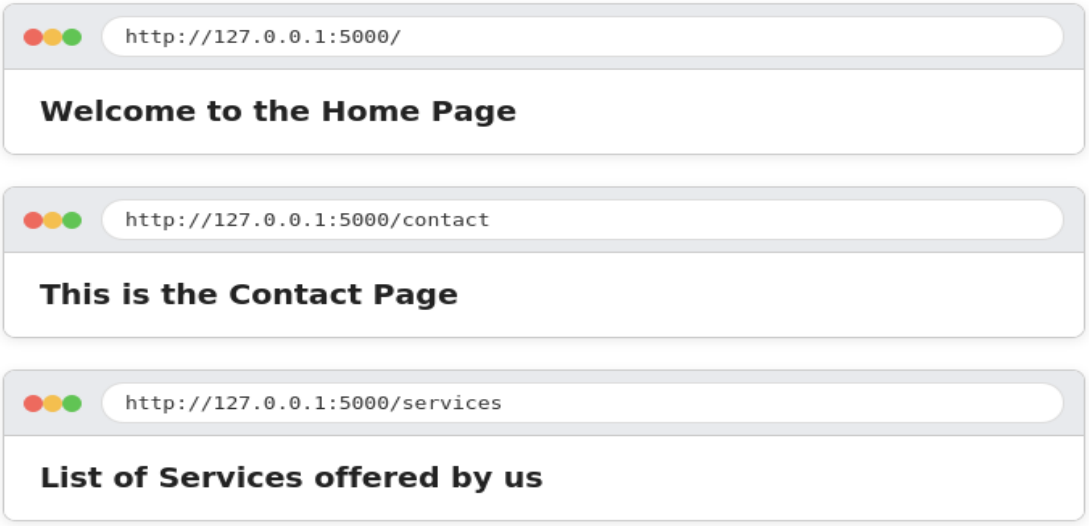


Fig 2.1: Browser output for the '/', '/contact' and '/services' routes

Practice Questions

Set A:

1. What is the purpose of the @app.route() decorator?
2. What happens when a user tries to access a URL that has not been defined as a route?

Set B:

1. Fill in the blank: Flask routes are _____-sensitive, so '/Contact' and '/contact' are treated as different routes.
2. State True or False: 'The same view function cannot be used for two different routes.'

Set C:

1. Write a Flask program with three routes - '/', '/courses' and '/faculty' - each displaying a different message about your college.
2. Add a fourth route '/timetable' to the program written above, displaying the message 'Timetable will be uploaded soon.'

2. Dynamic Routes

Dynamic routes allow part of the URL to be treated as a variable, which is passed as an argument to the view function. This is useful when the same logic needs to handle multiple similar requests, such as displaying details of different students or products.

A variable section in the route is marked using angle brackets <variable_name>. Flask also supports converters to specify the data type of the variable, such as int, float, string and path.

Syntax:

```
@app.route('/url/<variable_name>')
def function_name(variable_name):
    return "Value received: " + variable_name

@app.route('/url/<int:variable_name>')
def function_name(variable_name):
    return "Value received: " + str(variable_name)


# app.py
```

```
# Demonstrating Dynamic Routes

from flask import Flask

app = Flask(__name__)

# string type dynamic route
@app.route('/student/<name>')
def student(name):
    return "Welcome, " + name

# integer type dynamic route
@app.route('/marks/<int:score>')
def marks(score):
    return "Your marks are: " + str(score)

# float type dynamic route
@app.route('/percentage/<float:per>')
def percentage(per):
    return "Your percentage is: " + str(per)

if __name__ == '__main__':
    app.run(debug=True)
```

Output:

```
http://127.0.0.1:5000/student/Riya       ->  Welcome, Riya
http://127.0.0.1:5000/marks/85           ->  Your marks are: 85
http://127.0.0.1:5000/percentage/76.5    ->  Your percentage is: 76.5
```

More information about converters:

- string (default): accepts any text without a slash. Used when no converter is specified, e.g., <name>.
- int: accepts positive integer values only, e.g., <int:score>.
- float: accepts positive floating point (decimal) values, e.g., <float:per>.
- path: similar to string but also accepts slashes, useful for capturing file paths.
- If the URL value does not match the specified converter (for example, passing text where an int is expected), Flask returns a '404 Not Found' error for that route.

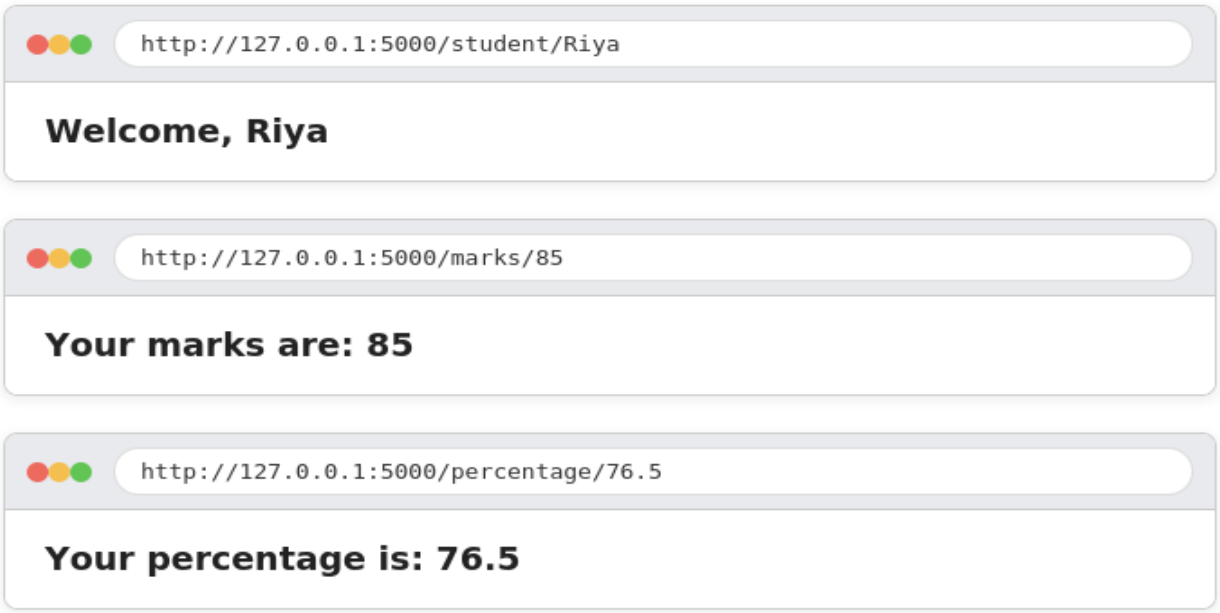


Fig 2.2: Browser output for the dynamic routes with string, int and float converters

Practice Questions

Set A:

1. What is a dynamic route in Flask? Give an example.
2. What is the difference between <name> and <int:age> in a route definition?

