24/11/21

→ **functions/Methods** (class fn) : Block of code which only runs when it is called.

: Reuse code - define code once, use it many times.

Syntax - definition of fn

```
public class Main {                    function name
    static void functionex() {
        //code
    }
}
```

return type (null) → void

OR

```
public class Main {
    accessmodifier returntype methodname(){
        //code
        return statement;
    }
}
```

: call fn - methodname();

~~[illegible]~~

ReturnType - used for returning the resulted value
- void (null value)
- returning data must be compatible with return type specified by function
- variable receiving value returned by fn must be compatible with return type specified for function.

function parameters - Arguments/parameters are variable/literals specified in functions.

* pass by value -

eg. public class Example {
public static void main(String[] args){
String name = "Ishika";
greet(name);
}

object/value

zaruri nahi same ho

copy of value of name passed to definition variable/parameter to call of fn

static void greet(String naam){
System.out.println(naam);
}
}

** No pass by reference in java, only pass by value

* You cannot modify strings, they are immutable.

* primitive data types just pass value (no references) or data but object & reference pass value of reference variable

non-primitive data types

→ Scope : fn scope - variables declared/created inside a fn can't be accessed outside fn.

: Block Scope -

```
public svm (){
    int a = 10;
    int b = 20;
    {// int a = 5;}
    a = 100;
    int c = 20;
    // c = 10;
    int c = 15;
    a = 80;
}
```

variables outside the block can be updated inside the box.

outside block cannot be updated inside block

inside block variables cannot updated outside block

variable inside block can be reinitialized outside the block

∴ loop scope - variables declared inside loop are having loop scope

→ Shadowing ∴ using variables in overlapping scopes with same name where the variable in low-level scope overrides the variable in high-level scope.
variable at high-level is shadowed by low-level scope variable.

→ Variable Arguments (Varargs) ∴ to input number of args/params in fy.

Syntax -

```
static void fun(int ....a){
    // method body
}
```

Ambiguities ∴ issues not defined clearly in Java language.

or

eg. psvm(---){
    fun(a:2, b:3, ...v : "Ishika", "Ishan");
}
(This should be inputted according to definition sequence only.)

```
void fun(int a, int b, String ...v){
}
```

→ function overloading ∴ Two functions have same name but different types & number of parameters.
∴ It decides at compile time, which fy should be run.

eg. fun(int a){ // code }
fun(int a, int b, String c){ }

∴ loop scope - variables declared inside loop are having loop scope

→ Shadowing ∴ using variables in overlapping scopes with same name where the variable in low-level scope overrides the variable in high-level scope.
variable at high-level is shadowed by low-level scope variable.

→ Variable Arguments (Varargs) ∴ to input number of args/params in fy.

Syntax -

```
static void fun(int ....a){
    // method body
}
```

Ambiguities ∴ issues not defined clearly in Java language.

or

eg. psvm(---){
    fun(a:2, b:3, ...v : "Ishika", "Ishan");
}
(This should be inputted according to definition sequence only.)

```
void fun(int a, int b, String ...v){
}
```

→ function overloading ∴ Two functions have same name but different types & number of parameters.
∴ It decides at compile time, which fy should be run.

eg. fun(int a){ // code }
fun(int a, int b, String c){ }