~~[illegible]~~ {string}.equals("text") →

· checks if the value of string is equal to the given text.

There are two cases
string a = "Jaideep"
string b = "Jaideep"

case I ↳

$$\begin{tikzcd} a \arrow[dr] & \\ & \text{Jaideep} \\ b \arrow[ur] & \end{tikzcd}$$

two ref. variable pointing to same ~~[illegible]~~ object

case II → a ⟶ Jaideep
b ⟶ ~~[illegible]~~ Jaideep

two ref. variables pointing to ~~[illegible]~~ seperate objects.

· equals function will give true for both cases
If we use ~~string~~ a == b then,
for first case →

$$\begin{tikzcd} a \arrow[r] & \text{Jaideep} \\ b \arrow[ur] & \end{tikzcd}$$

will give True

for second case ⟶
a ⟶ Jaideep
b ⟶ Jaideep
will give False

* Switch case →

Syntax →

```
switch (expression) {
    case one :
            //do something
            break;
    case two :
            // do something
            break;
    default :
            //do something
}
```

Some points :→ ① Cases have to be the same type as expression must be a constant or literal. ② Duplicate case values are not allowed ③ Break is used to terminate the sequence. ④ If break is not used, it will continue to next case. ⑤ Default will execute when none of the above ~~exists~~ does. ⑥ If default is not at end, put a break.

⑭ Print fruit characteristics using switch case.

```
import java.util.Scanner;
public class Fruit{
        public static void main(String[] args) {
            Scanner input = new Scanner (System.in);
            String ch = input.next();
            switch (ch) {
                    case "Apple" :
                            System.out.println("A sweet fruit");
                            break;
                    case "Orange" :
                            System.out.println("A round shaped fruit");
                            break;
                    case "Mango" :
                            System.out.println("King of fruits");
                            break;

                    default :
                            System.out.println("Enter a valid fruit")
            }
    }
}
```

INPUT→
Orange
OUTPUT→
A round shaped fruit

type as expression
values
ate the sequence
next case. ⑤
ove ~~[illegible]~~ does.

case.

n);

fruit");

nd shaped fruit");

of fruits");

a valid fruit");

The above switch can be replaced with a newer switch case syntax

```
switch(ch){
      case "Apple" → System.out.println("A sweet fruit");
      case "Mango" → System.out.println("King of fruits");
      case "Orange" → System.out.println("A round shaped
                                                  fruit");
      default → System.out.println("please enter a valid fruit");
}
```

⑮ Given day number print if its weekend or weekday.

```
import java.util.Scanner;
public class Week{
    public static void main (String [] args){
          Scanner input = new ~~[illegible]~~ Scanner(System.in);
          int n = input.nextInt();
          switch (ch){
                case 1,2,3,4,5 → System.out.println("Weekday");
                case 6,7 → System.out.println("Weekend");
          }
    };
}
```

INPUT →
3
OUTPUT →
Weekday

⑯ Nested Switch implementation

```
import java.util.Scanner;
public class Emp{
    public static void main(String[] args){
          Scanner input = new Scanner(System.in);
          int empid = input.~~[illegible]~~ nextInt();
          String dep = input.next();
          switch (empid){
                case 1 :
                    System.out.println(" Jaideep");
                    break;
                case 2:
                    System.out.println(" Mayank" );
                    break;
                case 3:
                    switch (dep){
                        case " IT " → System.out.println
                                        (" IT department");
                        case " CS " → System.out.println
                                        (" CS department");
                    }
                    break;
            default :
                System.out.println(" Enter valid data");
            }
        }
}
INPUT→
3
CS
OUTPUT→
CS department
```

FUNCTIONS/METHODS

Syntax →

```
<access modifier> <return type> <name> (arguments){
            // body
            <return ~~type~~ statement;
}
```

Fuction can also called be a method as java is completely OOP and any function in class is called a method.

```
sideep);
"Mayank");
ystem.out.println
T department");
System.out.println
department");
"Enter valid data");
ments){
java is completely
```

Example of function →

```
import java.util.Scanner;
public class Sum{
  public static void main(String[] args) {
      sum();
  }
  static void sum(){
      Scanner input = new Scanner (System.in);
      System.out.println(" Enter two numbers : ");
      int a,b,sum;
      a = input.nextInt();
      b= input.nextInt();
      sum = a+b;
      System.out.println("Sum is : " + sum);
  }
```

OUTPUT →

```
Enter two numbers : 7 5
Sum is : 12
```

* Returning a String →

```
psvm{
      String message = greet ();
            System.out.println (message);
}
static String greet(){
      return "How are you?".
}
```

OUTPUT →

```
How are you?
```

* Passing values when calling function →

```
psvm{
int ans = sum (1, 7);
System.out.println(" sum is : + ans);
}
static int sum(int a, int b){
  return a+b;
}
```

