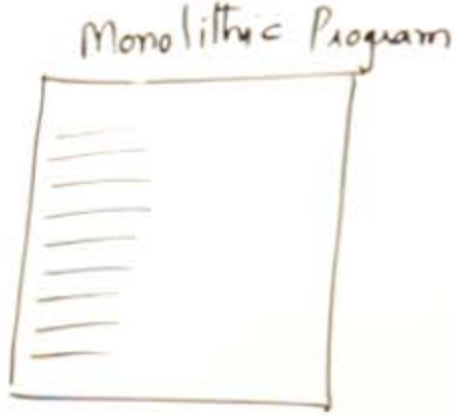
MeThods

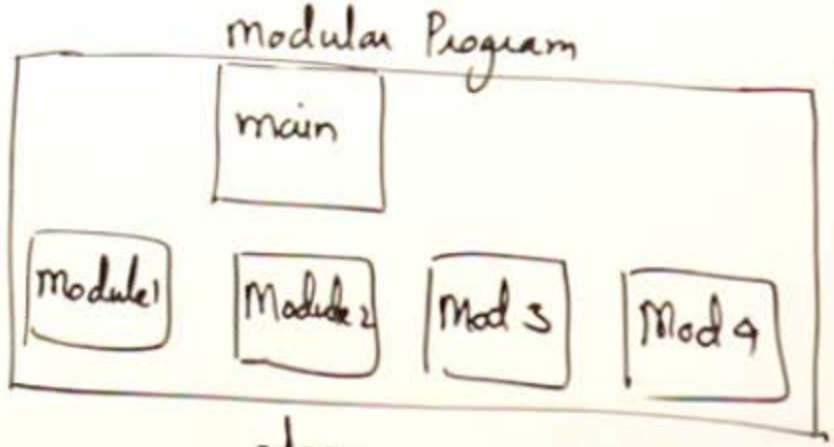
length( )
substring( )
IndexOf( )
equals( )

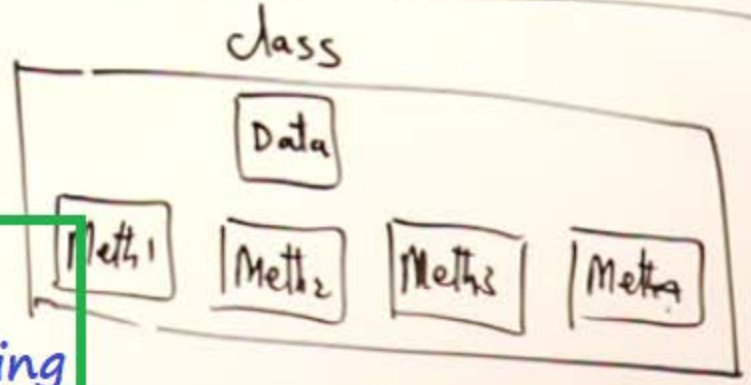
Some pre-defined methods.

Methods are members of classes which provide functionality for classes.

