

# El tiempo (o la experiencia) vale oro

*"Si logras construir una gran experiencia, los clientes se lo dicen unos a otros. El boca a boca es muy poderoso"*

Jeff Bezos - CEO de Amazon.com



Photo by [NordWood Themes](#) on [Unsplash](#)

Ya conoces unos cuantos conceptos nuevos: variables, condicionales, ciclos, switch, arrays, objetos y funciones. En el último encuentro has trabajado en equipo para convertir esos conocimientos en un *saber hacer*, bajo la consigna de armar un E-commerce. ¡Ahora es momento de continuar!

Cuando desarrollamos, nos interesa poder gestionar el modo en que las instrucciones que le damos a un programa son ejecutadas, porque estaremos determinando –o al menos seremos conscientes de– el **tiempo de ejecución**. Podremos hacer que nuestro programa sea ‘multitasker’ o **asincrónico** (ejecutará una acción tras otra sin esperar respuestas) o que sea ‘unitasker’ o **sincrónico** (ejecutará una acción, esperará la respuesta, y recién ahí ejecutará una nueva acción). JavaScript por default es asincrónico (o ‘multitasker’), por lo tanto, si



necesitamos la dependencia de una respuesta, necesitamos forzar a que sea sincrónico (o 'unitasker').

Cuando un programa desea acceder a un recurso material, no necesita enviar información específica a los dispositivos periféricos: simplemente envía la información al [sistema operativo](#), el cual la transmite a los periféricos correspondientes a través de su controlador (¡el famoso *driver*!). Si no existe ningún driver, cada programa debe reconocer y tener presente la comunicación con cada tipo de periférico.

Es aquí donde para los/as developers empieza lo divertido. Con lo que aprendiste sobre JavaScript en las bitácoras anteriores (arrays, funciones, objetos), sumado a lo que trabajaste en el workshop de e-commerce, estás preparado/a para lo que se viene. En esta bitácora aprenderás sobre el manejo de los tiempos de una aplicación con los diferentes métodos que utilizan los lenguajes de programación.

## El tiempo de las aplicaciones

Para cumplir su función, una aplicación se alimenta de entradas, realiza un proceso y produce una salida. Como ya te anticipamos, este conjunto de acciones ocurre en un tiempo de ejecución. Hay dos tipos de procesos de ejecución, que corresponden a dos tipos de métodos que los lenguajes de programación utilizan: el [método sincrónico](#) y el [método asincrónico](#).

- **Método sincrónico.** La ejecución espera a que un resultado sea devuelto para ejecutar la siguiente instrucción. La mayoría de los lenguajes de programación suelen ser sincrónicos y las instrucciones se ejecutan una detrás de la otra:

```
console.log("¿Hola ");  
  
console.log("cómo ");  
  
console.log("están?");
```

El resultado sería:



¿Hola  
cómo  
están?

Si tomamos como ejemplo al lenguaje PHP en el [artículo](#) de Panama Hitek sucede lo siguiente:

*“...Cuando un usuario intenta acceder a un archivo de PHP por medio de un navegador web, un subproceso de Apache2 (el cual se encarga de manejar todas las llamadas a los archivos web y dar soporte a PHP) es llamado. El subproceso no se cierra hasta que el código termina de ejecutarse. Esto quiere decir que si tu archivo tarda un poco en ejecutarse y miles de personas intentan acceder al mismo archivo, se llega a un punto donde no se pueden abastecer a más usuarios debido a la insuficiencia de recursos.”*

El problema de estos procesos es que ocupan más memoria, por lo que suelen ser menos eficientes.

- **Método asincrónico.** La ejecución no espera un resultado. El proceso es enviado y queda en cola hasta que el sistema operativo decida cuándo están dadas las condiciones para ser ejecutado, y es entonces cuando éste se reanuda.

Por ejemplo, si tenemos que ir a buscar un dato fuera de nuestro sitio web, todo el ida y vuelta tiene demora de “x” cantidad de tiempo, en el cual el proceso queda en “pausa” hasta recibir una respuesta. Para optimizar los tiempos de espera que se producen, una típica solución es asignarle a otro proceso al CPU para que pueda ejecutarse. De este modo el uso del CPU es optimizado. Siguiendo el ejemplo, si tenemos las siguientes líneas de programación:

```
console.log("¿Hola ");  
  
console.log("cómo ");
```



```
-- Realizar una búsqueda en Google (ya veremos las instrucciones)

console.log("están?");
```

