**MILESTONE 2**

● Case-sensitivity

● Identifiers

● Comments

● Strict Mode

● Statements

● Keywords and reserved words

● Variable

● The typeof Operator

● undefined

● null

● boolean

● number

● string

● object types

● Arrays / Json Objects

**ARRAYS -** [**Array methods**](https://davitdvalashvili1996.medium.com/javascript-array-methods-your-complete-guide-372b9c6f12cd#:~:text=In%20JavaScript%2C%20array%20indexing%20starts,the%20above%20array%20is%20four.)

**https://davitdvalashvili1996.medium.com/javascript-array-methods-your-complete-guide-372b9c6f12cd#:~:text=In%20JavaScript%2C%20array%20indexing%20starts,the%20above%20array%20is%20four.**

1. Case sensitivity

* JS is a case-sensitive language.
* JavaScript has a set of rules for writing JavaScript programs or codes. Using the correct capitalization for naming keywords, identifiers, functions, and variables is one of them that must be followed.
* EX: var a = “hii”;

Var A = “bye”

Console.log(a); //🡪 hii

1. Identifiers

* Identifiers are the name given to variables, objects, functions, arrays, classes etc.,
* EX:
  + var msg = “hi” 🡪 here ‘msg’ is an identifier.
  + function sayHello(){

console.log('Hi')

} 🡪 sayHello() is an identifier

* + There are set of rules to follow while naming an identifier.
    - 1st letter can be lowercase, uppercase, underscore, dollar sign
    - Followed by 1st letter, other letters also follows the same convention.
    - Reserved keywords, spaces, hyphen, numbers, @ should not be used to name an identifier.

1. Comments:

* Used to explain JS code, and to make it more readable.
* It is used for testing and to prevent execution.
* Single line comment starts with ‘//’
* Multi-line comments start with /\* and end with \*/.
* Hashbang comments 🡪#!, it is like a single line comment. But can be used only at the beginning of the document. It should not have any whitespaces at the beginning of the declarations.

1. Strict mode

* By default JS follows sloppy mode (no strict rules., variable need not to be declared etc.,
* Eliminates some silent errors in JS by changing them to throw errors.
* <script>

“use strict”;

const a = “hi”;

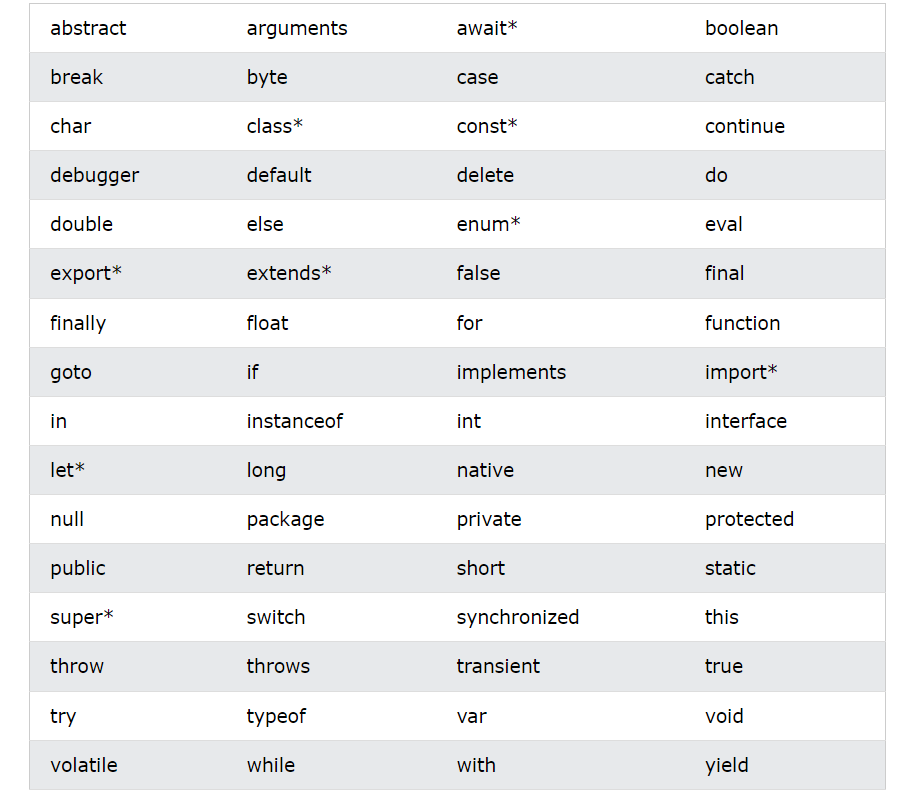
</script>

* If use strict is used inside a function, then that particular function implements use strict mode.
* Entire contents of JS modules are automatically in strict mode, with no statements needed to initiate it. JS modules 🡪 Complex projects need splitting JS programs into modules that can be imported when needed.
* Throws an error when duplicate parameters are used.