1. What are Methods
2. How to write a Method
3. Parameter Passing
4. Method Return type
5. Method Overloading
6. Varargs







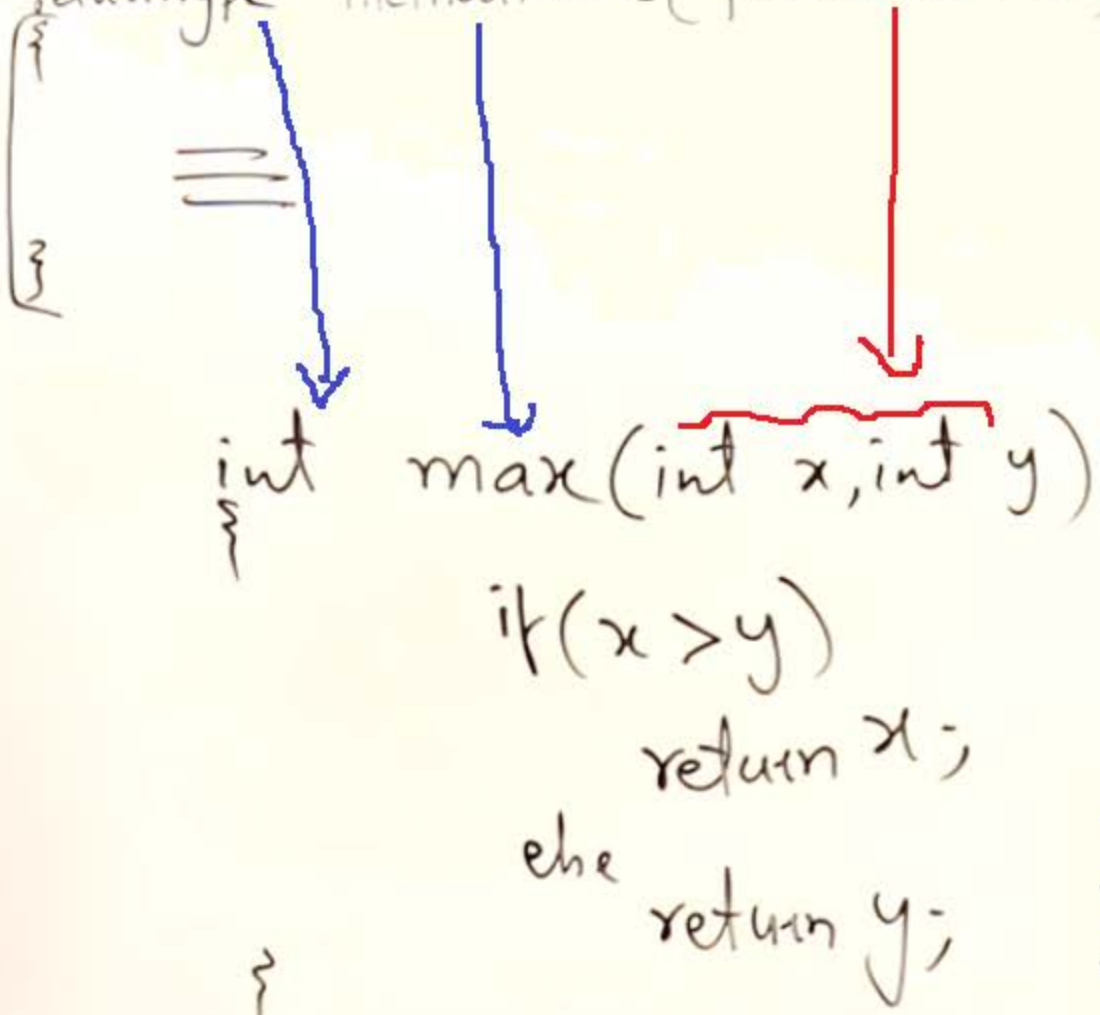
→ Modular program is used in "c" or before.

→ Class is mainly used in object oriented programming such as java and cpp(or c++).

Methods

signature

returnType methodName(parameter List)



→ Parameter is optional or use as per given condition.

→ method name is anything which defined by user.

→ return type mai wo data type likhta hai jo "method" hona ka baad hma milna wala hota hai. But agar koi data nahi milta hai then ham void ka use karta hai.

→ final type of data type(i.e. jo answer last mai aata hai) ya to return statement ka sath likhta hai. Aur aagar "void" hai to kuch bhi return nahi hota hai.

MeThods

→ method ko generally ham class kai andar hi define karta hai. Also, "public static void main (String args[ ])" ka bahar hi karta hai.

```
class Test
{
    static int max(int x,int y)
    {
        if(x>y)
            return x;
        else
            return y;
    }
    public static void main(String args[])
    {
        int a=10,b=15,c;
        c=max(a,b);
        S.o.p(c);
    }
}
```

ya method hai

15

----> → function ko call karna ka ek method hai.

