1. ECMAScript (June 2020) [ECMAScript® 2020 Language Specification](http://www.ecma-international.org/ecma-262/11.0/)
2. What is the code?
3. Statements, Variables (Symbolic placeholders for the value themselves, can be varied with time), literal values

What happens behind the scene when we do var a = 2 (Compilation Phase - Part of the phase is to find and associate all declarations with their appropriate scopes, e.g. var a; is processed in the compilation phase and a=2; is processed in the execution phase)?

1. [Var declaration in scope] Compiler checks if `a` exists in the current scope or not.

2. [Var lookup in scope] Performs an LHS Lookup to check if `a` has been declared, if yes then assigns the value to it.

LHS vs RHS Lookup: <https://drive.google.com/file/d/1w5heP03n_hWl9Vnx40QNUaie0AYvEf9V/view?usp=sharing>

Note: If RHS lookup fails. ReferenceError occurs if LHS lookup fails, the variable is declared in the global scope (if not in strict mode, if in strict mode, the same ReferenceError will be thrown!).

1. Value Types (7) (Not Variable Types - Variables are just containers for values) - number/boolean/string/null/undefined/object/symbol(ES6)
   1. Subtypes of type “object”: array/function

Note (Imp): Built-in type values can be wrapped to their object wrapper counterpart.

e.g. A `string` value can be wrapped by a `String` object, when we call .toUpperCase() on the primitive string value as the `String` object defines the method, .toUpperCase() on its prototype. Similarly, a `boolean` value can be wrapped into a `Boolean` object and a `number` value can be wrapped into a `Number` object.

Note: **Primitive Data types in JavaScript include Number, String, Boolean, Undefined, Null and Symbol.** **The Non-Primitive data type has only one member i.e. the Object**

1. Objects (Compound storage for any type of data)
2. Operators ( =, first RHS is calculated and then assigned to LHS)
   1. Operator types
      1. Assignment
      2. Math
      3. Compound Assignment (+=, etc)
      4. Increment / Decrement
      5. Object property access (.)
      6. Equality & Non-Equality (== loose-equals, === strict-equals, !==, !=)

(==) checks for value equality with coercion allowed and (===) checks for the value equality without allowing coercion.

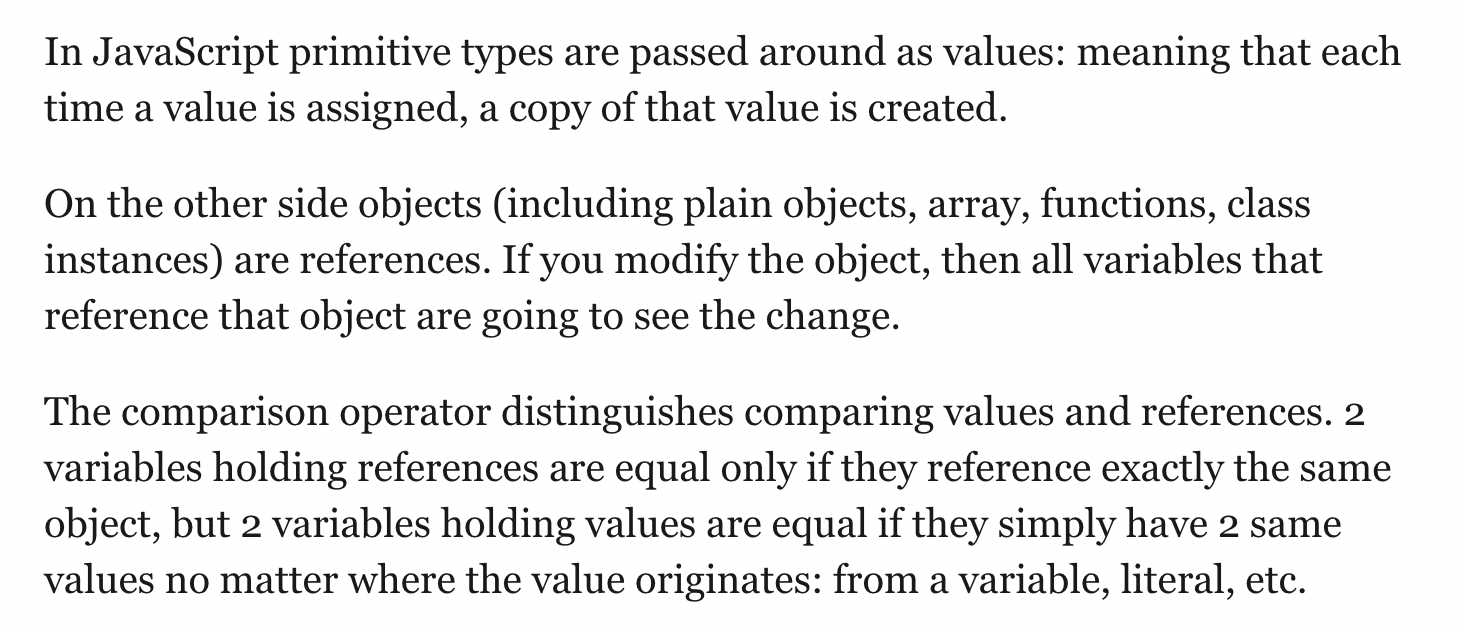
Why is “124” == 124? // JS implicitly converts LHS to its number equivalent. (aka Implicit Coercion). JS goes on through a number of steps to coerce one or both values to a different type until the types match, where then simple value equality can be checked. (Check the algorithm here, <http://www.ecma-international.org/ecma-262/5.1/#sec-11.9.3>)

