# How javascript works

**Overview :-** - The javascript Engine.
The call stack.
Browser or web APIs.
Asynchronous javascript
Event loop.
why need Asynchronous process.

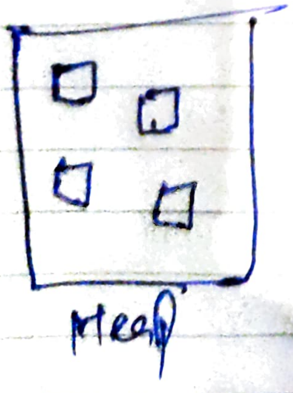
## Javascript Engine :-

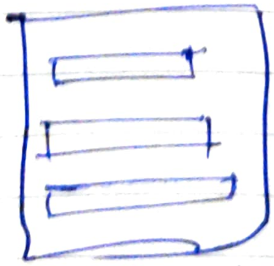
* Javascript use google V8 engine for compiler.
* The V8 engine is used inside chrome and
* Node.js for example.

The engine consist of two main component.

**Memory heap** — this is where the memory allocation happens.

**Call stack** — this is where your stack frames are as your code exectute


Heap


call stack.

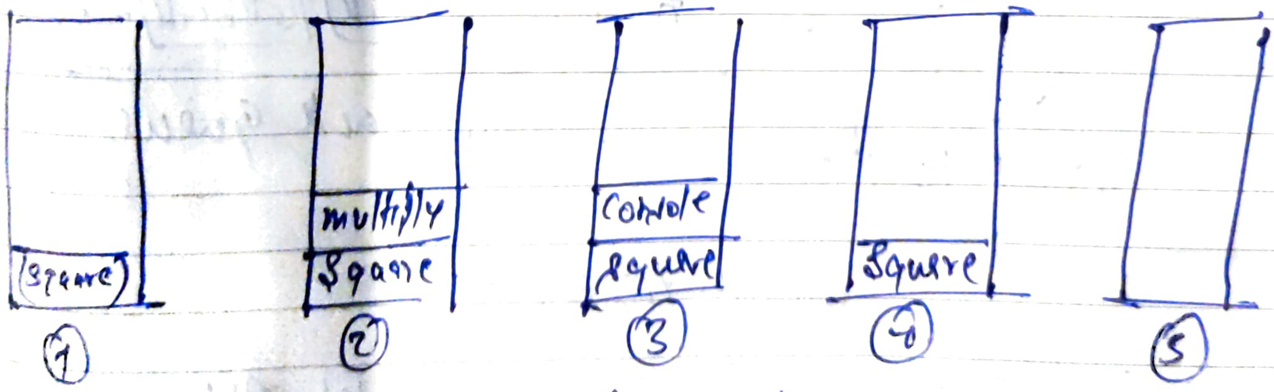
The call stack :- JS is a single-threaded programming language, which means it has a single call stack. Therefore it can do one thing at a time.

* If we step into a function, we put it on the top of the stack. If we return from a function we pop off the top of the stack.

Ex :-
```
function multiply (x,y){
        return x*y;
}

function square(x) {
        var s = multiply(x,x);
        console.log(s);
}

square(5);
```