MeThods

```
class Test
{
                                  formal param
                                  ↓
    static int max(int x, int y)
    {
        x++;
        if(x>y)
            return x;
        else
            return y;
    }

    public static void main(String args[])
    {
        int a=10, b=15, c;
            =max(a,b);
        S.o.p(c);
    }
}
```

actual Params

x: ~~10~~ 11 | y: 15

a: 10 | b: 15 | c: 15

→ Jaisa hmna pvsm mai user sai input a&b liya hai and then methods call kiya ha. To agar method mai koi bhi value update ho usesai 'a' & 'b' ki value ko koi farak nahi parta hai.

→ Bas as a result variable 'c' ki value update hogi.

```java
package methodpractice;

public class MethodPractice
{

    static int max(int x,int y)
    {
        if(x>y)
            return x;
        else
            return y;

    }

    public static void main(String[] args)
    {
        int a=10,b=15;

        System.out.println(max(a,b));

    }
}
```

→ Static mth. sirf method ko hi call kar sakta hai.
(Reason: next kuch slide mai hai)

```
compile:
run:
15
BUILD SUCCESSFUL (total time: 0 seconds)
```

```java
public class MethodPractice
{

    int max(int x,int y)
    {
        if(x>y)
            return x;
        else
            return y;

    }

    public static void main(String[] args)
    {
        int a=10,b=15;

        MethodPractice mp=new MethodPractice();
        System.out.println(mp.max(a,b));
```

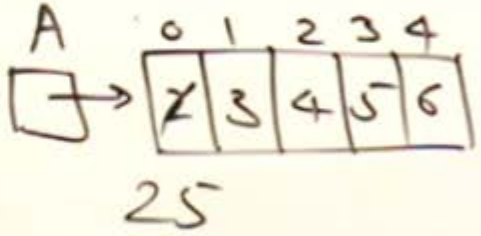
→ ham "method" koi bina static bnaya bhi use kar sakta hai. Uska liya hma class ka ek object ko bnana hoga.
Jaise ise code mai "MethodPractice" class hai and hmna "mp" name ka ek object kiya hai then usa print kiya ha same waise hi jaisa scanner class bna kar karta hai.

```
compile:
run:
15
BUILD SUCCESSFUL (total time: 0 seconds)
```

Methods

```
class Test
{
    static void update(int A[])
    {
        A[0]=25;
    }

    public static void main(String args[])
    {
        int A[]={2,3,4,5,6};
        update(A);
        S.O.P(A[0]);
    }
}
```



25

Methods → Example of various return data type as follows:

void

```
int max(int x, int y)
{
    return x>y?x:y;
}
```

※ for (i) --> return type is int.

※ for (ii) --> return type is string


```
String useName(String email)
{
    int a=email.indexOf('@');
    String name=email.substring(0,a);
    return name;
}
```

```java
public class MethodPractice
{
    static void change(int X[],int index,int value)
    {
        X[index]=value;
    }

    static void change2(int x,int value)
    {
        x=value;
    }

    public static void main(String[] args)
    {
        int A[]={2,4,6,8,10};

        change(A,2,20);

        for(int x:A)
        {
            System.out.println(x);
        }

        int x=10;
        change2(x,20);
        System.out.println("Value of Primitive "+x);
    }
}
```

ise case mai value update hoti hai.

→ When we pass primitive data type (refer data type note), then passed as value and no value updation take place in "main".

→ But when we passing object (such as array) as parameter then wo as reference pass hota hai and 'main' mai value update ho jati hai.

※ for more read next 2 slide in detail, you will understand everything easily.

ise case mai updation nahi hota hai.

Phela **Code:**

1. Aapne ek function `update` define kiya hai jo teen arguments leta hai: `a[]` (array), `index` aur `valu`. Yahan `a[]` ek array hai, aur `index` batata hai ki kaunse element ko update karna hai.
2. `update` function ke andar aapne `a[index] = valu;` likha hai. Yeh line array ke `index` position par `valu` ki value set kar degi.
3. `main` function mein, aapne ek `int` type ka array `a` banaya hai jisme 10 elements hain. Kuch elements ko aapne pehle se hi values assign ki hain.
4. Fir aapne `update` function ko call kiya aur `a` array mein `4` index par `63` ki value assign ki.
5. Ab aapne loop se array ke tamam elements ko print kiya. Aap dekhein ki `a[4]` ka value ab `63` ho gaya hai, jisse update kiya gaya tha.

