Functions

```
static/non-static return_type name (arguments){
    // body of the method/Function;
    `
    `
    `
    `
    return statement;
}
```

▼ Take input of 2 numbers and print the sum

```
import java.util.Scanner;

public class twoNumSum{
    public static void main(String[] args) {
        Scanner in=new Scanner(System.in);
        System.out.print("Enter the first number:");
        int first=in.nextInt();
        System.out.print("Enter the second number:");
        int sec=in.nextInt();
        int sum= first+sec;
        System.out.println("The sum of 2 numbers are:"+sum

    }
}
```

❓ WHAT if i need to do the same operation for N times ??

```
import java.util.Scanner;

public class twoNumSum{
    public static void main(String[] args) {
        sum();
    }

    static void sum(){
        Scanner in=new Scanner(System.in);
        System.out.print("Enter the first number:");
        int first=in.nextInt();
        System.out.print("Enter the second number:");
        int sec=in.nextInt();
        int sum= first+sec;
        System.out.println("The sum of 2 numbers are:"+sum
    }
}
```

Return Type

- Data type of anything that is getting as output from the method/Function.

```
import java.util.Scanner;

public class twoNumSum{
    public static void main(String[] args) {
        int answer=sum();
        System.out.println("Sum of 2 numbers are:"+answer);

    }
    static int sum(){
        Scanner in=new Scanner(System.in);
        System.out.print("Enter the first number:");
        int first=in.nextInt();
        System.out.print("Enter the second number:");
        int sec=in.nextInt();
        int sum= first+sec;
        return sum;

        // anything outside the return will not be EXECUTED..
        // return states that method is ended
    }
}
```

Return a String

```
import java.util.Scanner;

public class greetMethod {
    public static void main(String[] args) {
        String message=greet();
        System.out.println(message);
    }

    static String greet(){
        String greeting = "Good Morning";
        return greeting;
    }
}
```

❓ Passing the value of Numbers while calling a method without having Scanner multiple times inside the method .

```
//Sum of 2 num
import java.util.Scanner;

public class sumPara {
    public static void main(String[] args) {
        int answer=sum(10,20);
        System.out.println("Sum of given 2 numbers are:"+answ

    }

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

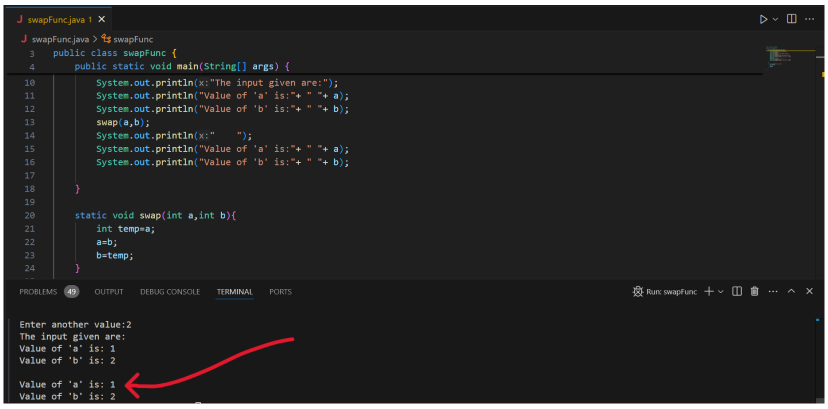
```


//String
import java.util.Scanner;

public class greetMethod {
    public static void main(String[] args) {
        System.out.print("Enter your name:");
        Scanner in=new Scanner (System.in);
        String your_name=in.next();
        String message=myGreet(your_name);
        System.out.println(message);
    }

    static String myGreet(String name){
        String msg= "Hey"+ " " +name+ " "+ "Good Morning"+ "
        return msg;
    }
}
```

▼ Swap numbers



```java
public class swapFunc {
    public static void main(String[] args) {
        System.out.println(x:"The input given are:");
        System.out.println("Value of 'a' is:"+ " "+ a);
        System.out.println("Value of 'b' is:"+ " "+ b);
        swap(a,b);
        System.out.println(x:"    ");
        System.out.println("Value of 'a' is:"+ " "+ a);
        System.out.println("Value of 'b' is:"+ " "+ b);

    }

    static void swap(int a,int b){
        int temp=a;
        a=b;
        b=temp;
    }
```

```
Enter another value:2
The input given are:
Value of 'a' is: 1
Value of 'b' is: 2