El resultado sería:

```
¿Hola
cómo
están?

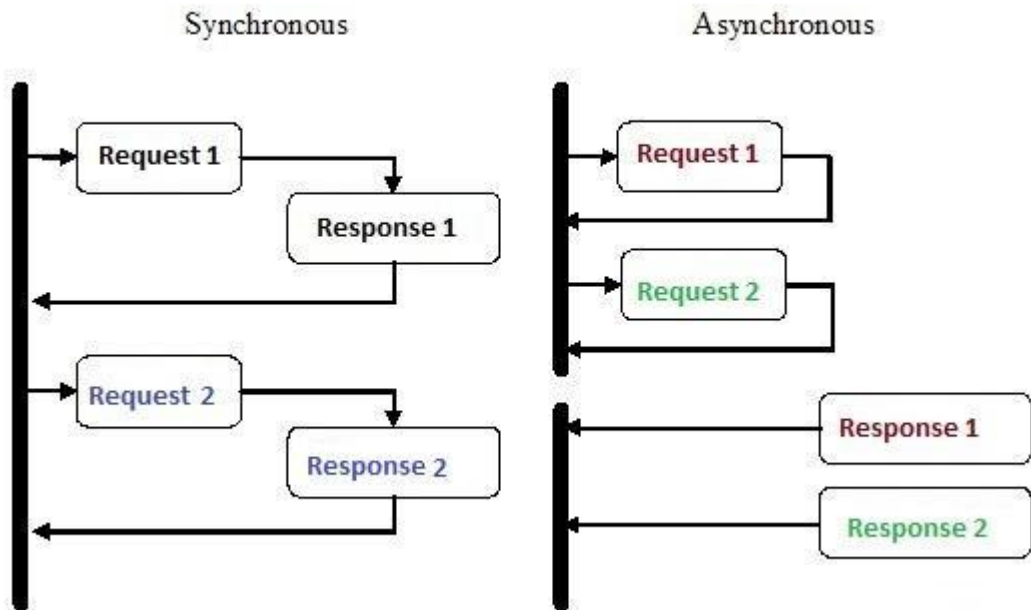
-- resultado de la búsqueda en Google
```

Como puedes ver, la instrucción para buscar algo en Google es la tercera pero nuestro output es la última línea.

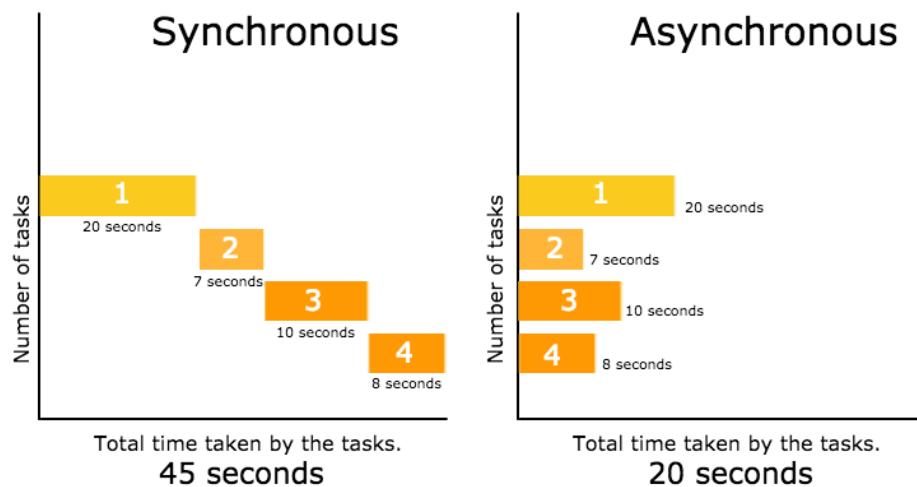
**JavaScript es en esencia un lenguaje asíncronico**, porque no se queda esperando que termine el proceso. En lugar de esperar a una respuesta para continuar, ejecutará un proceso tras otro sin esperar su respuesta. Su beneficio principal es el tiempo que se gana al reducir la espera del/a usuario/a al utilizar la aplicación. Si quisieras, podrías trabajar de manera síncrona pero en ese caso, deberías forzarlo a que lo haga. ¡Ya verás que en determinadas circunstancias lo necesitarás!

¡Todo sea por ganar tiempo! En el siguiente gráfico puedes ver las diferencias de tiempo entre ambos métodos:



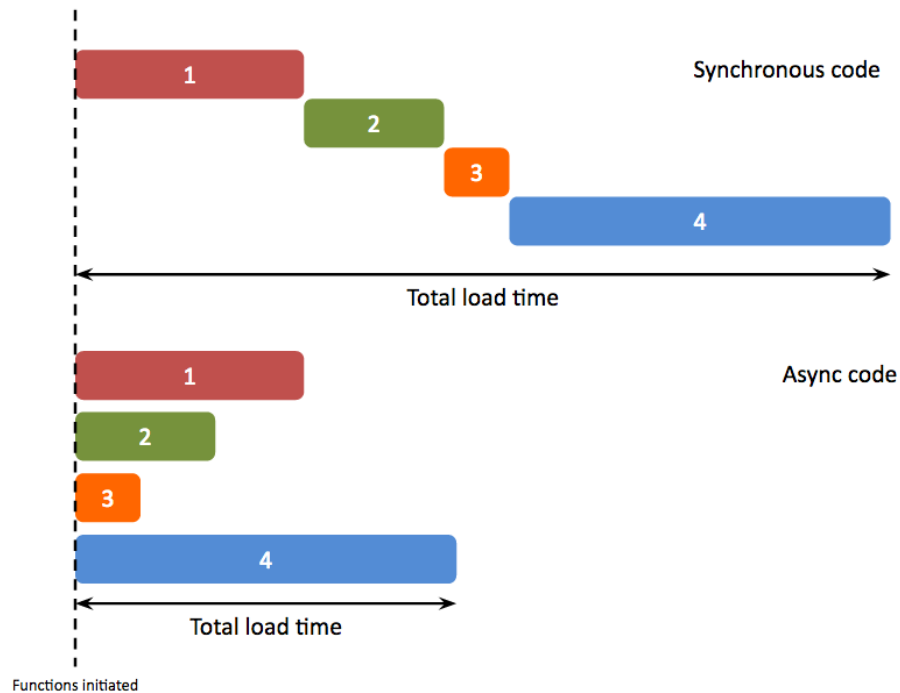


Otras formas de verlo...



Fuente: [PHP mind](#)





Fuente: [Cambridge Intelligence](#)

## Devolver llamadas con Javascript

Como sabes, JavaScript es un lenguaje orientado a objetos y las funciones son objetos. También entiendes que las funciones pueden tomar funciones como argumentos, y pueden ser devueltas por otras funciones. ¿Por qué es relevante? Porque esto te permitirá comprender cómo funciona la asincronía.

Una forma de capturar las respuestas de los procesos que enviamos y no sabemos cuándo van a terminar (la búsqueda en Google) es a través de [callbacks](#).

Un callback es una **función que recibe como argumento otra función y la ejecuta.**

Por ejemplo:

1. Imagina que definimos tres funciones: saludar, bienvenida y adiós. Todas son independientes entre sí y cualquiera podría ser ejecutada de manera independiente desde nuestro hilo principal de la aplicación.



```
function saludar(callback) {  
    callback();  
}  
  
function bienvenida() {  
    console.log("Hola mundo")  
}  
  
function adios() {  
    console.log("Adios mundo")  
}
```

La función `saludar` espera un parámetro que va a asignar a `"callback"` (si bien utilizar `callback` para llamar a ese parámetro es una convención muy utilizada, técnicamente puede ser cualquier otro nombre). La primera y única línea de esa función es tomar el dato del parámetro y ejecutarlo. Debido a que utilizamos los paréntesis `()`, entonces nuestra instrucción es `callback()`.

2. Pasemos a la ejecución: ejecutamos la función `saludar` y enviamos como parámetro `bienvenida`, sin comillas simples ni dobles porque es una función. Nuestra función `saludar` toma ese dato, lo ejecuta y muestra el mensaje.

```
saludar(bienvenida); // Imprime "Hola mundo"
```

Si reemplazamos `callback` por el valor que estamos enviando quedaría así:

```
bienvenida()
```



3. Si callback es una función que ejecuta otra función, a la función `saludar` solo necesitas enviarle una función para que la ejecute. Puedes tenerla previamente definida como hasta ahora o enviarla inline:

```
saludar(intermedio => console.log("Saludo intermedio"))
```

Como ves, estamos ejecutando un nuevo mensaje sin la necesidad de tener la función previamente definida.

Observa aquí el código completo y funcionando:

[Codepen: Ejemplo callback](#)

## La rapidez es la respuesta

En esta bitácora, conociste los diferentes métodos de ejecución de un programa e hicimos foco en la asincronía, mostrándote la importancia de tener un desarrollo que haga a una experiencia simple y rápida. Para aplicar la asincronía aprendiste acerca de los callbacks.

En el próximo encuentro profundizaremos en el uso de callbacks. Deberás tenerlos bien presentes para cuando salgas a buscar datos al exterior que demoren más tiempo.




Aunque avanzamos bastante sobre los temas de JavaScript ¡aún hay más por aprender! Ya te enterarás de algunos conceptos nuevos en las próximas bitácoras.






## ¡Prepárate para el próximo encuentro!

### Profundiza

-  [Programación Asíncrona](#)
-  [Asincronía en JavaScript](#)
-  [¿Qué es un callback y cómo utilizarlo?](#)

### Comunidad

 El desarrollo de e-commerce requiere numerosos callbacks para muchas funcionalidades. Googlea como un/a developer y ¡averigua cómo los developers del mundo utilizan los callbacks y comparte lo que aprendes con tu grupo!

### Challenge

 ¡Para pensar!

En el último workshop realizaste un E-commerce:

- ¿Qué fragmento de tu código requiere un callback? ¿En qué lugar imaginas que pueda ser necesario? ¿Añadir un artículo a la cesta? ¿El proceso de checkout?

En el próximo encuentro debatiremos sobre este punto. Te servirá de puntapié para los primeros intercambios sobre la programación asíncrona.