Set B:

1. Fill in the blank: The _____ converter is used in a route to accept only whole numbers.
2. State True or False: '<path:filename> can accept values containing a forward slash (/).'

Set C:

1. Write a Flask program with a dynamic route '/roll/<int:roll_no>' that displays 'Roll Number: <roll_no>'.
2. Write a Flask program with a dynamic route '/product/<name>/<float:price>' that displays the product name and its price.

3. HTML Templates using Jinja2

Instead of returning plain text or HTML directly from the view function, Flask allows rendering of separate HTML files known as templates. These templates are stored in a folder named 'templates' inside the project directory.

Flask uses the Jinja2 templating engine to render these HTML files. Jinja2 provides special delimiters to embed Python-like expressions and logic inside HTML:

- {{ ... }} - used to print the value of a variable or expression.
- {% ... %} - used for statements such as for loops, if conditions, etc.
- {# ... #} - used to write comments that are not rendered in the output.

Folder Structure:

```
project_folder/
   app.py
   templates/
      index.html
```

```
<!-- templates/index.html -->
<!DOCTYPE html>
<html>
<head>
   <title>My Flask Website</title>
</head>
<body>
   <h1>Welcome to Flask Templates</h1>
   <p>This page is rendered using Jinja2 templating engine.</p>
</body>
</html>
```

```
# app.py
# Rendering an HTML Template

from flask import Flask, render_template

app = Flask(__name__)

@app.route('/')
def home():
   return render_template('index.html')

if __name__ == '__main__':
   app.run(debug=True)
```

Output:

On opening http://127.0.0.1:5000/ in the browser, the contents of index.html are displayed as a formatted web page with the heading "Welcome to Flask Templates" and the paragraph below it.

More information:

- render_template () automatically looks for the given file inside the 'templates' folder - the folder name and location must be exactly as shown, otherwise Flask raises a 'TemplateNotFound' error.
- Separating HTML (templates) from Python code (app.py) keeps the application organized and follows good programming practice, making it easier for designers and developers to work independently.
- Comments written using {# ... #} are useful for leaving notes for the developer inside the template without affecting the output, since Jinja2 removes them before rendering.

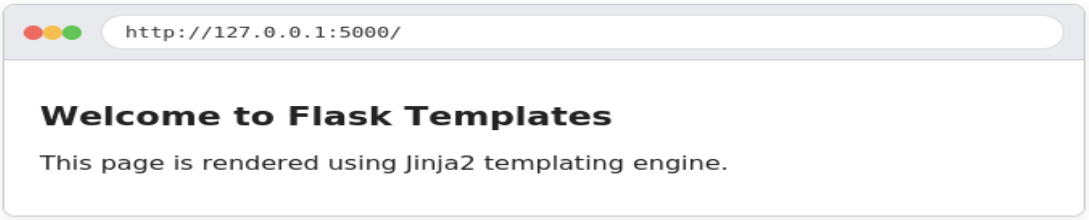


Fig 2.3: Browser output of index.html rendered using render_template()

Practice Questions

Set A:

1. What is the purpose of the render_template() function in Flask?
2. Where should HTML template files be stored in a Flask project?

Set B:

1. Fill in the blank: In Jinja2, _____ is used to print the value of a variable, while _____ is used for statements such as loops and conditions.
2. State True or False: 'Comments written using {# ... #} are displayed in the browser output.'

Set C:

1. Create a template named 'welcome.html' that displays your name and course, and write a Flask program to render it on the home page.
2. Create a template named 'rules.html' listing three rules of your college as plain text, and render it using a Flask route '/rules'.

4. Passing Data to Templates

Flask allows data from the view function (Python code) to be passed to the HTML template using the render_template() function. The data is passed as keyword arguments, which can then be accessed in the template using Jinja2 syntax.

Jinja2 also allows the use of control structures such as for loops and if-else conditions inside templates, which makes it possible to display lists, tables and conditional content dynamically.

Syntax:

return render_template('file.html', variable1=value1, variable2=value2)

```
<!-- templates/student.html -->
<!DOCTYPE html>
<html>
<head>
   <title>Student Details</title>
</head>
<body>
   <h2>Student Information</h2>
   <p>Name: {{ name }}</p>
   <p>Course: {{ course }}</p>

   <h3>Subjects</h3>
   <ul>
```

```
        {% for subject in subjects %}
            <li>{{ subject }}</li>
        {% endfor %}
    </ul>

    {% if percentage >= 50 %}
        <p>Result: PASS</p>
    {% else %}
        <p>Result: FAIL</p>
    {% endif %}
</body>
</html>

# app.py
# Passing Data to Templates

from flask import Flask, render_template

app = Flask(__name__)

@app.route('/')
def student():
    name = "Aarav Sharma"
    course = "TYBBA(CA)"
    subjects = ["Python", "Flask", "Database Management", "Networking"]
    percentage = 78.5
    return render_template('student.html', name=name, course=course,
                           subjects=subjects, percentage=percentage)

if __name__ == '__main__':
    app.run(debug=True)
```

Output:

The browser displays the Student Information page with the name, course, a bulleted list of subjects and the message "Result: PASS" since the percentage is greater than 50.

More information:

- Any Python data type - strings, numbers, lists, dictionaries, or even objects fetched from a database - can be passed to a template through render_template().
- The {% for %} loop in Jinja2 works similarly to a Python for loop and is commonly used to display lists or table rows generated from database records.
- The {% if %} ... {% else %} ... {% endif %} block allows the template to display different content based on conditions, exactly like an if-else statement in Python.