Note: Forcing numeric comparison (+a == +b)

Forcing string comparison (“” + a == “” + b)

Forcing boolean comparison (!a == !b)

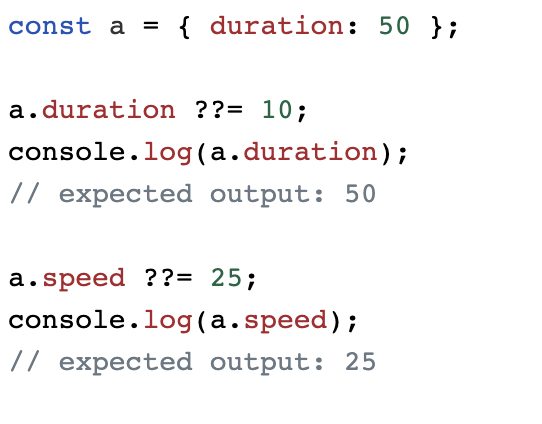
new String(“a”) == “a” (true) but new String(“a) == new String(“a”) (false) as Both the String Object are kept by reference and == or === only checks for the reference and not the actual value. And references here are different.

Explain 12

* + 1. Comparison / Inequality (<= less than or loose-equals)

Explain 13

* + 1. Logical (&&, etc)
    2. The string concatenation operator (+) Explicit Coercion
    3. Bitwise operator (<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Expressions_and_Operators#Bitwise>)
    4. Below: Nullish Coalescing Operator (?? - assign value if not null) and Optional Chaining Operator (?.)



Important link: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Nullish_coalescing_operator>

* + 1. Comma operator: (Evaluates multiple statements and assigns the value from the last statement, e.g.

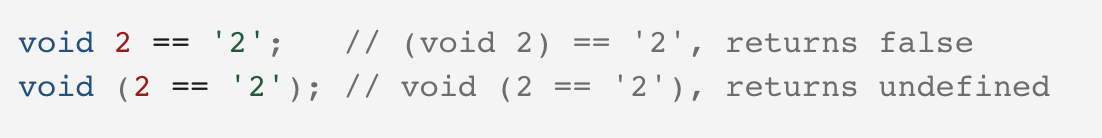
What are the outputs of let y = 24; y = (y+=2, y) // 26

let y = 24; y = (y+= 2, y+= 2) //28

and let x = (2,3) console.log(x) //3)