OUTPUT →

```
Sum is : 8
```

* Deep diving in functions (How functions work) → Ex – ①

Date.........
Page.........

```
psvm {
    int a = 10;
    int b = 20;
    swap (a,b);
    System.out.println(a + " " + b);      // a and b will not be swapped
}                                            as in function swap (a,b)
static void ~~[illegible]~~ swap (int a, int b){   a copy of value is
    int temp= a;                             created and the copy is
        a = b;                               swapped by original a,b
        b = temp;                            remains same.
}
```

$a \longrightarrow 10$

$a \longrightarrow 10$

$b \longrightarrow 20$

$b \longrightarrow 20$

But original values are not swapped

Inside fun →

temp = 10

a = 20
b = 10 { These are local scope variables }

Ex – ②

```
psvm {
    String name = "Jaideep";
    greet (name);   → only copy of value of ref variable
}                     is passed
static void greet (String naam){
    naam = "monu";     // Here we are not changing the object
}                         we are creating a new one.
```

Strings are immutable in java

name ⟶ Jaideep

naam ↗

naam ⟶ monu (new object created)

name ⟶ Jaideep

name remains same as Jaideep

* IMP Points →

1. Primitives like int, short, char, byte & ~~to~~ are ~~passed~~ passed by value.

2. Objects and ~~sto~~ stuffs are passed by value of the reference variable

~~8~~ Ex - 3

```
psvm{
    int[] arr = {1,2,3,4};
    change (arr)
    System.out.println(Arrays.toString(arr));
}
static void change( int[] nums){
    nums[0]= 10;
}
```

As arrays are non-primitive
hence it is passed by value of the
reference variables hence single
object having two ref. variables

$$\begin{tikzcd} arr \arrow[r] & {[1,2,3,4]} \\ nums \arrow[ur] & \end{tikzcd}$$

~~ucho~~ Inside fun
nums [0] = 10

$$\begin{tikzcd} arr \arrow[r] & {[10,2,3,4]} \\ nums \arrow[ur] & \end{tikzcd}$$

* Scoping →

```
psvm{
  int a = 10;
  int b = 20;
  System.~~o~~out.print(marks);   //will give error as scope of
}                                 marks is in fun random only
static void random(int marks) {   i.e. it cannot be accessed
    marks = 10;                   outside it.
}.
```

Ex - ②

```
psvm{
  int a = 10;
   {
      a = 70;       // can be used in block as it is defined in
      int c = 70;      the function and the block is inside
   }                   the function
   System.out.print(c);  // will give error as scope of c in
}                            in the block only
```

Ex - ③

For loops, while loops, if, else

Same conditions as in example ②

In simple words anything which is initialized outside the block can be used inside the block and anything initalized inside the block cannot be used outside it.

× × × × × ×

* Shadowing →

```
~~psvm{~~
  public class Scope {
     static int x = 40;  // This will be shadowed at line 6
     public static void main (String[] args){
           System.out.println(x); // 40
           int x = 90;
           System.out.println(x); // 90
           fun();         // 40
     }

     static void fun() {                    OUTPUT→
          System.out.println(x);            40
     }                                      90
}                                           40
```

Date.......
Page.......

Scope of a local variable begins from its initialization and not from when it is defined.

```
int x = 40;
psvm {
    System.out.println(x); //40
    int x;
    
    System.out.println(x); // error as x is not initialized
    x = 40;
}
```

* ## Variable length arguments →

```
psvm {
      Fun(1, 2, 3, 4, 5, 7, 9, 10);
}
static void fun(int...v){
      System.out.println(Arrays.toString(v));
}
```

OUTPUT→
```
[1, 2, 3, 4, 5, 7, 9, 10]
```

VarArgs is used when we do not know how many arguments are going to be there

if we want to use multiple data type arguments with VarArg then VarArg should be at last

Ex →
```
    static void multiple ( int a, String name, int...v){

    }
```

* ## Function Overloading →

Two or more function of same name with different arguments can exist. This is known as function overloading.

```
static void fun (int a){
}

static void fun (String name){
}
```

if any string is provided then this will run and if any int is provided then other one will run.

(17) Check if a number is prime or not using function.

```
import java.util.Scanner;
public class PrimeFun{
    public static void main(String[] args){
        Scanner input = new Scanner(System.in);
        int n = input.nextInt();
        System.out.println( prime(n));
    }
    static boolean prime(int a){
        if (a<=1){
            return false;
        }
        int c=2;
        while ( c*c <= a){
            if (a%c==0){
                return false;
            }
            c++;
        }
        return true;
    }
}
```

INPUT →
37
OUTPUT →
true

(18) Print all three digit armstrong number

Date.......................
Page.......................

```
~~import java.util.Scanner~~
public class Armstrong{
    public static void main(string[] args){
        armstrong();
    }
    static void armstrong(){
        int a = 100;
        while (a<1000){
            int x = a;
            int sum = 0;
            while (x>0){
                int N = x % 10;
                sum = sum + (int) Math. pow (N,3);
                x = x/10;
            }
            if (a = = sum){
                System. out. print ( sum+ "  ");
            }
            a++;
        }
    }
}
```

OUTPUT

```
153  370 ~~[illegible]~~ 371  407
```