

# FunctionUp Tech interview Prep Doc - Session 6 & 7

Javascript - Partha Roy

## What is the difference between '==' and '===' operators?

Double equals (==) is a **comparison operator**, which transforms the operands having the same type before comparison. So, when you compare a string with a number, JavaScript converts any string to a number. An empty string is always converted to zero. A string with no numeric value is converted to NaN (Not a Number), which returns false.

=== (Triple equals) is a **strict equality comparison operator** in JavaScript, which returns false for the values which are not of a similar type. This operator performs type casting for equality. If we compare 2 with "2" using ===, then it will return a false value.

Code -

```
var number = 100; // Here number variable assigned using =

// Here Comparison between two values using ==.
//This will not check datatype irrespective of datatype it will do comparison
if (number == 100)
    console.log("Both are equal");
else
    console.log("Both are not equal");

//Here Comparison between two values using ==.
//This will not check datatype irrespective of datatype it will do comparison
if (number == "100")
    console.log("Both are equal");
else
    console.log("Both are not equal");
```

Resource -  [JavaScript Comparison Operators Tutorial in Hindi / Urdu](#)

## What is the typeof operator?

typeof in JavaScript is an operator used for type checking and returns the data type of the operand passed to it. The operand can be any variable, function, or object whose type you want to find out using the typeof operator.


Since JavaScript is a dynamically typed language, meaning you do not have to specify the type of variables when declaring them, the typeof operator comes in handy to check the data type before execution.

You can use the JavaScript typeof operator in the following two ways:

- typeof operand
- typeof (operand)

Code -

```
console.log(typeof 42);  
// expected output: "number"  
  
console.log(typeof 'blubber');  
// expected output: "string"  
  
console.log(typeof true);  
// expected output: "boolean"  
  
console.log(typeof undeclaredVariable);  
// expected output: "undefined"
```

Resource -  [typeof Operator in JavaScript \(Hindi\)](#)

## What is NaN?

Unquoted literal constant NaN is a special value representing Not-a-Number.

Since NaN always compares unequal to any number, including NaN, it is usually

used to indicate an error condition for a function that should return a valid number.

**Note** – Use the `isNaN()` global function to see if a value is an NaN value.

Code -

```
function milliseconds(x) {  
  if (isNaN(x)) {  
    return 'Not a Number!';  
  }  
  return x * 1000;  
}  
  
console.log(milliseconds('100F'));  
// expected output: "Not a Number!"  
  
console.log(milliseconds('4'));  
// expected output: 4000
```

Resource - [YouTube: NaN and isNaN in JavaScript](#)

## What are `setTimeout` and `setInterval` functions and what is the difference between them?

We may decide to execute a function not right now, but at a certain time later. That's called "scheduling a call".

There are two methods for it:

- `setTimeout` allows us to run a function once after the interval of time.
- `setInterval` allows us to run a function repeatedly, starting after the interval of time, and then repeating continuously at that interval.

They are supported in all browsers and Node.js.

Resource - [YouTube: setInterval and setTimeout: timing events - Beau teaches Java...](#)

Code -

```
var intervalID = setInterval(alert, 1000); // Will alert every second.  
clearInterval(intervalID); // Will clear the timer.  
  
setTimeout(alert, 1000); // Will alert once, after a second.
```

## What are the functional scope and block scope?

In javascript, the whole document is the **global scope** and all the other functions and variables are contained in this global scope. Another is the **Local Scope**, variables declared inside the functions are considered to be of the local scope and it is further divided into **function scoped** and **block-scoped**.

**Function Scope:** When a variable is declared inside a function, it is only accessible within that function and cannot be used outside that function.

**Block Scope:** A variable when declared inside the if or switch conditions or inside for or while loops, are accessible within that particular condition or loop.

Resource - [▶ What is Scope in JavaScript?](#)

## What are shallow copy and deep copy?

Deep copy means that all of the values of the new variable are copied and disconnected from the original variable.


A shallow copy means that certain (sub-)values are still connected to the original variable.

Code -

```
var employeeDetailsOriginal = {  
  name: 'Manjula',  
  age: 25,  
  Profession: 'Software Engineer'  
};
```

```
//Shallow copy!
var employeeDetailsDuplicate = employeeDetailsOriginal;

//Deep copy!
var employeeDetailsDuplicate = {
  name: employeeDetailsOriginal.name,
  age: employeeDetailsOriginal.age,
  Profession: employeeDetailsOriginal.Profession
};
```

Resource -  [JavaScript Objects: Shallow and Deep Copy | All You Need to K...](#)

## What is the difference between map and filter?

The **filter()** method creates an array filled with all array elements that pass a test implemented by the provided function. The filter method is well suited for particular instances where the user must identify certain items in an array with a common characteristic.

The **map()** method creates a new array with the results of calling a function for every array element. The map method allows items in an array to be manipulated to the user's preference, returning the conclusion of the chosen manipulation in an entirely new array.