* + 1. Unary operator (delete, typeof)  
       Delete operator: Should not use on array elements as it just replaces the element with undefined; it returns either true/false)  
       Typeof operator (returns always a string (“”) value)  
        typeof(null) // “object” → Weird? You have to live with it!

typeof(String/Object/Array/Function/Boolean) // “function”  
 typeof(62) // “number”  
 typeof(doesnotexist) // “undefined”

typeof(undefined) // “undefined”  
Void operator (evaluates the expression and returns undefined)  
<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/void>

The void operator is often used merely to obtain the undefined primitive value

1. What does a = b\*2 (Statement); mean to the computer?
2. Expressions (Statements are made up of one or more expressions)
3. How many expressions are there in a = b\*2;
   1. 2 -> Literal value expression
   2. b -> variable expression
   3. b\*2 -> arithmetic expression
   4. a = b\*2 -> assignment expression
4. Expression Statements (A general expression that stands alone, e.g. b\*2;)
5. More on expressions, call expression (A function statement)
6. Why is JS an interpreted language? (Because JS source code is processed each time it’s run. But that’s not entirely accurate, the JS Engine actually compiles the program on the fly and then immediately runs the compiled code)
7. How do we output to the user? (console.log() -> Function Statement)
8. Taking input -> window.prompt from the console
9. Coercing Types

Let age = 24;

Let age1 = Number(age)

console.log(age) // “24” (number converted to a string when printed on-screen - Implicit Coercion)  
console.log(age1) //24 (forced coercion / explicit coercion)

1. JS Comments (Comments should explain why and not what!).
2. Static typing/type enforcement Number age = 42; (This does not work in JS though!)
3. JavaScript uses weak typing/dynamic typing (Allows a variable to hold any type of value at any time without any type of enforcement)
4. Arrays (It’s an object which holds values in numerically indexed positions):

Explain 4 5 5.1 5.2 5.3 5.4 5.5(Imp)  
Whenever you want to make an array, always think, Array.from()! E.g. Generating a sequence of numbers to a given number.

1. Conditional Statements: Something about if statement: The if statement expects a boolean but if you provide something else than boolean, implicit coercion will occur e.g. 0 & “” will become false and 99.99 & “free” will become true.
2. Loops: Explain how they work | Loop flowchart

Explain 8

1. Function: A function is nothing but a block of code that we can call anytime we want to perform a task. Now we do not need to write the same code again and again to perform the same task. | Explain parameters and return statements.

Explain 11 11.1

1. What are anonymous functions and what are their drawbacks?
   1. No useful name present in the stack trace, debugging is difficult.
   2. Can’t call itself now.
   3. Code less readable / understandable.
2. Scope (Technically called, Lexical Scope [write-time]) - A McDonald’s restaurant cannot serve you Thai food! It’s not in its scope!
   1. Each function has its own scope
   2. No two same variables can be declared in one scope (not let & const at least)
   3. The same name variables can exist in different scopes.

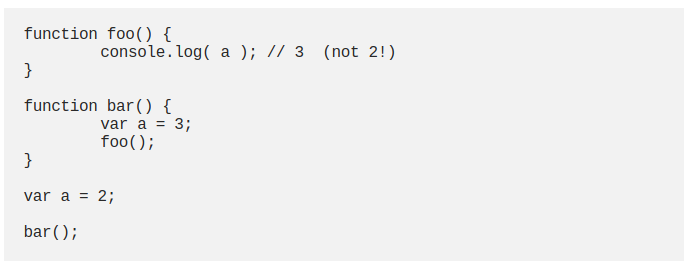
Note: 1) Engines stop scope look-ups when it finds the first match.