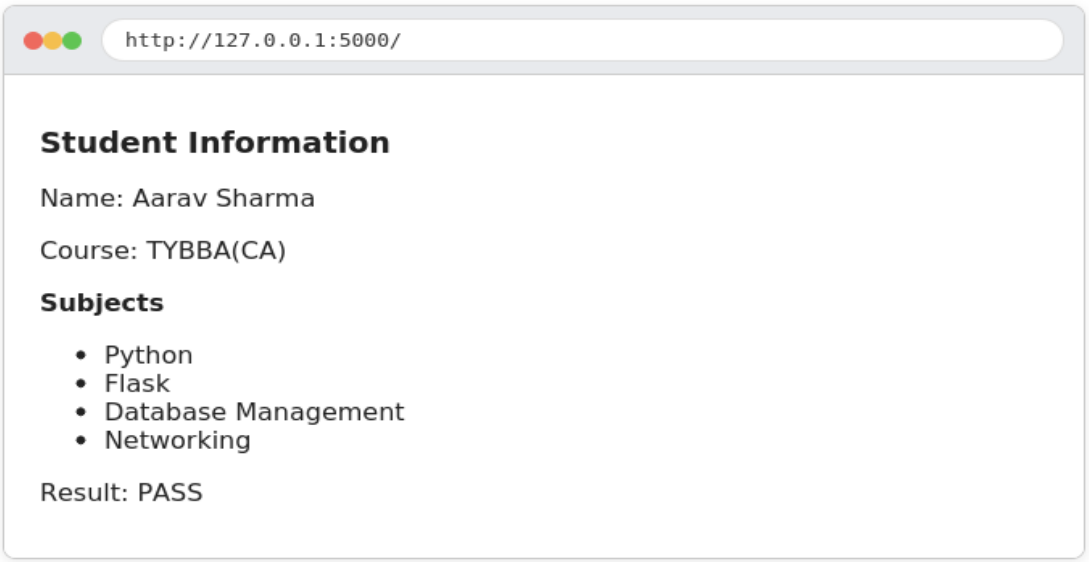


Fig 2.4: Browser output of student.html with data passed from app.py

Practice Questions

Set A:

1. How is data passed from a Flask view function to an HTML template?
2. What is the purpose of the {% for %} loop in Jinja2?

Set B:

1. Fill in the blank: The {% if %} ... {% else %} ... {% endif %} block in Jinja2 is used to display content based on a _____.
2. State True or False: 'Only strings and numbers can be passed to a Jinja2 template; lists cannot be passed.'

Set C:

1. Write a Flask program that passes your name and a list of three hobbies to a template, and display the hobbies using a {% for %} loop.
2. Write a Flask program that passes a number to a template and uses {% if %} ... {% else %} to display 'Pass' if the number is 40 or more, otherwise 'Fail'.

5.3 Forms and User Input

1. HTML Forms

An HTML form is used to collect input from the user and send it to the server for processing. A form is created using the <form> tag, which contains various input elements such as text boxes, radio buttons, checkboxes, drop-down lists and buttons.

- Action attribute: specifies the URL to which the form data should be submitted.
- Method attribute: specifies the HTTP method (GET or POST) used to send the data.
- Name attribute of input fields: used to identify the field's value on the server side.

Example: HTML Form (templates/form.html)

```
<!-- templates/form.html -->
<!DOCTYPE html>
<html>
```

```
<head>
   <title>Registration Form</title>
</head>
<body>
   <h2>Student Registration Form</h2>
   <form action="/submit" method="POST">
      Name: <input type="text" name="name"><br><br>
      Email: <input type="text" name="email"><br><br>
      Course:
      <select name="course">
         <option value="BBA(CA)">BBA(CA)</option>
         <option value="BCA">BCA</option>
         <option value="BSc-IT">BSc-IT</option>
      </select><br><br>
      <input type="submit" value="Register">
   </form>
</body>
</html>
```

Output:

The browser displays a registration form containing a text box for Name, a text box for Email, a drop-down list for Course and a Submit button. On clicking Submit, the entered data is sent to the '/submit' URL using the POST method.

More information about form elements:

- Common <input> types include text (single-line text), password (hides typed characters), checkbox, radio and submit (the button that sends the form).
- The <select> tag along with <option> tags is used to create a drop-down list; the value of the selected option is sent to the server using the name given to the <select> tag.
- Every input field that needs to be read on the server side must have a unique 'name' attribute - request.form['name_of_field'] in Flask retrieves the value using this attribute.

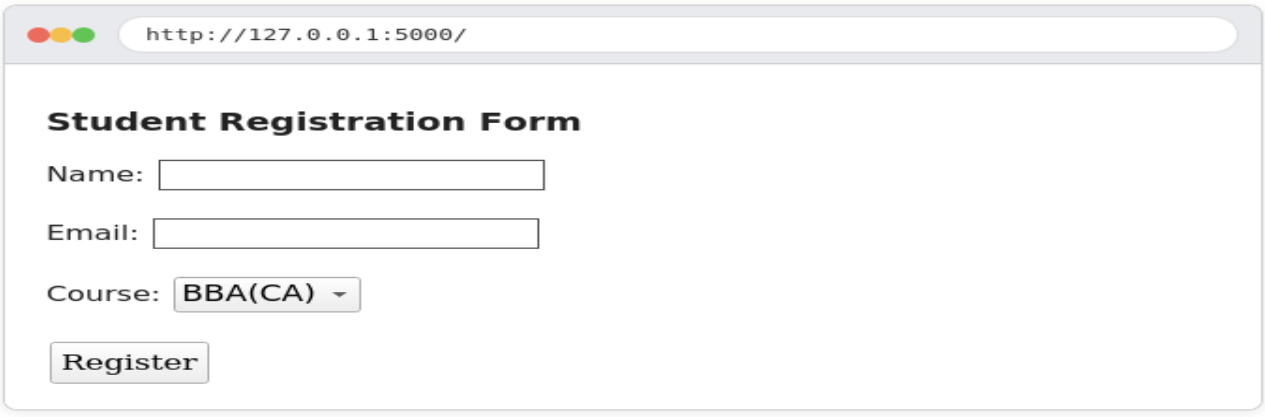


Fig 3.1: Student Registration Form rendered in the browser

Practice Questions

Set A:

1. What is the purpose of the 'action' and 'method' attributes of the <form> tag?
2. Why must every input field have a unique 'name' attribute?

