methods: A method is a named unit of a group of statements that can be invoked from other parts of the program. The advantages of using methods are:

· methods enable us to break a program down into a number of smaller and simpler units. This makes the program more organized and structured and easier to debug.

· It helps in reusability of code and thus reduces program size. If similar code is required at several places in the program, the use of a method allows this code to be written just once, and to be called wherever it is required.

· It helps to execute the same set of statements with different set of values.

<return type>: It is the data type of the value that is returned to the calling method after the method is executed. It can be any of the data types, available in Java.

<method name>: It is the name of the method, which is given with accordance to the naming conventions used for naming an identifier. A method is called with the help of it's name.

<data type of parameter>: These are the data types of the parameters/ inputs that the method receives when it is called. In method declaration it is optional to give the name of the parameters. The different data types specify to the compiler, the total number of parameters, and the data type of these parameters that the method will receive, when it is called.

method definition: The method definition contains the method header and the method body i.e. the set of statements which are executed when the method is called.

Syntax for method definition:

```
<return type> <method name> (<data type of parameter> <name of parameter>, …) //method header
{
        statement 1;
        statement 2;
        .
        .                                       // method Body
        .
        return (value/variable/expression);
}
```

Example:

```
public static int sum(int a, int b)
{
        int s=a+b;
        return s;
}
```

Note: The ***return statement*** is used to return a value to the calling method. It is always the last statement in the method definition (if the method has a return type). The data type of the value returned should match with the return type of the method.

//Example 1

//Program to find the factorial of a number using a method Fact( )

```
 1 import java.util.Scanner;
 2
 3 public class method
 4 { // Definition of the Called method fact( )
 5 public static long Fact(long n) //method header with n as a formal parameter
 6 {
 7         long i,f=1;
 8         for(i=1;i<=n;i++)                                          //method Body
 9                 f*=i;
10         return f; //return statement returning the value of variable f
11 }
12
13 public static void main( String [ ] args) //Calling method
14 {  Scanner input=new Scanner(System.in);
15    long num;
16    System.out.println("Enter a number:");
17    num=input.nextInt();
18    System.out.println(num + "!="  + Fact(num)); //method call with num as an actual
//parameter
19 }
20
21 }
```

<u>Order of Execution in the above program</u>

main( ) //Execution starts from the main( )

↓

method call to Fact( ) // Control is passed to the method definition of Fact( )

↓

Execution of the Fact( ) // All the statements in the method Definition of Fact( ) are executed

method Header: It is the first line of the method definition, which specifies the following:

- Return type of the method
- Name of the method
- Name of the parameters, along with their data types

Note: It is not terminated with semicolon.

method Body: It is the set of statements, which are executed when the method is called.

method Call: It is the statement in the calling method, through which the control is passed to the method definition of another method. This will execute the statements in the called method and then the control is returned back to the statement in the calling method, which follows the method call statement.. The method call consists of the method name, followed by actual parameters, if any, enclosed in parentheses.

Calling method: The method, which calls a method is called the calling method. In the above example **1**, **main( )** is the calling method.

Called method: The method which is called by another method is known as the called method. In the above example **1**, the method **Fact( )** is the called method.

Actual Parameters: Values/ Variables which are used while making a call to the method are called actual parameters. In the above example **1**, variable **num** is an actual parameter.

Formal Parameters: The parameters mentioned in the method header are called the formal parameters. In the above example **1**, parameter **n** is the formal parameter.

method Arguments/Parameters: Argument(s)/Parameter(s) of a method is(are) the data that a method receives when called from another method. It is not always necessary for a method to have arguments/parameters. Parameters allow a method to operate the same set of statements with different set of values.

Java has two kinds of methods:
Methods that return a single item
Methods that perform some action other than returning an item
If a method returns a single quantity, you can invoke it anywhere that you can use a value of the type returned by the method. For example, the following statement includes an invocation of the method getAgeInHumanYears, and the value returned is assigned to the int variable humanYears:

humanYears = scooby.getAgeInHumanYears();

If a method performs some action other than returning a single quantity, you write its invocation followed by a semicolon. The resulting Java statement performs the action defined by the method. For example, the following is an invocation of the method writeOutput for the object balto:
balto.writeOutput();
This method invocation displays several lines of output on the screen.

Java passes arguments to a method using call-by-value:

When a method is invoked, each parameter is initialized to the value of the corresponding argument in the method invocation. This type of substitution is known as the call-by-value parameter mechanism. The argument in a method invocation can be a literal constant, such as 2or ‘A'; a variable; or any expression that yields a value of the appropriate type.

Arguments must match parameters in number, order, and type

Within a method definition, formal parameters are given within the parentheses after the method name. In a method invocation, arguments are given within the parentheses after the method name. Arguments must match the formal parameters in the method heading with respect to their number, their order, and their data types.

The arguments are plugged in for their corresponding formal parameters. The first argument in the method invocation is plugged in for the first parameter in the method definition, the second argument in the method invocation is plugged in for the second parameter in the method definition, and so forth. Arguments should be of the same types as their corresponding formal parameters, although in some cases, Java will perform an automatic type conversion when the types do not match exactly.

Local Variables: Local variables are those variables which are declared within a method or a compound statement and these variables can only be used within that method/scope . They cannot be accessed from outside the method or a scope of it’s declaration. This means that we can have variables with the same names in different methods/scope. **Local variables are local to the method/scope in which they are declared.**
One method’s local variables have no meaning within another method. **Moreover, if two methods each have a local variable with the same name, they are considered two different variables.**