CLASSTEAM | Page No.
Date / /

JS - BASICS - 3

In-built Objects

1. Math

Math is a built-in object that has properties and methods for mathematical constants and functions.

It's not a function object.

Math works with the Number type. It doesn't work with BigInt.

Static Methods are :-

→ Math.random() :- generates a random number
→ Math.abs() :- Returns an absolute value of number.
→ Math.cbrt() :- Returns the cube root of number.
↳ ex:- Math.cbrt(64);
↳ 4 Ans

→ Math.ceil() :- Returns smaller integer greater than or equal to a given number.

ex:- Math.ceil(7.004) → 8
Math.ceil(-5.004) → 5

→ Math.floor() :- Returns largest integer less than or equal to x.

→ Math.max() → Returns maximum number.
→ Math.min() → Returns minimum number.
→ Math.pow() → Return base x to the exponent power y. (i.e. x^y)

→ Math.round() → Returns roundoff number.
→ Math.sqrt() → Return sqrt.

For More Methods, Go to MDN Site.

2. String

The String object is used to represent and manipulate a sequence of characters.

Here, we will discuss String object.

ex:-
let name = new String('Ankit');

① typeof(name);
↓
gives 'object'.

Note:-
String
↓
Primitive
↓
let name = 'Ankit';
This is primitive String.

typeof(name);
↓gives
'string'.

Note:- We use here String Constructor function to make object.

ex:- let name = new String('Ankit');
↓
Constructor.

Methods are:-

let lastname = "Babbar";
let firstname = new String("Love");

→ console.log(lastname.length) → 6
→ console.log(lastname[0]); → B
→ console.log(lastname.includes('Ba')); → True
→ lastname.startsWith('Babb'); → True
→ lastname.startsWith('love'); → False
→ lastname.endsWith('ar'); → True
→ lastname.indexOf('B'); → 0
→ lastname.toUpperCase(); → BABBAR.
→ lastname.toLowerCase(); → babbar
→ lastname.trim(); → Removes whitespaces from both ends.
→ lastname.replace('Babb', 'car'); → carar

→ Split() :-

```
let message = 'This is my first message';
let words = message.split(' ');
                          space split.
console.log(words);
```

Output:- ['This', 'is', 'my', 'first', 'message']

```
let words = messag.split('');
Console.log(words);
```

Output:- ['T', 'h', 'i', 's', ' ', 'm', 'y', ' ', 'f' ...].

→ Strike();
→ Substring()
→ Search()
→ trimStart()
→ trimEnd()
→ sub()

for more go to Mdn Site.

Escape Sequence in String

\0	→	null character.
\'	→	single quote
\"	→	double quote
\\	→	backslash
\n	→	newline
\r	→	carriage return
\v	→	vertical tabulation
\t	→	tab
\b	→	backspace

(3.) # Template Literals

Literals are delimited with backtick (`) character, allowing for multi-line strings and string interpolation.

ex:-
```
let message = `This is
               my first
               message`;

console.log(message);
```

Output:-

This is
my first
message

To get Exact line string

Ex-(2)

```
let lastname = 'Babbar';        → Placeholder

let message = `Hello ${lastname},

                Thanks for the Opportunity

                Regards,
                Babbar`;
```

Note:- To use name of variable, Here we use $ dollar symbol before { } inside ` back tick characters.

Output:

```
Hello Babbar,

Thanks for the Opportunity

Regards,
Babbar
```

8 Time

(4) Date Object :-

JavaScript Date objects represents a single moment in time in a platform-independent format.

→ Date() → 'Sun Feb 12 2023 19:22:49 GMT+0530 (India Standard Time)'

```
→ let date = new Date();
→ let date2 = new Date('June 20 1998 07:15');
→ let date3 = new Date(1998, 5, 20, 7);
  console.log(date3);
```

Output:- 20 June 1998 7:00 -GMT-+0530---

Methods are :-

→ Date.getDate()
→ Date.getDay()
→ Date.getFullYear()
→ Date.getMinutes()
→ Date.getMonth()
→ Date.getSeconds()

→ Date.setFullYear()
→ Date.setHours()
→ Date.setDate()
→ Date.setMonth()
→ Date.setTime()

Arrays

- In Javascript, arrays are not primitives but are instead Array object.

