# Lecture 18 | JavaScript Interview Series

## Agenda for today

- Null vs Undefined vs Not defined
- Comparison Operators and Truthy Falsy Values
- Shallow Copy and Deep Copy from every methord possible
- Array.isArray() Methord

## Undefined and Null in JavaScript

```
console.log(a); // but it is undefined, since it is not valued yet
var a = 2; // gets memory location

var x;
console.log(x); // undefined

function test() {
  // returns nothing
}

console.log(test()); // undefined

function test2() {
  return null; // explicitly assigned null
}

console.log(test2()); // returns null

let b = global.x; // using a global objects x which is non existent
console.log(b); // undefined

let person = {
  name: "Adam",
  age: null,
};

let p1 = person.name; // p1 is a string
let p2 = person.age; // p2 is a number
let p3 = person.address; // p3 is undefined

console.log(`${p1} is ${p2} years old from ${p3}`);
// Adam is null years old from undefined
```

> Output : `undefined`

```
> node undefined.js
undefined
undefined
undefined
null
undefined
Adam is null years old from undefined
```

> Footnotes : whenever we dont assign value to a variable, it is `undefined` whereas `null` is passed as a value explicitly.

## Null vs undefined

- Null is a value that is explicitly assigned
- undefined is a value that is not assigned

```
const log = console.log;

let formObj = {
  firstName: "Milind",
  middleName: null, // explicitly assigned null
  lastName: "Mishra",
};

log(formObj.middleName); // null : explicit assignment
log(formObj.age); // undefined : not assigned
```

> Output :

```
> node undefined.js
null
undefined
```

- `Null` explicitly tells that the value is empty
- `undefined` tells that the value is not assigned

## Comparison Operators == & ===

## Use of == and ===

- == is used to compare two values. It is a loose comparison. Where we can compare any type of values.
- === is used to compare two values. It is a strict comparison. Where we can compare only the same type of values.

```
const log = console.log;
// Conditional Operators : == and  ===

// Double equals (==) is used to compare two values. Its a loose
comparison.

let check = 2 == 2;
let check1 = 2 == "2";
log(check); // true
log(check1); // true

// Triple equals (===) is used to compare two values. Its a strict
comparison.

let check2 = 2 === "2";
let check3 = 2 === 2;
log(check2); // false (Strict comparison, compares the type of the values)
log(check3); // true
```

> Output :

```
> node truthyFalsy.js
true
true
false
true
```

## Truthy and Falsy

- all variables, array, objects, functions, etc. have boolean values in JavaScript.
- if the value is true, it is truthy.
- if the value is false, it is falsy.

```
const log = console.log;

function testTruthyFalsy(value) {
  return value ? log("Truthy") : log("Falsy");
}

testTruthyFalsy(0); // Falsy (0 is falsy) +- 0 is falsy in JavaScript
testTruthyFalsy(1); // Truthy (1 is truthy) Any + - number except 0 is
truthy in JavaScript
testTruthyFalsy(false); // Falsy (false is falsy) obviously
testTruthyFalsy(true); // Truthy (true is truthy) obviously
testTruthyFalsy(""); // Falsy (empty string is falsy)
testTruthyFalsy(" "); // Truthy (space is truthy)
testTruthyFalsy([]); // Truthy (Array is truthy)
testTruthyFalsy({}); // Truthy (Object is truthy)
```

```
testTruthyFalsy(function () {}); // Truthy (function is a type of object)
testTruthyFalsy(undefined); // Falsy (undefined is falsy)
testTruthyFalsy(null); // Falsy (null is falsy)
testTruthyFalsy(NaN); // Falsy (NaN is falsy) NaN : Not a Number its a
special value that is not equal to any other value.
testTruthyFalsy(Infinity); // Falsy (Infinity is falsy) Infinity :
Infinity is a special value that is greater than any other value.
testTruthyFalsy(-Infinity); // Falsy (-Infinity is falsy) -Infinity : -
Infinity is a special value that is less than any other value.
testTruthyFalsy(new Date()); // Truthy (Date is a type of object)
testTruthyFalsy(new Error()); // Truthy (Error is a type of object)
testTruthyFalsy(new RegExp()); // Truthy (RegExp is a type of object)
```

> Output :

```
> node truthyFalsy.js
Falsy
Truthy
Falsy
Truthy
Falsy
Truthy
Truthy
Truthy
Truthy
Falsy
Falsy
Falsy
Truthy
Truthy
Truthy
Truthy
Truthy
```

## List of Truthy and Falsy Values in Js

- List of Truthy values in Js

  > 1, true, " ", [], {}, function () {}, new Date(), new Error(), new RegExp()
  >
  > 1. Any Number other than 0
  > 2. Any String other than empty string
  > 3. New Object

