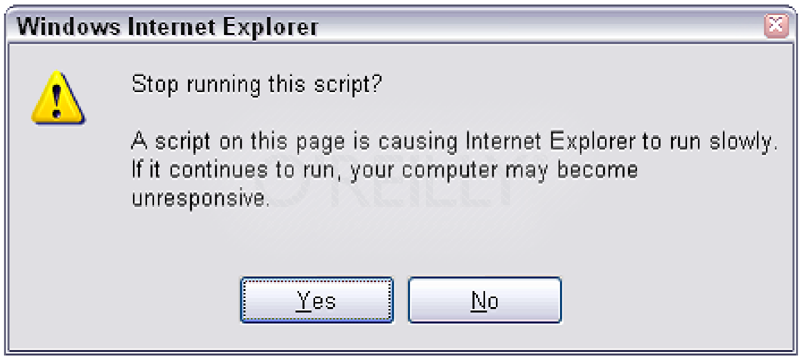
Web Workers

INTRODUCTION

In the browser, "normal" JavaScript code is run in a single thread (a thread is a light-weight process, see this Wikipedia page for details). This means that the user interface and other tasks are competing for processor time. If you run some intensive CPU tasks, everything is blocked, including the user interface. You have no doubt met this dialog during your Web browsing experiences.

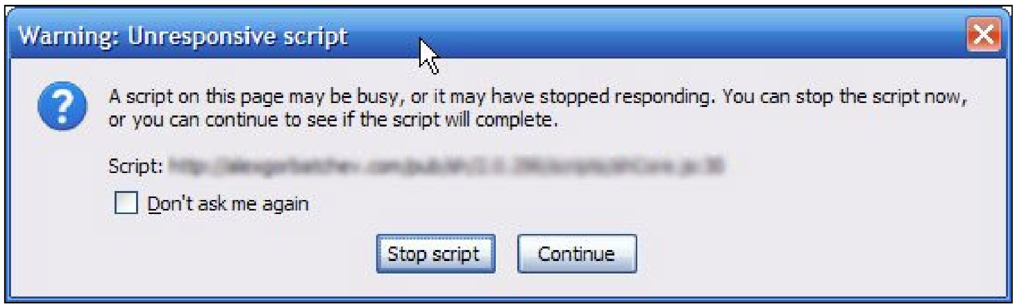
With Internet Explorer:


Windows Internet Explorer

Stop running this script?

A script on this page is causing Internet Explorer to run slowly. If it continues to run, your computer may become unresponsive.

Or maybe:


Warning: Unresponsive script

A script on this page may be busy, or it may have stopped responding. You can stop the script now, or you can continue to see if the script will complete.

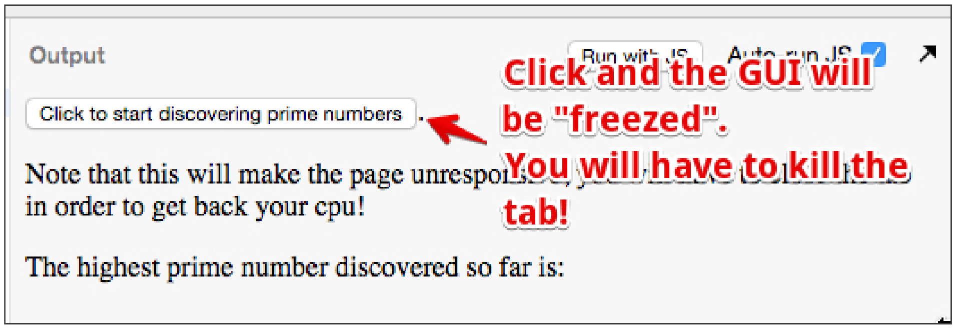
Script:

Don't ask me again

The solution for this problem, as offered by HTML5, is to run some intensive tasks in other threads. By "other threads" we mean "threads different from the UI-thread, the one in charge of the graphic user interface". So, if you don't want to block the user interface, you can perform computationally intensive tasks in one or more background threads, using the *HTML5 Web Workers*. So, Web Workers = threads in JavaScript

AN EXAMPLE THAT DOES NOT USE WEB WORKERS

This example will block the user interface unless you close the tab.Try it at JSBin but DO NOT CLICK ON THE BUTTON unless you are prepared to kill your browser/tab, as this will eat all the CPU time, blocking the user interface:



Note that this will make the page unrespo[illegible] in order to get back your cpu!

The highest prime number discovered so far is:

Click and the GUI will be "freezed".
You will have to kill the tab!