→ JS Arrays are resizable and contain a mix of different data types. ex:- int, String, bool etc.

→ It is a single variable that is used to store different elements.

Declaration of Array :- (square brackets)

① let arrayName = [value1, value2, ...]; //Method 1.

② let arrayName = new Array(); //Method 2.

Initialization of Array :-

① let house = ["1BHK", "2BHK", "3BHK", "4BHK"];
//method 1

//method 2

② let house = new Array (10, 20, 30, 40, 50);

* Indexing of Array :-

let numbers = [1, 3, 5, 7];

1	3	5	7
0	1	2	3

0 : 1
1 : 3
2 : 5
3 : 7

numbers[0] = 1
numbers[1] = 3
numbers[2] = 5
numbers[3] = 7

Insertion in Arrays :-

(i) At End.
(ii) At Beginning.
(iii) At Middle.

(i) Insertion at the End :- (Using push(x)).

let numbers = [1, 3, 5, 7];

numbers.push(9);
Console.log(numbers);

Output :- [1, 3, 5, 7, 9]

(ii) **Insertion at Begining :-** (using unshift(x))

let numbers = [1,3,5,7,9]

numbers.unshift(8);

Output:- [8,1,3,5,7,9].

(iii) **Insertion at Middle :-** (using splice())

* Syntax of splice (at index, no. of delete elements, 'values', 'v',--)
  - at index → Start number
  - no. of delete elements → deleteCount
  - 'values', 'v',-- → arr[].

let numbers = [8,1,3,5,7,9].

→ numbers.splice(3, 0, 'a', 'b', 'c', 'd', 'e');

Output:- [8,1,3,'a','b','c','d','e',5,7,9]

Searching → on PRIMITIVES → indexOf, includes()
↳ on Object/Reference → callback.

Searching in Array :- PRIMITIVES case

Method-1

* indexOf() → If value is found, Return POSITION of a specified value.
  ↳ Not found, Return -1.

```
let numbers = [1,2,3,4,5,6,7];
console.log(numbers.indexOf(6));   //found
Console.log(numbers.indexOf(10));  //Not found.
```

Output:-
5 → found.
-1 → Not found.

* Type 2 → indexOf(Searching Element, FROM Where to Search)

~~numbers.log~~

numbers.indexOf(4,2);
↓
Return position of 4 i.e. 3

numbers.indexOf(3,4);
↓
Return -1, No 3 after 4th index.

Method 2

* Includes() → If Found, Return "true"
  ↳ Not found, Return "false".

```
numbers.includes(7);   // True.
numbers.includes(10);  // false.
```

Searching in Array Object/Reference :-

To understand different References :-

Different References

```
let courses = [
    {no:1 , name:'Love'},
    {no:2 , name:'Rahul'}
];

console.log(courses);
```

output :-
```
0 : {no:1 , name:'Love'}
1 : {no:2 , name:'Rahul'}
```

```
console.log(courses.IndexOf({no:1 , name:'Love'}));
```

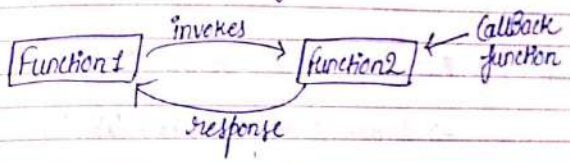
output :- -1 → Different References → Not Same Reference

When We search into Object,

* We use CALL BACK FUNCTIONS for searching in Array Object/Reference.

* CALL BACK FUNCTION :-

A Callback function is a function passed into another function as an argument, which is then invoked inside the outer function to complete some kind of routine or action.



Also called "Predicate function."

Example :-

①
```
let courses = [
    {no:1 , name:'Love'},
    {no:2 , name:'Rahul'}
];

let getCourse = courses.find(function(value){
    return value.name == 'Love';
});
console.log(getCourse);
```

output :- {no:1 , name:'Love'}

② let course = courses.find(function(course){
return course.name === 'Rahul';
});
console.log(course);
output:- {no:2, name: 'Rahul'}

③ If Not found, then return "Undefined".
→ return course.name === 'Ravi';
↓
returns
"Undefined".

