1. **List 5 difference between Browser JS(console) v Nodejs**

|  |  |  |
| --- | --- | --- |
| S.No | Javascript | NodeJS |
| **1.** | Javascript is a programming language that is used for writing scripts on the website. | NodeJS is a Javascript runtime environment. |
| **2.** | Javascript can only be run in the browsers. | We can run Javascript outside the browser with the help of NodeJS. |
| **3.** | We can run Javascript outside the browser with the help of NodeJS. | It is mostly used on the server-side. |
| **4.** | Javascript is capable enough to add HTML and play with the DOM. | Nodejs does not have capability to add HTML tags. |
| **5.** | Javascript can run in any browser engine as like JS core in safari and Spidermonkey in Firefox. | V8 is the Javascript engine inside of node.js that parses and runs Javascript. |
| **6.** | Javascript is used in frontend development. | Nodejs is used in server-side development. |
| **7.** | Some of the javascript frameworks are RamdaJS, TypedJS, etc. | Some of the Nodejs modules are Lodash, express etc. These modules are to be imported from npm. |
| **8.** | It is the upgraded version of ECMA script that uses Chrome’s V8 engine written in C++. | Nodejs is written in C, C++ and Javascript. |

1. **watch & summary 5 points -<https://www.youtube.com/watch?v=SmE4OwHztCc&ab_channel=JSConf>**

**ANS:**

1.Parsing HTML

There are a few things about parsing for HTML that a bit different to most languages, it’s very for giving by nature you can make a lot of mistakes and it well just work that’s one of the great things about web development. A lot of mistakes and it will still work for you, that means that the parsing is not straight forward in most language if you make a mistake or throw an error, it will error out, HTML will try to recover. It can be halted, so we will go into when it can stop parsing in certain situations, wilt do speculative parsing, that’s kind of a big word, will go into what that means soon and re-entering, that probably means nothing to you right now.

2.Render / Frame tree

DOM+CSSOM is called the Frame tree. It combines the two object models, style resolution. This is the actual representation of what will show on screen. Not a one to one mapping of our HTML. So, accommodation, you combine the two object models, style resolution, that means it will show on the screen.

3.Layout

Render tree with our visual information like the video I should where we could see all the element moving round the page, you might have heard it called fellow or re-layout as web if you transfer it.

4.Paint

This is what takes all the information from the render tree and will actually do the calls to actually paint something. So kind of think of it like the canvas API where you are drawing boxes, paint uses the process of taking all the information from the render tree and gives you the visual output. There’s a few things about it, it will take so it will layout the render tree, take all that information you have done the layout so it knows where it sits the Co-ordinates of the items, it will create layers, so layers of things you have got a position absolute, make that work from the bottom up things display correctly, it’s an incremental process, and actually there’s a 12 step phase, it needs to paint like the background colour and image or video it will put it into it’s own layer.

5.Consolation

We parse our CSS and our HTML that creates a DOM tree, the DOM tree gets converted into a render tree, so the CSS and the HTML combined. It,s actually four trees so the layers, the line boxes, the render objects, and the render styles.

1. To read
2. **Execute the below code and write your description in txt file**

# typeof

Console.log(typeof 42);

// expected output:”number”

Console.log(typeof `blubber`);

// expected output: “string”

Console.log(typeof true);

// expected output: “boolean”

Console.log(typeof undeclared Variable);

// expected output: “undefined”

## [Description](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/typeof" \l "description" \o "Permalink to Description)

| **Type** | **Result** |
| --- | --- |
| [Undefined](https://developer.mozilla.org/en-US/docs/Glossary/undefined) | "undefined" |
| [Null](https://developer.mozilla.org/en-US/docs/Glossary/Null) | "object" (see [below](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/typeof" \l "typeof_null)) |
| [Boolean](https://developer.mozilla.org/en-US/docs/Glossary/Boolean) | "boolean" |
| [Number](https://developer.mozilla.org/en-US/docs/Glossary/Number) | "number" |
| [BigInt](https://developer.mozilla.org/en-US/docs/Glossary/BigInt) (new in ECMAScript 2020) | "bigint" |
| [String](https://developer.mozilla.org/en-US/docs/Glossary/String) | "string" |
| [Symbol](https://developer.mozilla.org/en-US/docs/Glossary/Symbol) (new in ECMAScript 2015) | "symbol" |
| [Function](https://developer.mozilla.org/en-US/docs/Glossary/Function) object (implements [[Call]] in ECMA-262 terms) | "function" |
| Any other object | "object" |

