**27.09.2022**

**Installation**

**Download node js for windows from web**

Node js :- provide environment to run the java script code which is written visual studio code

Download visual code from browser and it provide environment to write the java script code

**Run the code in visual code**

Step1. create the folder and type cmd in the file path

Step2: create folder inside the visual studio and add or dump .js files

Step3: Inorder to run the .js file open terminal and create a new terminal and give command as **node absolute path of .js file**  and click enter ( Make sure auto save option is enabled to save automatically)

**Java Script data types**

Java script uses data types like primitive data type and non primitive data type

Primitive data type:

They are

**28.09.2022**

**Interview Question 1: why variables are loosely typed in JS?**

In java Script variables are loosely typed because unlike in java we don’t define the data type of the variable

Eg: var a = 10

var b = "helo"

Here only variables are declared and initialized but not pointed to any of the data type so it is called as loosely typed

**Interview Question 2: Explicitly converting the data types of the variable?**

Most commonly used and allowed conversions are string 🡪 number and number 🡪 to string

**String 🡪 Number**

var b = "helo" // string

console.log(typeof(b)); // string

console.log(typeof(Number(b))); // number

**Number 🡪 String**

var a = 10 // number

console.log(typeof(a)); // number

console.log(typeof(String(a))); // String

**Interview Question 3: Difference between == and ===**

== compares only the values or literals but not the datatype

=== compares the values an also the datatype of the varaiables

In my project while iterating account number in the active customer list , I will use == to match the account number and

**FOR ==**

var a = 10;

var b = '10'

if (a==b) {

    console.log('both are equal');

}

else{

    console.log('both are not equal');

}

**OUTPUT:**

**both are equal**

**FOR ===**

var a = 10;

var b = '10'

if (a===b) {

    console.log('both are equal');

}

else{

    console.log('both are not equal');

}

**OUTPUT**

**Both are not equal**

var a = '10';

var b = '10'

if (a===b) {

    console.log('both are equal');

}

else{

    console.log('both are not equal');

}

**OUTPUT**

**Both are equal**

**Interview Question 4: Pre Incremental and post incremental**

In pre incremental the values are incremented and then assigned to the variable

In post incremental the values are assigned first and then incremented.

**Interview Question 5: different types of variable:- var, let, const.**

**Var:-**The variable which is declared as var can be re-declared and reinitialized multiple times.

**Let:-** The variable which is declared as let cannot be re-declared but can be reinitialized multiple times

**Const:-** The variable which is declared as const cannot be re-declared and reinitialized.

|  |  |  |
| --- | --- | --- |
| Variable name | Re-declaration | Re- initialization |
| Var | Possible | Possible |
| Let | Possible | Not possible |
| const | Not possible | Not possible |

**JS EXECUTION**

**On browser:**

**1.In line embedded execution**

<!DOCTYPE html>

<html lang="en">

<head>

    <title>Document</title>

</head>

<body>

    <script>

        document.write('hello everyone') 🡪 to write anything in the body of the web page

    </script>

</body>

</html>

**2.external line embedded execution**

**This can be used to add the .js file and can debug the code in the browser**

<!DOCTYPE html>

<html lang="en">

<head>

    <title>Document</title>

</head>

<body>

    <script src="./Basic/DataType.js"></script>

</body>

</html>

we can add multiple scripts in the script tag

To select path of .js file

./ 🡪 suggests the file in the same folder

If u want to move on to next folder use

../🡪 navigates to the preceding or ancestor

**OFF Browser: No browsers will be involved for code execution**

**Program execution in memory**

In initial stages when java script code is executed an object of window will be created and global variable also been created. The global variable is called this and as per global level, window should be strictly equal to this

There are 2 phases will be created

1. Creation phase
2. Execution phase

**1.Creation phase**

In creation phase the memory will be allotted for all variables as memory address as a variable name and will be initialized as undefined and in case of functions , a memory will be allotted as function name and implementation of the function is stored.

**2.Execution phase**

In execution phase all undefined variables are overridden with the actual values and in functions if it is invoked it will be executed.

**Function execution in memory**

**In intial stages** When the function is executed, a function execution context is created above the global execution context in call stack. The two type of phases performed here are

**1.variable hosting**

The memory will be allotted for all variables as memory address as a variable name and will be initialized as undefined and In execution phase all undefined variables are overridden with the actual values.

**2.Function hosting**

Initially the implementation of the function is stored in the memory which address is a function name and when it is invoked it will be executed.

Finally after the execution of the function the function execution context will be teared down from the call stack.

**Functions**

Functions are the set of codes that can be reused to multiple times whenever it is invoked.