Set B:

1. Fill in the blank: The <select> tag along with <option> tags is used to create a _____ list.
2. State True or False: 'The <input type="submit"> element is used to send the form data to the server.'

Set C:

1. Create an HTML form with fields for Name, Age and City, with a Submit button.
2. Add a drop-down list of three favourite subjects to the form created above.

2. GET and POST Methods

GET and POST are the two most commonly used HTTP methods for sending data from the client to the server.

- GET Method: Appends the form data to the URL in the form of a query string (e.g., ?name=value). It is visible in the URL, has a limited amount of data that can be sent and is generally used for retrieving data.
- POST Method: Sends the form data as part of the request body, not visible in the URL. It can send larger amounts of data and is generally used for submitting sensitive information such as passwords or large forms.

By default, a Flask route only responds to GET requests. To allow a route to accept other methods, the methods argument is used in the @app.route() decorator.

Syntax:

```
@app.route('/url', methods=['GET', 'POST'])
def function_name():
    # code to handle both GET and POST requests
    pass
```

```
# app.py
# Demonstrating GET and POST methods

from flask import Flask, request

app = Flask(__name__)

@app.route('/greet', methods=['GET'])
def greet():
    # GET request - data comes from the query string
    name = request.args.get('name')
    return "Hello (via GET), " + str(name)

@app.route('/welcome', methods=['POST'])
def welcome():
```

```
    # POST request - data comes from the form body
    name = request.form.get('name')
    return "Hello (via POST), " + str(name)

if __name__ == '__main__':
    app.run(debug=True)
```

Output:

GET Request:
http://127.0.0.1:5000/greet?name=Riya
-> Hello (via GET), Riya

POST Request (submitted through a form with field name="name"):
-> Hello (via POST), Riya

More information:

- request.args is used to read data sent via the GET method (from the URL query string), while request.form is used to read data sent via the POST method (from the form body).
- GET requests can be bookmarked, cached and shared as a link because the data is part of the URL; POST requests cannot, since the data is hidden inside the request body.
- If a route is accessed with a method that is not listed in 'methods', Flask returns a '405 Method Not Allowed' error.

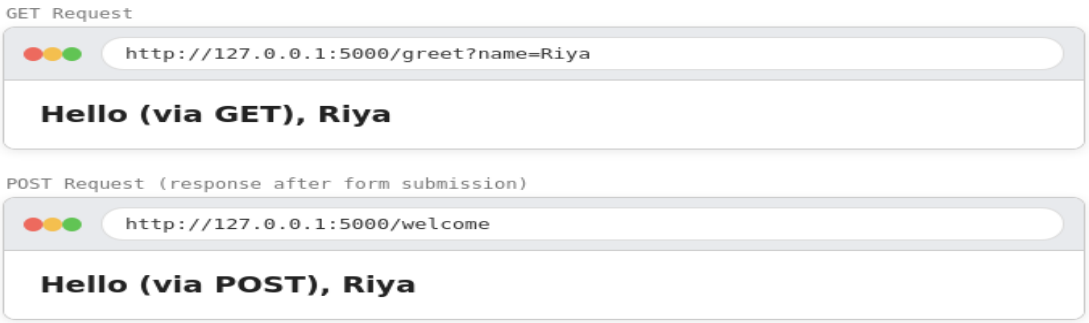


Fig 3.2: Browser output for a GET request and a POST request

Practice Questions

Set A:

1. What is the main difference between the GET and POST methods?
2. Which Flask object is used to read data sent via GET, and which is used for POST?

Set B:

1. Fill in the blank: By default, a Flask route responds only to the _____ method.
2. State True or False: 'POST data is visible in the URL of the browser.'

Set C:

1. Write a Flask program with a route '/search' that uses the GET method to accept a 'keyword' from the URL and displays 'Searching for: <keyword>'.

2. Write a Flask program with a route '/subscribe' that uses the POST method to accept an 'email' from a form and displays 'Subscribed: <email>'.

3. Handling Form Data in Flask

Flask provides the 'request' object (imported from the flask module) to access incoming request data. This object provides several attributes to retrieve data submitted by the user:

- request.form: a dictionary-like object containing data submitted through a POST form.
- request.args: a dictionary-like object containing data submitted through the URL query string (GET method).
- request.method: returns the HTTP method (GET or POST) used for the current request.

Example: Handling Form Data (app.py)

```
# app.py
# Handling Form Data submitted by the user

from flask import Flask, request, render_template

app = Flask(__name__)

@app.route('/')
def index():
    return render_template('form.html')

@app.route('/submit', methods=['POST'])
def submit():
    name = request.form['name']
    email = request.form['email']
    course = request.form['course']
    return render_template('result.html', name=name, email=email, course=course)

if __name__ == '__main__':
    app.run(debug=True)


<!-- templates/result.html -->
<!DOCTYPE html>
<html>
<head>
    <title>Registration Successful</title>
</head>
<body>
    <h2>Registration Successful</h2>
    <p>Name: {{ name }}</p>
    <p>Email: {{ email }}</p>
    <p>Course: {{ course }}</p>
</body>
```

```
</html>
```

Output:

After filling the registration form (created in the previous practical) and clicking Submit, the browser navigates to the '/submit' URL and displays a "Registration Successful" page showing the Name, Email and Course entered by the user.

More information:

- request.form ['fieldname'] raises an error if the field is missing from the submitted form; request.form.get ('fieldname') is safer as it returns none instead of raising an error when the field is absent.
- The same view function (e.g., submit ()) can both read the submitted data and decide which template to render next, allowing the application to respond differently based on the data received.
- It is good practice to keep the form page and the result page as two separate templates (form.html and result.html), which keeps the HTML for each screen organized and easy to maintain.