Value of 'a' is: 1
Value of 'b' is: 2
```

Change valid in this scope only. →method in swap

We see that the original values are not being changed even after swaping the values in the method

> ### ~~ TO REMEMBER ~~
>
> In Java, all methods/functions use pass-by-value by default. This is a fundamental characteristic of the language and applies to both primitive data types and objects. However, it's important to understand that:
>
> ### *Pass-by-Value vs Pass-by-Reference*
>
> 1. **Pass-by-Value**
>   - In pass-by-value, a copy of the variable's value is passed to the function. (copy of the actual value is passed to the method.)
>   - Changes made to the parameter inside the function do not affect the original variable.
>   - Java uses pass-by-value for primitive types (int, float, char, Byte etc.).
> 2. **Pass-by-Reference → Do not exist in JAVA**
>   - In pass-by-reference, the memory address of the variable is passed to the function.(copy of the

> reference to the object is passed to the method.)
>
> - Changes made to the parameter inside the function affect the original variable.
> - Java uses pass-by-value for object references, which can sometimes behave like pass-by-reference. (This can sometimes appear to behave like pass-by-reference, but it's still technically pass-by-value.)
> - The method receives a copy of the reference, not the original reference itself.
> - You can modify the object's internal state through this copied reference, but you cannot make the reference itself point to a different object.

▼ Example

```
import java.util.Arrays;

public class chnageValue {
    public static void main(String[] args) {
        String name="AISHWARYA";
        changeName(name);
        System.out.println(name);

    }
    static void changeName(){
        name="WHISKY"
    }

}
```

String ~~name~~ will not be changed even after calling the function to change the value of a ref variable. Because a new obj is created under a class.

```
import java.util.Arrays;

public class chnageValue {
    public static void main(String[] args) {
        int[] arr={1, 2, 3, 4, 5};
        change(arr);
        System.out.println(Arrays.toString(arr));

    }
    static void change(int[] num){
        num[0]=99;
    }

}
```

While in the arr, the existing arr is modified. No new obj has been created. This changes the original Value.

Scoping

- Access of vaiables.

```
//Method Scoping or Function scoping
public class Scope{
    public static void main(String[] args){
        int a=10;
        int b=20;
        System.out.println(num); //This gives an error. 'num'
    }
    static void random(){
        int num=67;
        System.out.println(num); //num can be accessed here
    }
// this also holds good for arguments as well.
}
```

```
//Block Scoping
public class Scope{
    public static void main(String[] args){
        int a=10;
        int b=20;
        {
            int a=90; // this gives an error too.  The value o
            a=80; // this seems right because the value of the
            int c=99; //values initialised remains in this blo
        }
        System.out.println(c); // throws an error
    }
```

```
//Scoping in for loop
public class Scope{
    public static void main(String[] args){
        for(int i=0;i<5;i++){
            System.out.println(i);
            int a=3;
        }
        System.out.println(i); // Gives a error
        int a=10; //gives an error
        a=4; // this doesnt throw an error, THIS IS UPDATING
}
```

Shadowing

```
// variable in higher level will be hided/shadowed. Lower leve
public class Scope{
    static int a=9; //acessible inside the entire class 'Scope
    public static void main(String[] args){
            System.out.println(x); // No error
            int a=70;
            System.out.println(x); // this prints 70 and prev
            random();
    }
    static void random(){
        System.out.println(x);
    }

    //This code snippet prints: 9 70 9
```

```
// variable in higher level will be hided/shadowed. Lower lev
public class Scope{
    static int a=9; //acessible inside the entire class 'Scop
    public static void main(String[] args){
            System.out.println(x); // No error
            int a;
            System.out.println(x); // gives an error
            /*SCOPE BEGINS WHEN THE VARIABLE IS INITIALISED*/
            a=80;
            System.out.println(x); // this prints 70 and prev
            random();
    }
    static void random(){
        System.out.println(x);
    }

    //This code snippet prints: 9 70 9

    /*SHAWDOWING DOES NOT WORK FOR METHODS*/
```

Variable Arguments (VarArgs)

- When the inputs are unknown

```
public class Scope{
    public static void main(String[] args){
        fun(2,3,4,5,6,7,8,9,...)// Can take any num of argume

    }
    static void random(int ...v){ //takes as array of integer
        System.out.println(Arrays.toString(v);
    }
```

```
//Mix of Args
public class Scope{
    public static void main(String[] args){
        fun(2,3,"Aish","Whisky","xyz"...)// Can take any num

    }
    static void random(int a ,int b, String ...v){ //takes as
        /*VarArgs should be given always at the end*/
        System.out.println(Arrays.toString(v);
    }

    /*here, '2' is taken as parameter for arg 'a' and
    '3' is taken as parameter for arg 'b'*/
```

Function Overloading

- Consider, there are 2 Fxns of same name (might be diff return type) → this can exist when PARAMETERS are DIFFERENT