### [typeof null](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/typeof" \l "typeof_null" \o "Permalink to typeof null)

// All constructor functions, with the exception of the Function constructor, will always be typeof 'object'let str = new String('String');let num = new Number(100);

typeof str; // It will return 'object'typeof num; // It will return 'object'

let func = new Function();

typeof func; // It will return 'function'

[Description](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/typeof" \l "description" \o "Permalink to Description)

In the first implementation of JavaScript, JavaScript values were represented as a type tag and a value. The type tag for objects was 0. null was represented as the NULL pointer (0x00 in most platforms). Consequently, null had 0 as type tag, hence the typeof return value "object"

typeof({})

The **function declaration** (function statement) defines a function with the specified parameters.

You can also define functions using the [Function](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function) constructor.

Function calcRectArea(width, height) {

Return width \* heigth;

}

Console.log(calcRectArea(5,6));

//expected output: 30

## [Description](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/function" \l "description" \o "Permalink to Description)

A function created with a function declaration is a Function object and has all the properties, methods and behavior of Function objects. See [Function](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function) for detailed information on functions.

A function can also be created using an expression (see [function expression](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/function)).

By default, functions return undefined. To return any other value, the function must have a [return](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/return) statement that specifies the value to return.

### [Conditionally created functions](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/function" \l "conditionally_created_functions" \o "Permalink to Conditionally created functions)

Functions can be conditionally declared, that is, a function statement can be nested within an if statement, however the results are inconsistent across implementations and therefore this pattern should not be used in production code. For conditional function creation, use function expressions instead.

var hoisted = "foo" in this;

console.log(`'foo' name ${hoisted ? "is" : "is not"} hoisted. typeof foo is ${typeof foo}`);if (false) {

function foo(){ return 1; }}

// In Chrome:// 'foo' name is hoisted. typeof foo is undefined//// In Firefox:// 'foo' name is hoisted. typeof foo is undefined//// In Edge:// 'foo' name is not hoisted. typeof foo is undefined//// In Safari:// 'foo' name is hoisted. typeof foo is function

Copy to Clipboard

The results are exactly the same for a condition that evaluates to true

var hoisted = "foo" in this;

console.log(`'foo' name ${hoisted ? "is" : "is not"} hoisted. typeof foo is ${typeof foo}`);if (true) {

function foo(){ return 1; }}

// In Chrome:// 'foo' name is hoisted. typeof foo is undefined//// In Firefox:// 'foo' name is hoisted. typeof foo is undefined//// In Edge:// 'foo' name is not hoisted. typeof foo is undefined//// In Safari:// 'foo' name is hoisted. typeof foo is function

Copy to Clipboard

### [Function declaration hoisting](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/function" \l "function_declaration_hoisting" \o "Permalink to Function declaration hoisting)

Function declarations in JavaScript are hoisted to the top of the enclosing function or global scope. You can use the function before you declared it:

hoisted(); // logs "foo"

function hoisted() {

console.log('foo');}

Copy to Clipboard

Note that [function expressions](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/function) are not hoisted:

notHoisted(); // TypeError: notHoisted is not a function

var notHoisted = function() {

console.log('bar');};

# Array.of()

The Array.of() method creates a new Array instance from a variable number of arguments, regardless of number or type of the arguments.

The difference between Array.of() and the Array constructor is in the handling of integer arguments: Array.of(7) creates an array with a single element, 7, whereas Array(7) creates an empty array with a length property of 7 (**Note:** this implies an array of 7 empty slots, not slots with actual [undefined](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/undefined) values).

