Las 9 preguntas que te hacen en cualquier entrevista de *Javascript*

(Sacadas de entrevistas reales)

Introducción

Hola, ¿cómo estás?, mi nombre es Gustavo y quiero compartir con vos algunas preguntas que me hicieron en muchas entrevistas a lo largo de mi carrera profesional. Preguntas que me hubiera gustado conocer en su momento y me hubieran ayudado mucho.

¡También te dejo las respuestas! Pero te sugiero que te tomes aunque sea unos minutos para pensarlas vos mismo y, de esta forma, ejercitás tu cerebro.

¡Suerte en tus entrevistas!

Las Preguntas

¿Qué es la consola de desarrollo?

¿Qué significa que *Javascript* sea Dynamic Typed?

¿Cuál es la diferencia entre == y ===?

¿Cómo se crea una clase en *Javascript*?

¿Qué es una promesa en *Javascript*?

¿Qué es un callback?

¿Qué imprime ésto en pantalla y por qué?:

```
console.log("1");

setTimeout(function() {
    console.log("2");
}, 0);

console.log("3")
```

¿Qué imprime ésto en pantalla y por qué?:

```
console.log(mivariable)
var mivariable = 'hola';
```

¿Qué es IIFE y para qué sirve?

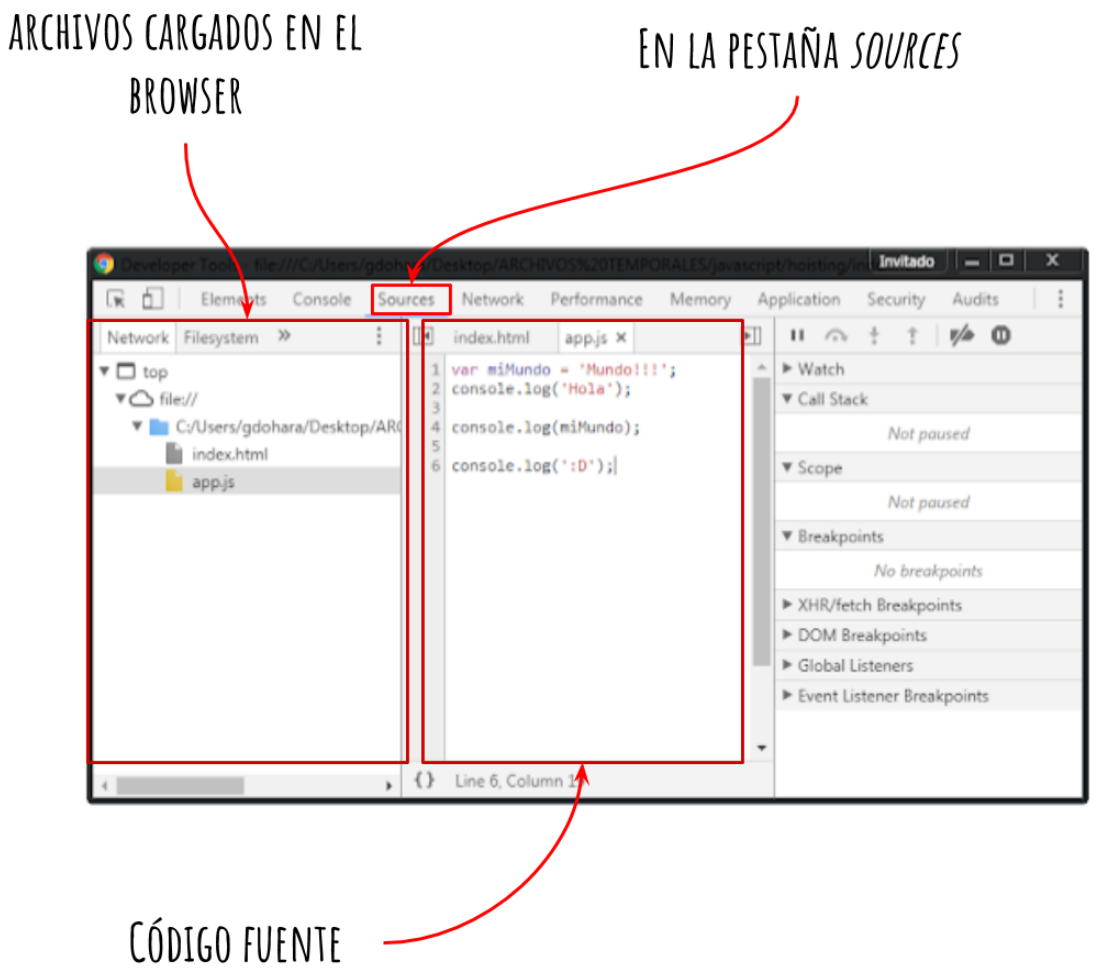
Las Respuestas

¿Qué es la consola de desarrollo?