1. Statements

* A program is a list of “instructions” executed by computer’s browsers.
* Those instructions are called statements.
* A statement contains values, operators, expressions, keywords, comments
* Semicolons separate JS statements.
* When separated by semicolons, multiple statements on one line are allowed:
* White spaces are ignored in JS.
* White spaces are added to make JS more readable.
* Statements often starts with a keyword.

1. Keywords and reserved words



* Keywords used in java, html, html event handler should not be used in js as variables.
* Some words are reserved as keywords for java or to be used in future. So those can’t also be used.
* Keywords will definitely have some functionality, whereas reserved words may or may not have functions implemented.

1. Variables:

* It is a named reference to a value.
* It is a container for storing variables.
* Declarations – var, let, const
  + var – used to declare variables both local, global. Mostly used in older versions of browsers
  + let – block-scoped local variable.
  + const – block-scoped, read only named constant.
  + EX: if (Math.random() > 0.5) {

const y = 5;}

console.log(y); // ReferenceError: y is not defined

* Var – used in old browsers.
* Let, const are keywords added to JS in 2015.

1. typeOf operator

* It returns the data type of JS variable.
* The datatypes are string, number, Boolean, null, undefined, object.
* Objects, arrays, sets, maps 🡪 returns *object* datatype.
* To determine whether given variable is object, array, set, map 🡪typeOf can’t be used.
* To recognize an array, it is given as *isArray()*
* *Instanceof* operator returns true/ false if an object is given as variable.
* When a variable has no value, it returns *undefined* as datatype.
* NaN is obtained when the given value is not a number.
* typeOf null 🡪 *object*
* Complex datatypes are object, function, arrays, set, maps.



* Datatype of “null” is an object. Null refers to nothing.
* Datatype of “undefined” is undefined.
* Null and undefined are equal in value but different in datatype.
* Typeof null 🡪object (datatype)
* It is possible to empty the value using both null and undefined. But it’s datatype will be varied. It maybe undefined/ object.
* .constructor 🡪 returns the default constructor of the variable.
* arr.constructor === Array 🡪 true . It is done to check whether the given variable is an array.
* Date.constructor === Date 🡪 true/ false. It is done to check whether the given variable is a date.
* Datatype of NaN is *number*
* Void operator evaluates an expression and returns undefined.

1. Undefined

* If a variable doesn’t have any value assigned to it, it returns undefined.
* A function returns undefined if a value was not returned.
* EX: let x;

if (x === undefined) {

// these statements execute

} else {

// these statements do not execute

}

* If (x==undefined) is given, then it will check whether x is a null or not also.
* By using typeof operator to check whether the variable is defined/ undefined doesn’t throw an error. So it is best to check a variable with typeof operator.
* To check the existence of global variable, if ("x" in window) {

// These statements execute only if x is defined globally

}

* Typeof undefined 🡪 undefined (datatype)

1. Null

* Null is primitive datatype we get when object is being missing.
* If an object couldn’t be created, returning null is a common practice.

1. Boolean

* JS Boolean value represents either true/ false.
* Boolean() can be used to check whether a function is true/ not.
* Everything with a value is true.
* Boolean value of 0/ -0/ “”/null/ Nan undefined is FALSE.
* Boolean can also be defined as an object. But it is not best practice.

EX: let y = new Boolean(false); 🡪 typeof y is object

* Comparing two JavaScript objects always return false.

1. Number

* Values of other types can be converted to numbers using the Number() function.
* Strings are converted by parsing them as if they contain a number literal. Parsing failure results in NaN.
* Number.parseFloat() and Number.parseInt() are similar to Number() but only convert strings, and have slightly different parsing rules. For example, parseInt() doesn't recognize the decimal point, and parseFloat() doesn't recognize the 0x prefix.
* Number("123"); // 123