Array.of(7); // [7]Array(7); // array of 7 empty slots

Array.of(1, 2, 3); // [1, 2, 3]Array(1, 2, 3); // [1, 2, 3]

## [Description](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/of" \l "description" \o "Permalink to Description)

# **Array.from() & Array.of()**

Original discussion began on Twitter and can be found by starting [here](http://twitter.com/" \l "!/littlecalculist/status/89848378682392576) and [here](http://twitter.com/" \l "!/littlecalculist/status/89855977838485504). Discussion was continued on ****es-discuss**** mailing list in the thread [Pure win: Array.from and Array.of](https://mail.mozilla.org/pipermail/es-discuss/2011-July/015831.html)

# undefined

The global undefined property represents the primitive value [undefined](https://developer.mozilla.org/en-US/docs/Glossary/undefined). It is one of JavaScript's [primitive types](https://developer.mozilla.org/en-US/docs/Glossary/Primitive).

| **Property attributes of undefined** | |
| --- | --- |
| Writable | no |
| Enumerable | no |
| Configurable | no |

Function test (t) {

If (t === undefined) {

Return ‘Undefined value!’;

}

Return t;

}

Let x;

Console.log(test(x));

//expected output: “Undefined value!”

## [Description](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/undefined" \l "description" \o "Permalink to Description)

undefined is a property of the global object. That is, it is a variable in global scope. The initial value of undefined is the primitive value [undefined](https://developer.mozilla.org/en-US/docs/Glossary/undefined).

In modern browsers (JavaScript 1.8.5 / Firefox 4+), undefined is a non-configurable, non-writable property, per the ECMAScript 5 specification. (Even when this is not the case, avoid overriding it.)

A variable that has not been assigned a value is of type undefined. A method or statement also returns undefined if the variable that is being evaluated does not have an assigned value. A function returns undefined if a value was not [returned](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/return).

# true

**[XSLT/XPath Reference](https://developer.mozilla.org/en-US/docs/Web/XSLT/Transforming_XML_with_XSLT/The_Netscape_XSLT_XPath_Reference)**: [XSLT elements](https://developer.mozilla.org/en-US/docs/Web/XSLT/Element), [EXSLT functions](https://developer.mozilla.org/en-US/docs/Web/EXSLT), [XPath functions](https://developer.mozilla.org/en-US/docs/Web/XPath/Functions), [XPath axes](https://developer.mozilla.org/en-US/docs/Web/XPath/Axes)

The true function returns a boolean value of true.

### [Syntax](https://developer.mozilla.org/en-US/docs/Web/XPath/Functions/true" \l "syntax" \o "Permalink to Syntax)

true()

# NaN

The global NaN property is a value representing Not-A-Number.

| **Property attributes of NaN** | |
| --- | --- |
| Writable | no |
| Enumerable | no |
| Configurable | no |

## Function sanitise(x) {

If (isNaN(x)) {

return NaN:

}

return x;

}

Console.log(sanitise(‘1’));

//expected output: “1”

Console.log(sanitise(‘NotANumber’));

//expected output: NaN

## [Description](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/NaN" \l "description" \o "Permalink to Description)

NaN is a property of the global object. In other words, it is a variable in global scope.

The initial value of NaN is Not-A-Number — the same as the value of [Number.NaN](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number/NaN). In modern browsers, NaN is a non-configurable, non-writable property. Even when this is not the case, avoid overriding it. It is rather rare to use NaN in a program.

There are five different types of operations that return NaN:

* Number cannot be parsed (e.g. parseInt("blabla") or Number(undefined))
* Math operation where the result is not a real number (e.g. Math.sqrt(-1))
* Operand of an argument is NaN (e.g. 7 \*\* NaN)
* Indeterminate form (e.g. 0 \* Infinity, or undefined + undefined)
* Any operation that involves a string and is not an addition operation (e.g. "foo" / 3)

**5.Prototypes**

**function Student()**

**{**

**this.name = 'John';**

**this.gender = 'Male';**

**}**

**var studObj1 = new Student();**

**studObj1.age = 15;**

**alert(studObj1.age);**

**// 15**

**var studObj2 = new Student();**

**alert(studObj2.age);**

**// undefined**

## **Description**

The prototype is an object that is associated with every functions and objects by default in JavaScript, where function's prototype property is accessible and modifiable and object's prototype property (aka attribute) is not visible.