Conditionals:

Ex:

```
int salary = 10000;
if (salary > 15000) {
    salary += 2000;
}
else if (salary > 20000){
    salary += 3000;
}
else {
    salary += 1000;
}
System.out.println(salary);
```

Loops:

print numbers from 1 to 5

```
for (int i=0 ; i<=5 ; i++){
    System.out.println(i);
}
```

print numbers from 1 to n

```
{
    Scanner input = new Scanner(System.in);
        int n = input.nextInt();
    for (int i = 1 ; i<=n ; i++){
        System.out.println(i);
    }
```

While:

```
    int num = 1;
    while (num <= 5){
        System.out.println(num);
        num++;
    }
```

do while:

```
    do {
        System.out.println(n);
    } while (n<=5);
```

Ex:

```
    int n=1;
    do{
        System.out.println("Hello World");
    } while (n!=1);
```

Executes atleast once inspective of the condition

Questions:

1. Largest Number:

```
    Scanner in = new Scanner(System.in);
    int a = in.nextInt();
    int b = in.nextInt();
    int c = in.nextInt();
```

```
int max = a;
if (b > max){
    max = b;
}
if (c > max){
    max = c;
}
System.out.println(max);
```

Math.max(a,b)

Math.max(c, Math.max(a,b))

2. Alphabbet case check

```
Scanner in = new Scanner(System.in);
char ch = in.next().trim().charAt(0);
```

trims the spaces before input

→ trims the first character

```
if (ch >= 'a' && ch <= 'z'){
    System.out.println("Lower case");
}else{
    System.out.println("Upper case");
}
```

3. Fibonacci numbers:

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

0, 1, 1, 2, 3

a	b
0	1
1	1
1	2
2	3

```
Scanner in = new Scanner(System.in);
int n = in.nextInt();
int a = 0;
int b = 1;
int count = 2;
while (count <= n){
int temp = b;
    b = b + a;
    a = temp;
```

we have two numbers
So count = 2

```
        count++;
    }
    System.out.println(b);
```

4. Counting Sequences:

```
int n = 64336432;
int count = 0;
while (n > 0) {
    int rem = n % 10;
    if (rem == 3) {
        count++
    }
    n = n / 10;
}
System.out.println(count);
```

5. Reverse:

```
int n = 234698;
int ans = 0;
while (n > 0) {
    int rem = n % 10;
    n = n / 10;
    ans = ans * 10 + rem;
}
System.out.println(ans);
```

Switch:

In switch statements, you can jump to various cases based on your expressions.

- cases have to be same type as expressions, must be a constant or literal.
- duplicate case values are not allowed.
- break is used to terminate the sequence.
- if break is not used, it will continue to next case.

- default will execute when none of the above does

Ex:

```
Scanner in = new Scanner(System.in);
String fruit = in.next();
switch (fruit) {
    case "Mango":
        System.out.println("Yellow colour");
        break;
    case "Apple":
        System.out.println("Red colour");
        break;
    case "Orange":
        System.out.println("Round Fruit");
        break;
    case "Grapes":
        System.out.println("Small Fruit");
        break;
    default:
        System.out.println("Invalid!");
}
```

Enhanced Switch:

```
switch (fruit) {
- case "Mango" -> System.out.println("Yellow colour");
  case "Apple" -> "Red colour";
  case "Orange" -> "Round fruit";
  case "Grapes" -> "Small fruit";
  default -> "Invalid!";
```

```
Switch (day){
    Case 1:
    Case 2:
    Case 3;
    Case 4:
    Case 5:
        System.out.println("Weekday");
        break;
    Case 6:
    Case 7:
        System.out.println("Weekend");
        break;
```

→ old syntax

```
Switch(day){
    case 1,2,3,4,5 → sout(
        "Weekdays");
    case 6,7 → sout(
        "weekend");
}
```

