A continuación, aprenderemos a utilizar métodos nativos de Javascript. Hay muchísimos. Nos vamos a detener en dos porque tienen utilidades prácticas muy evidentes. Son “setTimeout” y “setInterval”. El primero ejecuta algo, una función, una serie de instrucciones, luego de transcurrido un tiempo determinado. Determinado por nosotros. Eso hace “setTimeout”. “SetInterval”, por el contrario, ejecuta también una instrucción, una serie de instrucción, de instrucciones, lo que sea, una vez cada ‘n’ segundos o milisegundos. Es decir, es una estructura que repite nuestro código según el tiempo o el lapso que nosotros indicamos. Vamos a ver entonces, “setTimeout” y “setInterval”. Métodos nativos de javascript en el código. Vamos a comenzar con “setTimeout”. Esta es la sintaxis.

Recibe un callback y ese callback, esa función va a ejecutar algún proceso, alguna alguna funcionalidad. Y luego de la coma, aquí, vamos a indicar un tiempo en milisegundos. 2000 milisegundos son, bueno, dos segundos. Y aquí, podríamos indicar, simplemente, vamos a imprimir algo, vamos a imprimir “mundo”. Luego del setTimeout podríamos imprimir “hola”. Noten que al ejecutar javascript, esta línea, no se quedará detenido esperando. Seguirá ejecutando el código siguiente. Por lo tanto, esta línea se va a imprimir antes que ésta. Porque ésta que está marcada aquí, el “console.lock (“mundo”)” debe ejecutarse una vez que hayan transcurridos, que hayan transcurrido, los dos segundos, los 2000 milisegundos. Por lo tanto, si corremos esta porción de código, de la línea 3 a la línea 6, el resultado será “hola”, y luego de dos segundos, “mundo”. Vamos a verlo aquí en la consola. “hola” “mundo”

¿ok? Muy bien, como habrán notado ustedes, si este console.locke estuviese aquí arriba, el resultado sería exactamente el mismo. Ya que esta porción está ejecutando código de manera asincrónica. Es decir, lo que está aquí adentro en el callback se va a ejecutar luego de transcurrido este tiempo. Todo lo que esté luego de este setTimeout seguirá corriendo de manera normal sin aguardar absolutamente nada. Ok. Ahora, podría ser que nosotros deseamos interrumpir la ejecución de este código. Por ejemplo, presionando un botón antes de que esto se imprimiese. Todavía no manejamos el **dom**, lo vamos a hacer en un par de clases. Por lo tanto, no vamos a poner aquí un botón para presionarlo, pero vamos a mostrar cómo podríamos evitar que esto efectivamente corra, se ejecute. Tenemos un método que se llama “clearInterval” y recibe el nombre del intervalo. Es decir, si nosotros corremos esto aquí no vamos a obtener ningún resultado porque deberíamos asignar esta función a alguna variable que luego podamos pasar aquí adentro como parámetro. Entonces, podríamos hacer, por ejemplo, una constante que se llame ‘timer’. Bien, noten que si elimino esto,

aquí no va a haber ningún cambio. Va a seguir ejecutándose este código, pero si aquí nosotros ahora, hacemos un clearInterval y decimos que el intervalo que queremos limpiar o eliminar es este, se va a detener esta función antes de que lleguen los 2 segundos. Es decir, esto nunca va a correr. Ok. Perdón, aquí va el nombre ‘timer’. Y ahora ustedes van a ver en pantalla que esta línea se imprime, pero luego limpiamos el intervalo y esto nunca llega a imprimirse porque interrumpimos el proceso. Miren: “hola”, contamos dos segundos, y tres, y cuatro, y cinco, y se nos va el tren. Nunca se va a imprimir ésto porque hemos interrumpido la ejecución. Ok. Ahora setInterval hace algo totalmente distinto. SetInterval lo que hace es, es, bueno, está aquí determinado ¿verdad? Aquí lo tenemos lo tenemos descrito, antes de pensar de empezar la clase lo he descrito. Lo que hace es ejecutar una función o un fragmento de código de forma repetitiva. Es decir, también tendremos un reloj pero ese reloj no indicará, ok, ejecuta todo lo que esté aquí adentro pasado este tiempo. SetInterval dirá: “ejecuta todo lo que esté aquí adentro en este lapso”. Es decir, de manera repetitiva cada ‘n’ segundos, bien, o milisegundos en este caso, ¿no?. Podríamos hacerlo así, setInterval