```
public class Scope{
    public static void main(String[] args){
        rdm(67);
        rdm("Aish");
    }

    static void rdm(int a){
        System.out.println(a);
    }

    static void rdm(String name){
        System.out.println(name);
    }
```

QUESTION

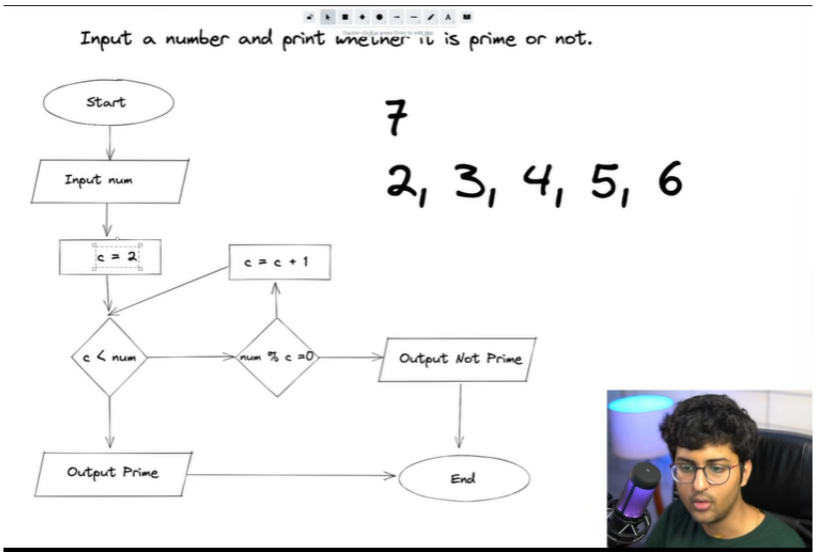
▼ Prime Numbers

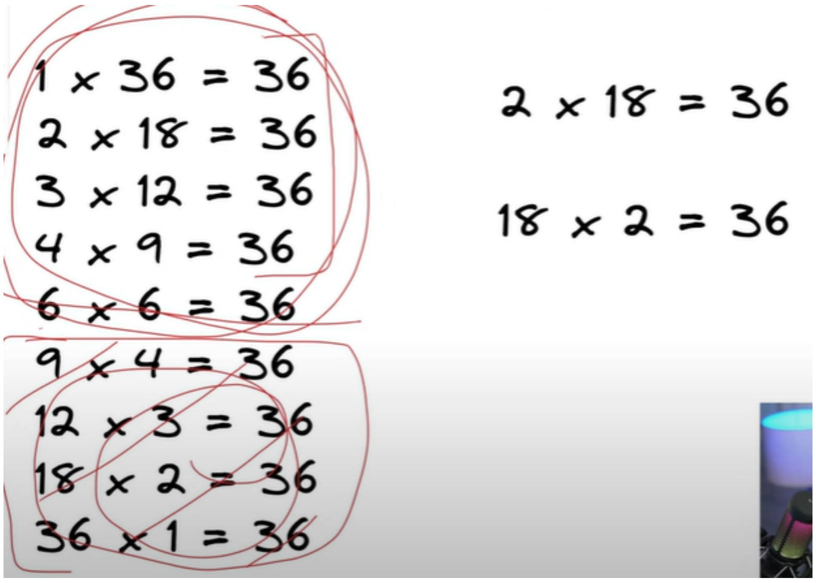
```
import java.util.Scanner;

public class Prime {
    public static void main(String[] args) {
        Scanner in=new Scanner(System.in);
        System.out.print("Enter a number:");
        int n=in.nextInt();
        boolean ans=isprime(n);
        System.out.println(ans);
    }
    static boolean isprime(int n){
       if(n<=1){
        return false;
       }
       int c=2;
       while(c * c <=n){ // or it can be while(c<n){
       /* Refer the pic*/
        if(n%c==0){
            return false;
        }
        c++; //inc and check for next number

        /*say if the n=9 ,
        i need to check if the numbers
        btw 2 and number <9 (2, 3, 4, 5,
        6, 7 and 8) will divide 9 or no).
        Now '3' divides 9. hence,
        its not a prime.
        PRIME ACTUALLY MEANS TO BE
        DIVIDED BY '1' and 'number itself'.
        Not by any other number*/
       }
       return (c* c>n);

    }
}
```





▼ Amstrong number [Cube of each digit in a number and sum of these giving a same number]

```
import java.util.Scanner;

public class armstrongNum {

    public static void main(String[] args) {
        // Scanner in = new Scanner(System.in);
        // System.out.print("Enter a 3 digit number: ");
        // int number = in.nextInt();
        // System.out.println(isArmstrong(number));

        // Loop through all 3-digit numbers
        for (int i = 100; i < 1000; i++) {
            if (isArmstrong(i)) {
                System.out.print(i + " ");
            }
        }
    }

    public static boolean isArmstrong(int n) {
        int original = n;  // Save the original number for
        int sum = 0;

        // Calculate the sum of cubes of digits
        while (n > 0) {
            int rem = n % 10;
            sum += rem * rem * rem;
            n /= 10;
        }

        // Compare the sum with the original number
        return sum == original;
    }
}
```