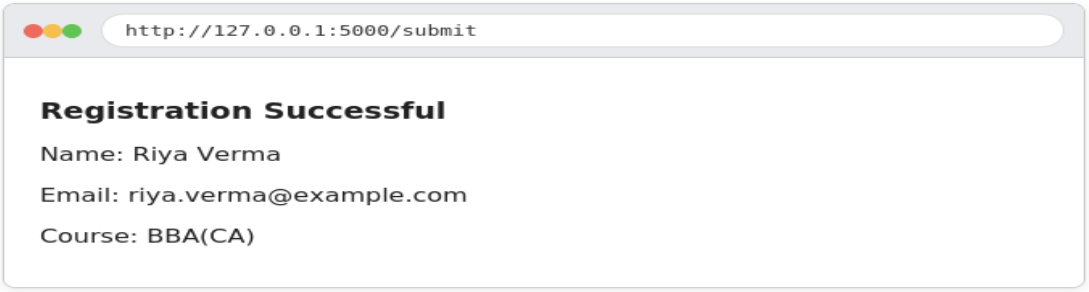


Fig 3.3: 'Registration Successful' page rendered after submitting the form

Practice Questions

Set A:

1. What is the difference between request.form ['name'] and request.form.get ('name')?
2. Why is it good practice to keep the form page and the result page as two separate templates?

Set B:

1. Fill in the blank: The _____ object in Flask is used to access data submitted through a form.
2. State True or False: 'request.form.get("name") raises an error if the field "name" is missing.'

Set C:

1. Write a Flask program with a form that accepts your Name and Age, and displays 'Name: <name>, Age: <age>' on submission.
2. Modify the program above to also display 'You are eligible to vote' if the age is 18 or more, otherwise 'You are not eligible to vote'.

4. Form Validation Basics

Form validation refers to checking whether the data entered by the user is correct, complete and in the expected format before it is processed further. Validation can be performed on the client side using JavaScript or on the server side using Flask.

Server-side validation is important because it cannot be bypassed by the user, unlike client-side validation.

In Flask, basic validation is performed inside the view function by checking the values received through request.form before processing them further. If validation fails, an error message can be passed back to the template and displayed to the user.

Example: Server-side Form Validation (app.py)

```
# app.py
# Demonstrating Basic Form Validation

from flask import Flask, request, render_template

app = Flask(__name__)

@app.route('/')
def index():
    return render_template('form.html')

@app.route('/submit', methods=['POST'])
def submit():
    name = request.form.get('name', '').strip()
    email = request.form.get('email', '').strip()
    course = request.form.get('course', '').strip()

    errors = []

    # checking for empty fields
    if name == '':
        errors.append("Name field cannot be empty.")
    if email == '':
        errors.append("Email field cannot be empty.")
    elif '@' not in email:
        errors.append("Please enter a valid email address.")

    # if there are validation errors, show them back to the user
    if errors:
        return render_template('form.html', errors=errors)

    # if validation passes, show the result page
    return render_template('result.html', name=name, email=email, course=course)

if __name__ == '__main__':
    app.run(debug=True)


<!-- templates/form.html (updated to display errors) -->
<!DOCTYPE html>
<html>
<head>
    <title>Registration Form</title>
</head>
```

```
<body>
  <h2>Student Registration Form</h2>

  {% if errors %}
    <ul style="color:red;">
      {% for error in errors %}
        <li>{{ error }}</li>
      {% endfor %}
    </ul>
  {% endif %}

  <form action="/submit" method="POST">
    Name: <input type="text" name="name"><br><br>
    Email: <input type="text" name="email"><br><br>
    Course:
    <select name="course">
      <option value="BBA(CA)">BBA(CA)</option>
      <option value="BCA">BCA</option>
    </select><br><br>
    <input type="submit" value="Register">
  </form>
</body>
</html>
```

Output:

If the Name or Email field is left empty, or the Email does not contain '@', the form is displayed again along with the relevant error message(s) in red. If all fields are valid, the Registration Successful page is displayed.

More information:

- .strip() removes any extra leading or trailing spaces entered by the user, so that a field containing only spaces is also treated as empty.
- The 'errors' list collects every validation message; passing this list to the template allows all problems to be shown to the user at once, instead of one at a time.
- This kind of validation is called server-side validation because it runs on the Flask server after the form is submitted. It is more reliable than client-side (JavaScript) validation, which a user can sometimes bypass.

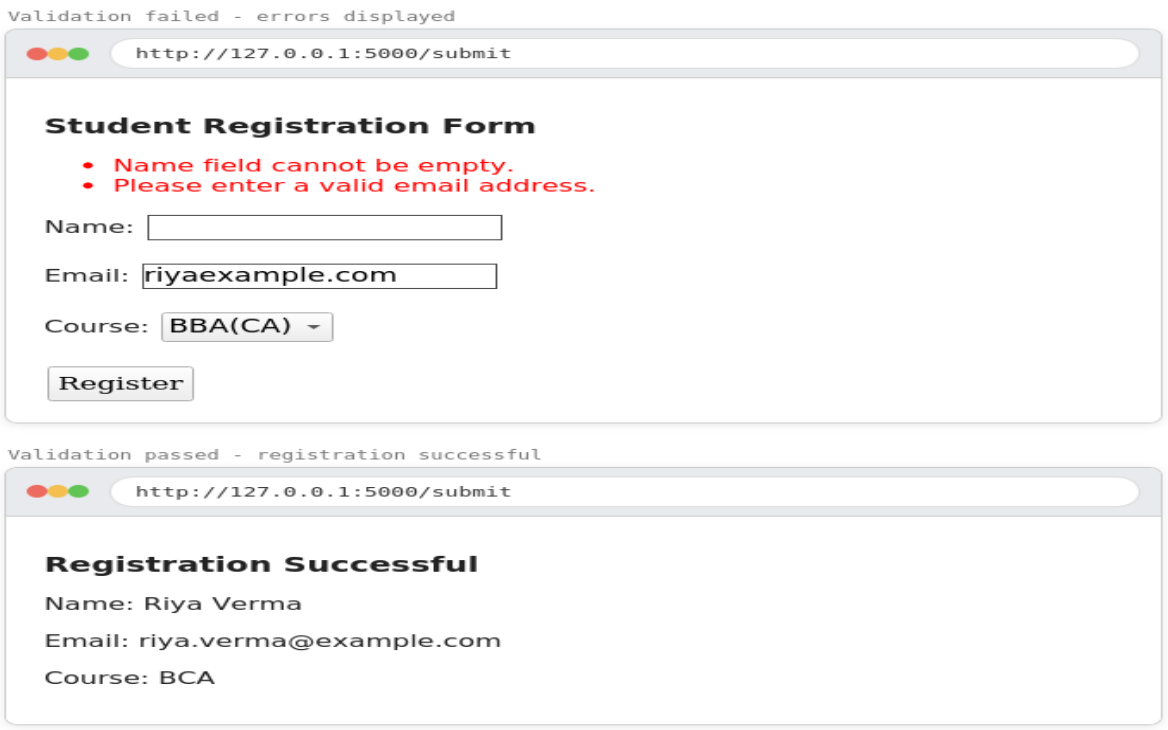


Fig 3.4: Form re-displayed with validation errors, and the success page shown once all fields are valid

Practice Questions

Set A:

1. Why server-side validation is considered more reliable than client-side validation?
2. What is the purpose of the .strip() method when validating form input?

