Principles of Programming Languages

Lab manual – COMP 316

Level - XIII

Department of Computer Science

College of Computer Science & Information Technology

Jazan University

1. Programs examine data types

Variables can hold values, and every value has a data-type.

A variable can hold different types of values. For example, a person's name must be stored as a string whereas its id must be stored as an integer.

Primary Data Types in C

The C programming language has five primitive or primary data types.

1. **Integer (int):** Refers to positive and negative whole numbers (without decimal), such as 10, 12, 65, 3400, etc.

```
#include <stdio.h>
void main()
{
 int i = 5;
 printf("The integer value is: %d \n", i);
}
```

2. **Character (char):** Refers to all the ASCII character sets within single quotes such as ‘a’, ‘A’, etc.

```
#include <stdio.h>
void main()
{
 char c = 'b';
 printf("The character value is: %c \n", c);
}
```

3. **Floating-point (float):** Refers to all the real number values or decimal points, such as 3.14, 10.09, 5.34, etc.

```
#include <stdio.h>
void main()
{
 float f = 7.2357;
 printf("The float value is: %f \n", f);
}
```

4. **Double (double):** Used when the range exceeds the numeric values that do not come under either floating-point or integer data type.

```
#include <stdio.h>
void main()
{
   double d = 71.2357455;
   printf("The double value is: %lf \n", d);
}
```

Primitive Data Types in Java

Primitive data types in Java are predefined by the Java language and named as the reserved keywords. A primitive data type does not share a state with other primitive values. Java programming language supports the following eight primitive data types.

1. Boolean data type
2. byte data type
3. int data type
4. long data type
5. float data type
6. double data type
7. char data type
8. short data type

Java program to demonstrate different primitive data types.

```
class Primitive
{
   Public static void main(String args[])
   {
         boolean isJtpBest=true;
         boolean isCold = false;
         System.out.println(isJtpBest);
         System.out.println(isCold);
         byte a = 100;
         System.out.println(a);
         int num= 50000;
         System.out.println(num);
         long l = 7000000000L;
         System.out.println(l);
         float num = 5.75f;
         System.out.println(num);
         double num= 19.99d;
         System.out.println(num);
         char myChar= 'H';
         System.out.println(myChar);
         short num = 5000;
         System.out.println(num);
    }
}
```

Data Types in Python

Python is a dynamically typed language; hence we do not need to define the type of the variable while declaring it. The interpreter implicitly binds the value with its type.

```
a = 5
```

The variable **a** holds integer value five and we did not define its type. Python interpreter will automatically interpret variables **a** as an integer type.

Python enables us to check the type of the variable used in the program. Python provides us the **type()** function, which returns the type of the variable passed.

Python program to define the values of different data types and checking its type.

```
a=10
b="Hi Python"
c = 10.5
print(type(a))
print(type(b))
print(type(c))
```

Output:

```
<type 'int'>
<type 'str'>
<type 'float'>
```

Standard data types in python

Python provides various standard data types that define the storage method on each of them. The data types defined in Python are given below.

- Numbers
- Sequence type
- Boolean
- Set
- Dictionary

Numbers

Number stores numeric values. The integer, float, and complex values belong to a Python Numbers data-type. Python provides the **type()** function to know the data-type of the variable. Similarly, the **isinstance()** function is used to check an object belongs to a particular class.

Python creates Number objects when a number is assigned to a variable. For example;

Python program to demonstrate int, float and complex data types.

```
a = 5
```

```
print("The type of a", type(a))

b = 40.5
print("The type of b", type(b))

c = 1+3j
print("The type of c", type(c))
print(" c is a complex number", isinstance(1+3j,complex))
```

Output:

```
The type of a <class 'int'>
The type of b <class 'float'>
The type of c <class 'complex'>
c is complex number: True
```

Python supports three types of numeric data.

1. **int -** Integer value can be any length such as integers 10, 2, 29, -20, -150 etc. Python has no restriction on the length of an integer. Its value belongs to **int**
2. **Float -** Float is used to store floating-point numbers like 1.9, 9.902, 15.2, etc. It is accurate upto 15 decimal points.
3. **complex -** A complex number contains an ordered pair, i.e., x + iy where x and y denote the real and imaginary parts, respectively. The complex numbers like 2.14j, 2.0 + 2.3j, etc.

2. **Program examine array:**

An array is defined as the collection of similar type of data items stored at contiguous memory locations.

