**JavaScript:**

J.S is a High level, object oriented and multi paradigm {Declarative and imperative} programming language. J.S is a High level, object oriented synchronous and multi paradigm {Declarative and imperative} programming language.

JavaScript is a scripting language that may be used to construct online and mobile apps, web servers, games, and more. It was created with the intention of being used in a browser. Even today, the server-side version of JavaScript known as Node.js may be used to create online and mobile apps, real-time applications, online streaming applications, and videogames.

A scripting language is a programming language that employs a high-level construct to interpret and execute one command at a time. In general, scripting languages are easier to learn and faster to code in than more structured and compiled languages such as C and C++.

**WHY?**

It is used to make webpages dynamic and interactive. Before JS webpages were static.

**Asynchronous and synchronous Languages**

JavaScript is a synchronous, blocking, single-threaded language. That just means that only one operation can be in progress at a time.

<https://ankitkamboj18.medium.com/is-javascript-synchronous-or-asynchronous-what-the-hell-is-a-promise-302ee008dfcd#:~:text=under%20the%20hood.-,JavaScript%20is%20Synchronous,in%20progress%20at%20a%20time>.

Java, python and c# are Asynchronous languages

**ECMAScript** is a standard for scripting languages, including **JavaScript**, **JScript**, and **ActionScript**. It is also best known as a [JavaScript](https://en.wikipedia.org/wiki/JavaScript) standard intended to ensure the [interoperability](https://en.wikipedia.org/wiki/Interoperability) of [web pages](https://en.wikipedia.org/wiki/Web_page) across different [web browsers](https://en.wikipedia.org/wiki/Web_browser).

**Lecture 9: Values and Variables**

Variables are used to store data which we can access for later use.

**Restrictions:**

* Always use camel Case style for variables declarations.
* Name of the variable should be declarative means expressing the cause or reason for variable.
* All the special characters and reserved keywords are not allowed for the name of variables e.g. new,@,#,%. But $ dollar special character is allowed by the JS but not recommended.

**Declaring Variables**

To declare (create) a variable, we need to use the var, let, or const keyword, followed by the name we want to provide to the variable.

{this is a block } of any loop or smt Var does not respect the block scope

Function{this is function scope} inside function Var respect the function scope means inside function var has functional scope

**Initialization**

Initializing is the term used to describe the process of assignment of a value to a variable (i.e. storing the value (piece of data) in the location in memory which the variable “points” to).

<https://www.interviewbit.com/problems/var-vs-let-vs-const/>

<https://www.freecodecamp.org/news/var-let-and-const-whats-the-difference/>

Better Understanding With The Help Of Quiz {perfect website }

<https://dandkim.com/js-var-let-const/>

**What are the different data types present in javascript?**

To know the type of a JavaScript variable, we can use the **typeof**operator.

1. **Primitive types**

BBSSNNU trick to remember

**String**- It represents a series of characters and is written with quotes. A string can be represented using a single or a double quote.

**var** str = "Vivek Singh Bisht";//using double quotes & **var** str2 = 'John Doe'; //using single quotes

* **Number**- It represents a number and can be written with or without decimals.

**var** x = 3; //without decimal & **var** y = 3.6; //with decimal

* **BigInt**- This data type is used to store numbers which are above the limitation of the Number data type. It can store large integers and is represented by adding “n” to an integer literal.

**var** bigInteger = 234567890123456789012345678901234567890;

* **Boolean**- It represents a logical entity and can have only two values : true or false. Booleans are generally used for conditional testing.

**var** a = 2; **var** b = 3; & **var** c = 2;(a == b) // returns false & (a == c) //returns true

* **Undefined**- When a variable is declared but not assigned, it has the value of undefined and it’s type is also undefined. **var** x; // value of x is undefined **var** y = undefined; // we can also set the value of a variable as undefined
* **Null**- It represents a non-existent or a invalid value. E.g. **var** z = null;

The concept of null exists only for reference types. It doesn't exist for value types.

|  |  |  |
| --- | --- | --- |
| **Difference between Null** | **undefined** |  |
| Variable is declared as Null | Variable is declared but not assign |  |
| Null | Undefined | NaN |
| var x = null;        console.log(x);//null  console.log(typeofx);//object | console.log("--------");        var y = undefined;        console.log(y);//undefined  console.log(typeofy);//undefined | console.log("--------");        var z = NaN;        console.log(z); //NaN  console.log(typeofz);//Number |
| console.log(Null==undefined); //true  bcz both are nothing | console.log(Null===undefined); //false  bcz both have different type |  |