Dusra Code:

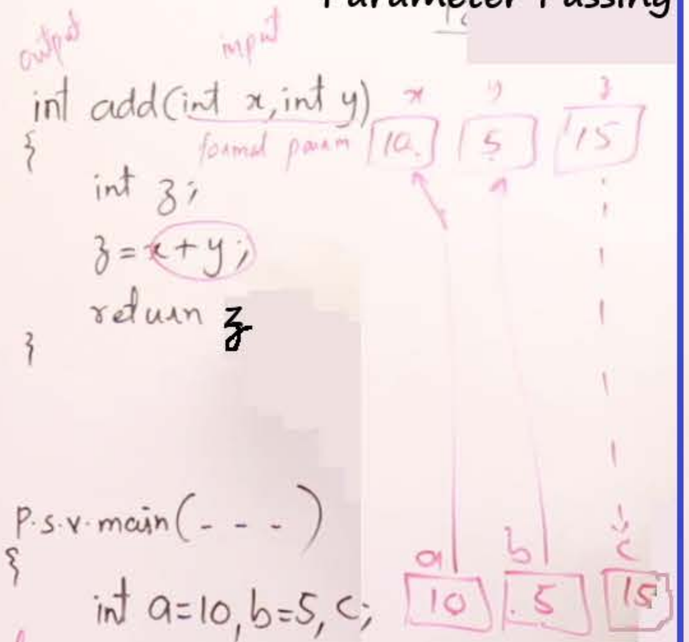
1. Aapne ek function `change` define kiya hai jo do arguments leta hai: `y` aur `value`. Dono arguments `int` type ke hain.
2. `change` function ke andar aapne `y = value;` likha hai. Yeh line lagta hai jaise `y` ki value ko `value` ki value se replace kar degi. Lekin yaad rakhein ki `y` ek local variable hai `change` function ke andar.
3. `main` function mein, aapne ek variable `x` ko value `10` assign kiya hai.
4. Fir aapne `change` function ko `x` ki value pass ki hai, jahan `y` ko `x` ki value aur `value` ko `20` ki value assign ki gayi hai.
5. `change` function ke andar, `y` ko `20` set kiya jata hai, lekin yeh change `main` function ke `x` ko nahi affect karta. Unka value ek lamhe ke liye samaan ho sakta hai, lekin yeh alag variables hain alag-alag scopes mein.
6. `change` function ko call karne ke baad, aap `x` ki value print karte hain, jo ab bhi `10` hai kyun ki `change` function ke andar `y` ko update karne se `main` function ke `x` ki value pe koi asar nahi padta.

   Mukhya baat samajhne wali yahan hai ki `x` aur `y` alag variables hain aur alag scopes mein hain. `change` function ke andar `y` ki value ko update karna `main` function ke `x` ko affect nahi karta kyun ki woh alag variables hain. Lekin, array ke case mein, aap array ko direct update kar sakte hain kyun ki woh ek reference type hota hai, aur changes usi array ke saath link hote hain.

Parameter Passing

```
int add(int x, int y)
{
    int z;
    z = x+y;
    return z;
}

P.s.v.main(- - -)
{
    int a=10, b=5, c;
    c=add(a,b);
    S.o.p(c);
}
```



→ Ise type mai actual parameter, formal parameter mai direct copy hota hai.

```
void welcome(String n)
{
    S.o.p("Welcome Mr/Miss "+n);
}

P.s.v.main(..)
{
    String name="victor";
    Welcome(name);
}
```

→ Ise type mai actual parameter, formal parameter mai direct copy nahi hota hai.
Wo phela "main" mai ek variable create karta hai jo string ki traf point karta hai. And jab "method" ko call karta hai to waha bhi ek variable create karta hai but phela wala string ki traf hi point karta hai. Koi new string create nahi nahi karta hai.