Set B:

1. Fill in the blank: The list used to collect all validation error messages before displaying them is commonly named _____.
2. State True or False: 'If the "errors" list is empty, the form should be displayed again with error messages.'

Set C:

1. Write a Flask program to validate a registration form so that the Name field cannot be empty and the Age field must be a number greater than 0.
2. Extend the validation program above to also check that the Email field contains the '@' symbol.

5.4 Database Connectivity

1. Introduction to SQLite

SQLite is a lightweight, serverless and self-contained relational database management system. Unlike other databases such as MySQL or Oracle, SQLite does not require a separate server process - the entire database is stored in a single file with a .db or .sqlite extension on the disk.

Python provides a built-in module named sqlite3 to work with SQLite databases, so no additional installation is required. SQLite is widely used for small to medium applications, prototyping and learning database concepts due to its simplicity.

Program: Creating a Database and Table (create_db.py)

```
# create_db.py
# Creating an SQLite database and a table

import sqlite3

# connecting to the database (creates the file if it does not exist)
conn = sqlite3.connect('student.db')

# creating a cursor object to execute SQL commands
cursor = conn.cursor()

# creating a table named 'students'
cursor.execute('''
   CREATE TABLE IF NOT EXISTS students (
      id INTEGER PRIMARY KEY AUTOINCREMENT,
      name TEXT NOT NULL,
      course TEXT NOT NULL,
      marks INTEGER NOT NULL
   )
''')

# saving the changes
conn.commit()

# closing the connection
conn.close()

print("Database and table created successfully.")
```

Output:

Database and table created successfully.

A file named 'student.db' is created in the project folder, containing
an empty table named 'students' with columns id, name, course and marks.

More information:

- INTEGER PRIMARY KEY AUTOINCREMENT automatically generates a unique id for each new row, so it does not need to be entered manually while inserting records.
- IF NOT EXISTS ensures that running the script again does not delete or recreate the table (and its data) if it already exists.
- conn.commit() permanently saves the changes to the database file; without it, the changes made during the connection may be lost when the connection is closed.

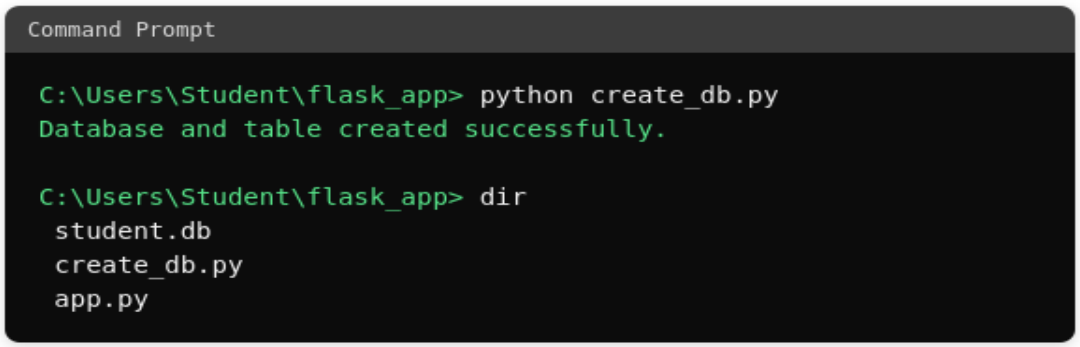


Fig 4.1: Running create_db.py to create the SQLite database and table

Practice Questions

Set A:

1. What is SQLite, and how is it different from MySQL?
2. Which Python module is used to work with SQLite databases?

Set B:

1. Fill in the blank: The SQL statement used to create a new table is _____.
2. State True or False: 'conn.commit() must be called to permanently save changes to the database.'

Set C:

1. Write a Python program to create a database 'library.db' with a table 'books' having columns id, title and author.
2. Run the program written above and verify that the file 'library.db' is created in your project folder.

2. Connecting Flask with SQLite

To use an SQLite database inside a Flask application, the sqlite3 module is imported along with Flask. A connection to the database file is established using sqlite3.connect(), and a cursor object is used to execute SQL queries.

It is good practice to open the connection inside the view function (or use a helper function) and close it after the required operation is performed, so that the database file is not left locked.

Syntax:

```
import sqlite3

def get_db_connection():
    conn = sqlite3.connect('student.db')
    conn.row_factory = sqlite3.Row   # allows accessing columns by name
    return conn


# app.py
# Connecting Flask application with SQLite database

from flask import Flask
```

```
import sqlite3

app = Flask(__name__)

def get_db_connection():
    conn = sqlite3.connect('student.db')
    conn.row_factory = sqlite3.Row
    return conn

@app.route('/')
def home():
    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute('SELECT COUNT(*) FROM students')
    total = cursor.fetchone()[0]
    conn.close()
    return "Total students in database: " + str(total)

if __name__ == '__main__':
    app.run(debug=True)
```

Output:

On opening http://127.0.0.1:5000/, the browser displays:
Total students in database: 0

(0 because the 'students' table created earlier is empty.)

More information:

- conn.row_factory = sqlite3.Row allows the columns of a fetched row to be accessed by their column name (e.g., student ['name']) instead of only by numeric index (e.g., student [1]), which makes the code in templates more readable.
- It is important to call conn.close () after every database operation. Leaving connections open can lock the database file, causing errors in subsequent requests.
- Defining get_db_connection() as a separate function avoids repeating the same connection code in every route, following the DRY (Don't Repeat Yourself) principle.

