Java Script

It's better to put < script > element below all the elements in the body section. There are two reasons to do so … The browser start parsing the file from top to bottom and if you have put the script code at top then browser will keep parsing that for long and your page will not show any thing and this will create a bad user experience… so it's better to not put the code at the top. Although there are some exceptions also.But still most of time we can go with this format;

And second thing all these operation of javascript will work on the elements ...so it is better to have the js code later than elements;

Type of Notation should use :

```
Camel notation : oneTwoThree
Pascal notation : ( for constructor functions)->OneTwoThird
```

Node is a program which contain javascript in google's v8 engine.

Variable in JS

We can use let to declare the variables in the javaScript;

For sure there are some rules while defining the name

1.) There should be no number in the beginning;
2.) No hyphen (-) should be used
3.) Cannot use reserved words (keywords)

A good software engineering ethic: We should also use meaning full name;

let variable_name=34;
If you don't initialize then there would be an error : undefined error;

```
Primitive Type:  number, string, bool, undefined, null
```

Java-Script is a dynamic type

Falsy and Truthy in JS;
Falsy means those which are equivalent to false -
Undefined, ' ' ,null, 0, false, NaN

Anything that is not falsy is truthy.

Factory Function :

```
// Create object by Factory Functions;
function createObj(name){
    return{
        name,
        view(){
            console.log('name is : ',name);
        }
    };
}
const obj1=createObj('sourav',20);
obj2.view();
-----------------------------------------------------------------
Output:
PS C:\Users\Sourav Sharma\javascript> node index.js
name is :  sourav
```

And the another way is using the constructor function :

```
// Constructor Function;
function Circle(radius){
    this.radius = radius;
    this.draw = function(){
        console.log('Ok I have draw a circle');
    }
}

const obj = new Circle(1);
// new is an 'operator' and it will return a 'this' pointer to the
object
obj.draw();
-----------------------------------------------------------------------
Output:
PS C:\Users\Sourav Sharma\javascript> node index.js
Ok I have draw a circle
```

Strictly saying there is no difference between these two pattern - so we can use any one of them.

Dynamic nature of Object;

It means that once you create object in JS you can always add new properties and new methods and as well as delete any one of them at any time ;

```
const obj = {
    name: 'Sourav'
}

obj.age = 20;
obj.view = function (){
    console.log('abc');
}

console.log(obj);

Output:
{ name: 'Sourav', age: 20, view: [Function] }

delete obj.age;
delete obj.view;
console.log(obj);

Output:
{ name: 'Sourav' }

// Looking at the constant object it may seem that how we can change this … but fine we are not changing this to some new object ... but we are only adding properties and functions to the current object;

So if try to change:

obj = {}
console.log(obj);

Then there will this in the output:
-------------------------------------------------------------
Output:
obj = {}
    ^
TypeError: Assignment to constant variable.
```

Constructor Properties:

Suppose this example:

```
let x = {};
// Inside JS engine change it to let x= new Object()
// So it is created by Object constructor function;
console.log(x.constructor);
----------------------------------------------------------------
Output:
PS C:\Users\Sourav Sharma\javascript> node index.js
[Function: Object]
```

Functions are objects:

```
// circle which we have created as a function is internally an
object in javascript

function Circle(radius){
    this.radius=radius;
    this.draw = function (){
        console.log('Circle drawn');
    }
}
// Internally this get converted to
//              |
//              |
//              V

const Obj = new Function('radius', `
    this.radius=radius;
    this.draw = function (){
    console.log('Circle drawn');
}
`)

const obj = new Obj(1);

console.log(obj.radius);
```

Also there are some other methods along with this object .
They are apply , call;
Object.call (this: arg, arguments.. explicitly)
Object.apply (this: arg, [arguments ,...]);

Value Type vs Reference Type:

```
 Value type :  Number, String , Boolean, undefined, null , symbols; These are copied by the value
 Reference type : Objects, Functions, Array : These are copied by the reference
let x = 10;
let y = x;
x=20;
console.log(x,y);
-------------------------------------------------------------
Output : 20, 10
```

Now Reference type (or an object there)

```
let x = { value: 10}; // This object is not stored in the variable x but instead x stores the address of the object ... and when we do let y = x then the address of that location is copy to the y
let y = x; // now x and y are pointing to the same location
x.value= 20 ;
console.log(x,y);

Output:
{ value : 20 } , { value : 20 }
```

See with an example: When we pass with reference and value ;

```
let x = 10;
function f(x){
    x++;
}
f(x);
console.log(x);
```

```
Output :  10
```