La sintaxis es idéntica al setTimeout y aquí podríamos decir

simplemente este mensaje. Y que esto, se ejecutase cada un segundo, mil milisegundos. Bien, lo que va a ocurrir es… ven, vamos a comentar esto, para que no corra ahora. Y aquí viene nuestro intervalo una vez, otra, y otra, y otra. Fíjense cómo está corriendo una y otra vez esta línea. Ok. Ahora aquí, estamos en un bucle infinito, esto nunca se va a detener. Para detener esto tenemos que detener también el intervalo.

Aquí en realidad, vamos a comentar esto, porque me estoy dando cuenta que he cometido un error aunque el código funciona igual. Es mucho más apropiado, aquí, indicar ClearTimeout, ¿si?. Había confundido las sentencias. Si bien de todas maneras estaba funcionando, no es lo más correcto. Lo más correcto es limpiar un Timeout con un cleartimeout, y limpiar un Interval con un clearinterval ¿ok?. Entonces, vamos a revisitar esta porción de código. Noten que funciona de todas maneras igual, pero hagámoslo bien. Hagámoslo de manera correcta. Afortunadamente me di cuenta a tiempo. Noten que si comentamos esta línea

Bien, ahora va a correr a los dos segundos "mundo", bien, pero si limpiamos el timer ¿ok? con cleartimeout esa es la manera correcta, bueno, nunca va a llegar a ejecutarse. Ok, bien.

Perfecto, continuemos con nuestro setinterval. Bien, esto se ejecuta una y otra vez, y nunca se detiene. Tenemos una manera de detenerlo y es igual que con el setTimeout. Podríamos, acá, decir 'intervalo'. Declarar esto, asignarlo aquí, a una, a una constante o a una variable, y luego tendríamos que hacer un clearinterval. Ahora sí es clearInterval del intervalo

Ok, y esto nunca jamás se va a ejecutar. Entiendo que tal vez esto es demasiado abstracto. Vamos a pensar, a ver, en un ejemplo que pueda tener un poco más de utilidad. Supongamos lo siguiente, miren, aquí dentro podríamos tener un, vamos a declarar aquí afuera: *let contador igual cero*. Y aquí dentro, digamos que "*valor de contador es*", y aquí pasamos "contador". La primera vez será 0, ¿verdad? Y luego de ejecutarse, vamos a decir que "contador es igual al contador más 1". Lo incrementamos en 1, o lo que es igual "contador++". Bien, entonces ahora ustedes noten como después de cada pasada se va incrementando el valor de nuestro contador, y me diréis con toda razón: "bueno, ¿y quién, en qué? no cambia demasiado a como estaba antes." Bueno, es que ahora tenemos una variable para poder limpiar nuestro intervalo desde aquí adentro. Podríamos decir, por ejemplo, que si contador es

igual a 5, ¿ok?, clearinterval y el nombre del intervalo es intervalo. Por lo tanto, ¿que haría este código? Bien, si no estoy equivocado, nuestro contador se dispondría en 0. Comenzaría el ciclo del interval, del intervalo, y tendríamos por pantalla "El valor de contador es 0". Ahora 'contador' valdría 1. Volvería a ejecutarse. Es decir, preguntaría: ¿'contador' es igual a 5? No. No es igual a 5. Ok. En algún momento 'contador' va a ser igual a 5 y en ese momento vamos a hacer el ClearInterval del intervalo, lo cual nos sacaría de aquí, ¿ok? Eso es lo que debería ocurrir. Vamos a ver
0, 1, 2, 3 y ahí viene 4, y viene 5, y ahora 5, ¿y que pasa? Bueno, cuando finalmente 'contador' es 5, se limpia el intervalo y salimos de aquí.