**Javascript can run various environments:**

* Web browsers
* Servers
* Mobile Apps
* Desktop Apps
* Command Line (CLI)
* Embedded Systems & IoT Devices
* Game development
* Cloud

Etc.

**Embedded Systems:**

* Micro oven’s control system.
* Car’s Engine Control unit.

Etc.

**IoT Devices:**

These are physical devices connected to the internet.

They are capable of collecting, sharing and acting on data.

* Smart lights
* Wearable like fitness trackers or watches
* Home cameras

Etc.

**NodeJS:**

* JavaScript Runtime environment.
* Built on chrome’s V8 Javascript Engine.
* Cross Platform and open source.
* Executes Javascript outside web browser.
* NodeJS has an event-driven architecture capable of asynchronous I/O.

**Outside web browser:** Can run on servers & other environments.

**Cross Platform:** Can run any OS like Windows, Mac, Linux Etc.

**Note:**

* Without JS Engine, you can’t run ant JavaScript code.
* Be it anywhere like browser, server & any devices, you need JS engine to run JavaScript.

**History:**

**Released:** 2009 – **Ryan Dahl** – **OpenJS foundation.**

**Name:** Initial name: web.js – NodeJS(later)

**JS engine:**

* Initial started - Spider Money (Firefox browser)
* Currently (Chrome’s V8 Engine)

**Blocking I/O & Non-Blocking I/O:**

* NodeJS was developed to create Non-blocking I/O servers.
* Before that there were Blocking I/O servers like Apache Http Server etc.

**Blocking I/O:**

* If a request ‘A’ comes to the server, one thread is assigned for the request.
* ‘A’- request’s thread wait for the request to complete (if request takes time) and will not share the tasks or work of another request.
* If a request B comes to the server, one more thread is assigned for the new request and the ‘A’-requests thread will not share the work of new ‘B’- request.
* Even if ‘A’-requests thread doesn’t have anything to do, it will wait until the request completes & will not share the work of another new request.
* Each thread blocks the server (traditional approach)

**Non-Blocking I/O:**

* In this case, ‘A’-request’s thread will not wait for the request (if request takes time) to complete.
* It delegates the task and move on to next task/request without waiting.
* Once the task completes, task notifies the server & it picks it up to send response or complete it.

**JS on Server:**

* Initially JavaScript was running on browsers only.
* Before NodeJS other languages like Java, Dot Net etc. were used on servers.
* When NodeJS was released, Javascript can run on servers & other environments.
* And now after NodeJS is released, with one language JavaScript we can run on Client & Server.
* A developer can learn one language and can develop whole application Front-End & Back-End.
* Full-Stack development came in to picture, may be one of these reasons also.

**NodeJS built on:**

* NodeJS is built with JavaScript & C++ majorly.

JavaScript: 61%

C++: 23%

Python: 10%

Others: Python, C, HTML, Shell

**V8 Engine:**

* Google’s open source high-performance JavaScript & WebAssembly engine.
* Used in Chrome & NodeJS.
* It implements or executes ECMAScript & WebAssembly.
* It runs on Windows, macOS & Linux Systems Etc.
* V8 can be embedded in to any C++ application.

**V8 Engine built on:**

* V8 Engine is built on C++ & Javascript majorly.

C++: 72%

JavaScript: 25%

Others: Python, C, etc.

**ECMAScript:**

* ECMA (European Computer Manufacturers Association) Script is standard for scripting languages like JavaScript, JScript etc.
* It is developed & maintained by EMA international Technical Committee (TC) 39.

**ECMA Standards:**

* ECMA standards for JavaScript means – For Ex: It define how variables, functions, objects, loops should work.
* In ensures that behavior of language is consistent across different implementations like browsers (Chrome, Firefox etc.) & environment like NodeJS.
* DOM manipulation & other web APIs are feature of JavaScript implemented in web browsers and is not part of ECMA Script standard.

**WebAssembly:**

* WebAssembly is a low level, binary instruction format code.
* It is compiled form high-level programming languages like C, C++ etc.
* It is compiled in to native machine code by V8 engine and executed.
* V8 Engine can also execute WebAssembly code alongside JavaScript.

**V8 can be embedded in to any C++ application:**

* It means integrating v8 engine in to C++ program to enable that program to execute Javascript code as part of its functionality like interact & pass data between C++ & JavaScript by exposing C++ functionality to JavaScript.