- List of Falsey values in Js

  > 0, false, null, undefined, NaN, Infinity, -Infinity

```
const log = console.log;
function homeWork() {
  if ((-100 && 100 && "0") || [] === true || 0) {
    log(1);
    if ([] || (0 && false)) {
      log(2);
    }
    if (Infinity && NaN && "false") {
      log(3);
      if ("") {
        log(4);
      }
    } else {
      log(5);
      if (({} || false === "") && !(null && undefined)) {
        log(6);
      }
    }
  }
}

homeWork();
```

> Output : (**Understanding is Important**)

```
> node homeWorkProblem.js
1
2
5
6
```

## Shallow and Deep Copy

> Explained thoroughly in the comments of the code.

```
const log = console.log;

// copying an array
let arr = [1, 2, 3, 4, 5];
let copyArr = arr;
copyArr[1] = 4;

log(arr); // [1, 4, 3, 4, 5]
log(copyArr); // [1, 4, 3, 4, 5]
// both arrays are pointing to the same array
// all object types (reference data types) are passed by reference (stores
just one reference to the object)
// all copies get points to the same object (the same reference)
```

```
// if you change the original array, the copy will also change and vice
versa

// Shallow copy is a upper level copy, but not deep copy which copies the
nested elements as well
// Shallow copy methord : use the spread operator `...`
let sports = [
  "soccer",
  "baseball",
  "football",
  {
    name: "Basketball",
    players: ["Lebron", "Curry", "Jordan"],
  },
];
let sportsCopy = [...sports]; // spread operator
sportsCopy[1] = "basketball"; // change the value of the copy
sportsCopy[3].players[1] = "Kobe"; // change the value of the copy
log(sports); // ["soccer", "basketball", "football", {name: "Basketball",
players: ["Lebron", "Kobe", "Jordan"]}] // original array is changed in
the nested array
log(sportsCopy); // ["soccer", "basketball", "football", {name:
"Basketball", players: ["Lebron", "Kobe", "Jordan"]}] // copy array is
changed in the nested array

// Shallow copy methord : `Array.from()`

let courses = [
  "Engineering",
  "Math",
  "Physics",
  "Chemistry",
  {
    name: "Biology",
    students: ["John", "Mary", "Peter"],
  },
];

let coursesCopy = Array.from(courses); // Array.from() method creates a
new array from an array-like or iterable object
coursesCopy[4].students[1] = "Jane"; // change the value of the copy
log(courses); // ["Engineering", "Math", "Physics", "Chemistry", {name:
"Biology", students: ["John", "Jane", "Peter"]}] // original array is
changed in the nested array
log(coursesCopy); // ["Engineering", "Math", "Physics", "Chemistry",
{name: "Biology", students: ["John", "Jane", "Peter"]}] // copy array is
changed in the nested array

// Shallow copy methord : arr.slice(0) method
// works the same as the spread operator

let coursesCopySlice = courses.slice(0); // slice() method creates a
shallow copy of an array
coursesCopySlice[4].students[1] = "July"; // change the value of the copy
```

```
log(courses); // ["Engineering", "Math", "Physics", "Chemistry", {name:
"Biology", students: ["John", "July", "Peter"]}] // original array is
changed in the nested array
log(coursesCopySlice); // ["Engineering", "Math", "Physics", "Chemistry",
{name: "Biology", students: ["John", "July", "Peter"]}] // copy array is
changed in the nested array

// Deep copy : It copies in the nested elements as well

// Deep copy methord : `JSON.parse(JSON.stringify(obj))`

let coursesDeepCopy = JSON.parse(JSON.stringify(courses)); //
JSON.stringify() method converts an object into a JSON string
coursesDeepCopy[4].students[1] = "Alice"; // change the value of the copy
log(courses); // ["Engineering", "Math", "Physics", "Chemistry", {name:
"Biology", students: ["John", "Jane", "Peter"]}] // original array is not
changed in the nested array
log(coursesDeepCopy); // ["Engineering", "Math", "Physics", "Chemistry",
{name: "Biology", students: ["John", "Alice", "Peter"]}] // copy array is
changed in the nested array

// Deep copy methord : `Object.assign(obj)`

let obj = {
  name: "John",
  age: 30,
  city: "New York",
  hobbies: ["movies", "music"],
  family: {
    wife: "Jane",
    son: "Peter",
    daughter: "Alice",
  },
};

let newObj = obj;
newObj.name = "Bob";
log(obj); // {name: "Bob", age: 30, city: "New York", hobbies: ["movies",
"music"], family: {wife: "Jane", son: "Peter", daughter: "Alice"}}
log(newObj); // {name: "Bob", age: 30, city: "New York", hobbies:
["movies", "music"], family: {wife: "Jane", son: "Peter", daughter:
"Alice"}}
// both objects are pointing to the same object (the same reference) and
both objects are changed

// shallow copy methord : `...` Spread operator

let objCopy = { ...obj };
objCopy.name = "Rock";
objCopy.family.wife = "Mary";
log(obj); // {name: "Bob", age: 30, city: "New York", hobbies: ["movies",
"music"], family: {wife: "Mary", son: "Peter", daughter: "Alice"}}
log(objCopy); // {name: "Rock", age: 30, city: "New York", hobbies:
["movies", "music"], family: {wife: "Mary", son: "Peter", daughter:
```

```
"Alice"}}
// upper level copy of the object but nested elements still get altered in
the original object

// Deep copy methord : `JSON.parse(JSON.stringify(obj))` :
JSON.stringify() method converts an object into a JSON string
// JSON.parse() method parses a JSON string, constructing the JavaScript
value or object described by the string
// JSON.parse() method creates a new object by parsing a JSON string

let person = {
  name: "Milind",
  age: 20,
  city: "Bangalore",
  hobbies: ["movies", "music"],
};

let personCopy = JSON.parse(JSON.stringify(person));
// any changes to the copy will not affect the original object
personCopy.name = "Mrinal";
personCopy.age = 23;
personCopy.city = "Mumbai";
personCopy.hobbies[0] = "cricket";
log(person); // {name: "Milind", age: 20, city: "Bangalore", hobbies:
["movies", "music"]}
log(personCopy); // {name: "Mrinal", age: 23, city: "Mumbai", hobbies:
["cricket", "music"]}

// Object Shallow Copy : `Object.assign()` method creates a shallow copy
of an object

let newShallowCopy = Object.assign({}, person);
newShallowCopy.name = "John";
newShallowCopy.age = 25;
newShallowCopy.city = "New York";
newShallowCopy.hobbies[0] = "coding";
log(person); // {name: "Milind", age: 20, city: "Bangalore", hobbies:
["coding", "music"]}
log(newShallowCopy); // {name: "John", age: 25, city: "New York", hobbies:
["coding", "music"]}
// nested elements get altered in the original object as it was a shallow
copy
```

