Typical use of Web Workers

1 - YOU CREATE WORKERS FROM A SCRIPT EXECUTED BY A "PARENT HTML5 PAGE"

The HTML5 Web Worker API provides the `Worker` JavaScript interface for loading and executing a script in background, in a thread different from the UI thread. The following instruction does exactly that: loading and creating a worker:

```
var worker = new Worker("worker0.js");
```

More than one worker can be created/loaded by a parent page. This is parallel computing after all :-)

2 - YOU USE A WORKER BY COMMUNICATING WITH IT USING MESSAGES

Messages can be strings or objects, as long as they can be serialized in JSON format (this is the case for most JavaScript objects, and is handled by the Web Worker implementation of recent browser versions).

Messages can be sent to a worker using this kind of code:

```
var worker = new Worker("worker0.js");
// String message example
worker.postMessage("Hello");
// Object message example
var personObject = {'firstName': 'Michel','lastName':'Buffa'};
worker.postMessage(personObject );
```

Messages can be received from a worker using this kind of code (code located in the JavaScript file of the worker):

```
worker.onmessage = function (event) {
   // do something with event.data
  alert('received ' + event.data.firstName);
};
```

The worker can also communicate messages back to the parent page:

```
postMessage("Message from a worker !");
```

And a worker can listen to messages from the parent page like this:

```
onmessage = function(event){
    // do something with event.data
};
```

3 - A COMPLETE, VERY SIMPLE, EXAMPLE

The HTML page of the most simple example of dedicated Web Workers:

```
<!DOCTYPE HTML>
<html>
 <head>
 <title>Worker example: One-core computation</title>
 </head>
 <body>
 <p>The most simple example of Web Workers</p>
 <script>
   // create a new worker (a thread that will be run in the background)
   var worker = new Worker("worker0.js");
   // Watch for messages from the worker
   worker.onmessage = function(e){
     // Do something with the message from the client: e.data
     alert("Got message that the background work is finished...")
```

```
    };
    // Send a message to the worker
    worker.postMessage("start");
 </script>
 </body>
</html>
```

The JavaScript code of the worker:

```
onmessage = function(e){
    if ( e.data === "start" ) {
        // Do some computation that can last a few seconds...
        // alert the creator of the thread that the job is
finished
        done();
    }
};
function done(){
    // Send back the results to the parent page
    postMessage("done");
}
```

4 - HANDLING ERRORS

The parent page can handle errors that may occur in the workers that it created by listening to the onError event on the worker objects:

```
var worker = new Worker('worker.js');
worker.onmessage = function (event) {
    // do something with event.data
 };
worker.onerror = function (event) {
    console.log(event.message, event);
 };
}
```

See also the section "how to debug Web Workers"...