EVENT LOOP

Before we dive into "**Event Loop**", we need to understand what an **EVENT** is**?**

An event is an announcement which is triggered as per the actions performed by the user or the browser.

Now before we get to know about 'what is Event Loop', we need to keep some points in mind.

1. JavaScript is a Single-Threaded language.
2. A single request is executed at one time.
3. Until the execution of first request gets finished, the second request won't be touched
4. This means JavaScript is a **Synchronous** language.

Now, the problem here is sometimes we get some functions to execute inside our call stack which blocks the execution of next lined up requests which makes our program to blow off (*HINDI - "FAT JAANA"*) and thus the next lined-up requests never get to the call stack, this is also known as **Blocking**.

Let's consider the following example where the execution is happening in a synchronous way

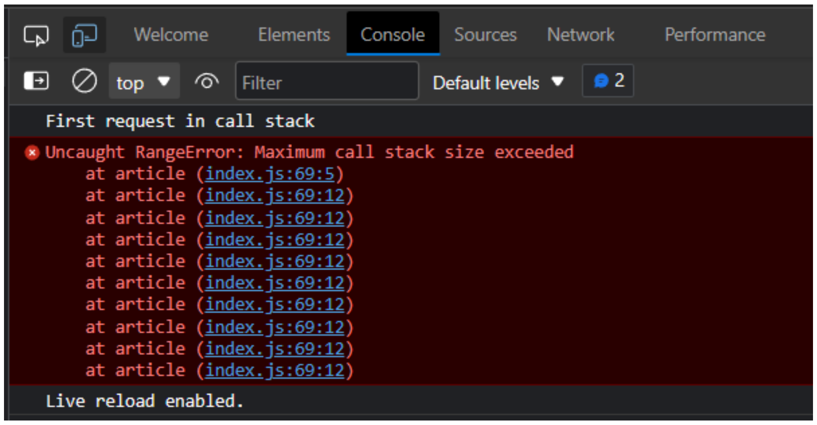
```
console.log("First request in call stack");

function article(){
    return article();
}

article();

console.log("This request never gets in call stack");
```

After execution in our console window we get...



```
First request in call stack
Uncaught RangeError: Maximum call stack size exceeded
    at article (index.js:69:5)
    at article (index.js:69:12)
    at article (index.js:69:12)
    at article (index.js:69:12)
    at article (index.js:69:12)
    at article (index.js:69:12)
    at article (index.js:69:12)
    at article (index.js:69:12)
    at article (index.js:69:12)
    at article (index.js:69:12)
Live reload enabled.
```

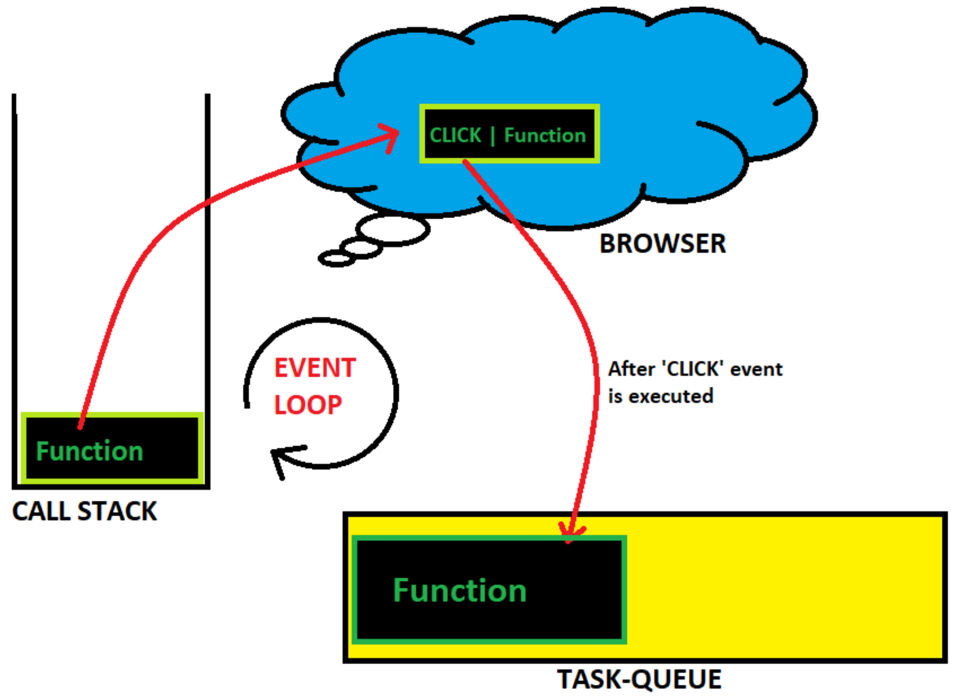
Since the requests are getting up lined in a synchronous way the last line of code never gets up to the call stack, because the call stack is already blocked at the function execution.

So, to tackle this problem we introduce "**EVENT LOOP"** to our JS which makes our execution Non-Blocking or ASYNC.

Typically when use functions like **setTimeOut()** or **.addEventListener()** in our code, the execution becomes an 'ASYNC Execution' whenever it hits the above mentioned functions, and by this way we can hit the next request into our call stack and execute them, and in the meantime the function in the *setTimeOut* or *addEventListener* gets processed inside the Browser and then moves into the '**task queue**' when the corresponding event is executed e.g. click, scroll, etc.