**JS Runtime Environment:** NodeJS includes several key components that allows it to function as a standalone environment for executing JavaScript.

* V8 JavaScript Engine.
* LibUV
* Core JavaScript APIs
* Event Loop
* NodeJS Modules

Etc.

**V8 Engine:**

**JavaScript (High level language)**

↓

**Byte Code,**

**WebAssembly**

↓

**Machine/Binary code**

**High level language | Low level intermediate code | Machine/CPU executable code**

Javascript | Byte code, WebAssembly code | Machine or Binary code

**High level language**:

* Human readable code.
* Examples: JavaScript, TypeScript, C++ etc.
* But, V8 Engine can only execute JavaScript & WebAssebly directly.
* Any other high-level languages like: Type Script (Transpiled to JavaScript), C++ (Compiled to WebAssembly) etc. to be executed by V8 Engine.
* WebAssembly code directly goes to Interpretation & Compilation Phase skipping preliminary steps in V8 engine like parsing in to AST tree, Tokenization etc.

**Install NodeJS:**

* Download installer file from NodeJS website and install it.
* To check, Run the Cmd: node -v

Ex-Output: 22.13.0

* NPM also will be installed along with NodeJS. To check, Run the Cmd: npm -v

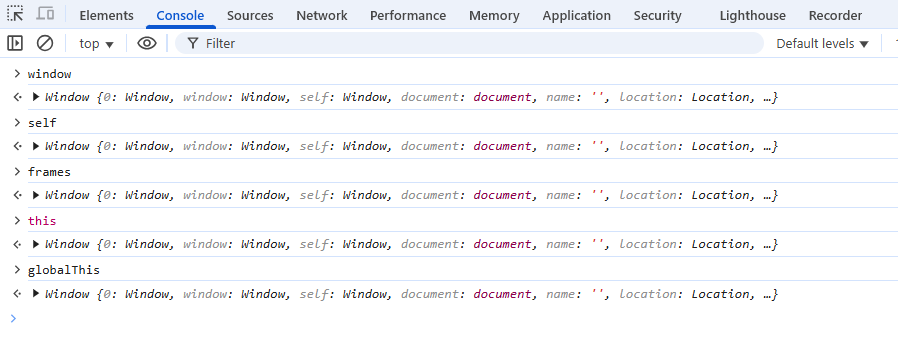
Ex-Output: 10.9.2

Node REPL: (Read-Evaluate-Print-Loop)

* When you enter command node in terminal, NodeJS runtime is created i-e Node REPL.
* You can write JavaScript & execute code there, just like in browser console.
* We can write only small pieces of code in REPL, difficult to develop projects there, So, we need to install VS code editor or ant other editors for easy development for projects.

**global object & this:**

* In Browser, global object cab be called with different names: window, self, frames, this, globalThis.



* In NodeJS, global is the name of the global object.
* Due to different reasons many global object names are defined in JavaScript.  
  In one of the latest versions, they came up with one single name “globalThis” to clear confusion & this name can be used as global object anywhere.
* Global object has methods like setTimeout, setInterval etc. and these are called super powers.
* Global object (and those methods or functionalities) is not part of V8 Engine or JavaScript.
* It is provided by browsers like chrome on browser environment and

In NodeJS environment provided by NodeJS.

* And this can be accessed by V8 engine during execution of JS code.

**Why “this” is empty object in NodeJS files?**

* In NodeJS, by default CommonJS module system is used for “.js” files. Each file is treated as separate module & wrapped inside a function by NodeJS internally.

Ex: (function (exports, require, module) {  
 //Your module code here  
 });

* NodeJS avoid binding “this” to the global object in module to main encapsulation & accidental leakage of variables into global scope.

Ex: If “this === global” in files/modules,

In file1.js:

this.x = 5 is same as global.x = 5;

Now, in file2.js:

this.x =5, because global cane be accessed in any file & global.x = 5;

This way, global variables can be polluted & so this is not global inside file in NodeJS.

* If we make this as global in files/modules, then encapsulation is lost, no isolation.
* In NodeJS REPL: this===global. Because there are no files/modules there.
* In ES module system: this===undefined.

Because: -> ECMA Script specification defines it that way.

-> It ensures consistency with browser ESM (In browsers “this” in a module is also undefined) & prevents global namespace pollution.