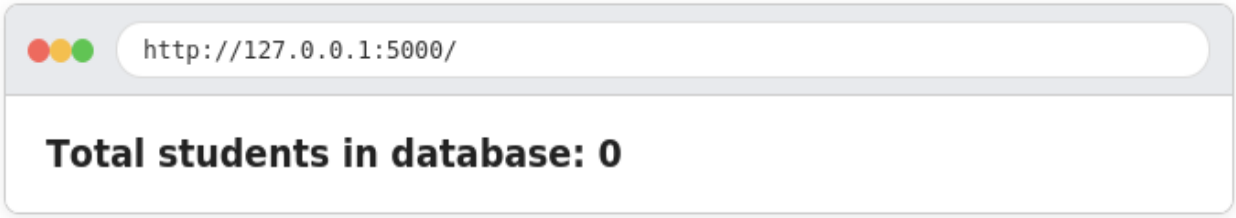


Fig 4.2: Browser output showing the total number of records in student.db

Practice Questions

Set A:

1. Why is it good practice to write a separate get_db_connection() function?
2. What is the purpose of conn.row_factory = sqlite3.Row?

Set B:

1. Fill in the blank: The SQL function used to count the number of rows in a table is _____.
2. State True or False: 'It is not necessary to close the database connection after performing an operation.'

Set C:

1. Write a Flask program that connects to 'library.db' (created in the previous topic) and displays the total number of books in the table.
2. Add a route '/library' to the program above that displays 'Library Database Connected Successfully' along with the total number of books.

3. CRUD Operations (Create, Read, Update, Delete)

CRUD stands for Create, Read, Update and Delete - the four basic operations that can be performed on data stored in a database table.

- Create: Adding new records to the table using the INSERT INTO SQL statement.
- Read: Retrieving existing records from the table using the SELECT SQL statement.
- Update: Modifying existing records using the UPDATE SQL statement.
- Delete: Removing records from the table using the DELETE FROM SQL statement.

In a Flask application, each of these operations is generally implemented as a separate route, and the corresponding SQL query is executed using the cursor object, followed by conn.commit() to save the changes (for Create, Update and Delete operations).

Program: CRUD Operations (app.py)

```
# app.py
# implementing CRUD operations using Flask and SQLite

from flask import Flask, request, redirect
import sqlite3

app = Flask(__name__)

def get_db_connection():
    conn = sqlite3.connect('student.db')
    conn.row_factory = sqlite3.Row
    return conn

# CREATE - add a new student record
@app.route('/add', methods=['POST'])
def add_student():
    name = request.form['name']
```

```
    course = request.form['course']
    marks = request.form['marks']

    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute('INSERT INTO students (name, course, marks) VALUES (?, ?, ?)',
                   (name, course, marks))
    conn.commit()
    conn.close()
    return redirect('/students')

# UPDATE - update marks of an existing student
@app.route('/update/<int:id>', methods=['POST'])
def update_student(id):
    new_marks = request.form['marks']

    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute('UPDATE students SET marks = ? WHERE id = ?', (new_marks, id))
    conn.commit()
    conn.close()
    return redirect('/students')

# DELETE - remove a student record
@app.route('/delete/<int:id>')
def delete_student(id):
    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute('DELETE FROM students WHERE id = ?', (id,))
    conn.commit()
    conn.close()
    return redirect('/students')

if __name__ == '__main__':
    app.run(debug=True)
```

Output:

- Submitting the Add Student form sends a POST request to '/add', a new row is inserted into the 'students' table, and the user is redirected to '/students'.
- Submitting the Update form for a particular student sends a POST request to '/update/<id>', updates the marks of that student, and redirects to '/students'.
- Clicking the Delete link for a student sends a request to '/delete/<id>', removes that record from the table, and redirects to '/students'.

More information:

- The '?' symbols in cursor.execute() are placeholders. Passing the actual values as a separate tuple (parameterized query) protects the application from SQL Injection attacks, which can occur if user input is directly inserted into the SQL string.
- redirect('/students') sends the browser to the '/students' route after the operation is completed, so that the user immediately sees the updated list of records.
- The Delete route uses a simple GET link here for simplicity; in real applications it is recommended to use a POST request (often with a confirmation step) for delete operations to avoid accidental deletion.

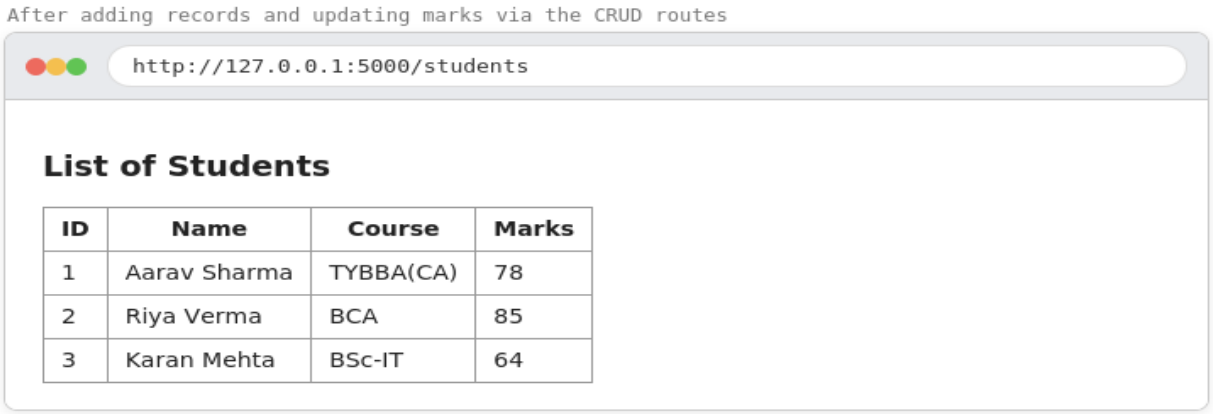


Fig 4.3: List of Students page after performing Add and Update operations

Practice Questions

Set A:

1. Expand the abbreviation CRUD and state the SQL statement used for each operation.
2. Why are '?' placeholders used instead of directly inserting values into the SQL query string?

