JavaScript is single-threaded. So **one task** can be executed at a time. Imagine you want to execute a task that takes a minute. During that task everything else in your website is **blocked**.

To deal with this problem, the browser give us WEB APIS!

These APIs include DOM, timers, HTTP requests etc

How it works

As we learned in a previous post when we invoke a function it gets added in the **Call Stack.**

When the function returns it pops from the stack.

How it works

What happens with the functions that are part of Web Apis?

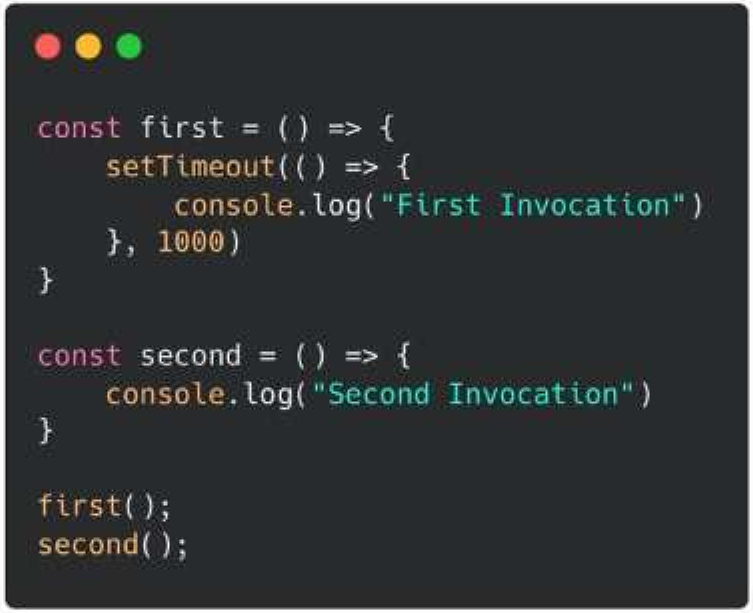
The Web API will take care of the callback and the function will pop immediately from the stack.

How it works

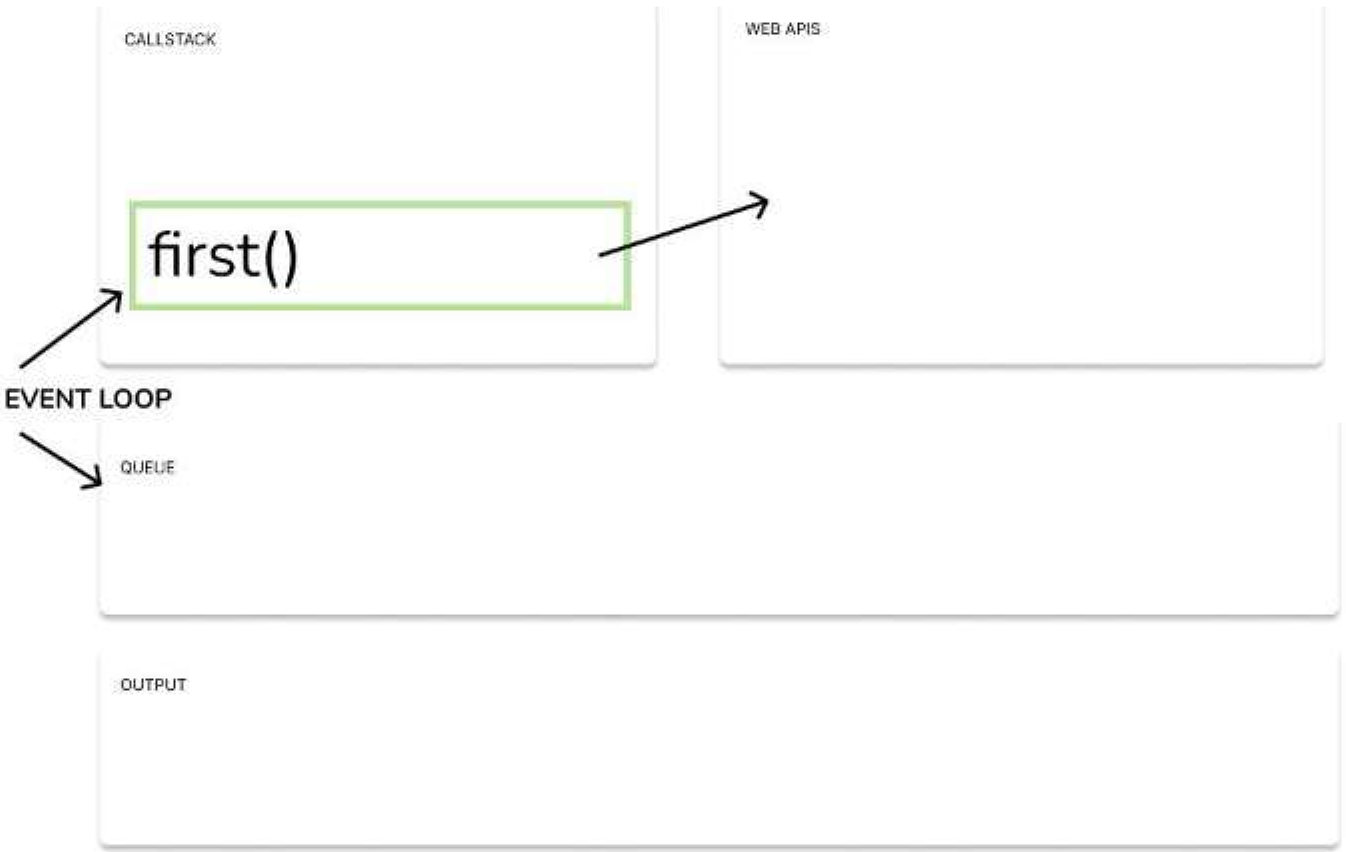
When the Web API finishes, the callback doesnt get added immediately to the Call stack. **Instead it’s passed to a callback (or task) queue.**