But if we pass an object - :

```
let x = {value: 10};
function f(x){
    x.value++;
}
f(x);
console.log(x);
-----------------------------------------------------------------
Output:
{ value: 11 }
```

Enumerating properties of an Object :

```
const circle = {
    radius : 1,
    draw(){
         console.log( ' Circle of radius : ', radius);
    }
}
for (let key in circle){
    console.log(key, circle[key]);
}
-----------------------------------------------------------------
OUtput:
radius 1
draw [Function: draw]
```

But on the other hand if we use for (let key of circle) then this will result in an error … The reason is that we can only use them on iterables … such as [array] and [maps] .

So

```
for (let key of circle){
    console.log(key);
}
Output:
for (let key of circle){
                ^
```

```
TypeError: circle is not iterable
```

However we want to do it with ‘for’ we have a method which returns an array and since arrays are iterable so we can use for.

```
for (let key of Object.keys(circle)){
    console.log(key);
}
----------------------------------------------------------------
Output:
PS C:\Users\Sourav Sharma\javascript> node index.js
radius
draw
```

Let’s look on this method more i.e (Object.keys(circle))
Earlier we have seen that this **object is a built-in constructor function**

```
// So somewhere in JS build-in we have constructor function
like this;

function Object() { }

// and whenever we create an object using object literal
syntax
// internally that is converted to this constructor function

// Object literal syntax like this :

const obj = { value : 10 , view : function ( ){
console.log(value);}}
// internally this is translated to a call to the constructor
function
// so that looks like :

obj = new Object ( );

// also we learn that every function in JS is an object ... so
they have properties and methods that we can access

// Beside of object.keys( ) which return a string of array of
the properties and methods of an objects we also have some
other
```

```
// Object.entries() - it returns a { key: value } pair
```

If we wanna see whether a given object has a given method or properties we can use ‘in’ operator.

if(‘radius’ in Circle) console.log(radius);

Cloning an Object :

```
const obj = {
    name: 'Sourav',
    age: 20,
    view(){
        console.log(name, age);
    }
}

const obj1 = {...obj } // ... means we spread.. we are
spreading all the properties and method of the obj in { }

// another way

 const obj2 = Object.assign({color:'red'},obj);

 console.log('obj1 : ', obj1);
 console.log('obj2 : ', obj2);
```

Garbage Collection :

We do not need to explicitly free memory which was allocated earlier… We have JS Garbage Collector and it works itself and clean all those variables which are not in current use and deallocate memory .

String Object and primitive :

```
// String primitive
let name = 'sourav';
console.log(typeof name);

JS engine convert the string primitive to string object
So we can evaluate all the methods that are provided with the
string Object ;
// String Object
const fName = new String('sourav');
```

```
console.log(typeof fName);
```

Example are :

```
let name = 'Sourav Sharma';
console.log('length of  ',name +' is '+ name.length);
console.log('1st character : ',name[0],' and 7th
Character',name[6]);
let store= name.includes('ou');
console.log(store);
-------------------------------------------------------------
OUtput:
PS C:\Users\Sourav Sharma\javascript> node index.js
length of   Sourav Sharma is 13
1st character :  S  and 7th Character
true
```

```
let name = ' Sourav Sharma ';

console.log('index of "a" = ' + name.indexOf('a')); // return
first index from starting;

let name1 = name.replace('Sharma', 'A'); // This returns a new
string ... not make any changes to the original one.
console.log('Replacing Sharma with A =' + name1);

let name2= name.toUpperCase();
console.log('upper Case: =' + name2);

let name3= name.trim(); // trims all the white spaces from
left and right;
console.log(name3);

console.log(name.trimRight());

// Incase if you wanna have single quote in your string;
// Then you have to use escape character;

console.log('my name is \'sourav \' \n and now we are on new
line');
```

```
console.log(name.split('o'));
```

```
----------------------------------------------------------------
OUtput:
index of "a" = 5
Replacing Sharma with A = Sourav A
upper Case: = SOURAV SHARMA
Sourav Sharma
 Sourav Sharma
my name is 'sourav '
 and now we are on new line
[ ' S', 'urav Sharma ' ]
```

Template Literals

```
// Template Literals; like we have object literals {} ,
boolean literals : true, false;
// Suppose you have to send mail to different users: and we
need not to worry about formats:
// like using /n , \' and other

name = 'Sourav';

const msg = `
Hi ${name}
Thanks for supporting cp community!

Regards Rachit Jain.
`;

console.log(msg);
```

```
----------------------------------------------------------------
Output:
Hi Sourav
Thanks for supporting cp community!

Regards Rachit Jain.
```