There are 4 types of functions

1. Standard function/ function declaration
2. Function expression
3. Immediate Invocable Function (IIF)
4. Arrow Function

**3.Immediate Invocable Function(IIF)**

Syntax: (function)(invoking usig parameters)

Only one IIF function can be

(function read(a,b){

    console.log(a+b);

})(5,6)

(function write(a,b){

    console.log(a+b);

})(5,6)

**Output as follows:**

PS C:\Users\Faster\Desktop\JavaS\_sdet40> node Functions\ImmediateInvokableFunction.js

**11**

C:\Users\Faster\Desktop\JavaS\_sdet40\Functions\ImmediateInvokableFunction.js:4

(function write(a,b){

^

TypeError: (intermediate value)(...) is not a function

at Object.<anonymous> (C:\Users\Faster\Desktop\JavaS\_sdet40\Functions\ImmediateInvokableFunction.js:4:1)

at Module.\_compile (node:internal/modules/cjs/loader:1126:14)

at Object.Module.\_extensions..js (node:internal/modules/cjs/loader:1180:10)

at Module.load (node:internal/modules/cjs/loader:1004:32)

at Function.Module.\_load (node:internal/modules/cjs/loader:839:12)

at Function.executeUserEntryPoint [as runMain] (node:internal/modules/run\_main:81:12)

at node:internal/main/run\_main\_module:17:47

**Interview Question: what is argumented array?**

when the invoking array contains more than the variable specified the argument array has been created.

function oper(a,b){

    console.log(a+b);

    console.log(arguments)

}

oper(9,8,1)

Output:

24

[Arguments] { '0': 9, '1': 8, '2': 7, '3': 6, '4': 5 }

We can fetch particular array elements using the arguments[index]

**Use of arguments array in project:** When the user fetches the multiple data but using only limited datas then he may go for the argumented array. In above oper(9,8,1) is the 3 inputs but he is making use of only first two then he can go for arguments array.

**Interview Question: Difference between break and continue**

**Break :**  When the break keyword is executed it will come out of the entire loop and execute the statements which is present outside the loop

var a = [2, 'hello', 34]

for (let index = 0; index < a.length; index++) {

    if (index<2) {

        console.log(index);

        break

    }

    console.log('outside break if');

}

Output:

0

**In the** above

**Continue:**

It will execute only the particular block of codes for the expected condition

var a = [2, 'hello', 34]

for (let index = 0; index < a.length; index++) {

    if (index<2) {

        console.log(index);

        continue

    }

    console.log('outside continue if');

}

In the above when the control comes inside the for and when it goes to if statement only if

**30.09.2022**

**Arrays Declaration: Its of 2 types**

**Var arr = new Array (elements) 🡪 Type 1**

**Var arr = [elements] 🡪 Type 2**

**Var arr = new Array (elements) 🡪 Type 1**

var arr = new Array(1, 'thaju' , 2.333) // declaration type 1

for (let index = 0; index < arr.length; index++) {

    console.log(index+"    "+arr[index]);

}

console.log("array size=" +arr.length);

// output:

// 0    1

// 1    thaju

// 2    2.333

// array size=3

var arr  = new Array(4) // declaration type 1 🡪 taking 4 as an size

for (let index = 0; index < arr.length; index++) {

    console.log(index+"    "+arr[index]);

}

console.log("array size=" +arr.length);

// output:

// 0    undefined

// 1    undefined

// 2    undefined

// 3    undefined

// array size=4

var arr  = new Array('hello') // declaration type 1

for (let index = 0; index < arr.length; index++) {

    console.log(index+"    "+arr[index]);

}

console.log("array size=" +arr.length);

// output:-

// 0    hello

// array size=1

In type 1 declaration the disadvantage is that when we enter only one element its that as an size instead of the element.

**Var arr = [elements] 🡪 Type 2**

var arr = [1,'hello',3.33] // declaration type 2

for (let index = 0; index < arr.length; index++) {

    console.log(index+"    "+arr[index]);

}

console.log("array size=" +arr.length);

// output

// 0    1

// 1    hello

// 2    3.33

// array size=3

var arr = [3] // declaration type 2 🡪 taking 3 as an element

for (let index = 0; index < arr.length; index++) {

    console.log(index+"    "+arr[index]);

}

console.log("array size=" +arr.length);

// output

// 0    3

// array size=1

**Diff b/w for loop, for –of loop, for in loop, for – each method**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **For** | **For Of** | **For In** | **For Each forEach(()=>{})** |
| What? | Loop | Loop | Loop | Method |
| Return | Index and value | Value/element | Index and value | index Value/element |
| Unassigned | Do not ignore | Do not ignore | Ignores | Ignores |
| Break and continue | Can be implemented | Can be implemented | Can be implemented | Cannot be implemented bcoz it’s a method |

var arr = [1, ,'thaju', , 2.333] 🡪 array has empty values and will see results in different scenarios like for, forin, for of, for each

for (let index = 0; index < arr.length; index++) {

    console.log(index+"    "+arr[index]);