Code -

```
let studentRecords = [
  { name: 'John', id: 123, marks: 98 },
  { name: 'Baba', id: 101, marks: 23 },
  { name: 'yaga', id: 200, marks: 45 },
  { name: 'Wick', id: 115, marks: 75 }
]

//Traditional for() loop
let names = [];
for (let index = 0; index < studentRecords.length; index++) {
  names.push(studentRecords[index].name.toUpperCase());
}
```

```

}
console.log(names); // logs: [ 'JOHN', 'BABA', 'JOHN', 'WICK' ]

//Solution using map() →
let namesArray = studentRecords.map(stu => stu.name.toUpperCase());
console.log(names); // logs: [ 'JOHN', 'BABA', 'JOHN', 'WICK' ]

//Array.filter()
let namesWithMarksMoreThan50 = studentRecords.filter(stu => stu.marks > 50);
console.log(names);
// logs: [ { name: 'John', id: 123, marks: 98 }, { name: 'Wick', id: 115, marks:
75 } ]

```

Resource - [YouTube map, filter & reduce](#) 🙏 *Namaste JavaScript Ep. 19* 🔥

## Explain math, the built-in javascript object, and functionalities.

Math is one of JavaScript's global or standard built-in objects and can be used anywhere you can use JavaScript. The math object provides you with properties and methods for mathematical constants and functions. Unlike other global objects, Math is not a constructor. All the properties and methods of Math are static and can be called by using Math as an object without creating it. It contains useful constants like  $\pi$  and Euler's constant and functions such as `floor()`, `round()`, and `ceil()`.

Code -

```

Math.E          // returns Euler's number
Math.PI         // returns PI

Math.round(x)   //Returns x rounded to its nearest integer
Math.ceil(x)    //Returns x rounded up to its nearest integer
Math.floor(x)   //Returns x rounded down to its nearest integer
Math.trunc(x)   //Returns the integer part of x (new in ES6)

```

Resource - [YouTube JavaScript Math](#) 📐

## What is ES6? Explain some features of ES6

ECMAScript 2015 was the second major revision to JavaScript. ECMAScript 2015 is also known as ES6 and ECMAScript 6.

Here are some features of ES6 -


**let and const keywords:** The keyword "let" enables the users to define variables and on the other hand, "const" enables the users to define constants. Variables were previously declared using "var" which had function scope and were hoisted to the top. It means that a variable can be used before declaration. But, the "let" variables and constants have block scope which is surrounded by curly-braces "{}" and cannot be used before declaration.

**Arrow Functions:** ES6 provides a feature known as Arrow Functions. It provides a more concise syntax for writing function expressions by removing the "function" and "return" keywords.

**Multi-line Strings:** ES6 also provides Multi-line Strings. Users can create multi-line strings by using back-ticks(`).

**Template Literals:** ES6 introduces very simple string templates along with placeholders for the variables. The syntax for using the string template is `${PARAMETER}` and is used inside of the back-ticked string.

**Classes:** Previously, classes never existed in JavaScript. Classes are introduced in ES6 which looks similar to classes in other object-oriented languages, such as C++, Java, PHP, etc. But, they do not work exactly the same way. ES6 classes make it simpler to create objects, implement inheritance by using the "extends" keyword and also reuse the code efficiently. In ES6, we can declare a class using the new "class" keyword followed by the name of the class.

Resource -  [ES6 Tutorial: Learn Modern JavaScript in 1 Hour](#)

## **Difference between typescript & javascript.**

When JavaScript was developed, the JavaScript development team introduced JavaScript as a client-side programming language. But as people were using JavaScript, developers also realized that JavaScript could be used as a server-side programming language.

However, as JavaScript was growing, JavaScript code became complex and heavy. Because of this, JavaScript wasn't even able to fulfill the requirement of an Object-Oriented Programming language. This prevented JavaScript from succeeding at the enterprise level as a server-side technology. So TypeScript was created by the development team to bridge this gap.

### **Differences:**

- TypeScript is known as an Object-oriented programming language whereas JavaScript is a scripting language.
- TypeScript has a feature known as Static typing but JavaScript does not support this feature.
- TypeScript supports Interfaces but JavaScript does not.
- TypeScript is Strongly typed, supports both static and dynamic typing. While JavaScript is Weakly typed, no option for static typing.
- TypeScript is Converted into JavaScript code to be understandable for browsers. Can be directly used in browsers.
- Supports modules, generics and interfaces to define data. No support for modules, generics or interfaces.

### **Advantages of using TypeScript over JavaScript:**

- TypeScript always points out the compilation errors at the time of development (pre-compilation). Because of this getting runtime errors is less likely, whereas JavaScript is an interpreted language.
- TypeScript supports static/strong typing. This means that type correctness can be checked at compile time. This feature is not available in JavaScript.
- TypeScript is nothing but JavaScript and some additional features i.e. ES6 features. It may not be supported in your target browser but the TypeScript compiler can compile the .ts files into ES3, ES4, and ES5 also.



Resource -  *Typescript vs Javascript*