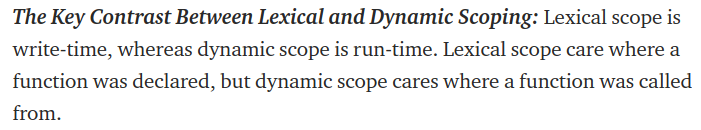
2) Block scope is a tool to extend the earlier Principle of Least Privilege from hiding information in functions to hiding information in blocks of our code.

Note: var variables are '**function scope.** ' *What does this mean?* It means they are only available inside the function they're created in, or if not created inside a function, they are 'globally scoped. '

Explain 1 1.1 1.2 3 11.2(Imp)

1. Scope Chaining - If not found in the current scope, the Engine looks into the next outer containing scope.
2. Dynamic Scope [run-time] - Dynamic Scope looks up the call stack (to check for the function through which it was called) in contrast to the lexical scope (which checks the next scope bubble) which looks up the nested scope chain to find the correct variable reference.





1. Garbage Collection

Explain 20

1. Shadow Variables - The same variable declared at multiple nested scopes is called shadowing (i.e. the inner identifier shadows the outer identifier).

Note: If want to access a global shadow variable, can use the window.a

If the variable is not a global variable then it can't access it. (Non-global shadow variables cannot be accessed.)

​​<https://www.geeksforgeeks.org/variable-shadowing-in-javascript/>

Explain 19(Imp)

1. What can we do to avoid scope collision?
   1. Try hiding variables in the scope
   2. Use global namespaces
   3. Using modules
2. Practice

<https://drive.google.com/file/d/16ZJHzNgDKT59rMWYF4Vw0srav7uvdE2M/view?usp=sharing>

Solution: <https://repl.it/@webber2408/BasicPracticeSolution#index.js>

1. Closures: Closures happen as a result of writing code that relies on the lexical scope. The closure is when a function is able to remember and access its lexical scope even when that function is executing outside its lexical scope.

Explain 10 10.1 10.2 (Great Example)

1. Modules (A way to implement closures!)

Explain 15 15.1 15.2(Imp) 15.3 15.4\_bar 15.4\_foo

1. Strict Mode: (ES5 added a “strict mode” to the language, which tightens the rules for certain behaviors. Generally, these restrictions are seen as keeping the code to a safer and more appropriate set of guidelines. Also, adhering to strict mode makes your code generally more optimizable by the engine.)