}

// Output:-

// 0    1

// 1    undefined

// 2    thaju

// 3    undefined

// 4    2.333

for (const index in arr) {

    console.log(index+"    "+arr[index]);

}

// Output:-

// 0    1

// 2    thaju

// 4    2.333

for (const index of arr) {

    console.log(index);

}

// Output:-

// 1

// undefined

// thaju

// undefined

// 2.333

arr.forEach((element, index, arr) => {

    console.log(index+"       "+element+"     "+arr);

});

// Output:-

// 0       1         1,,thaju,,2.333

// 2       thaju     1,,thaju,,2.333

// 4       2.333     1,,thaju,,2.333

**Array predefined methods:**

**1.Concat**

var arr = [2, 3.33, 'engine']

var arr1 = ['thaju']

console.log(arr);

console.log(arr.concat(arr1));

console.log(arr);

// output:-

// [ 2, 3.33, 'engine' ]

// [ 2, 3.33, 'engine', 'thaju' ]

// [ 2, 3.33, 'engine' ] 🡪 no changes will be done in the initial array

**2.Push**

var arr = [2, 3.33, 'engine']

var arr1 = 'thaju'

console.log(arr);

console.log(arr.push(arr1));

console.log(arr);

// Output:-

// [ 2, 3.33, 'engine' ]

// 4                                    --> length of the array

// [ 2, 3.33, 'engine', 'thaju' ]

**3.Pop**

var arr = [2, 3.33, 'engine']

console.log(arr);

console.log(arr.pop()); --> deletes last element or last element is removed

console.log(arr);

// output:-

// [ 2, 3.33, 'engine' ]

// engine               --> deleted element or removed element

// [ 2, 3.33 ]

**4.Shift**

var arr = [1, 'hello', 3]

console.log(arr);

console.log(arr.shift());  // shift()--> removes the 1st element in the array

console.log(arr);

// Output:-

// [ 1, 'hello', 3 ]

// 1                --> 1st element

// [ 'hello', 3 ]   --> 1st element is removed

**5.Unshift**

var arr = [1, 'hello', 3]

var arr1 = "ding"

console.log(arr);

console.log(arr.unshift(arr1));  // add the element to the 1st index

console.log(arr);

// Output:-

// [ 1, 'hello', 3 ]

// 4                        --> length of the array

// [ 'ding', 1, 'hello', 3 ]

**6. Every**

var arr = [1, 2, 4, 6]

console.log(arr);

console.log(arr.every((element)=>{

    return element<5

}))                                 // every(element) --> checks for all the element is lesser than 5 or not. If one fails all fails

console.log(arr);

// Output:-

// [ 1, 2, 'hello', 6 ]

// false

// [ 1, 2, 'hello', 6 ]

**7.Some**

var arr = [2, 9, 8, 6]

console.log(arr);

console.log(arr.some((element)=>{

    return element<5

}))                                 // every(element) --> checks for all the element is lesser than 5 or not. If atleast one pass all passes

console.log(arr);

// Output:-

// [ 2, 9, 8, 6 ]

// true

// [ 2, 9, 8, 6 ]

**8.Splice**

var arr = [1, 'hello', 3]

console.log(arr);

console.log(arr.splice(1,1,"hi there")); // (startingIndex, DeleteCount, ElementToBeAdded)

console.log(arr);

// Output:-

// [ 1, 'hello', 3 ]

// [ 'hello' ]          --> deleted element

// [ 1, 'hi there', 3 ]

console.log(arr);

console.log(arr.splice(1,0,"hi there"));

console.log(arr);

// Output:-

// [ 1, 'hi there', 3 ]

// []                               --> no element is deleted so empty array

// [ 1, 'hi there', 'hi there', 3 ]

It starts from the starting index and delete the element based on the count and add replaces with the new element.

**9.Slice**

var arr = [2.2, 'hello', 89, 'ding', 9786]

console.log(arr);

console.log(arr.slice(1,4)); // (1,4) --> (StartingIndex, endingIndex) and leaves the end index number

console.log(arr);

// Output:-

// [ 2.2, 'hello', 89, 'ding', 9786 ]

// [ 'hello', 89, 'ding' ]              --> The trimmed part

// [ 2.2, 'hello', 89, 'ding', 9786 ]

**10.Index of**

var arr = [2, 9, 8, 9,6]

console.log(arr.indexOf(9,2)); // indexOf(9,2)--> (element to be searched, From index number to start searching)

console.log(arr);

// output:-

// 3     --> result gives the index of the searched element

// [ 2, 9, 8, 9, 6 ]

**11.Last Index of**

var arr = [1, 2, 'hello',2, 3.45]

console.log(arr);

console.log(arr.lastIndexOf(2,4)); // (2,4) 2--> search for this element $ 4 --> from this index number to left

console.log(arr);

// output:-

// [ 1, 2, 'hello', 2, 3.45 ]