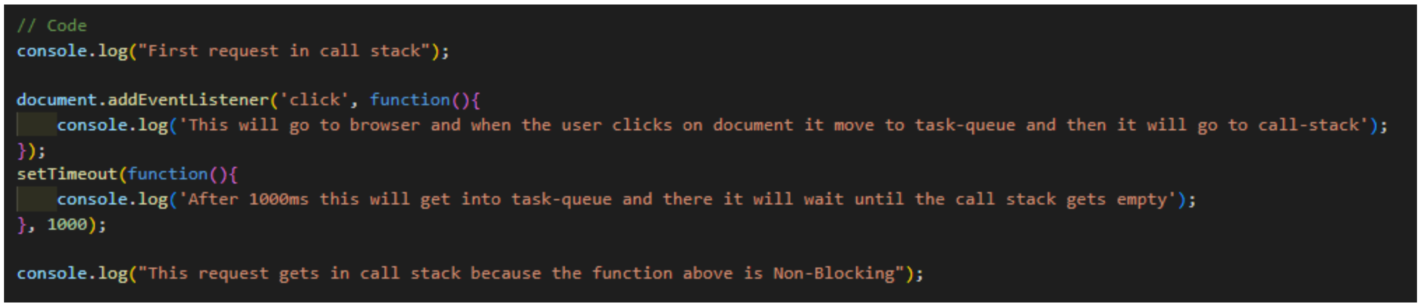
Inside the 'task queue' the function waits until the 'call stack' gets empty, and as soon as the 'call stack' gets empty the functions inside the 'task queue' are requested inside the 'call stack' where they get executed.

A simple working of event loop is shown here...



The "EVENT LOOP"

Let's understand the above working by the following code



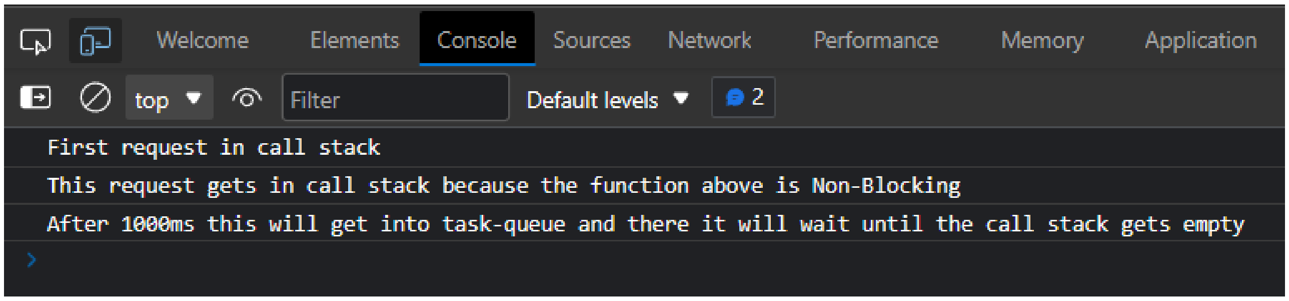
```
// Code
console.log("First request in call stack");

document.addEventListener('click', function(){
    console.log('This will go to browser and when the user clicks on document it move to task-queue and then it will go to call-stack');
});
setTimeout(function(){
    console.log('After 1000ms this will get into task-queue and there it will wait until the call stack gets empty');
}, 1000);

console.log("This request gets in call stack because the function above is Non-Blocking");
```

The CODE

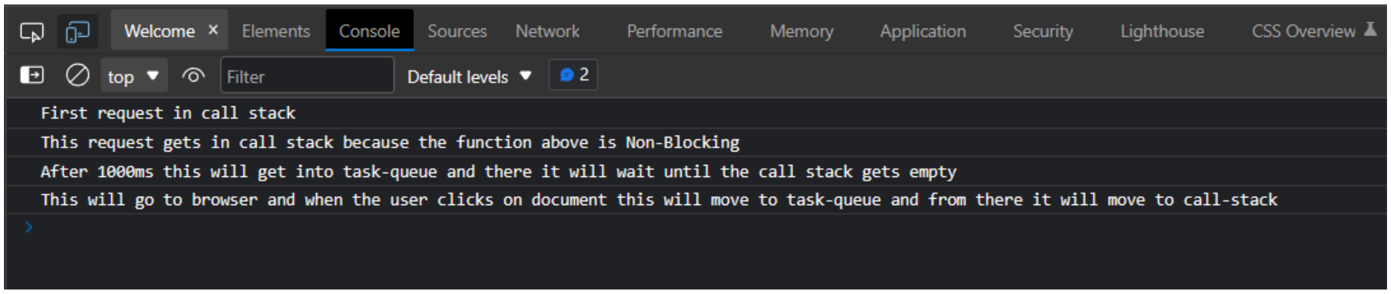
When the above code is executed in the console window we get...



```
First request in call stack
This request gets in call stack because the function above is Non-Blocking
After 1000ms this will get into task-queue and there it will wait until the call stack gets empty
```

The output **before** clicking on the document

The above result shows that the function inside the **eventListener** is still inside the '**task queue**', now when we click on the document,



```
First request in call stack
This request gets in call stack because the function above is Non-Blocking
After 1000ms this will get into task-queue and there it will wait until the call stack gets empty
This will go to browser and when the user clicks on document this will move to task-queue and from there it will move to call-stack
```

Output **after** clicking on the document

Thus, we got to understand that '**EVENT LOOP**' is a process of continuing the processing with the calls waiting for execution, whereas our function which made the process of execution slow goes to the '**browser**' and then executed according to the corresponding event through the '**task-queue**'.

Thank You!