Explain 14

1. this in JS

Explain 16

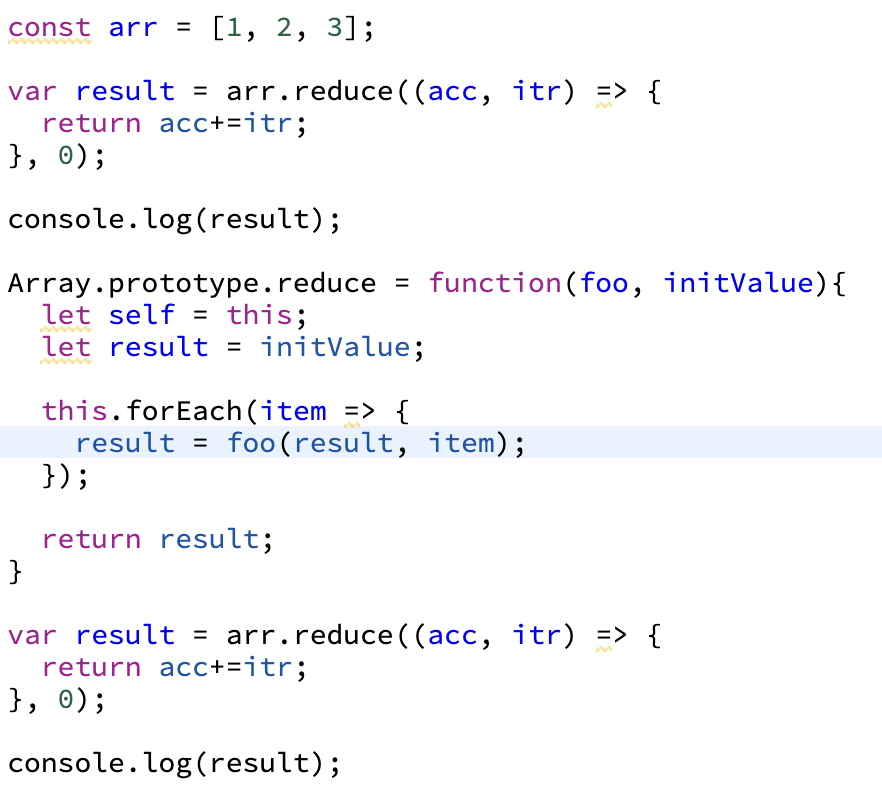
1. Prototype

Explain 17

1. Polyfilling vs Transpiling

Explain 18

1. Reduce Polyfill



**Object-Oriented JavaScript**

1. Object [A collection of related data or functionality]
2. Dot Notation & Bracket Notation
3. Objects are sometimes called **associative arrays** — they map strings to values in the same way that arrays map numbers to values.
4. **this** keyword refers to the current object the code is being written inside.

Why do we need `this`?

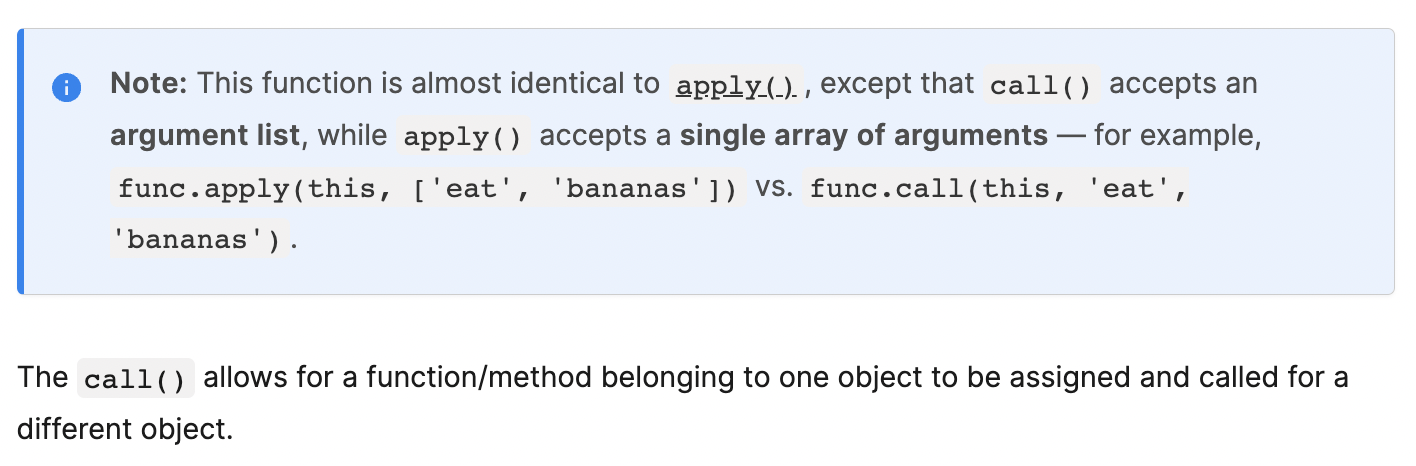
Because it ensures that the correct values are used when a member’s context changes, there can be two different object instances and we want to access their own methods and not of others.

1. Object names are called **namespaces**.
2. **abstraction** — creating a simple model of a more complex thing, which represents its most important aspects in a way that is easy to work with for our program's purposes.
3. **Object instances** - Objects that contain the data and functionality defined in the class.