// 3

// [ 1, 2, 'hello', 2, 3.45 ]

**12.Reverse**

var arr = [1, 2, 'hello',8, 3.45]

console.log(arr);

console.log(arr.reverse()); // --> reverse the array elements

console.log(arr);

// output:-

// [ 1, 2, 'hello', 8, 3.45 ]

// [ 3.45, 8, 'hello', 2, 1 ]

// [ 3.45, 8, 'hello', 2, 1 ]  ---> changes are committed

**13.Includes**

var arr = [1, 2, 'hello',8, 3.45]

console.log(arr);

console.log(arr.includes(2, 0)); // (2,0) 2--> element to be searched , 0 --> index from where to start searching

console.log(arr);

// output:-

// [ 1, 2, 'hello', 8, 3.45 ]

// true

// [ 1, 2, 'hello', 8, 3.45 ]

var arr = [1, 2, 'hello',8, 3.45]

console.log(arr);

console.log(arr.includes(2)); // if does not specify it automatically takes 0 index

console.log(arr);

// output:-

// hello', 8, 3.45 ]

// true

// [ 1, 2, 'hello', 8, 3.45 ]

**14.Join**

var arr = [1, 2, 'hello',8, 3.45]

console.log(arr);

console.log(arr.join("hi")); // concatinate to all the elements in the array

console.log(arr);

// output:-

// [ 1, 2, 'hello', 8, 3.45 ]

// 1hi2hihellohi8hi3.45

// [ 1, 2, 'hello', 8, 3.45 ]

**15.Call back fuctions**

var arr = [1, 2, 'hello',8, 3.45]

arr.forEach((element, index) => {

    console.log(index+"  "+element);

    console.log(element\*2);

});

console.log(arr);

// Output:-

// 0  1

// 2

// 1  2

// 4

// 2  hello

// NaN

// 3  8

// 16

// 4  3.45

// 6.9

// [ 1, 2, 'hello', 8, 3.45 ]

arr.forEach((element, index) => {

    console.log(element\*10);

});

console.log(arr);

//output:-

// 10

// 20

// NaN

// 80

// 34.5

// [ 1, 2, 'hello', 8, 3.45 ]

**16.Map**

var arr = [1, 2, 'hello',8, 3.45]

console.log(arr);

console.log(arr.map((value, index)=>{

    return value\*2

}));

console.log(arr);

// output:-

// [ 1, 2, 'hello', 8, 3.45 ]

// [ 2, 4, NaN, 16, 6.9 ]

// [ 1, 2, 'hello', 8, 3.45 ]

**17.filter**

var arr = [1, 2, 'hello',8, 3.45]

console.log(arr);

console.log(arr.filter((element, index)=>{

    if(element>3)

    return element

}));

console.log(arr);

// Output:-

// [ 1, 2, 'hello', 8, 3.45 ]

// [ 8, 3.45 ]                  ---> return the elments as per the if condition

// [ 1, 2, 'hello', 8, 3.45 ]

**18.Sort**

var arr = [10, 2, 'hello',8, 3.45]

console.log(arr);

console.log(arr.sort((a,b)=>{

    return a-b                  // arrange the array in ascending order

}));

console.log(arr);

// output:-

// [ 10, 2, 'hello', 8, 3.45 ]

// [ 2, 3.45, 8, 10, 'hello' ]

// [ 2, 3.45, 8, 10, 'hello' ]

console.log(arr);

console.log(arr.sort((a,b)=>{

    return b-a                  // arrange the array in descending order

}));

console.log(arr);

// output:-

// [ 2, 3.45, 8, 10, 'hello' ]

// [ 10, 8, 3.45, 2, 'hello' ]

// [ 10, 8, 3.45, 2, 'hello' ]

**19.Reduce**

var arr = [100, 56, 73, 64]

console.log(arr);

function nam(total, number) {

    return total-number

}

console.log(arr.reduce(nam)); //--> reduce the array element based on the operator sign

console.log(arr);

// output:-

// [ 100, 56, 73, 64 ]

// -93

// [ 100, 56, 73, 64 ]

**20.Right reduce**

var arr = [23, 56,32, 67]

console.log(arr);

function func(total, number) {

    return total-number

}

console.log(arr.reduceRight(func)); //--> start reducing from the right to left

console.log(arr);

// output:-

// [ 23, 56, 32, 67 ]

// -44

// [ 23, 56, 32, 67 ]

