**Data types supported by JavaScript**

* Undefined
* NULL
* Object
* Symbol
* Boolean
* String
* Number

**Difference between undefined and null**

Both represent empty values

Undefined ==> when we define a variable it would return undefined because we aren’t assign any value it would take automatically undefined

Null ==> we can manually empty a variable by using null. If we assign null means the variable will get empty value

**NAN javascript**

The NaN property represents "Not-a-Number" value. This property indicates that a value is not a legal number.

Use the [isNaN()](https://www.w3schools.com/jsref/jsref_isnan.asp) global function to check if a value is a NaN value.

EXAMPLE:

false: 123  
false: -1.23  
false: 5-2  
false: 0  
false: '123'  
true: 'Hello'  
true: '2005/12/12'  
false: ''  
false: true  
true: undefined  
true: 'NaN'  
true: NaN  
true: 0 / 0

**Arrow function**  
short syntax for writing function expression  
you don't need function keyword and the return keyword and the curly brackets  
Arrow functions are not hoisted.   
They must be defined before they are used.  
Using const is safer than using var, because a function expression is always constant value.  
ES5  
var x=function(x, y){  
rerurn x\*Y;  
}  
  
ES6  
var x=(x,y)=> x\*y  
x(5,5)

**Self-Invoking Functions:**  
invoked automatically without being called  
this is a anonymous self invoking function(function without name)  
(function(){  
var x="hai";  
})();

**Explain "use strict"**

* it is a JS directive and introduced in ES5
* used to enforce the code is execute in strict mode
* In strict mode we cannot use variable without declaring it.

**Comparison operator supported by JavaScript**

* > greater than
* < less than
* >=,<=,==,!=,=== equal with data type,!== not equal with data type

**What close () does in JavaScript**  
Used to close current window   
Window. close()

**Debugging JavaScript**

* Debugger
* console.log()

**Method VS Function**

There is a slight difference -

**Method** : Method is a function when object is associated with it.

var obj = {

name : "John snow",

work : function someFun(paramA, paramB) {

// some code..

}

**Function** : When no object is associated with it , it comes to function.

function fun(param1, param2){

// some code...

}

**This**

Refers object it belongs to

It has different values depending on where it is used:

* in a method🡺this refer to the own object
* alone🡺this refers to the global object
* in a function🡺this refers to the global object
* in a function🡺in strict mode, this is undefined
* in a event🡺this refers to the element that received the event
* methods likes call() and apply() can refer this to any object

**JavaScript | typeof operator**

In JavaScript, the typeof operator returns the data type of its operand in the form of a string. Operand can be any object, function or variable.

**Syntax:** typeof operand OR typeof (operand)

typeof "Siddharth";

typeof 3.14;

typeof true;

typeof NaN;

typeof [5, 10, 15, 20];

typeof {name:'Siddharth', age:20};

typeof new Date();

typeof function () {};

typeof workout;

typeof null;

**OUTPUT:**

string

number

boolean

number

object

object

object

function

undefined

object

**Array methods**  
push  
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.push("Kiwi")

pop  
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.pop("Kiwi")

shift🡺remove first element of array  
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.shift()  
"Orange", "Apple", "Mango"

unshift🡺add first element of array  
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.unshift("Lemon")

**String methods**  
string length  
<p id="demo"></p>  
var txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";  
document.getElementById("demo").innerHTML = txt.length;

indexOf🡺first occurrence position of the string  
var d=txt.indexOf("D");

lastIndexOf 🡺last occurrence position of the string  
var d=txt.lastIndexOf("D");

If the text is not found it would return -1

slice(start,end)🡺negative index will take from reverse

substring(start,end)🡺it would not accept negative values

substr(start,length)🡺

replace("hai","hi")🡺replace only first match

toLowerCase()🡺 Convert lower case

var text1=”Hello”;

var text2=text1. toLowerCase ();

toUpperCase()🡺Convert upper case

var text1=”Hello”;

var text2=text1. toUpperCase ();

concat()🡺String concatenation  
var text1 = "Hello";  
var text2 = "World!";  
var text3 = text1.concat(" ",text2);

trim()🡺Remove empty space

var text1=” hello ”;

var text2=text1.trim();

**Difference between var, let and const**  
Both are used to declare a variable/method  
**Var is function scope**

It is globally scoped when a var variable is declared outside a function. This means that any variable that is declared with var outside a function block is available for use in the whole window.

var is function scoped when it is declared within a function. This means that it is available and can be accessed only within that function.

**EXAMPLE:**

var greeter = "hey hi";

function newFunction() {

var hello = "hello";

}

Here, greeter is globally scoped because it exists outside a function while hello is function scoped. So we cannot access the variable hello outside of a function. So if we do this:

**EXAMPLE:**

var tester = "hey hi";

function newFunction() {

var hello = "hello";

}

console.log(hello); // error: hello is not defined

We'll get an error which is as a result of hello not being available outside the function.

**var variables can be re-declared and updated**

That means that we can do this within the same scope and won't get an error.

**EXAMPLE:**

var greeter = "hey hi";

var greeter = "say Hello instead";

console.log(greeter); // "say Hello instead";

and this also

**EXAMPLE:**

var greeter = "hey hi";

greeter = "say Hello instead";

console.log(greeter); // "say Hello instead";

**Hoisting of var**

Hoisting is a JavaScript mechanism where variables and function declarations are moved to the top of their scope before code execution. What this means is that if we do this:

**EXAMPLE:**

console.log (greeter);

var greeter = "say hello"

it is interpreted as this

**EXAMPLE:**

var greeter;

console.log(greeter); //greeter is undefined

greeter = "say hello"

So var variables are hoisted to the top of its scope and initialized with a value of undefined.

**Problem with var**

There's a weakness that comes with var. I'll use the example below to explain this.

var greeter = "hey hi";

var times = 4;

if (times > 3) {

var greeter = "say Hello instead";

}

console.log(greeter) //"say Hello instead"

So, since times > 3 returns true, greeter is redefined to "say Hello instead". While this is not a problem if you knowingly want greeter to be redefined, it becomes a problem when you do not realize that a variable greeter has already been defined before.  
If you have use greeter in other parts of your code, you might be surprised at the output you might get. This might cause a lot of bugs in your code. This is why the let and const is necessary.

Let is block scope