<!-- ! Data Types In js -->

<!-- ! Premitive data type -->

```
        1)Number
        2)Boolean
        3)String
        4)Null
        5)Undefined
        6)BigInt
```

<!-- ! Non-Premitive Data type -->

```
        1) array
        2) object
        3) function
```

<!--! Primitive Data Types -->

<!--? Number: -->

Represents both integer and floating-point numbers. Example: 42, 3.14.

<!--? Boolean: -->

Represents a logical entity and can have two values: true or false.

<!--? String: -->

Represents a sequence of characters used to represent text. Example: "hello", 'world'.

<!--? Null: -->

Represents the intentional absence of any identifier's value.

<!--? Undefined: -->

Indicates that a variable has not been assigned a value, or not declared at all.

```
 Example: let a;
 console.log(a);  // undefined.
```

<!--? BigInt: -->

Represents whole numbers larger than the Number type can safely represent.

```
Example: BigInt(1234567890123456789012345678901234567890n).



 <!--! NaN -->

     1. invalid mathametical operation
     2. parsing
     3. operation with NaN

   <!-- the type of NaN is number.   -->

<!-- ! Difference between null and undefined  -->

undefined typically indicates a variable that has been declared but hasn't
been
assigned a value, or a value that is missing or hasn't been returned from a
function.

null is used to explicitly indicate the absence of a value or to reset a
variable that
previously held a value.


<!-- !  take inputs from user and perform addition -->

var a = Number.parseInt( prompt("enter first number"));
var b = Number.parseInt( prompt("enter second number"))

alert(a+b);



                      <!-- ! falsy value -->


1. `false`:  The boolean value `false`.
2. `0`:  The number zero.
3. `-0`:  Negative zero.
4. `""` (empty string):  A string with no characters.
5. `null`:  Represents the absence of any object value.
6. `undefined`:  A variable that has been declared but not assigned a value.
7. `NaN`:  Represents a "Not-a-Number" value resulting from an undefined or
unrepresentable mathematical operation.

example :

console.log(Boolean(null))    <!-- false -->
```

<!-- ! Type coercion -->

Type coercion in JavaScript refers to the automatic or implicit conversion of values from one data type to another. This often happens in operations that involve different types of values, such as strings, numbers, or booleans. JavaScript tries to make sense of these operations by converting values to a common type.

Examples

<!--? String Coercion: -->

When we use the + operator with a string and a number, the number is converted to a string.

example :

```
let result = '5' + 10;
console.log(result);    <!-- "510" (number 10 is converted to string "10") -->
```

<!--? Number Coercion: -->

When you use arithmetic operators (other than +), strings that represent numbers are converted to numbers.

example :

```
let result = '5' - 2;
console.log(result);       <!-- 3 (string "5" is converted to number 5) -->
```

<!--? Boolean Coercion: -->

Values in conditional statements are converted to boolean.

```
if ('hello') {
    console.log('This is truthy');
}
```

<!-- "This is truthy" is logged because non-empty strings are truthy. -->

```
// !  number converting into string

let result = '5' + 2 ;

console.log(result)    // output : 52


// !  string converting to number

let result2 = '5' - 2
console.log(result2)     // output : 3


// ! falsy values =>

        //  false , null, undefined, 0 , -0 , NaN

let a = 10

if('san')                              //!  string converting into boolean
{
    console.log('hello')
}
else{
    console.log('hi')
}
```