**Instantiation** - creating an object instance from a class.

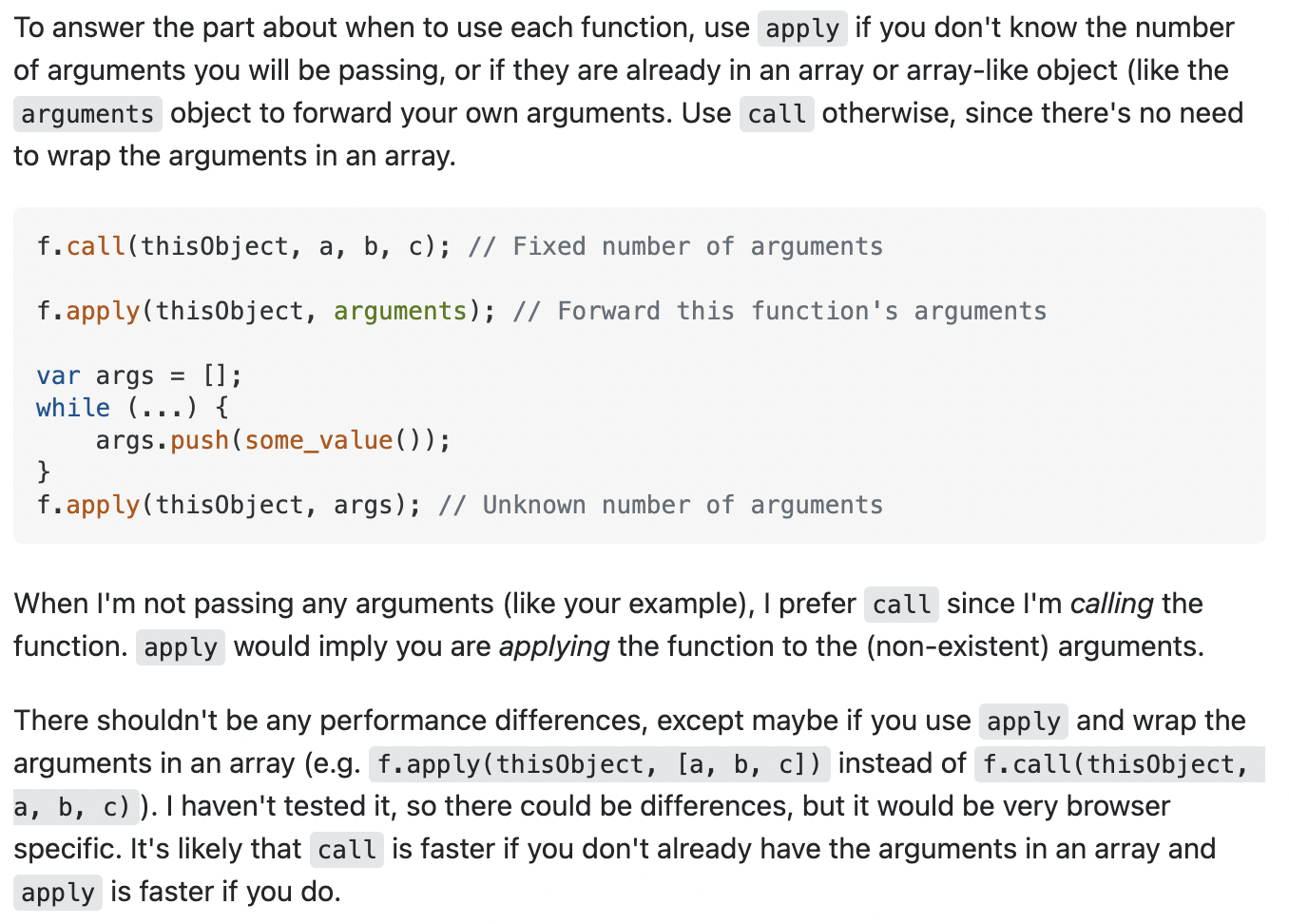
1. **Polymorphism** - The ability of multiple object types to implement the same functionality is called polymorphism.
2. **Constructors -** special functions to define and initialize objects and their features.