This is Event Loop job: EL connects the queue with the call stack. If it is empty, the first item from the queue gets added to the call stack.

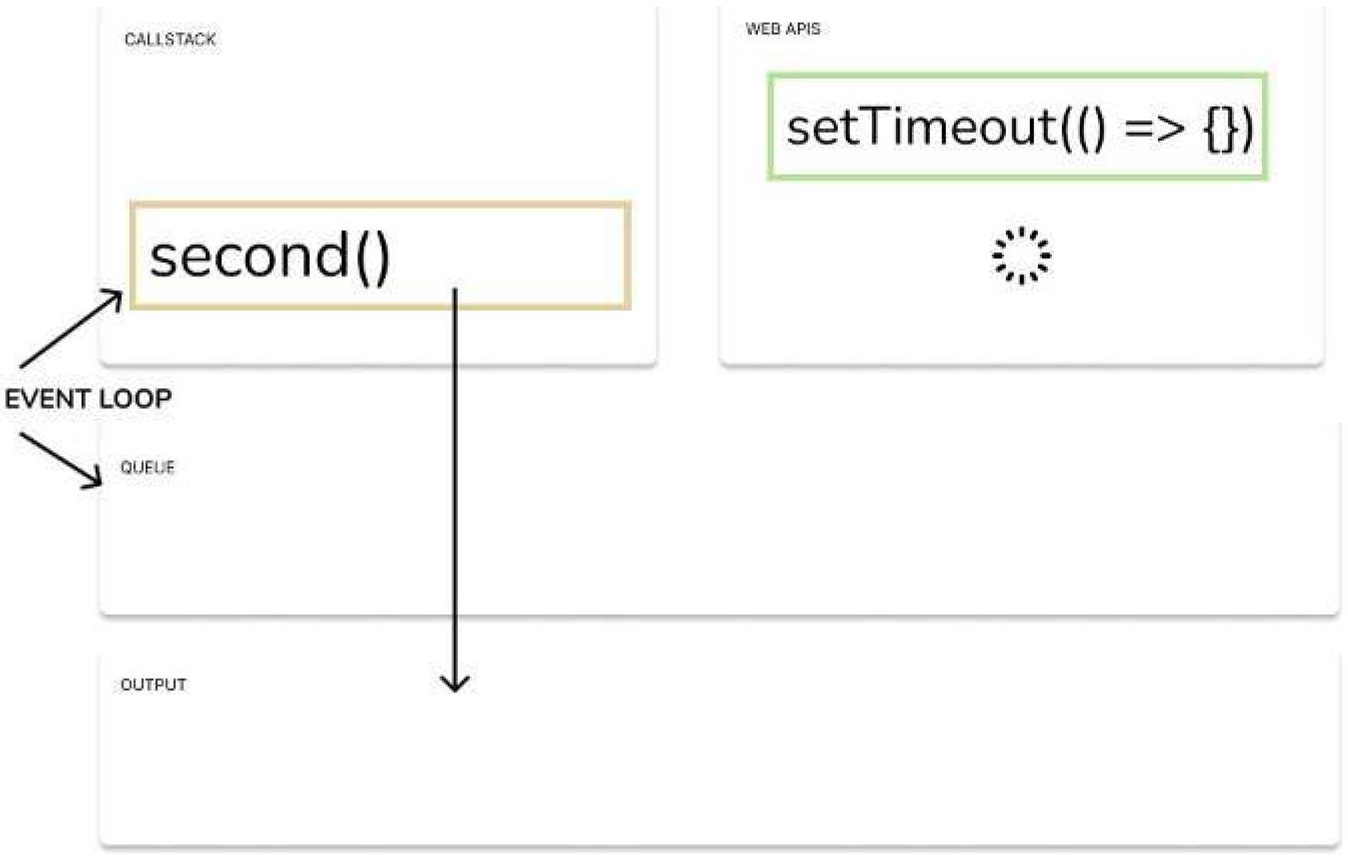
Example



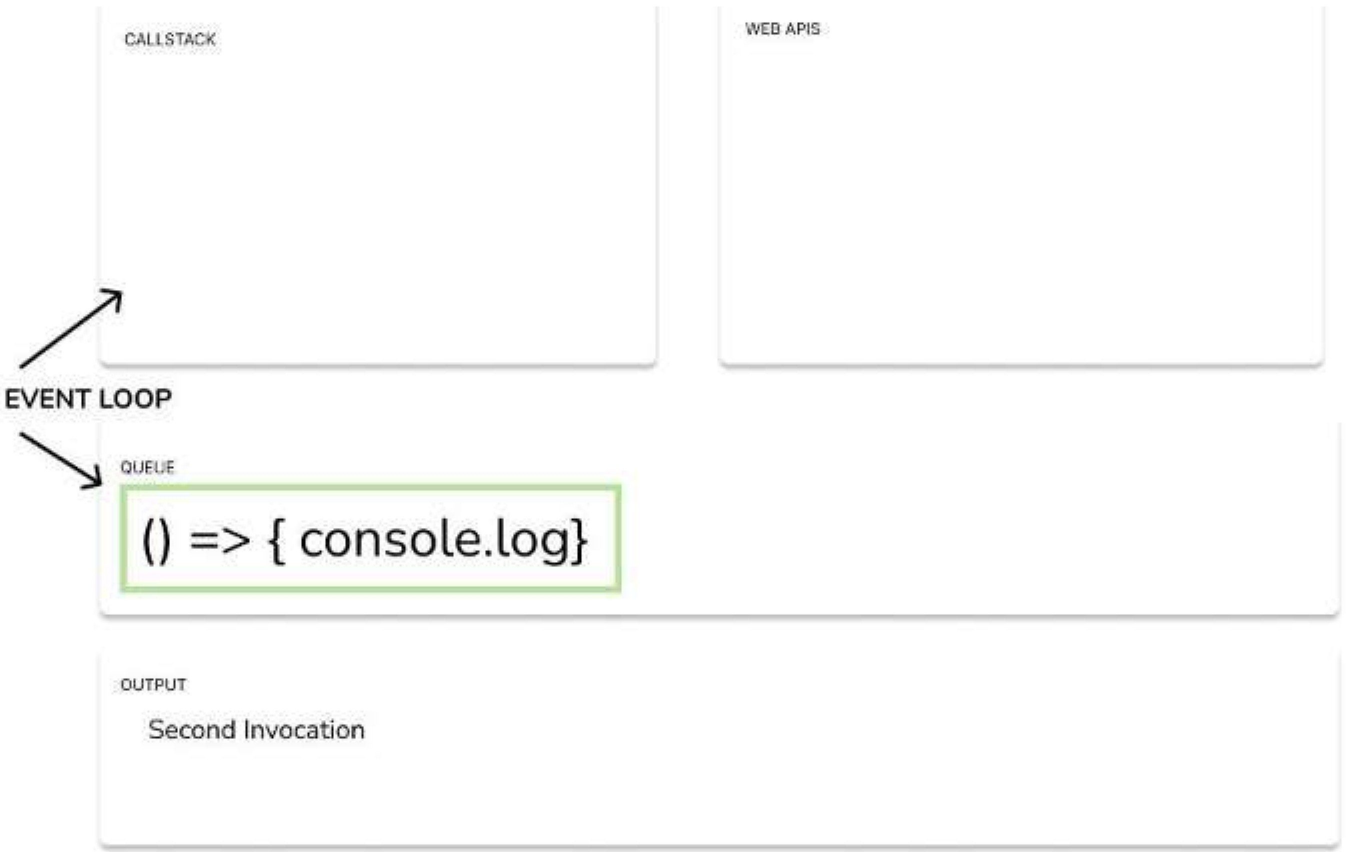
```
const first = () => {
    setTimeout(() => {
        console.log("First Invocation")
    }, 1000)
}

const second = () => {
    console.log("Second Invocation")
}

first();
second();
```



1. We invoke the **first function**, so it gets added to the call stack. First returns a setTimeout function.
2. The setTimeout callback is passed to the Web API, and the function is removed from the call stack.