**ARRAY PREDEFINED METHODS COMPARISON**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **S.No** | **Method name** | **Definition** | **Syntax** | **Return Type** | **Modifies the actual array** |
| **1** | **Concat** | Combines two or more arrays. | Array.concat(array1) | Array of elements | No |
| **2** | **Push** | Add new elements to the end of an array, and returns the new length of the array. | Array.push(element) | Length of an array | Yes |
| **3** | **Pop** | Removes the last element from an array and returns it. If the array is empty, undefined is returned | Array.pop() | Deleted element | Yes |
| **4** | **Shift** | Removes the first element from an array and returns it. If the array is empty, undefined is returned | Array.shift() | Removed 1st element | Yes |
| **5** | **Unshift** | Inserts new elements at the start of an array, and returns the new length of the array. | Array.unshift(element) | Length of an array | Yes |
| **6** | **Every** | Determines whether all the elements of an array satisfy the condition If one fails returns false. | (arr.every((element)=>{      return element<5  })) | Boolean values | No |
| **7** | **Some** | Determines whether any one of the elements of an array satisfy the condition If atleast one passes returns true. | (arr.some((element)=>{      return element<5  })) | Boolean values | No |
| **8** | **Slice** | Returns a copy of a section of an array | Array.slice(Startindex, EndIndex) | Delted elements in the form of an array | No |
| **9** | **Splice** | Removes elements from an array and, if necessary, inserts new elements in their place, returning the deleted elements. | Array.splice(StartIndex, deleteCount,”element to be added if”) | Delted elements in the form of an array | Yes |
| **10** | **Index of** | Returns the index of the first occurrence of a value in an array, or -1 if it is not present. | Array.indexof(element to be searched, Starting index) | Index of the searched element | No |
| **11** | **Last index of** | Returns the index of the last occurrence of a specified value in an array, or -1 if it is not present. | Array.lastindexOf(element to be searched, last Index) | Index of the searched element | No |
| **12** | **Reverse** | Reverses the elements in an array | Array.reverse() | Array of elements | Yes |
| **13** | **Includes** | Checks whether the element is present in array or not and returns Boolean values | Array.includes(element to be searched, Starting index) | Boolean values | No |
| **14** | **Join** | Adds all the elements of an array into a string and concatenate the element with all the values in the array | array.join(element) | string | No |
| **15** | **Call back function** | Performs the specified action for each element in an array. | arr.forEach((element, index) => {      console.log(index+"  "+element);      console.log(element\*2);  }); | element | No |
| **16** | **Map** | Calls a defined callback function on each element of an array, and returns an array that contains the results. | array.map(value, index) | Array of elements | No |
| **17** | **Filter** | Returns the elements of an array that meet the condition specified in a callback function. | (arr.filter((element, index)=>{      if(element>3)      return element  })); | Array of elements | No |
| **18** | **Sort** | Sort the array to either ascending order or descending order based on the caller function | arr.sort((a,b)=>{      return a-b                  // arrange the array in ascending order  })  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  arr.sort((a,b)=>{      return b-a                  // arrange the array in descending order  }) | Array of elements | Yes |
| **19** | **Reduce** | Based on call back function it convert an array into an accumulated array performing the operation from left to right | function nam(total, number) {      return total-number  }  console.log(arr.reduce(nam)); | String or element | No |
| **20** | **Right reduce** | Based on call back function it convert an array into an accumulated array performing the operation from right to left | function func(total, number) {      return total-number  }  console.log(arr.reduceRight(func)) | String or element | No |

**STRING PREDEFINED METHODS**

1. **Conact**

var a = "hello everyone"

console.log(a);

console.log(a.concat("hi")); // the string is added to the last in the actual string

console.log(a);

// output:-

// hello everyone

// hello everyonehi

// hello everyone

1. **CharAtPosition**
2. var a = "thajudeen"
3. console.log(a);
4. console.log(a.charAt(3));
5. console.log(a);
6. // output:-
7. // thajudeen
8. // j
9. // thajudeen
10. **Ends with**
11. var a  = "divine"
12. console.log(a);
13. console.log(a.endsWith("e"));
14. console.log(a);
15. console.log(a.endsWith("a"));
16. // output:-
17. // divine
18. // true
19. // divine
20. // false
21. **Includes**

var q  = "severity"

console.log(q);

console.log(q.includes("e"));

console.log(q);

// output:-

// severity

// true

// severity

1. **Index Of**

var a = "dramatict"

console.log(a);

console.log(a.indexOf("t", 2)); // Returns the position of the first occurrence of a substring.

console.log(a);

// output:-

// dramatict

// 5

// dramatict

1. **Last Index Of**

var a = "thajudeen"

console.log(a);

console.log(a.lastIndexOf("j", 7)); // Returns the last occurrence of a substring in the string.

console.log(a);

// Output:-

// thajudeen

// 3

// thajudeen

1. **Length**

var a = 'thajudeen'

console.log(a);

console.log(a.length); // Returns the length of a String .