JavaScript Revision:

1. **Go through JS Masters.**
2. **Read YDKJS - Objects.**
3. **Read Mocha / Chai Blog** - <https://javascript.info/testing-mocha>, <https://www.chaijs.com/api/assert/#method_equal>
4. **Call(), Bind(), Apply(). - YDKJS Ch:2 Pg-18,19**

**Call()** - <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/call>

**Apply()** - <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/apply>



**Bind()** - The bind() method creates a new function that, when called, has its keyword set to the provided value, with a given sequence of arguments preceding any provided when the new function is called. *Advantage? Can pre-set some of the arguments of the target function.*

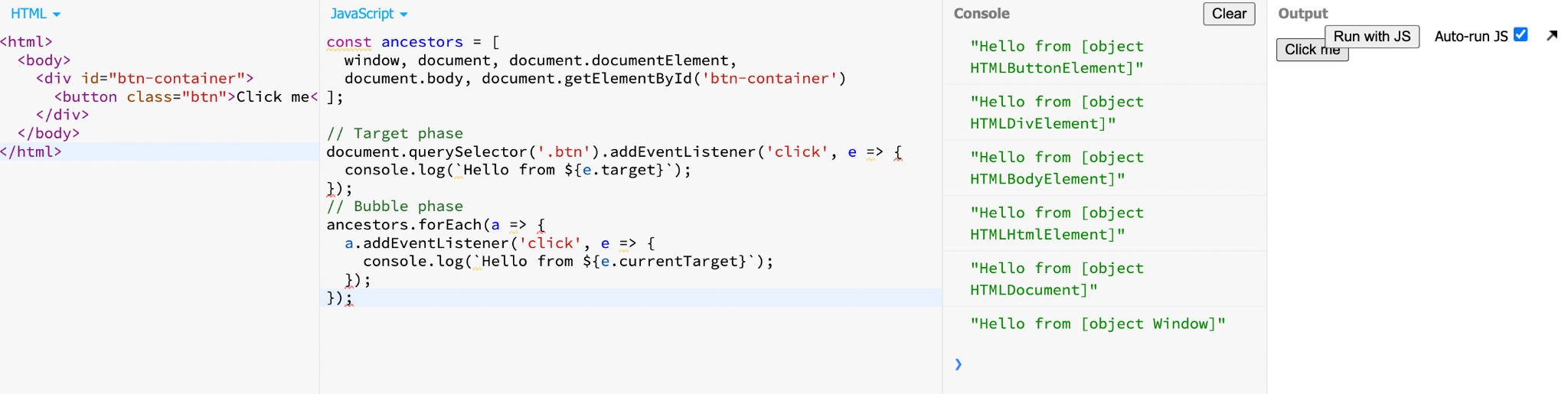
1. **Event Delegation / Propagation / Bubbling / Capturing.**

<https://www.30secondsofcode.org/articles/s/javascript-event-bubbling-capturing-delegation>

<https://jsbin.com/kaqagederi/2/edit?html,js,console,output>

Most events bubble but some like “focus” do not! (<https://codepen.io/cferdinandi/pen/pqJZdK?editors=1111>)