Array in C

C program to declare and traverse one dimensional array

```
#include<stdio.h>
int main()
{
  int i=0;
  int marks[5]={20,30,40,50,60};//declaration and initialization of array
  //traversal of array
  for(i=0;i<5;i++)
  {
        printf("%d \n",marks[i]);
   }
   return 0;
}
```

C program to declare and traverse two dimensional array

```
#include<stdio.h>
int main()
{
    int i=0,j=0;
    int arr[4][3]={{1,2,3},{2,3,4},{3,4,5},{4,5,6}};
    //traversing 2D array
     for(i=0;i<4;i++)
```

```
        {
                for(j=0;j<3;j++)
                {
                        printf("arr[%d] [%d] = %d \n",i,j,arr[i][j]);
                 }//end of j
         }//end of i
         return 0;
    }
```

***C program to sort the array of elements in ascending order using bubble sort method*.**

```
    #include<stdio.h>
    void main ()
    {
        int i, j,temp;
        int a[10] = { 10, 9, 7, 101, 23, 44, 12, 78, 34, 23};
        for(i = 0; i<10; i++)
        {
            for(j = i+1; j<10; j++)
            {
                if(a[j] > a[i])
                {
                    temp = a[i];
                    a[i] = a[j];
                    a[j] = temp;
                }
            }
        }
        printf("Printing Sorted Element List ...\n");
        for(i = 0; i<10; i++)
        {
            printf("%d\n",a[i]);
    }
}
```

Array in Java

Java array is an object which contains elements of a similar data type. Additionally, The elements of an array are stored in a contiguous memory location. It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array.

Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on.

Java Program to illustrate the use of declaration, instantiation

```
class Testarray1
{
  public static void main(String args[])
  {
          int a[]={33,3,4,5};//declaration, instantiation and initialization
          //printing array
          for(int i=0;i<a.length;i++)//length is the property of array
          System.out.println(a[i]);
  }
}
```

Output:

```
33
3
4
5
```

Java program to declare and traverse two dimensional array

```
class GFG
{
   public static void main(String[] args)
   {

      int[][] arr = { { 1, 2 }, { 3, 4 } };
```

```
        for (int i = 0; i < 2; i++)
        {
           for (int j = 0; j < 2; j++)
            {
              System.out.print(arr[i][j]);
            }
            System.out.println();
         }
      }
}
```

Java Program to demonstrate the addition of two matrices in Java

```
    class Testarray5
    {
       public static void main(String args[])
       {
                //creating two matrices
                int a[][]={{1,3,4},{3,4,5}};
                int b[][]={{1,3,4},{3,4,5}};

                //creating another matrix to store the sum of two matrices
                int c[][]=new int[2][3];

                //adding and printing addition of 2 matrices
                for(int i=0;i<2;i++)
                {
                        for(int j=0;j<3;j++)
                        {
                                c[i][j]=a[i][j]+b[i][j];
                                System.out.print(c[i][j]+" ");
                        }
                        System.out.println();//new line
                }
```

```
        }
    }
```

Output:

```
2 6 8
6 8 10
```

Java Program to multiply two matrices

```
public class MatrixMultiplicationExample
{
  public static void main(String args[])
  {
        //creating two matrices
        int a[][]={{1,1,1},{2,2,2},{3,3,3}};
        int b[][]={{1,1,1},{2,2,2},{3,3,3}};

        //creating another matrix to store the multiplication of two matrices
        int c[][]=new int[3][3];  //3 rows and 3 columns

        //multiplying and printing multiplication of 2 matrices
        for(int i=0;i<3;i++)
        {
                for(int j=0;j<3;j++)
                {
                        c[i][j]=0;
                        for(int k=0;k<3;k++)
                        {
                                c[i][j]+=a[i][k]*b[k][j];
                        }//end of k loop
                        System.out.print(c[i][j]+" ");  //printing matrix element
                }//end of j loop
                System.out.println();//new line
        }
  }
}
```

Output:

```
6 6 6
12 12 12
18 18 18
```

3.Programs examine expressions

An expression is a combination of operators and operands that is interpreted to produce some other value. In any programming language, an expression is evaluated as per the precedence of its operators. So that if there is more than one operator in an expression, their precedence decides which operation will be performed first.

Expressions in C

There are four types of expressions exist in C:

- Arithmetic expressions
- Relational expressions
- Logical expressions
- Conditional expressions

Each type of expression takes certain types of operands and uses a specific set of operators. Evaluation of a particular expression produces a specific value.