↓ Enhanced Syntax

Nested Switch Case:

```
Scanner in = new Scanner (System.in);
int empID = in.nextInt();
String department = in.next();
Switch (empID){
    Case 1:
        System.out.println("Sri");
        break;
    Case 2:
        System.out.println("Harsha");
        break;
    Case 3:
        System.out.println("Emp ID: 3");
        Switch (department){
            Case "IT":
                System.out.println("Information Tech");
                break;
            Case "Med":
                Sout("Medicine");
                break;
            default:
                Sout("Invalid dept");
```

```
        }
        break;
    default:
        sout (" Enter correct ID");
}
```

Methods in Java:

To bundle a repeated code in a sort of a format we use methods/functions. So instead of re writing the code we can use the method again and again.

Ex:

```
public class Sum{
    public static void main(String[] args){
        sum();
    }
static void sum() {
        Scanner in = new Scanner(System.in);
        System.out.println(" Enter num1: ");
        int num1 = in.nextInt();
        System.out.println(" Enter num2: ");
        int num2 = in.nextInt();
        int sum = num1 + num2;
        System.out.println(sum)
    }
}
```

When the function finishes execution, the function call sum() is going to have a value returned by the function.

void ⟶ No returning

No statement will execute in the function after return statement.

Page No.: YOUVA
Date:

Returning values:

```
. . . . main( . . . ) {
    int ans = sum2();
}
static int sum2(){
    Scanner in = new Scanner(System.in);
    Sout (" Enter num1: ");
    int num1 = in.nextInt();
    .
    .
    .
    sum = num1 + num2;
    return sum;
```

Returning Strings:

```
psvm ( ){
    String message = greet();
    sout(message);
}
static String greet(){
    String greeting = "how are you?";
    return greeting;
}
```

Passing the parameters in integer func

```
PSVm {
    int ans = sum(20, 30);
    sout(ans);
}
static int sum(int a, int b){
    int sum = a+b;
    return sum;
}
```

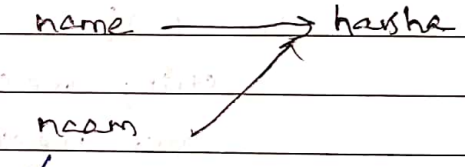
Page No.: YOUVA
Date:

Passing parameters in string function:

```
psvm {
    Scanner in = new Scanner(System.in);
    Sout("Enter your name: ");
    String name = in.next();
    String message = myGreet(name);
    Sout(message);
}
static String myGreet(String name){
    String msg = "Hello " + name;
    return msg;
}
```

Internal working of passing:

```
main() {
    name = "harsha";
    greet(name);
}
greet(naam) {
    print(naam)
}
```



This also point to object "harsha" because naam is copying the value of reference variable name

Ex:

```
{String name = "harsha";
 changeName(name);
 Sout(name)
}
static void changeName(naam){
    naam = "Sri";
}
```

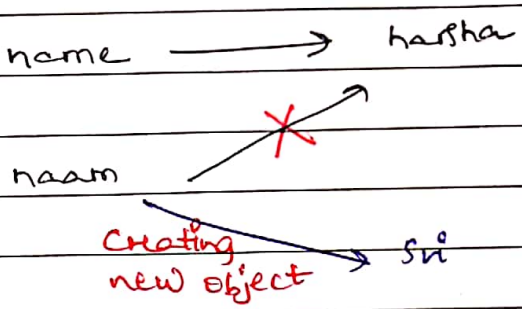
↓

This does not change name = "harsha". Why?

The parameter name and naam initially point to the same object.

Then in function the parameter naam is creating a new value

So here it is not changing but creating a new object.




```
psum ( ) {
    a = 10
    b = 20
    swap (a, b)
}
swap(num1, num2) {
    temp = num1
    num1 = num2
    num2 = temp
}
```

a → 10

b → 20

temp → 10

num1 → 20

num2 → 10

These are swapped only in the scope of swap(num1, num2)