* **Symbol**- It is a new data type introduced in the ES6 version of javascript. It is used to store an anonymous and unique value. E.g **var** symbol1 = Symbol('symbol');
* typeof **of primitive types**:

**typeof** "John Doe" // Returns "string"

**typeof** 3.14 // Returns "number"

**typeof** true // Returns "boolean"

**typeof** 234567890123456789012345678901234567890n // Returns bigint

**typeof** undefined // Returns "undefined"

**typeof** null // Returns "object" (kind of a bug in JavaScript)

**typeof** Symbol('symbol') // Returns Symbol

**2. Non-primitive types**

* Primitive data types can store only a single value. To store multiple and complex values, non-primitive data types are used.
* Object - Used to store collection of data.
* Array is a non-primitive value
* Function
* Data
* Regex
* Example:

// Collection of data in key-value pairs

**var** obj1 = {

x: 43,

y: "Hello world!",

z: **function**(){

**return** this.x;

}

}

// Collection of data as an ordered list

**var** array1 = [5, "Hello", true, 4.1];

**Note- It is important to remember that any data type that is not a primitive data types, is of Object type in java script like arrays and functions.**

|  |  |
| --- | --- |
| **Primitive** | **Non primitives** |
| Data types that pass by value | Data type that pass by reference |
| Immutable once it created | Mutable |
| <https://www.freecodecamp.org/news/mutability-vs-immutability-in-javascript/>  O.S theory start after this statement let b=9;  Let a=b; > now mutate immutate start. Hint which I make concept.  one thing from w3Schools is  let a=3.14; a is changeable but 3.14 is not  O.S> let obj={name:”hamza”} now obj is changeable and {name:hamza} also change able. | |
| <https://blog.devgenius.io/mutable-and-immutable-in-javascript-78a3cbc6187c> | |
| Values stored on **call stack** | Value store on the **Heap** but reference to value is stored on **call stack** |
| Brief Discription about Stack and Heap:  ->When we write the program compiler convert it into binary code/source code/ machine level code.  -> Loader load the machine level code in memory. This Memory is RAM and it is divided into three parts [machine code, stack and Heap]. Stack is for static values and its size is static/fixed. If stack gets full then Stack Overflow error occurred.  ->Heap it dynamic memory here dynamic allocation take place. | |
| <https://www.geeksforgeeks.org/primitive-and-reference-value-in-javascript/> | |
| Primitives are compared by value.  Two values are strictly equal if they have the same value.  var number1 = 5; var number2 = 5;number1 === number 2; // true | Non-primitives are not compared by value.  Values can also be referred to as reference types because they are being compared by reference instead of value. Two objects are only strictly equal if they refer to the same underlying object.  var obj3 = { 'car' : 'purple' } var obj4 = obj3;  obj3 === obj4; // true  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  var obj1 = { 'cat': 'playful' }; var obj2 = { 'cat': 'playful' };  obj1 === obj2; // false  var arr1 = [ 1, 2, 3, 4, 5 ]; var arr2 = [ 1, 2, 3, 4, 5 ];  arr1 === arr2; // false |
| [**https://medium.com/@junshengpierre/javascript-primitive-values-object-references-361cfc1cbfb0**](https://medium.com/@junshengpierre/javascript-primitive-values-object-references-361cfc1cbfb0) | |
| Data that is not an object and has no methods | -- |
| Can store only one data type | Can store multiple data types in single entity |

**Best Explanation of deep copy I have learned with spread operator**

## 5 Ways to Deep Copy Objects in JavaScript

In JavaScript, we can perform a copy on objects using the following methods:

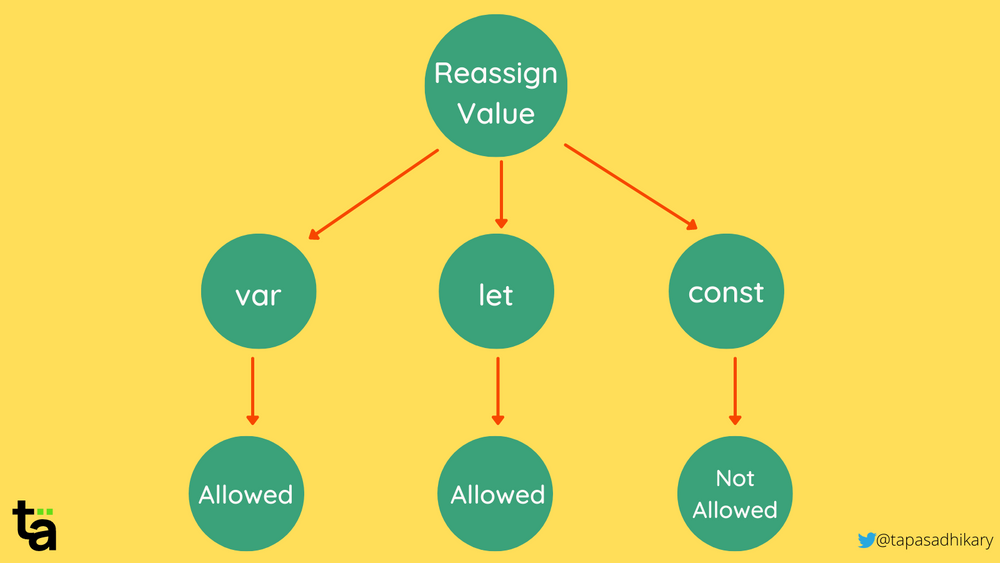
|  |  |  |
| --- | --- | --- |
| **Method** | **Pros** | **Cons** |
| [shallow copy with =](https://code.tutsplus.com/the-best-way-to-deep-copy-an-object-in-javascript--cms-39655a#shallow) | clear and direct, the default | only shallow copies objects |
| [JSON.stringify() and JSON.parse()](https://code.tutsplus.com/the-best-way-to-deep-copy-an-object-in-javascript--cms-39655a#json) | deep copies nested objects | doesn't copy functions |
| [Object.assign()](https://code.tutsplus.com/the-best-way-to-deep-copy-an-object-in-javascript--cms-39655a#object-assign) | copies the immediate members of an object—including functions | doesn't deep copy nested objects |
| [the ... spread operator](https://code.tutsplus.com/the-best-way-to-deep-copy-an-object-in-javascript--cms-39655a#spread) | simple syntax, the preferred way to copy an object | doesn't deep copy nested objects |
| [Lodash cloneDeep()](https://code.tutsplus.com/the-best-way-to-deep-copy-an-object-in-javascript--cms-39655a#lodash) | clones nested objects including functions | adds an external dependency to your project |

These methods all have their pros and cons. Let's take a closer look at each of them.

[**https://code.tutsplus.com/the-best-way-to-deep-copy-an-object-in-javascript--cms-39655a**](https://code.tutsplus.com/the-best-way-to-deep-copy-an-object-in-javascript--cms-39655a)

**Let, Var and const**

****

****

<https://www.interviewbit.com/javascript-interview-questions/>

**Quiz var, let & const**

[**https://dandkim.com/js-var-let-const/**](https://dandkim.com/js-var-let-const/)

**Interview Questions for Data types**

<https://backbencher.dev/javascript-interview-questions-data-types>

**Operators in JS**

For string comparison

<https://www.freecodecamp.org/news/javascript-string-comparison-how-to-compare-strings-in-js/>

**Interview Questions for OPERATORS**

<https://levelup.gitconnected.com/javascript-interview-questions-operators-37c8c35d5eeb>

**Perfect one**

<https://www.hellojavascript.info/docs/general-javascript-questions/javascript-fundamentals/basic-math-operators-in-javascript>

**Switch Statement:**  The switch statement evaluates an expression, matching the expression's value against a series of case clauses, and executes statements after the first case clause with a matching value, until a break statement is encountered. The default clause of a switch statement will be jumped to if no case matches the expression. (It use the strict equality comparison).

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/switch>

**What is Hoisting?**

In JS all the declaration perform at the top of the program but not initialized.

# JavaScript Functions are First-Class Citizens

In JavaScript, you can create and modify a function, use it as an argument, return it from another function, and assign it to a variable. All these abilities allow us to use functions everywhere to place a bunch of code logically.