For example:

1. x = 9/2 + a-b;

The entire above line is a statement, not an expression. The portion after the equal is an expression.

Arithmetic Expressions

An arithmetic expression is an expression that consists of operands and arithmetic operators. An arithmetic expression computes a value of type int, float or double.

When an expression contains only integral operands, then it is known as pure integer expression when it contains only real operands, it is known as pure real expression, and when it contains both integral and real operands, it is known as mixed mode expression.

Evaluation of Arithmetic Expressions

The expressions are evaluated by performing one operation at a time. The precedence and associativity of operators decide the order of the evaluation of individual operations.

When individual operations are performed, the following cases can be happened:

- When both the operands are of type integer, then arithmetic will be performed, and the result of the operation would be an integer value. For example, 3/2 will yield 1 not 1.5 as the fractional part is ignored.
- When both the operands are of type float, then arithmetic will be performed, and the result of the operation would be a real value. For example, 2.0/2.0 will yield 1.0, not 1.
- If one operand is of type integer and another operand is of type real, then the mixed arithmetic will be performed. In this case, the first operand is converted into a real operand, and then arithmetic is performed to produce the real value. For example, 6/2.0 will yield 3.0 as the first value of 6 is converted into 6.0 and then arithmetic is performed to produce 3.0.

Let's understand through an example.

6*2/ (2+1 * 2/3 + 6) + 8 * (8/4)

Evaluation of expression	**Description of each operation**
6*2/(2+1 * 2/3 +6) +8 * (8/4)	An expression is given.
6*2/(2+2/3 + 6) + 8 * (8/4)	2 is multiplied by 1, giving value 2.
6*2/(2+0+6) + 8 * (8/4)	2 is divided by 3, giving value 0.
6*2/ 8+ 8 * (8/4)	2 is added to 6, giving value 8.
6*2/8 + 8 * 2	8 is divided by 4, giving value 2.
12/8 +8 * 2	6 is multiplied by 2, giving value 12.
1 + 8 * 2	12 is divided by 8, giving value 1.
1 + 16	8 is multiplied by 2, giving value 16.
17	1 is added to 16, giving value 17.

Relational Expressions

- A relational expression is an expression used to compare two operands.
- It is a condition which is used to decide whether the action should be taken or not.
- In relational expressions, a numeric value cannot be compared with the string value.

- The result of the relational expression can be either zero or non-zero value. Here, the zero value is equivalent to a false and non-zero value is equivalent to true.

Relational Expression	**Description**
x%2 = = 0	This condition is used to check whether the x is an even number or not. The relational expression results in value 1 if x is an even number otherwise results in value 0.
a!=b	It is used to check whether a is not equal to b. This relational expression results in 1 if a is not equal to b otherwise 0.
a+b = = x+y	It is used to check whether the expression "a+b" is equal to the expression "x+y".
a>=9	It is used to check whether the value of a is greater than or equal to 9.

C program to check the number is odd or even.

```
#include <stdio.h>
 int main()
{

    int x=4;
    if(x%2==0)
    {
        printf("The number x is even");
    }
    else
    printf("The number x is not even");
    return 0;
}
```

Output

```
The number x is even
```

Logical Expressions

- A logical expression is an expression that computes either a zero or non-zero value.
- It is a complex test condition to take a decision.

Let's see some example of the logical expressions.

Logical Expressions	**Description**
(x > 4) && (x < 6)	It is a test condition to check whether the x is greater than 4 and x is less than 6. The result of the condition is true only when both the conditions are true.
x > 10 || y <11	It is a test condition used to check whether x is greater than 10 or y is less than 11. The result of the test condition is true if either of the conditions holds true value.
! (x > 10) && (y = = 2)	It is a test condition used to check whether x is not greater than 10 and y is equal to 2. The result of the condition is true if both the conditions are true.

C program to perform operation on "&&" logical operator.

```
#include <stdio.h>
int main()
{
   int x = 4;
   int y = 10;
   if ( (x <10) && (y>5))
   {
      printf("Condition is true");
   }
   else
   printf("Condition is false");
   return 0;
}
```

Output

Condition is true

C program to perform operation on "||" logical operator.

```
#include <stdio.h>
int main()
{
   int x = 4;
   int y = 9;
   if ( (x <6) || (y>10))
   {
      printf("Condition is true");
   }
   else
   printf("Condition is false");
   return 0;
}
```

Output

```
Condition is true
```

Conditional Expressions

- A conditional expression is an expression that returns 1 if the condition is true otherwise 0.
- A conditional operator is also known as a ternary operator.