Number("123") === 123; // true

Number("12.3"); // 12.3

Number("12.00"); // 12

Number("123e-1"); // 12.3

Number(""); // 0

Number(null); // 0

Number("0x11"); // 17

Number("0b11"); // 3

Number("0o11"); // 9

Number("foo"); // NaN

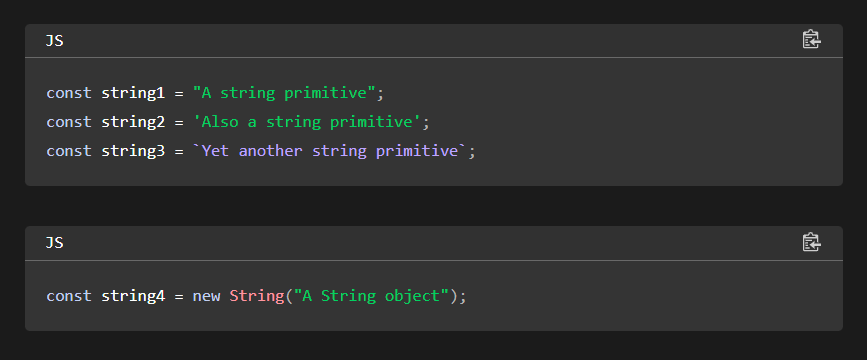
Number("100a"); // NaN

Number("-Infinity"); // -Infinity

* Number.EPSILON 🡪 it is the smallest interval between 2 representable numbers.
* Number.MAX\_SAFE\_INTEGER 🡪 The maximum safe integer in JavaScript (2^53 - 1).
* Number.MAX\_VALUE 🡪 largest positive number.
* Number.MIN\_SAFE\_INTEGER 🡪 The minimum safe integer in JavaScript (-2^53 - 1).
* Number.MIN\_VALUE 🡪 smallest positive number. Smallest number closest to 0. But not 0.
* Number.NaN 🡪 not a number
* Number. NEGATIVE\_INFINITY 🡪 represents negative infinity value.
* Number. POSITIVE\_INFINITY 🡪 represents positive infinity value.
* Number.isFinite() 🡪 determines whether the passed value is a finite number.
* Number.isInteger() 🡪determines whether passed value is an integer.
* Number.isNaN() 🡪 determines whether passed value is NaN.
* Number.isSafeInteger() 🡪 whether the passed value is a safe integer (number between -(253 - 1) and 253 - 1).
* Number.parseFloat() 🡪 changes to number.
* Number.parseInt() 🡪 changes to int.

1. String

* String holds data in text form.
* String can be created using primitives, string literals, objects.



* Character in a string can be accessed by using .charAt()
* Other way to get character from a string is by using an array. string[1]
* To compare both strings, it is common to convert both strings either to uppercase/ lowercase.
* It is done by toUpperCase(), toLowerCase().
* Intl.Collator() is used for comparing and sorting strings acc to specific language and local rules.
* It compares string in a way that respects cultural rules of different language.
* const collator = new Intl.Collator([locales], [options]); 🡪 locale is optional if it is not mentioned, it takes the locale of the runtime.
* Options for case-sensitivity, numeric sorting, ignoring dialects.
* sensitivity: Determines how strictly strings are compared.
  + 'base': Ignores accents and case differences (e.g., "a" == "á").
  + 'accent': Considers accents but ignores case (e.g., "a" != "á", but "a" == "A").
  + 'case': Considers case but ignores accents (e.g., "a" != "A", but "a" == "á").
  + 'variant': Considers both accents and case (e.g., "a" != "A" and "a" != "á").
* {ignorePunctuation} can be used to ignore punctuation and compare the string
* Sorting of array can be done with fruits.sort(collator.compare)
* console.log(new Intl.Collator().compare("a", "c")); // -1, or some other negative value because c comes after a
* console.log(new Intl.Collator().compare("c", "a")); // 1, or some other positive value because a comes before c
* console.log(new Intl.Collator().compare("a", "a")); // 0.
* Literal – list of values

1. Objects

* Object is JS datatype.
* It is used to store various key, value pair collections
* It can be created using new Object(), {}.
* Object type refers to blueprint for creating multiple objects that share similar properties and behaviour.
* Object constructor function is a JS function used with new keyword to create an object.
* EX- // Constructor function

function Car(brand, model, year) {

this.brand = brand; // Property

this.model = model; // Property

this.year = year; // Property

this.drive = function() { // Method

console.log(this.brand + ' ' + this.model + ' is driving!');

}; }

const car1 = new Car('Toyota', 'Corolla', 2022);

const car2 = new Car('Honda', 'Civic', 2023);

console.log(car1.brand); // Toyota

car1.drive(); // Toyota Corolla is driving!

car2.drive(); // Honda Civic is driving!

* In constructor function, this has no value. This will come into effect only when instance of a new object is created.
* JS Object Methods
  + General Methods
  + Property Management Methods
  + Object Protection method
* General Methods:
  + Object.assign() 🡪 copies properties from 1/ more source object to target objects.
  + EX: const person1 = {  
      firstName: "John",  
      lastName: "Doe",  
      age: 50,  
      eyeColor: "blue"  
    };  
      
    // Create Source Object  
    const person2 = {firstName: "Anne",lastName: "Smith"};  
      