Code from the example:

```
    <!DOCTYPE HTML>
    <html>
     <head>
     <title>Worker example: One-core computation</title>
     </head>
     <body>
     <button id="startButton">Click to start discovering prime
    numbers</button><p> Note that this will make the page unresponsive, you
    will have to close the tab in order to get back your CPU!
     <p>The highest prime number discovered so far is: <output id="result">
    </output></p>
     <script>
10.    function computePrime() {
          var n = 1;
          search: while (true) {
             n += 1;
             for (var i = 2; i <= Math.sqrt(n); i+= 1)
             if (n % i == 0)
             continue search;
             // found a prime!
             document.getElementById('result').textContent= n;
          }
20.    }

    document.querySelector("#startButton").addEventListener('click', computePrime);
     </script>
     </body>
    </html>
```

Notice the infinite loop in the function `computePrime` (*line 12*, in bold). This will -for sure- block the user interface. And if you are brave enough to click on the button that calls the `computePrime()` function, you will notice that the *line 18* execution (that should normally modify the DOM of the page and display the prime number that has been found) does nothing visible. The UI is unresponsive.*This is really, really, bad JavaScript programming, and should be avoided at all costs.*

Shortly we will see a "good version" of this example that uses Web Workers.

THREAD SAFETY PROBLEMS? NOT WITH WEB WORKERS!

When programming with multiple threads, a common problem is "thread safety". This is related to the fact that several concurrent tasks may share the same resources (i.e., JavaScript variables) at the same time. If one task is modifying the value of a variable while another one is reading it, this may result in some strange behavior. Imagine that thread number 1 is changing the first bytes of a 4 byte variable, and thread number 2 is reading it at the same time: the read value will be wrong (1st byte that has been modified + 3 bytes not yet modified).

With `Web Workers`, the carefully controlled communication points with other threads mean that it's actually very hard to cause concurrency problems. There's no access in a worker to non-thread safe components or to the DOM, and you have to pass specific data in and out of a thread through serialized objects. Y*ou share different copies* so the problem with the four bytes variable explained in the previous paragraph cannot occur.

Different kinds of Web Workers:

There are two different kinds of Web Workers described in the specification:

1. **Dedicated Web Workers**: threads that are dedicated to one single page/tab. Imagine a page with a given URL that runs a Web Worker that counts in the background 1-2-3- etc. It will be duplicated if you open the same URL in two tabs for example, and each independent thread will start counting from 1 at startup time (when the tab/page is loaded).
2. **Shared Web Workers**: these are threads that can be shared between different pages of tabs (they must conform to the same-origin policy) on the same client/browser. These threads will be able to communicate, exchange messages, etc. For example, a shared worker, that counts in the background 1-2-3- etc. and sends the current value to a page/tab, will display the same value in all pages/tab that share a communication channel with it, and if you refresh all the pages, they will display the same value. The pages don't need to be the same (with the same URL), however they must conform to the "same origin" policy.

Shared Web Workers will not be studied in this course. They are not yet supported by major browser vendors, and their complete study would require a whole week's worth of study material. We may cover this topic in a future version of this course, when implementations are more stable/available.

EXTERNAL RESOURCES

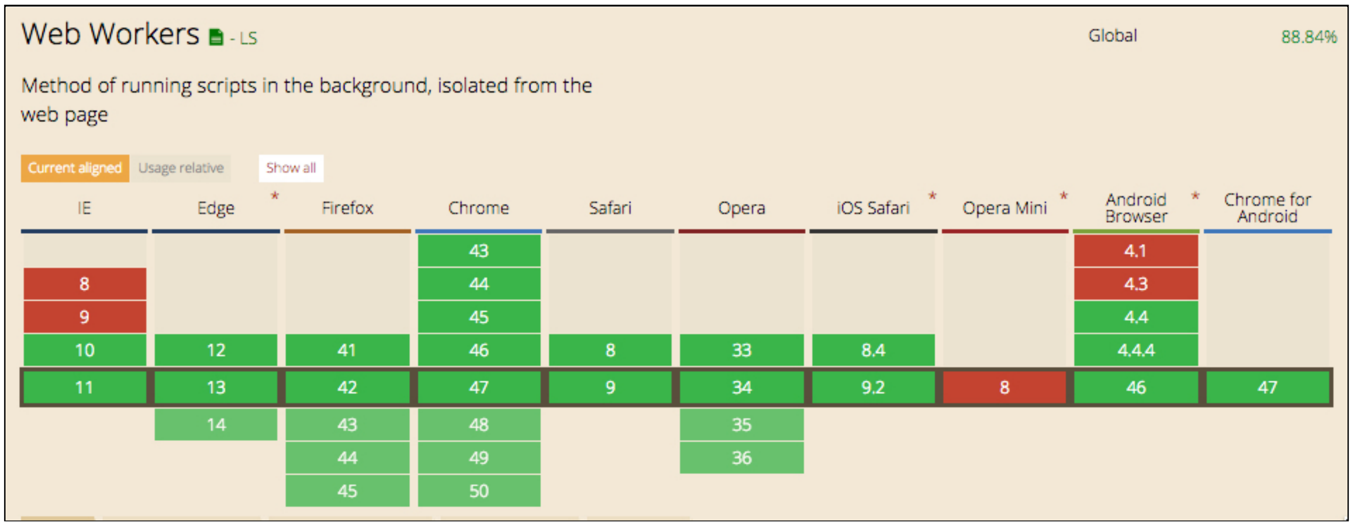
- W3C specification about Web Workers

- Using Web Workers, article in the Mozilla Developer Network

CURRENT SUPPORT

Dedicated Web Workers

Support as at December 2015:



Web Workers - LS | Global 88.84%

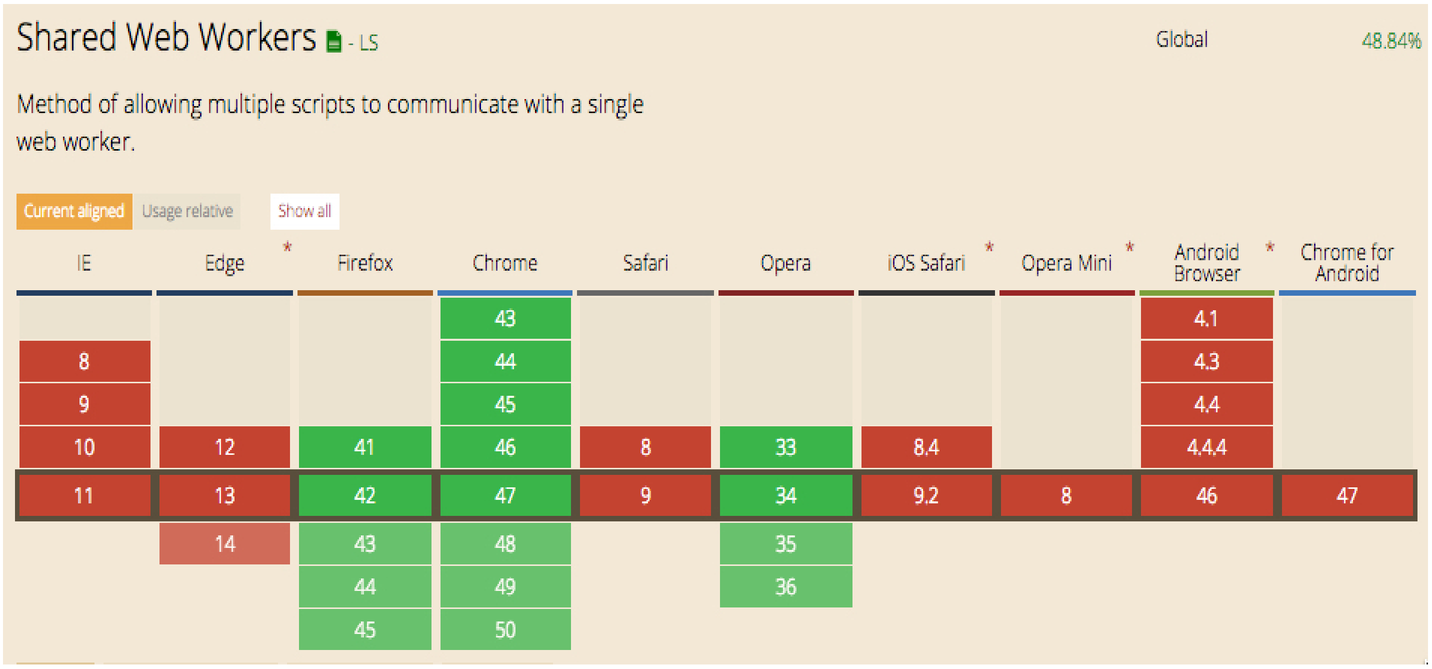
Method of running scripts in the background, isolated from the web page

Current aligned | Usage relative | Show all

IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Chrome for Android
			🟩 43					🟥 4.1	
🟥 8			🟩 44					🟥 4.3	
🟥 9			🟩 45					🟩 4.4	
🟩 10	🟩 12	🟩 41	🟩 46	🟩 8	🟩 33	🟩 8.4		🟩 4.4.4	
🟩 11	🟩 13	🟩 42	🟩 47	🟩 9	🟩 34	🟩 9.2	🟥 8	🟩 46	🟩 47
	🟩 14	🟩 43	🟩 48		🟩 35				
		🟩 44	🟩 49		🟩 36				
		🟩 45	🟩 50						

Up to date version of this table:http://caniuse.com/#feat=webworkers

Shared Web Workers (not studied), only in Chrome and Opera so far:



Shared Web Workers - LS | Global 48.84%

Method of allowing multiple scripts to communicate with a single web worker.

Current aligned | Usage relative | Show all

IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Chrome for Android
			🟩 43					🟥 4.1	
🟥 8			🟩 44					🟥 4.3	
🟥 9			🟩 45					🟥 4.4	
🟥 10	🟥 12	🟩 41	🟩 46	🟥 8	🟩 33	🟥 8.4		🟥 4.4.4	
🟥 11	🟥 13	🟩 42	🟩 47	🟥 9	🟩 34	🟥 9.2	🟥 8	🟥 46	🟥 47
	🟥 14	🟩 43	🟩 48		🟩 35				
		🟩 44	🟩 49		🟩 36				
		🟩 45	🟩 50						