The Syntax of Conditional operator

Suppose exp1, exp2 and exp3 are three expressions.

exp1 ? exp2 : exp3

The above expression is a conditional expression which is evaluated on the basis of the value of the exp1 expression. If the condition of the expression exp1 holds true, then the final conditional expression is represented by exp2 otherwise represented by exp3.

C program to check the entered age is suitable to married or not.

```
#include<stdio.h>
#include<string.h>
int main()
{
    int age = 25;
    char status;
    status = (age>22) ? 'M': 'U';
    if(status == 'M')
    printf("Married");
    else
    printf("Unmarried");
    return 0;
}
```

Output

```
Married
```

Expressions in java

Operator in Java is a symbol that is used to perform operations. For example: +, -, *, / etc.

There are many types of operators in Java which are given below:

- Unary Operator,
- Arithmetic Operator,
- Shift Operator,
- Relational Operator,
- Bitwise Operator,
- Logical Operator,
- Ternary Operator and

- Assignment Operator.

Java Operator Precedence

Operator Type	**Category**	**Precedence**
Unary	postfix	*expr*++ *expr*--
	prefix	++*expr* --*expr* +*expr* -*expr* ~ !
Arithmetic	multiplicative	* / %
	additive	+ -
Shift	Shift	<< >> >>>
Relational	comparison	< > <= >= instanceof
	equality	== !=
Bitwise	bitwise AND	&
	bitwise exclusive OR	^
	bitwise inclusive OR	|
Logical	logical AND	&&
	logical OR	||
Ternary	ternary	? :

Assignment	assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Java Unary Operator Example: ++ and –

Java program to show increment and decrement operators

```
    public class OperatorExample
    {
        public static void main(String args[])
        {
                int x=10;
                System.out.println(x++);//10 (11)
                System.out.println(++x);//12
                System.out.println(x--);//12 (11)
                System.out.println(--x);//10
        }
    }
```

Output:

```
10
12
12
10
```

Java Unary Operator Example 2: ++ and --

```
public class OperatorExample
{
    public static void main(String args[])
    {
        int a=10;
        int b=10;
        System.out.println(a++ + ++a);//10+12=22
        System.out.println(b++ + b++);//10+11=21
     }
    }
```

Output:

```
1. 22
2. 21
```

Java Arithmetic Operator Example

Java program to demonstrate arithmetic operators

```
public class OperatorExample
{
    public static void main(String args[])
    {
        int a=10;
        int b=5;
        System.out.println(a+b);//15
        System.out.println(a-b);//5
        System.out.println(a*b);//50
        System.out.println(a/b);//2
        System.out.println(a%b);//0
    }
}
```

Output:

```
15
5
50
2
0
```

Java program to perform Arithmetic Expression

```
    public class OperatorExample
    {
        public static void main(String args[])
        {
                System.out.println(10*10/5+3-1*4/2);
        }
    }
```

Output:

```
21
```

Java Left Shift Operator

The Java left shift operator << is used to shift all of the bits in a value to the left side of a specified number of times.

Java program to demo Left Shift Operator

```
    public class OperatorExample
    {
        public static void main(String args[])
        {
                System.out.println(10<<2); //10*2^2=10*4=40
                System.out.println(10<<3); //10*2^3=10*8=80
                System.out.println(20<<2); //20*2^2=20*4=80
                System.out.println(15<<4); //15*2^4=15*16=240
        }
}
```

Output:

```
40
80
80
240
```

Java Right Shift Operator

The Java right shift operator >> is used to move the value of the left operand to right by the number of bits specified by the right operand.

Java program to demo Right Shift Operator

```
public OperatorExample
```

```
  {
        public static void main(String args[])
        {
                    System.out.println(10>>2);    //10/2^2=10/4=2
                    System.out.println(20>>2);    //20/2^2=20/4=5
                    System.out.println(20>>3);    //20/2^3=20/8=2
        }
  }
```

Output:

```
2
5
2
```

Java AND Operator

The logical && operator doesn't check the second condition if the first condition is false. It checks the second condition only if the first one is true.

The bitwise & operator always checks both conditions whether first condition is true or false.

