**Java script run time**

# **What is the runtime environment?**

A runtime environment is where your program will be executed.

The JavaScript runtime environment provides access to built-in libraries and objects that are available to a program so that it can interact with the outside world.

JavaScript is a single threaded at runtime unlike the other programming language. That mean the code execution is done sequentially and any code that takes a long time to execute will block anything that be executed after that. When JavaScript executing is blocked, the browser will stop doing all things like scrolling the web page, printing something on the web page, listen to DOM events, and it won’t respond to anything until that task is completed.

In this research, two JavaScript runtime environments will be covered:

* The runtime environment of a browser
* The node runtime environment

# A browser runtime environment

The most common place where executed JavaScript code is in a browser.

The browser runtime environment is comprised of the following elements:

* The JavaScript engine
* Web APIs
* The callback queue
* The event loop

## The java script engine

To start writing JavaScript we don’t need to install any specific software because each web browser has its own version of the JS engine that parses the code. For example chrome uses the V8 JS engine and Firefox uses SpiderMoneky engine. The JavaScript engine is translate source code that developers write into machine code that allows a computer to perform specific tasks.

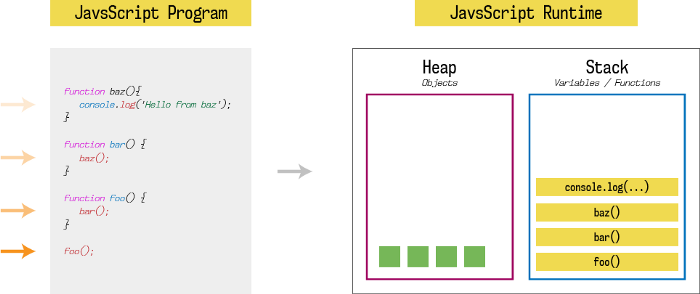
Netscape browser used the SpiderMoneky JavaScript engine was a rudimentary interpreter with no optimizations. Running the JavaScript code with this engine was slow but it worked.

V8 is written in C++, and it's continuously improved. It is portable and runs on Mac, Windows, Linux and several other systems.

The V8 JS engine has many version. In each version added many features to optimize JS code. And **V8** is one of the most popular JavaScript engines, and used by Google Chrome browser and nodeJS.

JavaScript runtime has one stack and one heap storage per process. A **heap** is a free memory storage unit where you can store memory in random order. Heap is managed by the JavaScript runtime and cleaned up by the garbage collector.

A **stack** is **LIFO** data storage that stores the current function execution **context** of a program. And when execute it popped from stack.



In the above figure, we have a simple JavaScript code that contains three functions foo, bar and baz. When we run this code, first the function foo called and then the call chain begins until console.log () is executed.

Until a function returns something (while the function is executing), it won’t be popped out from the stack. The stack will pop entries one by one as soon as that entry (function) returns some value.

## Web APIs

The Web APIs are not a part of the JavaScript engine, but they are part of the runtime environment provided by the browser. There are a large number of APIs available in modern browsers that allow us to a wide variety of things. Like **Manipulate documents** as DOM API and draw graphics as Canvas API.

Features like event listeners, timing functions and AJAX requests all sit in the Web APIs container until an action gets triggered. A request finishes receiving its data, a timer reaches its set time or a click happens and this triggers a callback function to be sent to the callback queue.

## Callback queue

It’s also called a message queue or task. The callback queue stores the callback functions sent from the Web APIs in the order in which they were added.

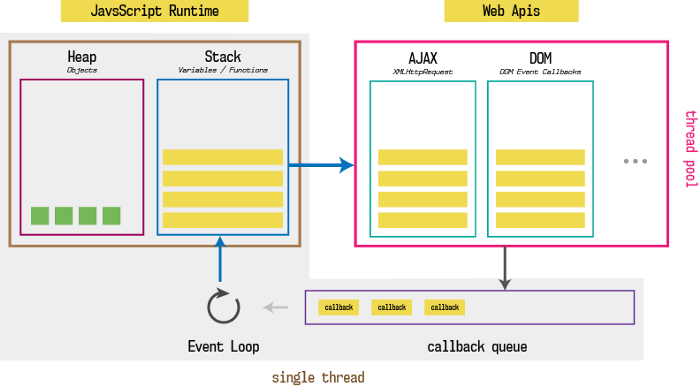
Callback functions will sit in the queue until the call stack is empty, they are then moved into the stack by the event loop.

## Event loop

The job of the event loop is to constantly monitor the state of the call stack and the callback queue. If the stack is empty it will grab a callback from the callback queue and put it onto the call stack, scheduling it for execution.

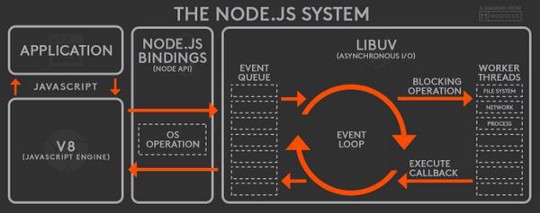
This is why JavaScript often gets described as being able to run asynchronously, even though it is a single-threaded language.

In general, in most browsers there is an event loop for every browser tab, to make every process isolated and avoid a web page with infinite loops or heavy processing to block your entire browser.



# Node runtime environment

The Node runtime environment was created in 2009 to execute JavaScript code without a browser. This enabling programmers to create full-stack applications using only JavaScript language.  
  
Node.js uses promises more than browser. So we can do must of things in the background. Node.js uses Google’s V8 engine to provide JavaScript runtime and employs its own event loop using the libuv library which written in c language. Node follows the same callback approach like Web APIs and works in a similar fashion as the browser.



From the figure above the LIBUV section looks like Web API but it also contains event queue (callback queue or message queue) and the event loop. V8, event queue, and event loop runs on the single thread while worker threads are responsible to provide asynchronous I/O operation. That’s why Node.js is said to have as**non-blocking event-driven asynchronous I/O architecture.**

# Difference between node and browser runtime environment

There are several differences between them, mostly because the node is outside of the browser. The browser environment data values and functions, like window.alert (), can’t be used in node. Instead, the node runtime environment gives back-end applications access to many of features unavailable in a browser, such as access to server’s file system, database and network.

Also, the Node contains some of functions not present in browser, such like process.nextTick (), and setImmediate () (not standards on all browser). The nextTick function used to instruct the engine to invoke this function at the end of the current operation, before the next event loop tick starts. And setImmediate function used when you want to execute some piece of code asynchronously, but as soon as possible.

Event loop executes tasks in process.nextTick queue first, and then executes promises microtask queue, and then executes macrotask (setImmediate and setTimeOut) queue.

# References:

1. <https://medium.com/@gemma.croad/understanding-the-javascript-runtime-environment-4dd8f52f6fca>
2. <https://www.codecademy.com/article/introduction-to-javascript-runtime-environments>
3. <https://medium.com/jspoint/how-javascript-works-in-browser-and-node-ab7d0d09ac2f>
4. <https://nodejs.dev/en/learn/the-nodejs-event-loop>