→ Caller: Jisna mth. ko call kiya hai.
Ise case mai caller "main" hai.

→ Calling method: Jis method ko bulaya gya hai.

→ Parameter inside "**main**" called actual parameter.

→ Parameter inside "**calling method**" called formal parameter.

→ Java mai parameter pass karna ka sirf ek method hota hai jisa simply parameter passing method bolta hai.
Ise mai actual parameter copy ho jata hai formal parameter mai.

→ Ise mai parameter pass by value, reference and so on jaisa kuch nahi hota hai.

Method Overloading

```
int max(int x, int y)
{
    return x>y?x:y;
}

float max(float x, float y)
{
    return x>y?x:y;
}

int max(int x, int y, int z)
{
    return x>y && x>z ? x : (y>z ? y : z);
}
```

isko ternary operator bolta hai

→ method overloading ek aisa concept hai jisse hum apne program mein ek hi naam se kai saare functions bana sakte hain, lekin unke parameters alag-alag type aur alag no. mai ho. Matlab, agar humein ek function ko alag-alag tarikon se use karna hai, toh hum method overloading ka istemal kar sakte hain.

→ Example ke liye, maan lo aapko ek class banana hai "Calculator" namak. Ismein aapko do numbers add karne ka function banane hain. Lekin, aap chaahte hain ki yeh function integers aur decimals dono ke liye kaam kare. Toh aap ek hi naam se do functions create kar sakte hain - ek integer wala aur ek decimal wala.

→ Aap dekhte hain, jab aap call karoge "add" function ko integer values ke saath, toh woh integer wala function chalega. Aur jab aap decimals ke saath call karoge, toh decimal wala function chalega.

※ Aur yeh aapko code likhne mein flexibility deta hai aur program ko zyada maintainable bhi banata hai.

Variable Arguments varargs

```
void show(int...x)
{
    for(int a : x)
    {
        s.o.p(a);
    }
}
```

int []x

--> variable arguments ek aisa concept hai jo hume allow karta hai ki hum ek function mein alag-alag number of arguments pass kar sakai, bina har baar function ko modify kia.

--> printf kuch ise concept par kaam karta hai.

--> "C & Cpp" mai ise concept ko ellipses bolta hai.

--> ise mai int, float, aur bhi sabhi type ka data type ka use kar sakta hai.

```
show();
show(10);
show(10,20,30);
show(10,20,30,40);
show(10,20,30,40,50,----);
```

- `show(new int[]{10,20,30,40});`

--> inka element ko access karna ka liya, ham array ka sabhi method ka use kar sakta hai.

Ek example se samjhata hoon. Maan lo aapko ek function banana hai "totalSum" naam se, jo numbers ki sum nikale. Lekin, aap chahte hain ki kitne bhi numbers aayein, woh function handle kar sake. Toh variable arguments ka use karke woh function aisa dikhega:

java Copy code

```java
public class VariableArgumentsExample {
    static int totalSum(int... numbers) {
        int sum = 0;
        for (int num : numbers) {
            sum += num;
        }
        return sum;
    }

    public static void main(String[] args) {
        int sum1 = totalSum(5, 10, 15);
        int sum2 = totalSum(2, 4, 6, 8, 10);

        System.out.println("Sum 1: " + sum1);
        System.out.println("Sum 2: " + sum2);
    }
}
```

Dekho, yahaan par humne "totalSum" function mein "int... numbers" likha hai, jo variable arguments represent karta hai. Aap chahe jitne numbers pass kar sakte hain. Fir loop mein humne saare numbers ko add kiya aur sum return kiya.

Is tarah se, variable arguments ka use karke hum aasani se alag-alag quantity ke arguments pass karke function ko use kar sakte hain. Ye code aapko different sums dega har baar jab aap alag-alag numbers pass karenge.