Java program to demo Logical && and Bitwise &

```
  public class OperatorExample
  {
        public static void main(String args[])
        {
                    int a=10;
                    int b=5;
                    int c=20;
                    System.out.println(a<b&&a<c);        //false && true = false
                    System.out.println(a<b&a<c);         //false & true = false
        }
  }
```

Output:

```
false
false
```

Java OR Operator Example: Logical || and Bitwise |

The logical || operator doesn't check the second condition if the first condition is true. It checks the second condition only if the first one is false.

The bitwise | operator always checks both conditions whether first condition is true or false.

Java program to show OR Operator example: Logical || and Bitwise |

```
  public class OperatorExample
  {
      public static void main(String args[])
      {
              int a=10;
              int b=5;
              int c=20;
              System.out.println(a>b||a<c); //true || true = true
              System.out.println(a>b|a<c);  //true | true = true
              System.out.println(a>b||a++<c);      //true || true = true
              System.out.println(a);          //10 because second condition is not checked
              System.out.println(a>b|a++<c);       //true | true = true
              System.out.println(a);          //11 because second condition is checked
      }
 }
```

Output:

```
true
true
true
10
true
11
```

Java Ternary Operator

Java Ternary operator is used as one line replacement for if-then-else statement and used a lot in Java programming. It is the only conditional operator which takes three operands.

Java program Ternary Operator example

```
  public class OperatorExample
  {
      public static void main(String args[])
      {
              int a=2;
              int b=5;
              int min=(a<b)?a:b;
              System.out.println(min);
      }
 }
```

Output:

```
2
```

Java Assignment Operator

Java assignment operator is one of the most common operators. It is used to assign the value on its right to the operand on its left.

Java program to demonstrate assignment operator example

```
  public class OperatorExample
  {
      public static void main(String args[])
      {
              int a=10;
              int b=20;
              a+=4;//a=a+4 (a=10+4)
              b-=4;//b=b-4 (b=20-4)
              System.out.println(a);
              System.out.println(b);
      }
  }
```

Output:

```
14
16
```

Expressions in Python

Python program to perform arithmetic operations.

```
x = 40
y = 12

add = x + y
sub = x - y
pro = x * y
div = x / y

print(add)
print(sub)
print(pro)
print(div)
```

output:

```
52

28

480

3.3333333333333335
```

Python program to show Integral Expressions

```
a = 13
b = 12.0

c = a + int(b)
print(c)
```

output:

```
25
```

Python program to perform Relational Expressions

```
a = 21
b = 13
c = 40
d = 37

p = (a + b) >= (c - d)
print(p)
```

output:

```
True
```

Python program to perform logical expressions

```
P = (10 == 9)
Q = (7 > 5)

# Logical Expressions
R = P and Q
S = P or Q
T = not P

print(R)
print(S)
print(T)
```

output:

```
False
True
True
```

Python program to perform multi-operator expression

```
a = 10 + 3 * 4
print(a)
```

```
b = (10 + 3) * 4
print(b)

c = 10 + (3 * 4)
print(c)
```

output:

```
22

52

22
```

4. Programs examine subprograms:

A subprogram is a sequence of instructions whose execution is invoked from one or more remote locations in a program, with the expectation that when the subprogram execution is complete, execution resumes at the instruction after the one that invoked the subprogram. In high-level languages, subprograms are also called subroutines, procedures, and functions. In object-oriented languages, they are usually called methods or constructors. In most modern high-level languages, subprograms can have parameters, local variables, and returned values.

Function in C

The syntax of creating function in c language is given below:

```
return_type function_name(data_type parameter...)
{
//code to be executed
}
```

Example without return value:

```
void hello()
{
printf("hello c");
}
```

Example with return value:

```
int get(){
return 10;
}


float get(){
return 10.2;
}
```

C programs for Function without argument and return value

Example 1

```
#include<stdio.h>
void printName();
void main ()
{
   printf("Hello ");
   printName();
}
void printName()
{
   printf("Jazan University");
}
```

Output

```
Hello Jazan University
```

Example 2

```
#include<stdio.h>
void sum();
void main()
{
   printf("\nGoing to calculate the sum of two numbers:");
   sum();
}
void sum()
{
   int a,b;
   printf("\nEnter two numbers");
   scanf("%d %d",&a,&b);
```

```
    printf("The sum is %d",a+b);
}
```

Output

```
Going to calculate the sum of two numbers:

Enter two numbers 10
24

The sum is 34
```