Date Object in JS:

```
let date =new Date();
console.log(date);
-----------------------------------------------------------
Output:
PS C:\Users\Sourav Sharma\javascript> node index.js
2020-01-03T14:50:15.085Z
```

Array in JS

Constant with the declaration doesn't stop us modifying our array - like add new elements, and delete some elements… It only stop to modify it to some brand new array;

Array in JS are objects:

```
const arr = [1,2,3];

// Insert at end;
arr.push(4,5);

// Insert at beginning;
arr.unshift(-1,0);

// Insert at any position ;
arr.splice(2,0,'a','b');

console.log(arr);
-----------------------------------------------------------
OUtput:
[-1, 0, 'a', 'b', 1, 2,  3, 4,   5]
```

Finding elements (Primitive)

```
const arr = [1,2,3,4,5,4,4,4];

const find = arr.indexOf(4); // returns first index at which 4 is founded
// we can also pass second argument which mean from where to start

console.log(arr.indexOf(4,4));

if(find==-1){
  console.log('Not found!');
}
else {
  console.log('Found at index : ',find);

}

const lastFind = arr.lastIndexOf(4);
console.log(lastFind);

// to check whether an element is without checking for return -1 ;

We have new method;

console.log(arr.includes(1));

console.log(arr.sort());
-----------------------------------------------------------------
Output:
PS C:\Users\Sourav Sharma\javascript> node index.js
5
Found at index :  3
7
true
[
  1, 2, 3, 4,
  4, 4, 4, 5
]
```

Finding elements in Reference Types:

```
const courses = [
  {id: 3, name: 'CS-3003'},
  {id: 4, name: 'CS-3004'}
];

const course = courses.find(function(i){
  return i.name==='CS-3003';
});

console.log(course);
-----------------------------------------------------------------
Output:
PS C:\Users\Sourav Sharma\javascript> node index.js
{ id: 3, name: 'CS-3003' }

// If not found then it will return undefined
-----------------------------------------------------------------

:In case if we used findIndex() It will return index of first
appearance :
And in case we don't find the element then it will return -1;

const course = courses.findIndex(function(i){
  return i.name==='abcdef';
});
console.log(course);
-----------------------------------------------------------------
Output :
-1;
```

Arrow function

Whenever you want to pass a function as a call back function(or an argument to a different function) : like we have done in previous

```
// course.find( function(){ } ) , we can use the arrow function syntax ( => ) see how we can do that:
const courses = [
  {id: 3, name: 'CS-3003'},
  {id: 4, name: 'CS-3004'}
];

const course = courses.findIndex( (obj) => {
  return obj.name==='abcdef';
});
```

If you have a single parameter in the call-back function then even you can remove parentheses from there... so code will be more cleaner and if you don't have any parameter then pass this : ()=> an empty parentheses and if you have one line of code inside and it is returning something then you can remove return keyword from there and { } these also; So your code will appear:

Remove elements from array:

```
const arr = [1, 2, 3, 4];

// Remove from end:
let store = arr.pop();

// Remove from beginning:
store = arr.shift();

// From any position :
arr.splice(1,1);

console.log(arr);

Output:
PS C:\Users\Sourav Sharma\javascript> node index.js
[ 2 ]
```

Emptying an array

```
const arr = [1, 2, 3, 4];

// First thing:
arr = [];

// Second :
arr.length(0);

//Third:
arr.splice(0,arr.length());

// Another loops:
// although it is not good approach
while(arr.length()){
  arr.pop();
}
```

Combining two arrays

```
const arr = [1, 2, 3, 4];
const arr2 = [5,6,7,8,9];


const arrNew = arr.concat(arr2); // Both array remain
unaffected and result is in new array :
console.log(arrNew);

const slice = arrNew.slice(2,4); // [x,y) : WHERE x is
starting index (0 based indexing) and y is excluded :
console.log(slice);
In slice if we don't pass any x,y  arrNew.slice() then the
whole array is copied and if we don't pass the second i.e y
```

```
then whole array from x index (included) will be copied into
new location :

-----------------------------------------------------------------
OUtput:
PS C:\Users\Sourav Sharma\javascript> node index.js
[ 1, 2, 3, 4, 5, 6, 7, 8, 9]
[ 3, 4 ]
```