// output:-

// thajudeen

// 9

1. **Repeat**

var a = "hello"

console.log(a);

console.log(a.repeat(4)); // repeat the string based on the specified count

console.log(a);

// output:\_

// hello

// hellohellohellohello

// hello

1. **Replace**

var a = "energetic"

console.log(a);

console.log(a.replace("tic","out")); // replace a particular substring with another string

console.log(a);

// output:-

// energetic

// energeout

// energetic

console.log(a.replace("e","a"));

console.log(a);

// output:

// anergetic

// energetic

1. **Separator**

var a= "welcome to heaven"

console.log(a);

console.log(a.split(" ")); //Split a string into substrings using the specified separator and return them as an array.

console.log(a);

// output:-

// welcome to heaven

// [ 'welcome', 'to', 'heaven' ]

// welcome to heaven

1. **Trim**

var a = " knowledge is power "

console.log(a);

console.log(a.trim()); // --> trim the spaces at front and back

console.log(a);

// output:-

//  knowledge is power

// knowledge is power

//  knowledge is power

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **S.No** | **Method name** | **Definition** | **Syntax** | **Return type** | **Modifies the actual string** |
| **1** | **Concat** | Combines two or more strings. | Array.concat(“string”) | Returns modified string | No |
| **2** | **CharAt** | Returns the character at the specified index. | String.CharAt(index) | Returns character | No |
| **3** | **Ends with** | Checks whether the string ends with the specified string. Returns Boolean values | String.endswith(“string”) | Boolean values | No |
| **4** | **Includes** | Check whether the string contains the specified string, return boolean values | String.includes(“string”) | Boolean values | No |
| **5** | **Index of** | Returns the position of the first occurrence of a substring. | String.indexof(“string to be searched”, starting index) | number | No |
| **6** | **Last index of** | Returns the last occurrence of a substring in the string. | String.lastindexof(“string to be searched”, last index) | number | No |
| **7** | **Length** | Returns the length of a String . | String.length() | Length in number | No |
| **8** | **Repeat** | repeat the string based on the specified count | String.repeat(count) | string | No |
| **9** | **Replace** | replace a particular substring with another string | String.replace(substring to be replaced, replaced with string) | string | No |
| **10** | **separator** | Split a string into substrings using the specified separator and return them as an array. | String.split(“based on character”) | Array of elements | No |
| **11** | **Trim** | Removes the leading and trailing white space | String.trim() | string | No |

**Date Method**

Its used to get the system date

It has predefined methods like

1. toDateString()
2. getDate()
3. getDay()
4. getFullYear()
5. getMonth()

var a = new Date() //--🡪 Date object should be created 1st it has 4 inbuilt methods

console.log(a.toDateString()); //🡪 gives the complete date as Sun Oct 02 2022

console.log("today's date="+ a.getDate()); // fetches the today’s date

console.log(a.getDay()); // fetches today’s day as in index eg Sunday=0, Monday=1 etc

console.log(a.getFullYear()); // fetches the full year

console.log(a.getMonth()); // fetches month also in index eg:- 1st month= 0, etc

// output:-

// Sun Oct 02 2022

// today's date=2

// 0

// 2022

// 9

**Math Method**

This math method is used to perform operation on numbers

It has inbuilt methods like

1. abs 🡪 Returns the absolute value of a number (the value without regard to whether it is positive or negative).
2. Round 🡪 Returns a supplied numeric expression rounded to the nearest integer.
3. Min 🡪 Returns the smaller of a set of supplied numeric expressions
4. Max🡪 Returns the larger of a set of supplied numeric expressions.
5. Floor 🡪 Returns the greatest integer less than or equal to its numeric argument
6. Ceil 🡪 Returns the smallest integer greater than or equal to its numeric argument.
7. Random 🡪Returns a pseudorandom number between 0 and 1.

console.log(Math.abs(-4));

console.log(Math.abs(3));

console.log(Math.round(37.34567));

console.log(Math.round(5.9876789));

console.log(Math.min(7.5));

console.log(Math.max(5.5));

console.log(Math.floor(9.975678));

console.log(Math.ceil(6.2345678));

console.log(Math.random());

// output:-

// 4

// 3

// 37

// 6

// 7.5

// 5.5

// 9

// 7

// 0.1646136891388046

**03.10.2022**

**Object declaration**

Objects can be declared in 5 ways

1. **Object literals**

syntax:- var obj\_ref = { prop1:key1, prop2:key2}

var personalDetails = {

    name:"Thajudeen",

    Age:25,

    Address:{

        streetName:"karur main road",

        doorNo:"61/70",

        district:"tiruppur"

    },

    qual:function(){

        return "B.E"

    }

}

// console.log(personalDetails); -🡪 prints the entire object

// personalDetails.maritalStatus="single" 🡪 add the retrieve the property

// console.log(personalDetails);

// delete personalDetails.maritalStatus 🡪 delete the particular property

// console.log(personalDetails);

// console.log(personalDetails.Address);

personalDetails.Address.languagesKnown=["english", "hindi","kannada"] 🡪 adding the property

console.log(personalDetails.Address.streetName); 🡪 retrieving the property which is inside an object