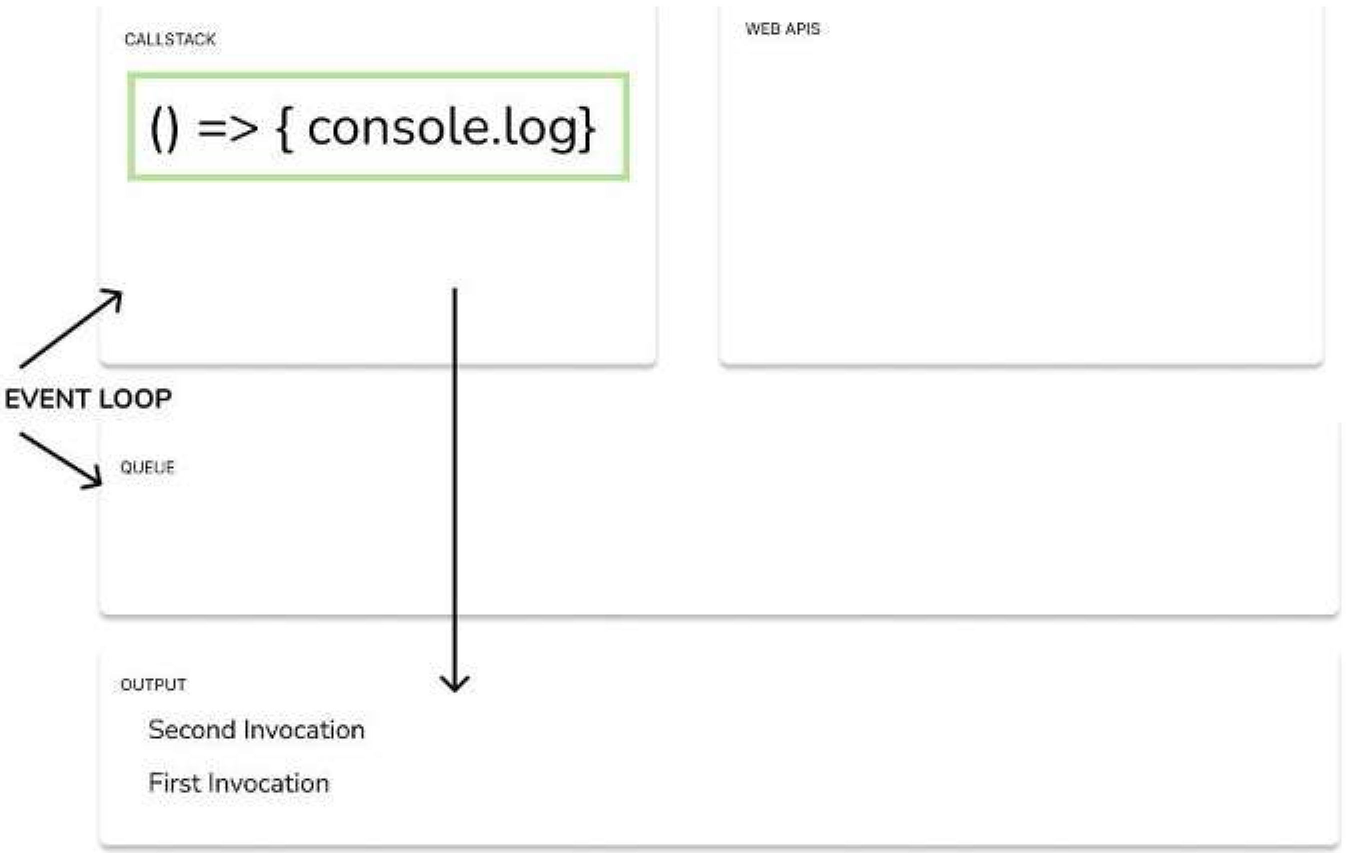
3. The timer runs. At the same time **second function** gets invoked and added to the call stack. It returns and logs “Second Invocation”
4. The timer finishes, the callback is added to the queue.



```
() => { console.log}
```

```
Second Invocation
```

5. The **event loop** sees that the call stack is empty.
The callback is added to the call stack.



6. The callback logs “First Invocation”