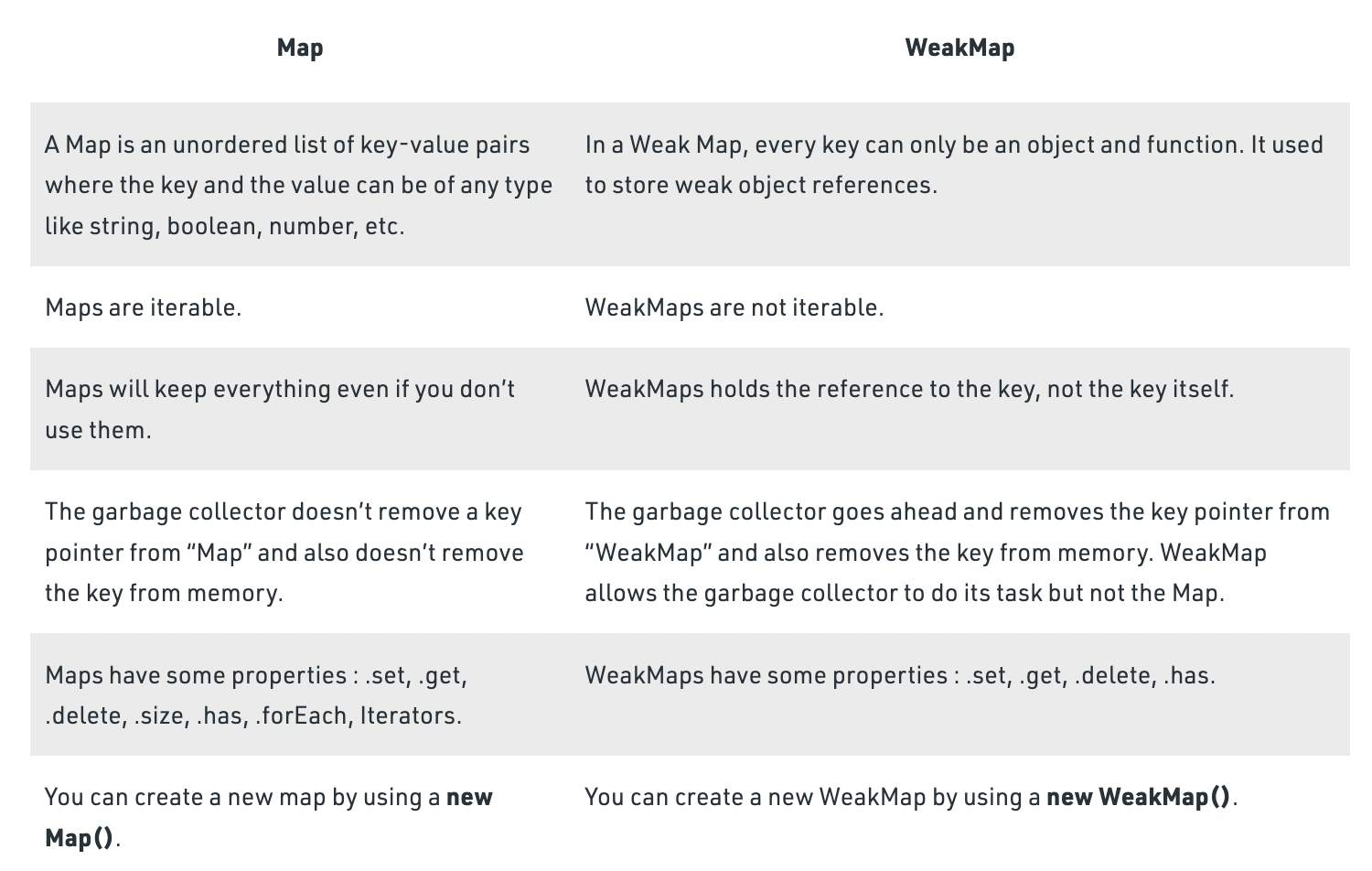
<https://gomakethings.com/whats-the-difference-between-javascript-event-delegation-bubbling-and-capturing/#:~:text=tl%3Bdr%3A%20event%20delegation%20is,events%20that%20don't%20bubble>.



*The event propagation mode determines in which order the elements receive the event.* ***With bubbling, the event is first captured and handled by the innermost element and then propagated to outer elements****. With capturing, the event is first captured by the outermost element and propagated to the inner elements.*

1. Throttling and Debouncing.
2. **Difference between Map() and WeakMap()**

<https://www.geeksforgeeks.org/what-is-the-difference-between-map-and-weakmap-in-javascript/>



1. Reduce Polyfill.
2. Event Loop.
3. How DOM Works - Google Keep.
4. Currying in JS.
5. Latest ECMAScript Developments.
6. Revise OOPS Concepts.
7. What is TDD vs BDD?
8. JS Design Patterns.
9. Vanilla JS Handling forms.
10. Revise CSS.