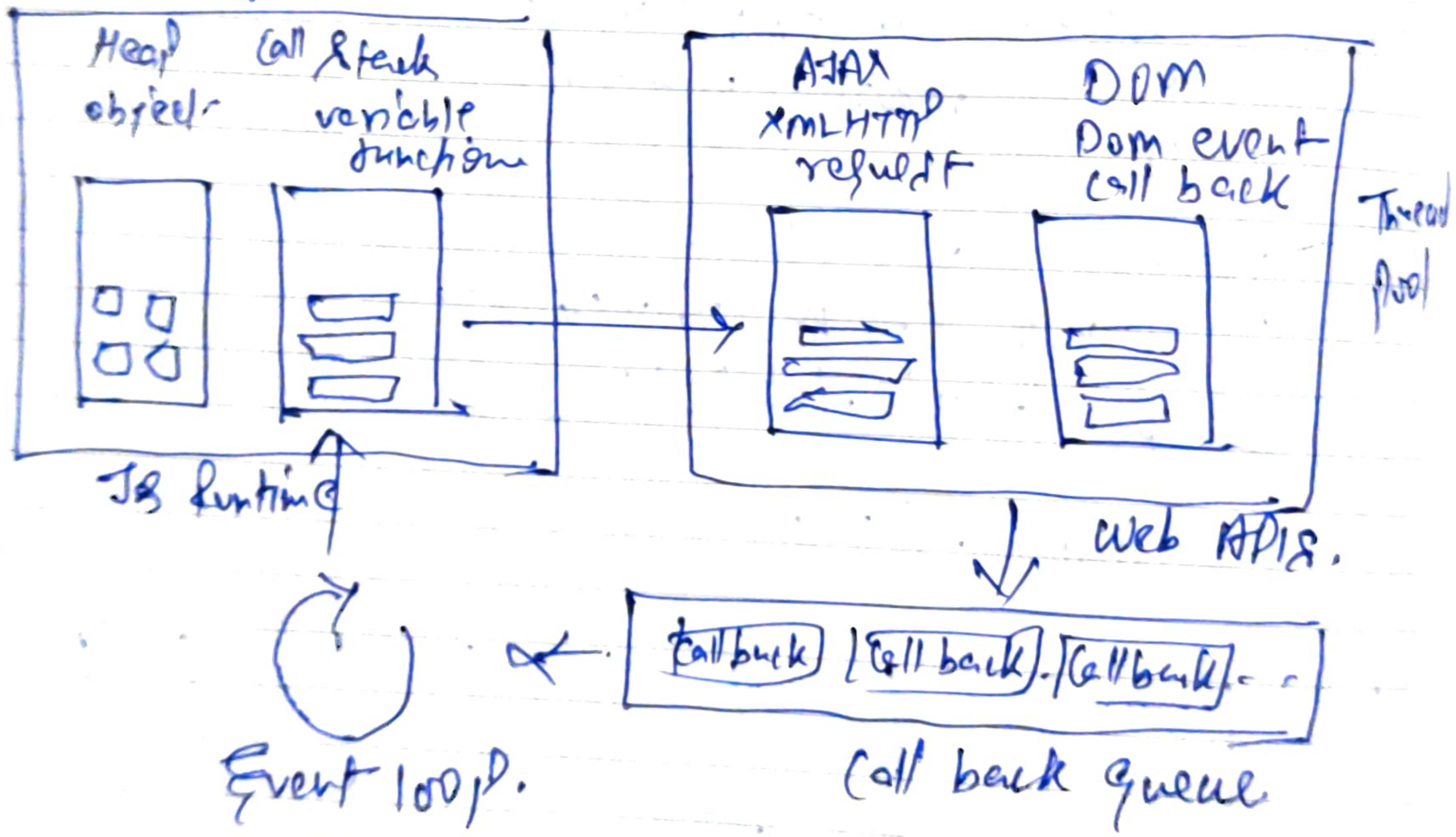


Call stack

AST → Abstract syntax tree.

## Browser or web APIs :-

* There are not provided by engine. they are provided by browser.
* They may take more time as usual js task
* These web APIs are asynchronous. That means you can instruct these APIs to do something in background and return data once done.



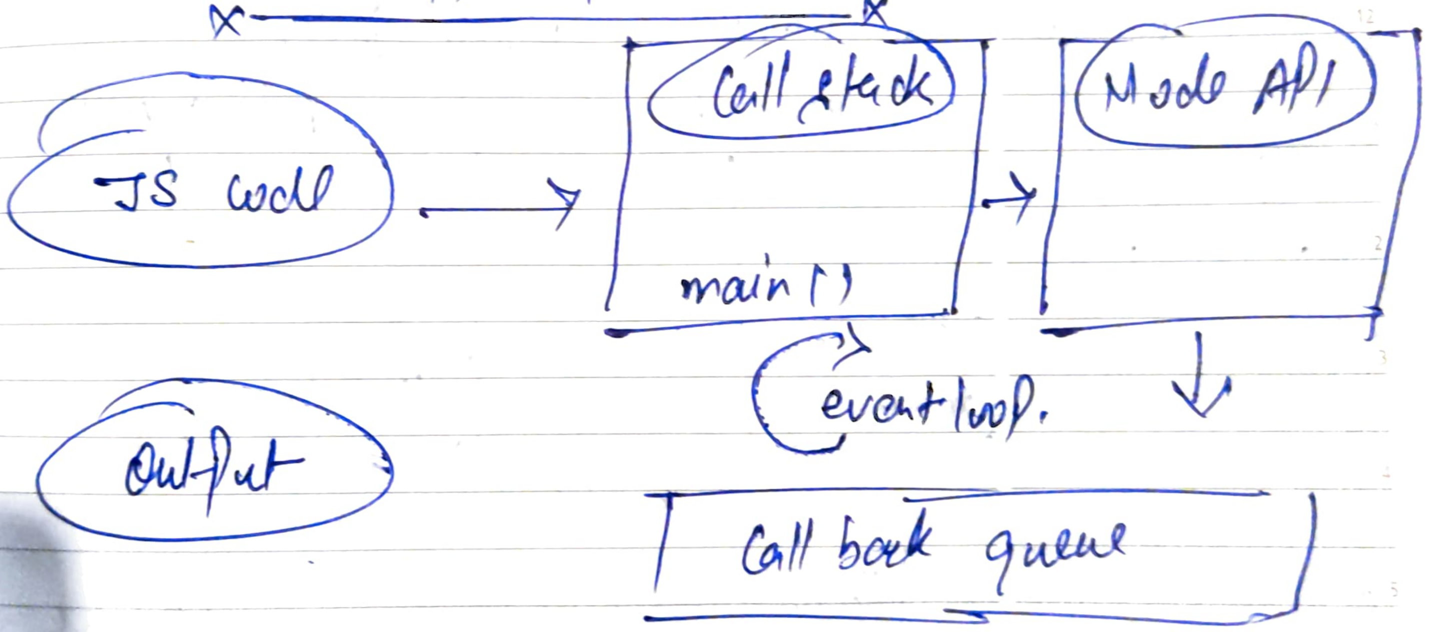
Single thread.

