→ **Property of Javascript :—**

1) Javascript is synchronous in nature ← default nature
2) Javascript is single threaded language.

→ Javascript is synchronous in nature means that the code is executed one line at a time, and the next line cannot be executed until the current has completed.

It behaves synchronously, it is not asynchronous in nature. This is something that have to completely feed in our mind that Javascript is sy synchronous in nature that means it will execute the code line by line and if there is some piece of code that is going to take lot of time, it is going to stop there, give the code whatever time is required and then only move forward.

But only if we execute valid ecmascript code which is given by the standards

Example :—

```
console.log("Hi, We are starting");
for(let i=0; i<1000000000; i++){
    // some task
}
console.log("Done");
```

Till the time for-loop is going to execute, it is going to taking that amount of time and then only move ahead and then print is done.

because var-loop is a native piece of javascript code

→ Javascript is single threaded language, which means that it can only process one task at a time.

In other words, it can only execute one line of code at a time, and if a line of code takes a long time to execute, the rest of the code will be blocked until the line of code has finished.

This single threaded nature of Javascript can cause performance issues when dealing with long running tasks.

To overcome these limitations, javascript uses asynchronous programming techniques, such as callbacks and promises, to handle multiple tasks concurrently.

Example:—

```
console.log("hi");
setTimeout(function(){
    console.log("time done");
}, 5000);
console.log("by");
```

// output:

hi
by
time done

Previously, there was a for-loop which was time consuming, it was waiting but we have a setTimeout which was not waiting.

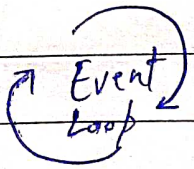
* **Javascript Runtime :—**

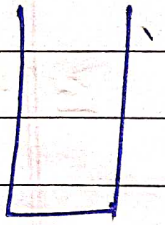
Javascript runtime refers to the environment in which javascript code is executed. It consists of the javascript engine, which is responsible for parsing, compiling and executing the javascript code, as well as the APIs and libraries that provide additional functionality to the javascript code.

Inshort, the javascript runtime is the environment in which Javascript code is executed, and it provides the necessary resources and APIs for executing and interacting with that code.

⇒ " We never pause a synchronous piece of code execution, we never pause something that is native to javascript at any cost.

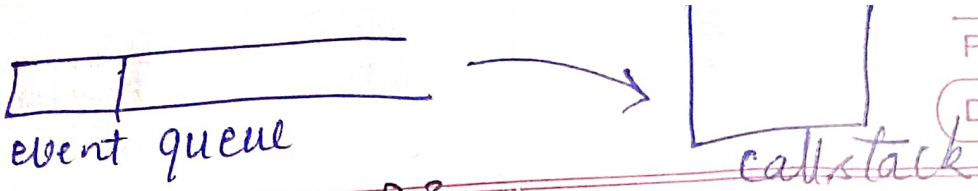
* **Event loop :—**





It keeps on checking wether the call stack is empty or not and no global code is left.

It infinitly times checks, is the callstack is do? is the callstack is done? oh there is nothing in the callstack, is the global piecce of code is done? it constantly checks.



Page No.
Date / /

So, why it checks ??

So whatever callbacks you we have in our queue, can not execute immediately. They can execute if and if only, there is nothing in the callstack, (callstack is empty) and no global piece of code is left. If everything is done, event loop is going to pick one callback from the event queue, pushes to the callstack and the callstack now starts its execution.

If there is any other function in event queue, it will keep waiting so and so for.