An important point :

```
const arr = [ {id: 4} ];
const arr2 = [5,6,7,8,9];

Now here we are concatenate the array where it contains object
i.e { id: 1} : and till now we know objects in JS are not
copied but they are passed by reference so the newArr will not
contain the copy of these values but only a reference: So if
we change at original location then changes will reflect at
both locations : because both arr and arrNew both are pointing
to the same location on their 0th index; So If they are
primitive type they are copied by value else by reference;

const arrNew = arr.concat(arr2);

arr[0].id = 'sourav';
console.log(arr);
console.log(arrNew);
-----------------------------------------------------------------
OUtput:
PS C:\Users\Sourav Sharma\javascript> node index.js
[ { id: 'sourav' } ]
[ { id: 'sourav' }, 5, 6, 7, 8, 9 ]
```

A bit more flexible is spread operator :

Spread operator IN (ES6)

```
const arr = [1, 2, 3, 4];
const arr2 = [5, 6, 7, 8, 9];
// Spread operator ... : we are spreading the elements of arr,
arr2
const arrNew = [...arr, 'Sourav Sharma', ...arr2];
// We can also add elements in between easily
console.log(arrNew);

Output :
PS C:\Users\Sourav Sharma\javascript> node index.js
[ 1, 2, 3, 4, 'Sourav Sharma', 5, 6, 7, 8, 9 ]
```

Let's see how to iterate on the array :

Iterate on array

```
const arr = [1, 2, 3, 4];
// We are using
 forEach(  function(element){console.log(element); })
arr.forEach( value => console.log(value));
// If want to do it with index:

arr.forEach( (value, index) => console.log(index,value));
```

Joining array

```
const arr = [1, 2, 3, 4];
const joined = arr.join(' SD ');
console.log(joined);

const msg = 'This is a message';
const str = msg.split(' ');
console.log(str);
```

```
// Now let's join them back :

const strJoin = str.join('-');
console.log(strJoin);


// This technique is useful in urls because url does not
contain white spaces : but a user when search for any thing
for sure he/she leaves white spaces :
```

Sort array

```
const arr = [28,11,34,56,78];
arr.sort();
console.log(arr);

arr.reverse();
console.log(arr);
```

But working with object's is a little bit different :

```
const arr = [
  {id: 1, name: 'Sourav'},
  {id: 2, name: 'Abcdef'}
]
//arr.sort( (a,b) => b.name < a.name ? 0 : -1 )
function sortArray(){
  arr.sort( (a,b) => {
    if(a.name <b.name)return -1;
    if(a.name > b.name)return 1;
    return 0;
  })
  console.log(arr);
}
sortArray();
// see if we change the name of id: 2
arr[0].name ='abcdef';
sortArray();
```

```
-------------------------------------------------------------
OUtput:
PS C:\Users\Sourav Sharma\javascript> node index.js
[ { id: 2, name: 'Abcdef' }, { id: 1, name: 'Sourav' } ]
[ { id: 1, name: 'Sourav' }, { id: 2, name: 'abcdef' } ] //
see Sourav appear before because ASCII(a)> ASCII(S): and if we
want to ignore this we can :
const name1 = arr[0].name.toUpperCase;
const name2 = arr[1].name.toUpperCase;
```

Testing the elements of the array

```
const arr = [ 1, -2, 3, 4, 5];

let allPositive = arr.every( value => value>=0);
console.log(allPositive);

let atLeastOnePositive = arr.some( value => value>=0);
console.log(atLeastOnePositive);
```

Filtering elements

```
const arr = [ 1, -2, 3, 4, 5];

let storeArr = arr.filter( value => value>=0);

console.log(storeArr);
```

Mapping an array

```
const arr = [1, -1, 2, 3];

const allPositive = arr.filter( n => n>=0);
```

```
const obj = allPositive.map( n => ({ value: n}));

console.log(obj);
// Both filter and map returns a new array : they don't modify
the existing one :
// and these functions can be chained :

// arr
//  .filter( n => n>=0)
//  .map( n => ({value:n}))
//  .filter( n => n.value>1);
//  .map( n => n.value);
// So Our final ans will be an array = [2,3];
```

Reducing an array

```
const arr = [1, 1, 2, 3];

let xorValue = arr.reduce((accumulator,currValue)
=>accumulator ^ currValue ,0); // this 0 which is the 2nd
parameter is acting as an initial value for ans = 0;

// If we don't initialize accumulator then it will initialize
with the first element
console.log(xorValue);
```