We can make callback function in short :-
By using Arrow Function

→ Steps of Arrow Function
① Remove 'function' and add ⇒ after Input parameter.
② If Input parameter is only one, then Remove '()'
③ If return code is in single line, then Remove '{}, return and place near to this ⇒ symbol. codes
↳ {}, return s;

Example:-
let course = courses.find(function(course){
return course.name == 'love';
});

steps
①
courses.find((course) => {
return course.name == 'love';
});

②
courses.find(course => {
return course.name == 'love';
});

③
let course = courses.find(course => course.name == 'love');
ARROW FUNCTION

~~let course = courses.find(value => value.~~

Deletion in Array :-

(i) At End
(ii) Begining
(iii) At middle.

(i) Removing at the End :- (Using pop(*))

```
let numbers = [1,2,3,4];

numbers.pop();
console.log(numbers);
```

Output :- [1,2,3]

(ii) Removing at the Begining :- (Using shift())

```
numbers.shift();
```

Output :- [2,3]

(iii) Removing at the middle :- (using splice())

Syntax :- splice(starting index, delete count)

deleted

```
let numbers = [1,2,3,4,5]

numbers.splice(1,3);
console.log(numbers);
```

Output :- [1,5]

Emptying an Array :-

(i) By Array.length = 0 | Array is Array Name

example :-
```
let numbers = [1,2,3,4,5];
console.log(numbers);      //5 length output
                           //with array.

numbers.length = 0;
console.log(numbers);
```

Output :-
[1,2,3,4,5] → 5
[] → 0

(ii) By splice() method :-

ex:- let numbers = [1,2,3,4,5];

numbers.splice(0, numbers.length);

consale.log(numbers);

output:-

[]

Combining Arrays :- (ON Primitive Value)

* By Using concat() method

let numbers = [1,2,3,4,5];
let second = [6,7,8,9];

let combined = numbers.concat(second);

consale.log(combined);

output:-

[1,2,3,4,5,6,7,8,9]

Slicing Arrays :- ON Primitive Value

* By using slice() method :- slice(start, end-1)

start → includes
end → exclude

let numbers = [1,2,3,4,5,6,7,8,9];

let sliced = numbers.slice(2, 6);

includes ↓ ↓ exclude

console.log(sliced);

output:-

[3,4,5,6]

→ Full Slicing Concept :-

let sliced = numbers.slice();
consale.log(sliced);

output:-

[1,2,3,4,5,6,7,8,9]

H.W.

* Combining Array on Object Type

① Using Call Back function

```
let arr1 = [
        {id: "abc", date: "2023-02-12"},
        {id: "def", date: "2023-02-14"}
    ];

let arr2 = [
        {m_no: 12345, name: "Ankit"},
        {m_no: 45678, name: "Rishabh"}
    ];

// Call back function                    key  value
let combined = arr1.map(function(item, i) {
        return Object.assign({}, item, arr2[i]);
    });

console.log(combined);
```

Output:-

```
0: {id: "abc", date: "2023-02-12", m_no: 12345, name: "Ankit"}
1: {id: "def", date: "2023-02-14", m_no: 45678, name: "Rishabh"}
```

② using Arrow Function

```
let combined = arr1.map((item, i) => Object.assign({}, item,
                                        arr2[i])
                        );
```

✓ Same output

H.W.

* Slicing Array on Object Type :-

```
let obj = {
        0: 'zero', 1: 'one', 2: 'two',
        3: 'three', 4: 'four'
    };
```

① // Call Back function :-

```
let Sliced = Object.keys(obj).slice(0,2).map(function(key)
        {
            return ({[key]: obj[key]})
        }
    );

console.log(Sliced);
```

Output:-

```
[ {0: 'zero'}
  {1: 'one'}
]
```

(ii) Arrow function :-

```
let sliced = Object.keys(obj).slice(0,2).map(
        key => ({[key]: obj[key]})
        );
```

✓✓

Combining Using Spread Operator (...)

