What is JavaScript?

It is a:

1. High-Level: By this I mean you have to worry about managing resources such as creating space / memory for your variable that you declare, JavaScript will take care of that.
2. Garbage-Collected: Can be regarded as one of the powerful tool that takes memory management away from us developers. It’s a built in algorithm inside JavaScript engine that removes old and unused objects from the memory in order not to block it with unnecessary stuff, more like having a cleaner.
3. Interpreted or Just in Time: Ability of JavaScript to interpret and compile human written code into machine code of 0’s and 1’s
4. Multi-paradigm: This is an approach and mindset of structuring your code. 3 Popular paradigms are:
   1. Procedural programming: This is organizing your code in a linear way
   2. Object Oriented Programming: Prototype Object oriented approach, Almost everything in JavaScript is an object except for primitive values such numbers, strings etc. but and array it’s an object
   3. Functional Programming: Reusable code
5. Prototype-based object – oriented: Ability to create blueprint with class and constructor methods.
6. First-class functions: Functions are treated as just variables, you have the ability to pass a function into other functions and return them from functions. Example Event listeners that takes two arguments the event and the function.
7. Dynamic: JavaScript is dynamically typed, you don’t have to specify data type of your variables, they are only known when we execute the code unlike TypeScript and there is a lot of controversy about this being good or bad whether or not JavaScript should be a strongly typed language, like TypeScript.
8. Single-threaded: Synchronous architecture, the execution of each operation depends on completing the one before it. Each task require answer before moving to the next iteration
9. Non-blocking event loop: This is Asynchronous architecture, the execution of one task doesn’t depend on another. (Call Back Functions)

JavaScript engine it’s a computer program that executes JavaScript code. Every browser has its own engine such as V8 for Google Chrome that also powers NodeJS.

Any JavaScript engine contains a Call Stack and Heap.

1. Call Stack: It’s where our code is executed, using execution context.
2. Heap: Unstructured memory pull which stores all the objects that application needs

Just in Time compilation of JavaScript

Modern JavaScript use Just-In-Time(JIT) compilation.

When the piece of code enters the engine first that data is:

1. Parsed (Reading the code). The code is parsed into AST (Abstract Syntax Tree)
2. Compilation: This step takes AST and compile it into machine code.
3. And that machine code gets Executed right away.

Cell C Contracts Exp dates :Jan & Sep next year.

How JavaScript is Executed in Call Stack:

* This is Asynchronous architecture

After compilation and when our code is ready to be executed, Global Execution Context is created i.e. Top Level Code.

Top Level Code; Is the code that is not inside any function.! And it makes sense because functions should be executed when they are called.

ⁿᵇ Call back functions that are executed after top level code. Asynchronous architecture.

Global and Local variables!

Execution Context has:

1. Variable Environment
2. Scope Chain
3. this keyword.

Scope and Scope Chains

Scoping controls how our program variables are organized and accessed by JavaScript engine.

* *Where do variables live?*

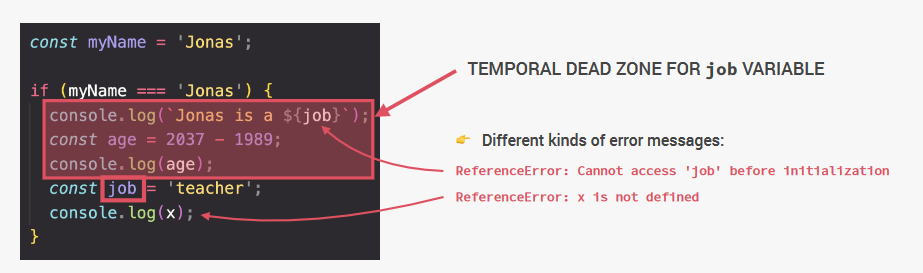
Scope is the space or environment in which a certain variable is declared. There are

1. Global scope: This is for Top Level Code. These are variables outside any function or block. Variables declared in global scope are accessible anywhere.
2. Function scope also called local scope: These are variables only accessible inside function, NOT outside.
3. Block scope: Variables that are only accessible inside block, however this only applies to *let* and *const* variables *var* are accessible.

Hoisting

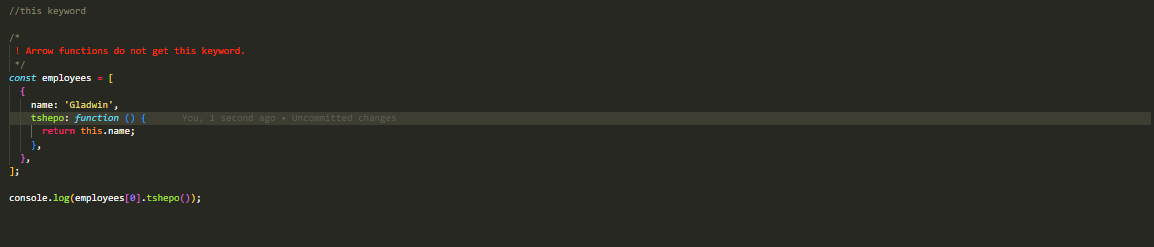
It makes some types of variables accessible/usable in the code before they are actually declared. i.e. Variables lifted to the top of their scope.

TDZ (Temporal Dead Zone): Attempting to access variable before it’s declared and initialized. Variable will be available after declaration thus area or lines prior declaration it’s TDZ (Temporal Dead Zone)

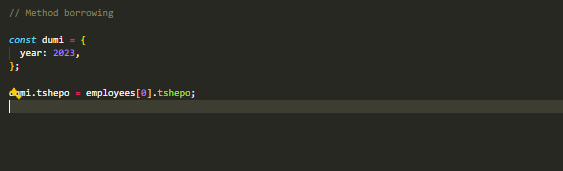


this keyword

It’s a special variable that is created for every execution context(function)



Method borrowing



this keyword always points to the object that’s calling the method.