C program for Function without argument and with return value

Example 1

```
#include<stdio.h>
int sum();
void main()
{
    int result;
    printf("\nGoing to calculate the sum of two numbers:");
    result = sum();
    printf("%d",result);
}
int sum()
{
    int a,b;
    printf("\nEnter two numbers");
    scanf("%d %d",&a,&b);
    return a+b;
}
```

Output

```
Going to calculate the sum of two numbers:

Enter two numbers 10
```

```
24
The sum is 34
```

C program to calculate the area of the square using function

```
#include<stdio.h>
int sum();
void main()
{
    printf("Going to calculate the area of the square\n");
    float area = square();
    printf("The area of the square: %f\n",area);
}
int square()
{
    float side;
    printf("Enter the length of the side in meters: ");
    scanf("%f",&side);
    return side * side;
}
```

Output

```
Going to calculate the area of the square
Enter the length of the side in meters: 10
The area of the square: 100.000000
```

C program for function with argument and without return value

```
#include<stdio.h>
void sum(int, int);
void main()
{
    int a,b,result;
    printf("\nGoing to calculate the sum of two numbers:");
    printf("\nEnter two numbers:");
    scanf("%d %d",&a,&b);
    sum(a,b);
}
void sum(int a, int b)
{
    printf("\nThe sum is %d",a+b);
}
```

Output

```
Going to calculate the sum of two numbers:

Enter two numbers 10
24

The sum is 34
```

C program for function with argument and with return value

```
#include<stdio.h>
int sum(int, int);
void main()
{
    int a,b,result;
    printf("\nGoing to calculate the sum of two numbers:");
    printf("\nEnter two numbers:");
    scanf("%d %d",&a,&b);
    result = sum(a,b);
    printf("\nThe sum is : %d",result);
}
int sum(int a, int b)
{
    return a+b;
}
```

Output

```
Going to calculate the sum of two numbers:
Enter two numbers:10
20
```

```
The sum is : 30
```

Functions in Java

The **method in Java** is a collection of instructions that performs a specific task. It provides the reusability of code. We can also easily modify code using **methods.**

There are two types of methods in Java:

- Predefined Method
- User-defined Method

Predefined Method

In Java, predefined methods are the method that is already defined in the Java class libraries is known as predefined methods. It is also known as the **standard library method** or **built-in method**. We can directly use these methods just by calling them in the program at any point. Some pre-defined methods are **length(), equals(), compareTo(), sqrt(),** etc. When we call any of the predefined methods in our program, a series of codes related to the corresponding method runs in the background that is already stored in the library.

```
public class Demo
{
    public static void main(String[] args)
    {
            // using the max() method of Math class
            System.out.print("The maximum number is: " + Math.max(9,7));
    }
}
```

Output:

```
The maximum number is : 9
```

User-defined Method

The method written by the user or programmer is known as **a user-defined** method. These methods are modified according to the requirement.

How to Create a User-defined Method

Let's create a user defined method that checks the number is even or odd. First, we will define the method.

Java program function to check the number is odd or even using runtime input

```
import java.util.Scanner;
public class EvenOdd
{
    public static void main (String args[])
    {
            //creating Scanner class object
            Scanner scan=new Scanner(System.in);
            System.out.print("Enter the number: ");
            //reading value from user
            int num=scan.nextInt();
            //method calling
            findEvenOdd(num);
    }
    //user defined method
    public static void findEvenOdd(int num)
    {
            //method body
            if(num%2==0)
                    System.out.println(num+" is even");
            else
                    System.out.println(num+" is odd");
        }
}
```

Output 1:

```
Enter the number: 12
12 is even
```

Output 2:

```
Enter the number: 99
99 is odd
```

Java program to add two integers in a function and return the result to the calling function

```
public class Addition
{
    public static void main(String[] args)
    {
            int a = 19;
            int b = 5;
            //method calling
            int c = add(a, b);   //a and b are actual parameters
            System.out.println("The sum of a and b is= " + c);
    }
    //user defined method
    public static int add(int n1, int n2)   //n1 and n2 are formal parameters
    {
            int s;
            s=n1+n2;
            return s; //returning the sum
    }
}
```

Output:

```
The sum of a and b is= 24
```

Static Method

A method that has static keyword is known as static method. In other words, a method that belongs to a class rather than an instance of a class is known as a static method. We can also create a static method by using the keyword **static** before the method name.