Event loop :- * Once job is done, web APIs bind result of that job to call back function and publishes a message to message queue with that call back.

* The only job of event loop is to look at callback queue and once there is something pending in callback queue, push the callback to the stack.

* Event loop pushes one callback function at a time, to the stack.

How Node is work



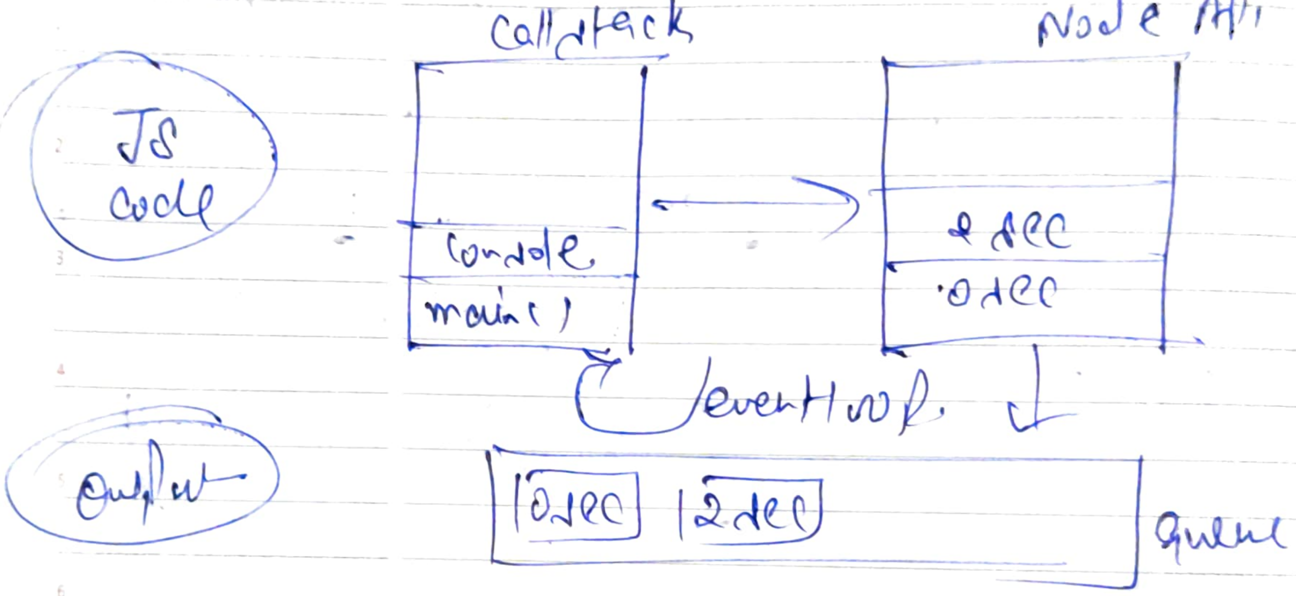
Ex:-
```
console.log ("starting")
setTimeOut (() => {
   console.log ("2 sec");
 }, 2000);
setTimeOut (() => {
     console.log ("0 sec");
 }, 0);
```

console.log ("finish");

Output:-)

Starting
finish
0 sec
2 sec..

Note:-) setTimeout inherited from C, C++ so it is part of Node APIs.



* Variables declared with "var" in js are function scoped.

* Variables declared with "let & const" are block scoped.

Ex:)

function scoped

```
function modJS (x){
    if (x){
        let name = "shirish";
        const sname = "patel";
    }
console.log ("my name is"+ name + " " + sname);
}

modJS (True);
```

Block scoped.

Output:) shows error.

---

Execution context, Execution stack

$$\begin{tikzcd}
\boxed{\text{JS Code}} \arrow[r] & \boxed{\begin{array}{c}\text{Execution context}\\ \text{The enviroment in which}\\ \text{our code is executed}\\ \text{and is evaluated}\end{array}} \arrow[r] & \boxed{\begin{array}{c}\text{Global Execution context}\\ \text{Default}\end{array}}
\end{tikzcd}$$

- JS engine creates the global execution context before it starts to execute any code.
- Variables and functions that is not ~~any~~ inside any function. A new execution context gets created every time a function is executed.

Execution Stack (ES) Execution stack, also known as "calling stack" is a stack with a LIFO (last in, ~~first~~ first out) structure. which is used to store all the execution context created during the code execution.