    // Assign Source to Target  
    Object.assign(person1, person2);
  + Object.entries() 🡪 return key, value pairs in an object. More useful in loops.
  + Object.fromEntries() 🡪 return value of particular property.
  + Object.values() 🡪 returns value of a property.
  + The Object.groupBy() method groups elements of an object according to string values returned from a callback function.
  + EX: const fruits = [  
      {name:"apples", quantity:300},  
      {name:"bananas", quantity:500},  
      {name:"oranges", quantity:200},  
      {name:"kiwi", quantity:150}  
    ];  
      
    function myCallback({ quantity }) {  
      return quantity > 200 ? "ok" : "low";  
    }  
    const result = Object.groupBy(fruits, myCallback);
  + Object.keys() 🡪 returns keys of an object
* Property Management Methods
  + Object.defineProperty() 🡪 used to add new property, change value, change property of a metadata, change getters, setters.
  + EX: const person = {  
      firstName: "John",  
      lastName : "Doe",  
      language : "EN"  
    };  
    // Add a Property  
    Object.defineProperty(person, "year", {value:"2008"});

// Change a Property  
Object.defineProperty(person, "language", {value : "NO"});

// Define a getter  
Object.defineProperty(person, "fullName", {  
  get: function () {return this.firstName + " " + this.lastName;}  
});

* + Meta data 🡪 writable : true         
    enumerable : true

configurable: true

* + Object.defineProperty(person, "language", {writable:false}); 🡪 with meta data
  + Object.getOwnPropertyNames() 🡪 used to get all properties of an object.
* Object Protection Methods
  + Most common way to protect an object from being changed is by using const keyword.
  + Object.preventExtensions() 🡪 prevents adding properties to an object.
  + EX: const person = {firstName:"John", lastName:"Doe"};  
    // Prevent Extensions  
    Object.preventExtensions(person);  
    // This will throw an error  
    person.nationality = "English";

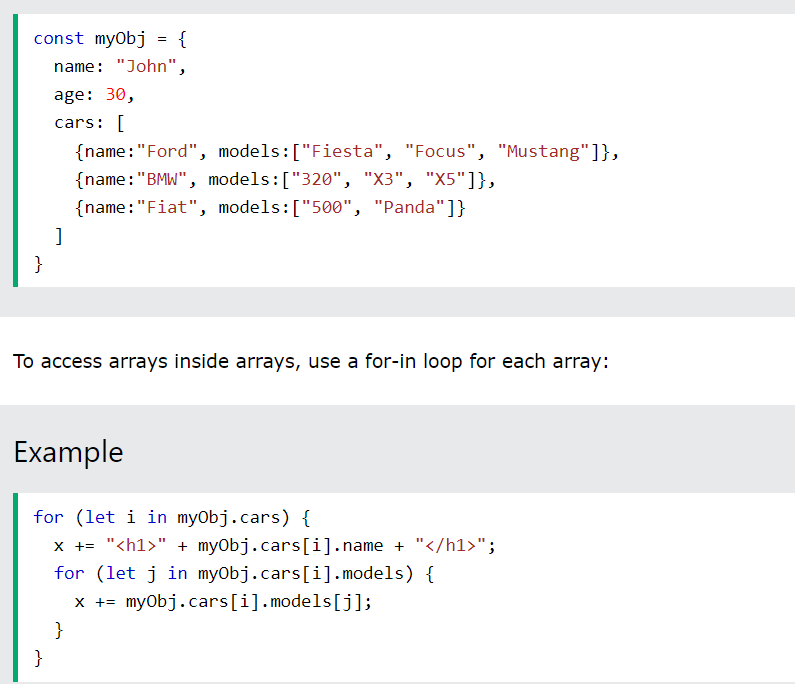
const fruits = ["Banana", "Orange", "Apple", "Mango"];  
Object.preventExtensions(fruits);  
// This will throw an error:  
fruits.push("Kiwi");

let answer = Object.isExtensible(person); **//false**

* + Object.seal(), Object.isSealed() 🡪 adding, deleting properties is not allowed. Modification of properties is allowed.
  + Object.seal() is same as Object.freeze() but modification is allowed in freeze().

12. Arrays/ JSON Objects

* Array is a variable which can hold more than one value.
* It can be created by declaring a variable, also with new keyword.
* Elements can be accessed by referring to its index number. Car[0]
* Array can be converted to a String by using .toString() method
* Arrays are special type of objects.
* Arrays have inbuilt .length(), .sort() method
* Elements can be added in array by using arr.push(“Lemon”)
* Array with named indexes are called as associative arrays/ hashes
* Arrays use “numbered indexes” whereas objects use “named indexes”
* To check whether the given variable s an array or not, we can use isArray() and instanceOf. Typeof for an array returns object



13. JSON

* JSON – Javascript Object Notation
* It is a light-weight, text-format for storing and data interchange format
* It is language independent.
* JSON is “self-describing” and easy to understand
* Written as key, value pair. Key value should be in double quotes.
* JSON array can contain objects
* Use of JSON is to read data from web server and display data in web page.
* JSON.parse() can be used to convert string into js object
* JSON.stringify() is used for converting an object to JSON string.