Syntax of function :

[Access Specifier] [Access Modifier] <return type / void> <function name> (parameter list)
{
Body of function
}

public static void main (String args[])
{
}

* Access Specifier

(a) **Public** : Public data and methods can be accessed everywhere within as well as outside the class in which they were declared.

It can be inherited.

(b) **Protected** : Protected data and methods can be accessed everywhere within their class, other classes in their package and sub classes in other packages.

It can be inherited.

Packages → class → methods / data members

(c) default (friendly / Package): Only in the same package

(d) Private: Private data & member methods can only be accessed in the class in which they they were declared.
It is the most secure access specifier.
It cannot be inherited.

* Access Modifiers :

(i) **Static :** Any static method or variable is of class type and it can be directly called or accessed within any function in the same class without help of any object.

A non static function or variable is required to be called through object from any static function of the class.

(ii) final

Eg:

```
class fun
{
public static void calc()
{
int a = 10;
int b = 20;
Sopln(a+b);
}
public void acc()
{
calc();
}
public static void main(String args[])
{
fun obj = new fun();
obj.acc();
}
}
```

30

* Formal parameters ⇒ The parameters, appear in the function definition statement are called formal parameters.

Function definition statement

Eg:

```
public static void Sum (int a, int b)
{ int s = a + b;
  System.out.println(s);
}
```

17 → a, 16 → b

a + b → 33

→ 33

Formal Parameters

* Actual Parameter ⇒ The parameters which appear in function calling statements are called actual parameters.

Eg:
```
public static void main (String args[])
{
    int x = 17;
    int y = 16;
    sum (x, y);
    sum (10, 20);
}
```

Actual parameters

sum (x, y); → 33

sum (10, 20); → 30

* Return data type :

It tells about the data type of the value that has to be returned to the calling function.

Eg:

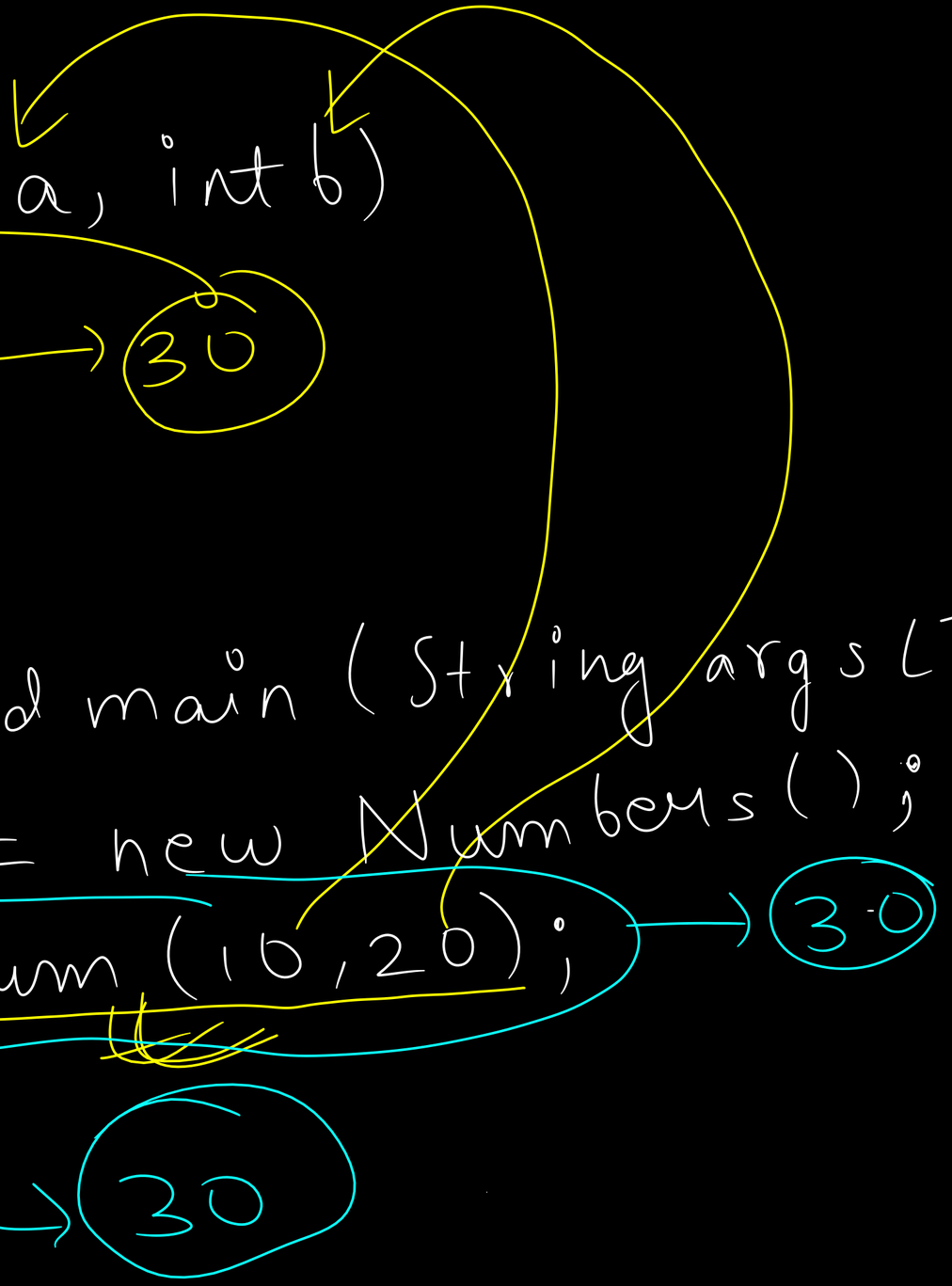
```
int sum( )
{

}
```

*return : It is used to return a value to the calling function.

*Note : A method cannot return more than one value at a time.
It sends the control back to the calling function.

```
Class Numbers
{
    int sum(int a, int b)
    {
        int s = a + b;
        return s;
    }
    public static void main(String args[])
    {
        Numbers obj = new Numbers();
        int s = obj.sum(10,20);
        Sop ln(s);
    }
```



Q WAP to print the factorial of 5.

factorial of 2 = 2 × 1
" " 3 = 3 × 2 × 1
factorial of 1 = 1

5 × 4 × 3 × 2 × 1 = 120

Soln ⇒

```
class factorial
{
  psvm(---)
  { int f = 1;
    for(int i = 1; i <= 5; i++)
    {
      f = f * i;
    }
    Sopln(f);
  }
```

	f
1	
2 × 1 =	2
3 × 2 =	6
4 × 6 =	24
5 × 24 =	120

```
class factors
{
    psvm (String args[])
    {
        int c=0;
        for(int i=1; i<=10; i++)
        {
            if(10%i==0)
            c++;
        }
    }
}
```

Q Design a class Numbers as follows:

void factorial(): to input a no. through Keyboard & print its factorial.

Void prime(): to input a no. through Keyboard & check & print whether its a prime no. or not.

Also write main method to input 10 no.s using above functions & print factorial of each no. & check wether it is prime or not.

```
import java.util.Scanner;
class Numbers
{
 void factorial()
 {
   Scanner obj= new Scanner(System.in);
   Sopln("Enter a no");
   int n= obj.nextInt();
   int f=1
   for(int i=1; i<=n; i++)
   {
     f=f*i;
   }
   Sopln(f);
 }
void prime()
{
 Scanner obj= new Scanner(System.in);
 int n= obj.nextInt();
 int c= 0;
 for(int i=1; i<=n; i++)
 {
  if(n%i==0)
  c++;
 }
 if(c==2)
 Sopln(n+" is a prime no");
 else
 Sopln(n+" is not a prime no.");
}
public static void main(String args[])
{ Scanner obj= new Scanner(System.in);
  Numbers ob= new Numbers();
  for(int i=1; i<=10; i++)
  { ob.factorial();
    ob.prime();
  }
}
```

H/w will be sent by tomorrow!

Thank You!