The main advantage of a static method is that we can call it without creating an object. It can access static data members and also change the value of it. It is used to create an instance method. It is invoked by using the class name. The best example of a static method is the **main()** method.

Example of static method

Display.java

```
public class Display
{
    public static void main(String[] args)
    {
            show();
    }
    static void show()
    {
            System.out.println("It is an example of static method.");
    }
}
```

Output:

```
It is an example of a static method.
```

Instance Method

The method of the class is known as an **instance method**. It is a **non-static** method defined in the class. Before calling or invoking the instance method, it is necessary to create an object of its class. Let's see an example of an instance method.

Java program to demonstrate instance method

```
public class InstanceMethodExample
{
```

```
                int s;
                public static void main(String [] args)
                {
                        //Creating an object of the class
                        InstanceMethodExample obj = new InstanceMethodExample();
                        //invoking instance method
                        System.out.println("The sum is: "+obj.add(12, 13));
                }
                //user-defined method because we have not used static keyword
                public int add(int a, int b)
                {
                        s = a+b;
                        //returning the sum
                        return s;
                }
    }
```

Output:

```
The sum is: 25
```

Functions in Python

A function is a collection of related assertions that performs a mathematical, analytical, or evaluative operation.

The functions are broad of two types, user-defined and built-in functions. It aids in keeping the software succinct, non-repetitive, and well-organized.

Syntrax:

```
def name_of_function( parameters ):
    """This is a docstring"""
    # code block
```

The following elements make up define a function, as seen above.

- The beginning of a function header is indicated by a keyword called def.
- name_of_function is the function's name that we can use to separate it from others. We will use this name to call the function later in the program. The same criteria apply to naming functions as to naming variables in Python.
- We pass arguments to the defined function using parameters. They are optional, though.
- The function header is terminated by a colon (:).
- We can use a documentation string called docstring in the short form to explain the purpose of the function.
- The body of the function is made up of several valid Python statements. The indentation depth of the whole code block must be the same (usually 4 spaces).
- We can use a return expression to return a value from a defined function.

Python program to find out square of a number using function

```
def square( num ):
    return num**2
object_ = square(9)
print( "The square of the number is: ", object_ )
```

Output:

```
The square of the number is:  81
```

Python program to define a function to return length of a string

```
# Defining a function
def a_function( string ):
    return len(string)
# Calling the function we defined
print( "Length of the string Functions is: ", a_function( "Functions" ) )
print( "Length of the string Python is: ", a_function( "Python" ) )
```

Output:

```
Length of the string Functions is:  9
Length of the string Python is:  6
```

Python program to demo for passing one and two arguments for same function

```
# Python code to demonstrate the use of default arguments
# defining a function
def function( num1, num2 = 40 ):
    print("num1 is: ", num1)
    print("num2 is: ", num2)



# Calling the function and passing only one argument
print( "Passing one argument" )
function(10)

# Now giving two arguments to the function
print( "Passing two arguments" )
function(10,30)
```

Output:

```
Passing one argument
num1 is:  10
num2 is:  40
Passing two arguments
num1 is:  10
num2 is:  30
```

Python code to demonstrate the use of return statements

```
# Defining a function with return statement
def square( num ):
   return num**2

# Calling function and passing arguments.
print( "With return statement" )
print( square( 39 ) )

# Defining a function without return statement
def square( num ):
    num**2

# Calling function and passing arguments.
print( "Without return statement" )
print( square( 39 ) )
```

Output:

```
With return statement
1521
Without return statement
None
```

5.Programs examine Scope

Scope of Variable

Each variable is defined and can be used within its scope and determines that wherein the program this variable is available to use. The scope means the lifetime of that variable. It means the variable can only be accessed or visible within its scope.

Scope in C

The scope of variables can be defined with their declaration, and variables are declared mainly in two ways:

- **Global Variable:** Outside of all the functions
- **Local Variable:** Within a function block:

What is a Global Variable?

- Global variables are those variables which are declared outside of all the functions or block and can be accessed globally in a program.
- It can be accessed by any function present in the program.
- Once we declare a global variable, its value can be varied as used with different functions.
- The lifetime of the global variable exists till the program executes. These variables are stored in fixed memory locations given by the compiler and do not automatically clean up.
- Global variables are mostly used in programming and useful for cases where all the functions need to access the same data.

Example:

```
#include<stdio.h>
int a=50, b=40;
void main()
{
  printf("a = %d and b=%d",a,b);
}
```

In the above example, a and b are the global variables.

