5. Conditional Statements & Loops

① if-else :-

```
if (true) {
    Condition-1;
}
```

```
if (true) {
    Condition;
}
else {
    Condition;
}
```

② while :-

```
while (Condition) {
    Condition;
    increment;
}
```

- when you don't know how many times loop is going to run.

③ for :-

```
for (initialization; Condition; increment) {
    Statements;
}
```

④ do-while :-

```
do {
  statements;
} while(Condition);
```

Q Find largest in 3 numbers

A ① using 4th variable,

a=10
b=20
c=30

let max = a, then
if b > max:
max = b
if c > max;
max = c
else max = a

② without 4th variable,

if a>b & a>c,
max = a
if b>a & b>c
max is b
if c>a & c>b
max is c
~~else~~.

```
int max=0;
if (a>b){
    max=a
} else{
    max=b;
}
if (c>max){
    max=c;
}
```

③ int max= Math.max(c, Math.max(a,b))

Q) fibonacci numbers.

0, 1, 1, 2, 3, 5, 8, 13, 21 - - - -

A)

```
a=0;
b=1;
~~temp=. while (b!=0) n=0;~~
whik (n!=0){
    int temp = a+b;
    Sout("n+"th fiboracci number= " + temp);
    a=b;
    b=temp;
    n++;
}
```

b ← a
1 ← b
1 ← temp2
2

~~Q) Find nth fibonacci number.~~

Q) Count occulances? n = 1385757879

A) how many '7' ⇒ 3

A)

```
while (n>0){
    int rem = n%.10;
    if (rem == occur){
        Count ++;
    }
}
Sout(Count);
```

Q) Reverse n = 1234
ans = 4321

A) ⇒ while (n > 0) {
int temp = n % 10;
ans = ans * 10 + temp
n = n / 10;

n = 1234
ans = 0
⇒ temp = 4
ans = 0 * 10 + 4
= 4

n = 123
ans = 4
temp = 3
ans = 4 * 10 + 3 = 43
n = 12

6. Switch

In switch case, we can directly jump to various cases based on condition given.

```
Switch(expression){
    case one;
        // statement;
        break;
    Case two;
        //statement;
        break;
    default;
        // Statement;
```

```
Switch (fruit){
    Case "Mango":
        System.out.println("yellow fruit");
        break;
    Case "Apple";
        System.out.println("red fruit");
        break;
    default:
        System.out.println("enter valid fruit");
}
```

7. Functions / Methods

→ Everything in java is a class. And functions in classes are called methods.

static/access modifiers returntype function()

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

    Static void sum(){
        Scanner in = new Scanner(System.in);
        int num1 = in.nextInt();
        int num2 = in.nextInt();
        int sum = num1 + num2;
        sout("sum = " + sum);
    } }
```

does not depend on object

Return type :-

① When a function is called, what datatype is the output

② When function execution ends, output is converted to return type.

① Just return output value.

```
Static int sum2(){
    //same as sum but
    return sum;
}
```

→ now "sum" points to integer, so in psvm, we need to assign this to another variable & print it.

→ no other statement will be executed after this.

② Return string

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

String message = greet();
    sout (message);
```

Parameters:-

Pass values when calling method.

```
Static int sum3(int a, int b){
    int sum = a+b;
    return sum;
}
```

eg:-

```
public class Names{
    public static void main(String[] args){
        Scanner in = new Scanner(System.in);
        String name = in.nextLine();
        System.out.println("
        String message = greetName(name);
    }
    static String greetName(name){
        String message = "Hello, " + name;
        return message;
    }
}
```

Q. Swap function (a=10, b=20 ; a=20, b=10)

A.
```
public class Swap {
    public static void main(String[] args) {
        System.out.print("Enter first number ");
        Scanner in = new Scanner(System.in);
        int a = in.nextInt()
        System.out.println("Second number ");
        int b = in.nextInt()
        int temp = b ;          //temp = 20 ; b = 20 ; a = 10
        b = a;                  // b = 10 ; a = 10 ; temp = 20
        a = temp;               // b = 10 ; a = 20 ; temp = 20
    }
}
```

```
int[] arr = {1, 3, 2, 45, 9};
change(arr)
Sout (Arrays.toString(arr))

Static void change (int[] nums) {
    nums[0] = 99;
}
```

↳ Copy of value of ref variable arr.

⇒ arr = [99, 3, 2, 45, 9]

arr → [1, 3, 2, 45, 9]

nums → [99, 3, 2, 45, 9]

nums[0] = 99

∴ arr = [99, 3, 2, 45, 9]

But in primitives,
variables get values instead of reference variable copy.
So, swap can't be a function

Scoping :-

Scope → where we can access our variables.
Global scope → can be accessable anywhere
Method scope → only be accessed in method.
Loop scope → only in loops.

eg:-

```
public class Scope {
    public static void main(String[] args) {
        int a = 10;          // global scope
        int b = 20;
    }

    static void random(int nums) {
        int num = 67;        // methode scope
    }
}

for(int i=0; i<20; i++)      // i = b/t loop scope.
```

{ int g=78; } ← only valid in block

↪ any of variable initialized outside the block can be used inside to read, update & use it. But anything initialized inside box is permitted to it.

Shadowing :-

```
public class Shadow{
static int x=90; public static void main(String[] args){

    }
}
```

→ Shadowing is a concept/practice of using 2 variables with same name within the scope that overlaps. The variable with higher level scope is hidden.

→ Shadowing happens when the scope is initialized.

Variable Length Argument (VarArgs)

→ When we don't know how many inputs would be given.

```
static void fun (<datatype>... (variable))
```

```
Static void func (int a, int b, String ...v)
```

Function overloading:-

Two or more functions of same name can exist but with different parameters/arguments, at compile time decides which one to run based on arguments.

① Type of argument should be different
② Arguments should be different.

Q) Prime?

A)
```
public class Prime{
    public static void main(String[] args){
        Scanner in = new Scanner(System.in);
        int num = in.nextInt();
        isPrime(num);
    }
    static boolean isPrime(int num){
        if(num<=1){
            return false;
        }
        int c = 2;
        while(c*c<=num){
            if(n%.c==0){
                return false;
            }
            c++;
        } else {
            return true;}}}
```

Q) Print all 3 digit Armstrong numbers?

A) $153 \Rightarrow 1^3 + 5^3 + 3^3 = 153$, then it's armstrong number.

n = 153

sum = 0

⇒ rem = n % 10 (153) % 10 → 3 | 5 | 1

n = n/10 (15 | 1 | 0

sum = sum + rem³ 0 + 3³ = 27 | 27 + 5³ → 125 + 27 | 153

If sum = num, then armstrong.