// console.log(personalDetails);

console.log(personalDetails.qual()); 🡪 invoking the function

console.log(personalDetails.Address.languagesKnown[2]); 🡪 retrieving the array element value

1. **New object**

// Type 2:

// syntax :

// var obj\_ref  = new Object()

// obj\_ref.prop1 = value1

// obj\_ref.prop2 = value2

var digitalLibrary = new Object()

digitalLibrary.languageBooks={Tamil:"PS-1", English:"Novels", Arabic:["scripts", "literatures"]}

digitalLibrary.authorStandard='international'

digitalLibrary.cost=function(){return "Rs 8000/-"}

console.log(digitalLibrary.languageBooks.Arabic[1]);

console.log(digitalLibrary.cost());

1. **Constructor method**

// Syntax:

// function ref\_name(prop1, prop2, prop3,..){

//     this.prop1=prop1

//     this.prop2=prop2

//     this.prop3=prop3

// }

// var obj\_ref1 = new ref\_name(value1, value2, value3,..)

// var obj\_ref2 = new ref\_name(value1, value2, value3,..)

function personalDetails( name,age,address,qual){

    this.name=name

    this.age=age

    this.address=address

    this.qual=function(){return "B.E"}

}

var PersonalDetails1 = new personalDetails("Thajudeen","25",{streetName:"karur main road", doorNo:"61/70", district:"tiruppur"})

PersonalDetails1.languageKnown=["english","Tamil"]

console.log(PersonalDetails1);

console.log(PersonalDetails1.languageKnown[1]);

1. **Class [ES6 feature]**

// syntax:

// class obj\_ref

// {

//     constructor(prop1,prop2,prop3,..)

 //    {

        // this.prop1=prop1

        // this.prop2=prop2

        // this.prop3=prop3

//     }

// }

// var ref\_name = new obj\_ref(value1, value2, value3,..)

class personalDetails

{

    constructor(name,age,address,qual,languageKnown)

    {

        this.name=name

        this.age=age

        this.address=address

        this.qual=function(){return "B.E"}

    }

}

var PersonalDetails1 = new personalDetails("Thajudeen","25",{streetName:"karur main road", doorNo:"61/70", district:"tiruppur"})

console.log(PersonalDetails1.qual());

1. **Object.create method [ES6 feature]**

var bookStore = {

    bookName:"Novels",

    price:"8000",

    status:"available",

    publication:["international","local","global"],

    standard:function(){

        return this.bookName+"  "+this.price+"  "+this.status

        }

}

var bookStore1 = Object.create(bookStore)

bookStore1.bookName="literature"

bookStore1.price="10000"

bookStore1.status="Unavailable"

console.log(bookStore1.standard());

console.log(bookStore.standard());

console.log(bookStore1.status);

console.log(bookStore.status);

**Method overriding using Object.create method**

var bookStore = {

    bookName:"Novels",

    price:"8000",

    status:"available",

    publication:["international","local","global"],

    standard:function(){

        return this.bookName+"  "+this.price+"  "+this.status

        }

}

var bookStore1 = Object.create(bookStore)

bookStore1.bookName="literature"

bookStore1.price="10000"

bookStore1.status="Unavailable"

console.log(bookStore1.standard());

console.log(bookStore.standard());

console.log(bookStore1.status);

console.log(bookStore.status);

**Method overriding using prototype:**

var personalDetails={

    name:"Thajudeen",

    age:"25",

    address:{

        streetName:"karur main road",

        doorNo:"61/70"

    },

    qual:function(){

        return "B.E"

    }

}

var personalDetails1 = {}

personalDetails1.\_\_proto\_\_ = personalDetails

personalDetails1.name="faiz"

personalDetails1.age="8"

console.log(personalDetails1)

**Method overloading:**

It is not possible in case of java script because whenever we create a functions of same name the latest functions always overrides the old one. So in that case overloading cannot be achieved in java script.

**Array destructuring**

**Object destructuring**

**Classes:**

This keyword in non-static method of the class points to the non-static members of the class “biodata”

This keyword in static method of the class points to the static members of the class “biodata”

This keyword inside the static methods points to all static variables

This keyword inside the non static methods points to all non static variables

* accessing non static variable 🡪 obj\_ref.variable name
* accessing static variable 🡪 classname.variable name
* invoking the non-static method 🡪 obj\_ref.function name
* invoking the static method 🡪 classname.function name
* accessing static variable inside non static method

using classname.variable name we can access static variable inside non static method

class netflix{

    constructor(movie, category,languages){

        this.movie = movie

        this.category = category

        this.languages = languages

    }

    static a  = 10

    b=20

    static duno = "bush"

    watch(){

        console.log(netflix.a); // accessing static variable inside non static method

    }

}

netflix.wiki()

