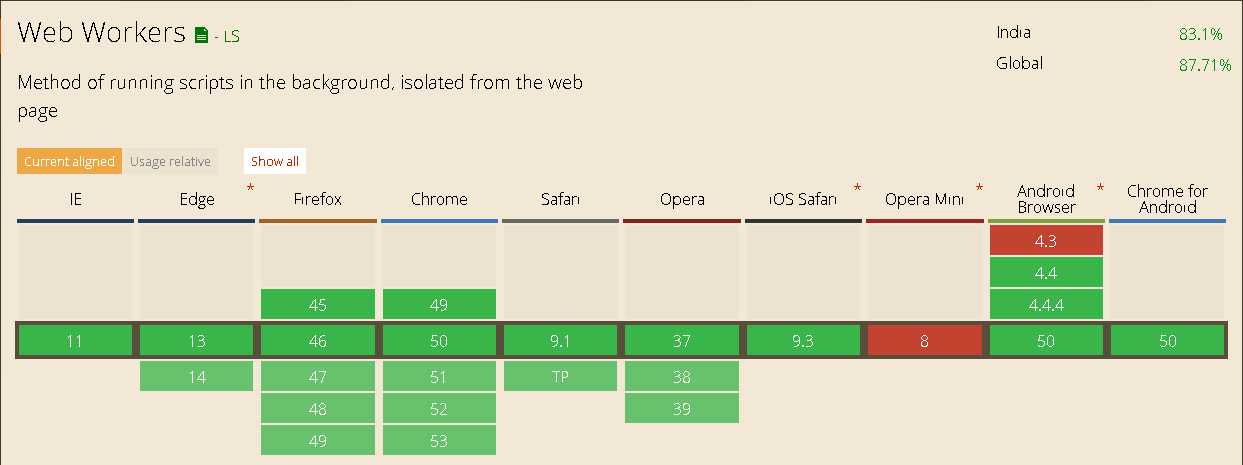
**WEB WORKER**

A web worker is a JavaScript running in the background, without affecting the performance of the page, in short we can say web workers provide us to execute multithreading in JavaScript.

**So what we did before:** Developers mimic 'concurrency' by using techniques like setTimeout(), setInterval(), XMLHttpRequest, and event handlers. Yes, all of these features run asynchronously, but non-blocking doesn't necessarily mean concurrency. Asynchronous events are processed after the current executing script has yielded. The good news is that HTML5 gives us something better than these hacks!

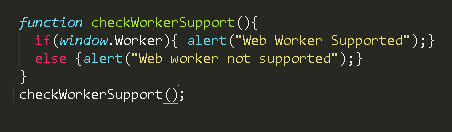
**Type of web workers and Difference:** Here we have 2 kinds of workers **Dedicated Workers** and **Shared Workers** and the very basic difference between these workers is a Worker can only be accessed from the script that created it, a Shared Worker can be accessed by any script that comes from the same domain.

**Support of web worker:**



Checking for Browser Support:

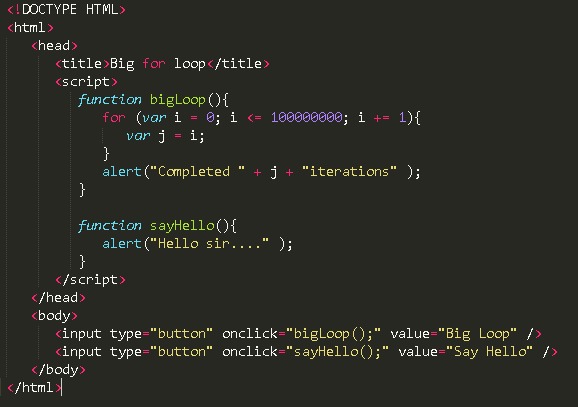
Following is the syntax to detect a Web Worker feature support available in a browser



**How a web worker help you:**

JavaScript was designed to run in a single-threaded environment, meaning multiple scripts cannot run at the same time. Consider a situation where you need to handle UI events, query and process large amounts of API data, and manipulate the DOM.

JavaScript will hang your browser in situation where CPU utilization is high. Let us take a simple example where JavaScript goes through a big loop −



Once you click big Loop button your page will block and you’ll not able to click on Say Hello.  
So what will happen if we execute same code but add bidLoop function in a separate file called bigLoop.js and load this with web worker. In this example you will also able to know how to terminate a web worker and handle the error scenario of web worker.