Output :

```
> node shallowDeep.js
[ 1, 4, 3, 4, 5 ]
[ 1, 4, 3, 4, 5 ]
[
  'soccer',
  'baseball',
  'football',
```

```
    { name: 'Basketball', players: [ 'Lebron', 'Kobe', 'Jordan' ] }
  ]
  [
    'soccer',
    'basketball',
    'football',
    { name: 'Basketball', players: [ 'Lebron', 'Kobe', 'Jordan' ] }
  ]
  [
    'Engineering',
    'Math',
    'Physics',
    'Chemistry',
    { name: 'Biology', students: [ 'John', 'Jane', 'Peter' ] }
  ]
  [
    'Engineering',
    'Math',
    'Physics',
    'Chemistry',
    { name: 'Biology', students: [ 'John', 'Jane', 'Peter' ] }
  ]
  [
    'Engineering',
    'Math',
    'Physics',
    'Chemistry',
    { name: 'Biology', students: [ 'John', 'July', 'Peter' ] }
  ]
  [
    'Engineering',
    'Math',
    'Physics',
    'Chemistry',
    { name: 'Biology', students: [ 'John', 'July', 'Peter' ] }
  ]
  [
    'Engineering',
    'Math',
    'Physics',
    'Chemistry',
    { name: 'Biology', students: [ 'John', 'July', 'Peter' ] }
  ]
  [
    'Engineering',
    'Math',
    'Physics',
    'Chemistry',
    { name: 'Biology', students: [ 'John', 'Alice', 'Peter' ] }
  ]
  {
    name: 'Bob',
    age: 30,
    city: 'New York',
```

```
  hobbies: [ 'movies', 'music' ],
  family: { wife: 'Jane', son: 'Peter', daughter: 'Alice' }
}
{
  name: 'Bob',
  age: 30,
  city: 'New York',
  hobbies: [ 'movies', 'music' ],
  family: { wife: 'Jane', son: 'Peter', daughter: 'Alice' }
}
{
  name: 'Bob',
  age: 30,
  city: 'New York',
  hobbies: [ 'movies', 'music' ],
  family: { wife: 'Mary', son: 'Peter', daughter: 'Alice' }
}
{
  name: 'Rock',
  age: 30,
  city: 'New York',
  hobbies: [ 'movies', 'music' ],
  family: { wife: 'Mary', son: 'Peter', daughter: 'Alice' }
}
```

## `arr.isArray()` Method

```
const log = console.log;

let str = "Hello";

let obj = {
  name: "John",
  age: 30,
};

let num = 20;

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

log(typeof str); // string
log(typeof obj); // object
log(typeof num); // number
log(typeof arr); // object! Refrence types are all objects

// the catch here is that all references are objects and since array is an
Reference type, it is also an object
// so `arr.isArray()` method used to check if the variable is an array

log(Array.isArray(str)); // false // string is not an array
log(Array.isArray(obj)); // false (object is not an array)
```

```
log(Array.isArray(num)); // false - number is not an array
log(Array.isArray(arr)); // true (arr is an array)
```

Output :

```
> node isArray.js
string
object
number
object
false
false
false
true
```