What is a Local Variable?

- **Variables that are declared within or inside a function block are known as Local variables.**
- These variables can only be accessed within the function in which they are declared.
- The lifetime of the local variable is within its function only, which means the variable exists till the function executes. Once function execution is completed, local variables are destroyed and no longer exist outside the function.
- The reason for the limited scope of local variables is that local variables are stored in the stack, which is dynamic in nature and automatically cleans up the data stored within it.
- But by making the variable static with "static" keyword, we can retain the value of local variable.

Example:

```
#include<stdio.h>
void main()
{
  int x=50, y=40;
  printf("x = %d and y=%d",x, y);
}
```

In the above example, we have declared x and y two variables inside the main function. Hence these are local variables.

Local vs Global in C

Example-1:

```
#include<stdio.h>

   // Global variables
   int a;
   int b;
   int Add()
   {
           return a + b;
   }

   int Mul()

   {
            int c=10;  //Local Variable
            int d=20;   //Local Variable
           return c*d;
   }
   void main()
   {
           int Ans1, Ans2, c=30;// Local variable
           a = 50;
           b = 70;

            Ans1 = Add();
            Ans2= Mul();

           printf("The addition result is: %d\n",Ans1);
           printf("The Multiplication result is: %d\n",Ans2);
           printf("%d\n", c);
   }
```

Scope of goto

The goto statement is a jump statement which is sometimes also referred to as unconditional jump statement. The goto statement can be used to jump from anywhere to anywhere within a function.

```
// C program to check if a number is
// even or not using goto statement
#include <stdio.h>
```

```
// function to check even or not
void checkEvenOrNot(int num)
{
   if (num % 2 == 0)
      // jump to even
      goto even;
   else
      // jump to odd
      goto odd;

even:
   printf("%d is even", num);
   // return if even
   return;
odd:
   printf("%d is odd", num);
}

int main()
{
   int num = 26;
   checkEvenOrNot(num);
   return 0;
}
```

Scope in Python

Local Vs. Global in Python

```
v1 = "Hey, I am Global Variable."   #globalvariable
def func1():
   v2="Hey, I am Local Variable." #localvariable
   print(v2)
func1()    #calling func1

def func2():
   print(v1)
```

```
func2()    #callin func2
```

Output:

```
Hey, I am a Local Variable
Hey, I am Global Variable
```

In the above program, we have taken one global variable v1 and one local variable v2. Since v1 is global, it can be easily accessed in any function, and v2 is local; it is used only within its declared function. But if we try to use v1 in func1, it will give an error. Let's see the below example:

Example-2

```
v1 = "Hey, I am Global Variable"   #globalvariable
def func1():
   v2="Hey, I am Local Variable" #localvariable
   print(v2)
   print(v1)
func1()    #calling func1

def func2():
   print(v1)
   print(v2)
func2()    #callin func2
```

If we try accessing v1, it can easily be accessed in fun1 and func2. But if we try to access v2 outside its function, which means in func2, it will give the runtime error. We will get the below output after executing the above code:

Runtime Error:

```
NameError: global name 'v2' is not defined
```

Scope in Java

Local Vs. Global Variable in Java

In Java, there is no concept of global variables; since Java is an Object-oriented programming language, everything is a part of the Class. But if we want to make a variable globally accessible, we can make it static by using a **static** Keyword.

```
class Demo
{
        // static variable
        static int a = 10;
        // non-static or local variable
        int b = 20;
}

public class Main
 {
         public static void main(String[] args)
         {
                 Demo obj = new Demo();
                // accessing the non-static variable
                 System.out.println("Value of non-static variable is: " + (obj.b));
                // accessing the static variable
                 System.out.println("Value of static variable is:" + (Demo.a));
        }
}
```

Output:

```
Value of non-static variable is: 20
Value of static variable is:10
```

In the above program, we have used one local variable or non-static variable and one static variable. The local variable can be accessed by using the object of the Demo class, whereas the static variable can be accessed using the name of the class.

```
// in java, jump into/out of the block


Class Test
{
  Public static void main(String args[])
  {
        First:
        {
                Second:
                {
                        Third:
                        {
                                System.out.println("Third Block");
                                If(true)
                                {
                                        Break second;
                                }
                        }
                        System.out.println("Second block");
                }
                System.out.println("First Block");
        }
  }
}
```