(1) Ex:-
```
let first = [1,2,3];
let second = [4,5,6];

let combined = [...first, ...second];
Console.log(combined);
```

Output:- [1, 2, 3, 4, 5, 6] ✓

(2) if we want to add more,

```
let combined = [...first, 'a', ...second, 'b'];
Console.log(combined);
```

output:- [1, 2, 3, 'a', 4, 5, 6, 'b']

Iterating an Array :-

(1) for - of loop :-

```
let arr = [10, 20, 30, 40, 50];
for (let value of arr)
{
    Console.log(value);
}
```

Output:-
10
20
30
40
50

(2) forEach loop works CallBack Function :-

```
arr.forEach (function (number) {
        console.log(number);
})
```

output:-
10
20
30
40
50

(5) forEach loop with Arrow function :-

```
arr.forEach ( number => console.log(number))
```

output:-

10
20
30
40
50

Joining in Arrays :- (join()) ✓

→ join elements with some special character

(1) let numbers = [10,20,30,40,50];

let joined = numbers.join(',');

console.log(joined);

Output:-

10,20,30,40,50

(ii) let message = ['This', 'is', 'my', 'message'];

let join2 = Sentence.join('_');

console.log(join2);

Output :-

This_is_my_message

Sorting Array :- (Sort()) ✓

The Sort() method sorts the elements of an array and returns the reference to the same array.

The default sort order is ascending, built upon converting the elements into strings, then comparing their sequences of UTF-16 code units value.

If we do let numbers = [10, 5, 4, 20];
numbers.sort();

output → [10, 20, 4, 5] X

↓

This sorting is according to UTF-16 code after converting into Strings.

Acc. to numbers, It is wrong order sorting.

let arr = [10, 5, 4, 25];

So, we use Sort() with comparison function.

let St = arr.Sort(function(a,b) {
    return a-b;
}); → (all Back function here.

a-b → -ve (a<b) → a before b.
    → 0 (a=b) → a before b OR b before a.
    → +ve (a>b) → b before a.

console.log(St);

Output:- [4, 5, 10, 25] ✓ "This is Correct order.

→ To filter the elements from Array.

Filtering Arrays :- ~~map()~~ filter()

using callback function / Arrow function.

* let numbers = [1, 2, -1, -4];

let filtered = numbers.filter(function(value) {
    return value >= 0;
});

console.log(filtered);

filter: value >= 0 → [-1, -4].

Output:- [1, 2]

* Arrow function :-

let filtered = numbers.filter(value ⇒ value >= 0); ✓

Mapping Arrays :- map()
(Primitive)

Map each elements of array to something else.

example :-

```
let numbers = [1, 2, 3, 4, 5];

let items = numbers.map(function(value){
        return 'Student No. ' + value;
});

console.log(items);
```

Call back

Output :-

['Student No. 1', 'Student No. 2',
'Student No. 3', 'Student No. 4',
'Student No. 5']

//ARROW FUNCTION

```
let items = numbers.map(value => 'Student No. ' +
                        value);
```

Mapping Objects :- To make objects.

example :-

```
let numbers = [1,2,3,4];
let items = numbers.map(function(num){
        return {value: num};
});

console.log(items);
```

Output :-

[{value:1}, {value:2}, {value:3}, {value:4}]

---

* Chaining Concept of using two or more Callback function / Arrow function.

example :-

```
let numbers = [1,2,-6,-9];

let items = numbers.filter(value => value >= 0)
                   .map(num => ({value: num}));

console.log(items);
```

Output :-

[{value:1}, {value:2}]

Reducing an Array :-

reduce() :-

* executes a reducer function for array element.
* returns a single value : the function's accumulated result.
* reduce() does not execute the function for empty array elements.
* It does not change the original array.

let a = [1,2,3,4];

↓

total Sum = ?

We can done by using FOR-OF loops.

But we do here using reduce().

Example :-

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

let totalSum = arr.reduce((accumulator, currentValue) =>
                    accumulator + currentValue , 0);

console.log(totalSum);
```

(sum → accumulator; iterate on each value → currentValue; 0 → accumulator = 0 Initially.)

Output :- 15

Dry Run :-

accumulator = 0, currentValue = 1
accu. + c.v. = 1

accu. = 1, c.v. = 2
accu. + c.v. = 3

accum. = 3, c.v. = 3
accum. + c.v. = 6

accum = 6, c.v. = 4
accum. + c.v. = 10

accum. = 10, c.v. = 5
accum. + c.v. ⇒ 15 Ans