Set B:

1. Fill in the blank: The SQL statement used to remove a record from a table is _____.
2. State True or False: 'redirect() is used to send the browser to a different route after an operation is completed.'

Set C:

1. Write a Flask program with a route '/add_book' that adds a new book (title, author) to the 'books' table using a form.
2. Write a Flask program with a route '/delete_book/<int:id>' that deletes a book record with the given id.

4. Displaying Database Records

To display the records stored in the database on a web page, the SELECT statement is used to fetch all rows from the table, and the result is passed to an HTML template using render_template(). The template then uses a Jinja2 for loop to display each record, typically inside an HTML table.

Program: Displaying Records (app.py and template)

```
# app.py
# Displaying records from the database

from flask import Flask, render_template
import sqlite3

app = Flask(__name__)

def get_db_connection():
    conn = sqlite3.connect('student.db')
    conn.row_factory = sqlite3.Row
    return conn

@app.route('/students')
def students():
    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute('SELECT * FROM students')
    all_students = cursor.fetchall()
    conn.close()
    return render_template('students.html', students=all_students)

if __name__ == '__main__':
    app.run(debug=True)


<!-- templates/students.html -->
<!DOCTYPE html>
<html>
<head>
    <title>Student Records</title>
</head>
<body>
    <h2>List of Students</h2>
    <table border="1" cellpadding="8">
        <tr>
            <th>ID</th>
            <th>Name</th>
            <th>Course</th>
            <th>Marks</th>
        </tr>
        {% for student in students %}
        <tr>
            <td>{{ student['id'] }}</td>
            <td>{{ student['name'] }}</td>
            <td>{{ student['course'] }}</td>
            <td>{{ student['marks'] }}</td>
        </tr>
```

```
        {% endfor %}
    </table>
</body>
</html>
```

Output:

On opening http://127.0.0.1:5000/students, the browser displays a table titled "List of Students" with columns ID, Name, Course and Marks, listing every record currently present in the 'students' table of the SQLite database.

More information:

- cursor.fetchall() returns all matching rows as a list; cursor.fetchone() can be used instead when only a single row is needed (for example, fetching one student's details by id).
- The {% for student in students %} loop runs once for every row returned by the query, generating one <tr> (table row) for each student record.
- If the 'students' table is empty, the loop simply does not execute, and the table is displayed with only the header row (ID, Name, Course, Marks) and no data rows.

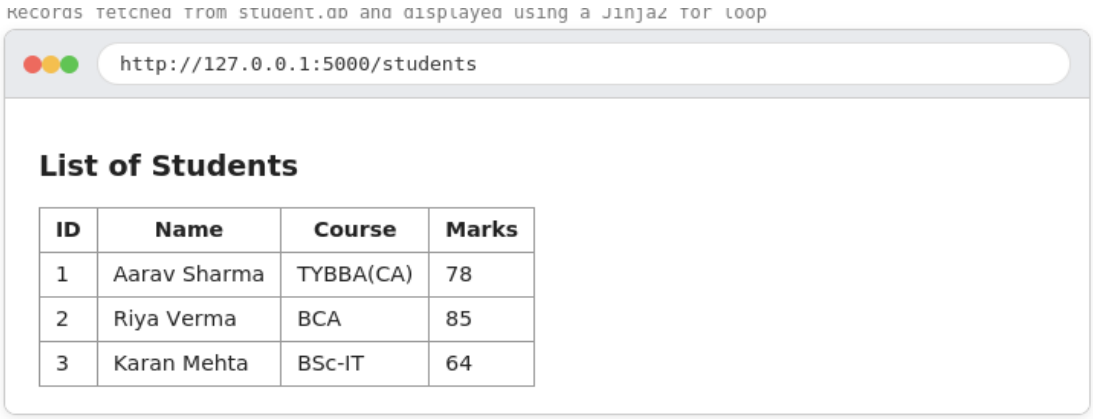


Fig 4.4: List of Students page displaying records fetched from the SQLite database

Practice Questions

Set A:

1. Which cursor method is used to fetch all rows returned by a SELECT query?
2. What happens if the {% for %} loop is used on an empty list of records?

Set B:

1. Fill in the blank: Inside a Jinja2 template, a column value of a row can be accessed using row['_____'].
2. State True or False: 'render_template() can only accept one variable from the view function.'

Set C:

1. Write a Flask program with a route '/books' that fetches all records from the 'books' table and displays them in an HTML table.
2. Modify the program above to display only the books whose price is less than 500 (Hint: use a WHERE clause in the SQL query).

Practice Set:

a. Write a Flask program to create a Student Registration form (Name, Email, Course, Year) and store the submitted data in an SQLite database table named 'registrations'. Display all registered students in an HTML table.
b. Write a Flask program to create a Simple Interest Calculator. Accept the Principal amount, Rate of Interest and Time (in years) through an HTML form, and display the Simple Interest and the Total Amount.
c. Write a Flask program to create a simple To do List application. Accept a task name through an HTML form, store it in an SQLite database table named 'tasks', and display all the tasks as a numbered list.

Set A

a. Write a Flask program to display 'Welcome to Python Flask Framework' on the home page ('/') and 'This is the Student Portal' on the '/student' page
b. Write a Flask program to display the current date and time on the home page
c. Write a Flask program to accept a person's name through the URL (dynamic route) and display a greeting message 'Hello, <name>! Welcome to our website.
d. Write a Flask program to pass a list of fruit names to a template and display them as a bulleted list using a Jinja2 for loop

Set B

a. Write a Flask program to create a simple feedback form with Name and Feedback fields using the GET method, and display the submitted values
b. Write a Flask program to accept two numbers through an HTML form (POST method) and display their sum
c. Write a Flask program to accept a number through an HTML form and check whether it is Positive, Negative or Zero.

Set C

a. Write a Python program to create an SQLite database 'employee.db' with a table named 'employee' having columns id, name and salary. .
b. Write a Flask program to insert a new employee record (name and salary) into the 'employee' table (created in the previous slip) using an HTML form
c. Write a Flask program to render an HTML template (college.html) that displays the name, address and contact number of a college.

Evaluation

0: Not Done [ ] **1: Incomplete [ ]** **2: Late Complete [ ]**

3: Needs Improvement [ ] **4: Complete [ ]** **5: Well Done [ ]**

Signature of Instructor