* accessing non static variable inside non static method

using this keyword which loads all the non static variable inside the non static method

class netflix{

    constructor(movie, category,languages){

        this.movie = movie

        this.category = category

        this.languages = languages

    }

    static a  = 10

    b=20

    static duno = "bush"

    watch(){

        console.log(this.b);  // accessing non static variable inside non static method

    }

}

netflix.wiki()

* accessing static variable inside static method

using this keyword which loads all static variable inside the static method

class netflix{

    constructor(movie, category,languages){

        this.movie = movie

        this.category = category

        this.languages = languages

    }

    static a  = 10

    b=20

    static duno = "bush"

    static wiki(){

        console.log(this.a); // accessing static variable inside static methods

    }

}

netflix.wiki()

* accessing non static variable inside static method

Inorder to access non static variable inside static method an object has to be created with the class name

Var variable\_name = new classname() 🡪 instantiation

class netflix{

    constructor(movie, category,languages){

        this.movie = movie

        this.category = category

        this.languages = languages

    }

    static a  = 10

    b=20

    static duno = "bush"

    static wiki(){

        var x = new netflix() // create an object to access non static variable inside static method

        console.log(x.b); // access through the reference\_variable.variable name

    }

}

netflix.wiki()

* **accessing non static method inside non static method**

class netflix{

    constructor(movie, category,languages){

        this.movie = movie

        this.category = category

        this.languages = languages

    }

    static a  = 10

    b=20

    static duno = "bush"

    watch(){

        console.log(netflix.a); // accessing static variable inside non static method

        console.log(this.b);  // accessing non static variable inside non static method

    }

    demo(){ // creating one more non static method

        this.watch(); // calling non static method

    }

}

var screen1 = new netflix("vikram","thriller",["tamil","telugu","english"])

screen1.demo();

* **accessing static method inside static method**

class netflix{

    constructor(movie, category,languages){

        this.movie = movie

        this.category = category

        this.languages = languages

    }

    static a  = 10

    b=20

    static duno = "bush"

    static wiki(){

        console.log("inside static method");

    }

    static run(){

        this.wiki() // accessing static method inside static method

    }

}

netflix.run()

* **accessing static method inside non static method**

class netflix{

    constructor(movie, category,languages){

        this.movie = movie

        this.category = category

        this.languages = languages

    }

    static a  = 10

    b=20

    static duno = "bush"

    static wiki(){

        console.log("inside static method");

    }

    demo(){

        netflix.wiki()// accessing static method inside non static method

    }

}

var screen1 = new netflix("vikram","thriller",["tamil","telugu","english"])

screen1.demo();

* **accessing non static method inside static method**

class netflix{

    constructor(movie, category,languages){

        this.movie = movie

        this.category = category

        this.languages = languages

    }

    static a  = 10

    b=20

    static duno = "bush"

    watch(){

        console.log("inside non static method");

    }

    static run(obj\_ref){

        obj\_ref.watch() // accessing non static method inside static method using obj\_ref pass

    }

}

var screen1 = new netflix("vikram","thriller",["tamil","telugu","english"])

netflix.run(screen1)

**Rest and spread parameter**

**Rest parameter**

function bike (brand,model,...otherDetails){

    console.log(otherDetails);

}

bike("BMW","2022","200cc","TDSI engine","20KMPL","1lakh")

var {name,type,...addmenus}={

    name:"Rice",

    type:"veg",

    spicy:"medium",

    style:"andhra",

    variety:["var1","var2","var3"]

}

console.log(addmenus);

It uses “…” which should be strictly used in the last variable

Its stores the rest datas to the last arguments and creates an array. The difference between argument array and rest array is that in arguments array in built methods cannot be used where in rest array is possible.

**Rules:**

A function can have only one rest parameter

The rest parameter symbol must be used for the last argument in the function.

**Spread parameter**

* The spread operator allows us to spread the value of an array (or any iterable)

**Difference b/w rest and spread parameter**

Rest operator puts the rest of some specific user specified values into a JavaScript array

But the spread syntax expands iterables into individual elements**.**

**07.10.2022**

**Default parameter**

function sum(a,b){

    return a+b

}

console.log(sum(10)); // here b is undefined so we get output as not an number NaN

output:- Nan

function sum(a,b=2){

    return a+b

}

console.log(sum(10)); // here b is already initialized

output:- 12

function sum(a,b=2){

    return a+b

}

console.log(sum(10,4));  // b which is initialized as 2 will be overridden as 4

output:- 14

function sum(a=1,b){

    return a+b

}

console.log(sum(10)) // 10 will be passed only to a not to b. hence it’s not possible

class fun{

    constructor(a,b){

        console.log(a+b);

    }

}

new fun(10)

**Null is a falsy value**

const data = null

if (data) {

    console.log(" null");

} else {

    console.log("not null");

}