La **consola de desarrollo** es una herramienta con la que cuentan los browsers modernos. Esta consola nos permite, entre otras cosas, ver el código que muestra nuestro browser: *html*, *css* y *javascript*, aunque también podría mostrar otros lenguajes como *typescript*, *less* y *sass,* entre otros.

Para ver esa consola en *Chrome* y *Firefox* presiona la tecla F12.

Esta consola también es muy útil a la hora de debuguear código *Javascript*, ya que se puede poner un breakpoint en cualquier lugar de nuestro código, que luego se detendrá en ese punto, permitiendo ver el estado de cada variable.



¿Qué significa que *Javascript* sea Dynamic Type?

Dynamic Type o **Tipo Dinámico**. En nuestro código *Javascript* no es necesario indicar el **tipo** (*string*, *boolean*, *number*, etc) de dato que se guardará en una variable, es más, el **tipo** de una variable puede ir cambiando a lo largo de nuestro programa, pudiendo la misma variable ser numérica, y luego string, un par de líneas más adelante. Esto es perfectamente normal en *Javascript*.

El permitir este cambio de **tipo** todo el tiempo, hace que nuestro código sea muy dinámico, pero nos puede llevar a cometer errores si no somos cuidadosos y no chequeamos el tipo de dato de nuestras variables.

¿Cuál es la diferencia entre == y ===?

El Operador doble igual (==): compara 2 valores PERO lo hace sin chequear el **tipo** de las variables.

En cambio, el Operador triple igual (===) chequea el contenido de la variable y además, chequea que las dos variables sean del mismo **tipo**.

Ejemplos:

```
1 == '1' // verdadero
1 === '1' // falso
false == 0 // verdadero
false === 0 // falso
'' == 0 // verdadero
'' === 0 // falso
```

Para *Javascript,* cuando usamos el operador doble igual (==), false y 0 son iguales, lo mismo que una cadena vacía y 0. ¿Raro? No, sólo es *Javascript* ;)

¿Cómo se crea una clase en *Javascript*?

La forma de definir una clase es la siguiente:

```
class Poligono {
   constructor(alto, ancho) {
     this.alto = alto;
     this.ancho = ancho;
   }
}
```

Y luego, para usarla es:

```
var p = new Poligono()
```

Nota: cabe aclarar que ésta no es la misma clase, conceptualmente hablando, que se usa en la programación orientada a objetos. Ésta es una forma de simplificar la herencia usando prototipos.

¿Qué es una promesa en *Javascript*?

Una promesa es un objeto que representa la terminación o error de un evento asincrónico. Un evento asincrónico es “algo” que sucede en otro momento distinto al que estamos ejecutando nuestro código. Si bien las promesas se pueden crear con un constructor, es más común usarlas.

La forma de usarlas es la siguiente:

```
miPromesa.then(function() {
   //codigo a ejecutar despues de que la promesa finalice
})
```

Este código lo que hace ejecutar el código que está en el método “then”, una vez que la promesa “miPromesa” haya finalizado correctamente.

¿Qué es un callback?

Un callback es una función (**callback**) que la pasás a otra función (miFuncion) como parámetro. El callback se ejecuta cuando la “miFuncion” finalice.

Un ejemplo sería:

```
var callback = function() {
    console.log('termine!!!')
}


var miFuncion = function(callback) {
```

```
    // codigo de la funcion
    // la funcion finalizo
    callback();
}
```

Hay muchas formas de implementar esto, éste es sólo un ejemplo simple.

La idea de usar callback es decirle a la función “miFuncion” que “avise” cuando finaliza, y la forma de hacerlo es llamando a función **callback.**

¿Qué imprime ésto en pantalla y por qué?

```
console.log("1");

setTimeout(function() {
    console.log("2");
}, 0);

console.log("3")
```

Ésto imprime en pantalla.

```
1
3
2
```

¿Por qué? Porque *setTimeout* es una función asincrónica, esto significa que se ejecuta “más tarde”. Cuando *Javascript* encuentra una función asincrónica la coloca en una cola de “cosas asincrónicas” y sigue ejecutando el resto del código que le falte, es por eso que se imprime 1 y luego 3. Cuando termina de ejecutar el código principal, revisa esa cola de cosas asincrónicas y, si hay algo para ejecutar, lo ejecuta, en nuestro ejemplo ejecuta *setTImeout*. Es por eso que finalmente termina imprimiendo 2.

¿Qué imprime ésto en pantalla?

```
console.log(mivariable)
var mivariable = 'hola';
```

Ésto imprime *undefined* en pantalla.

¿Por qué? porque *Javascript,* cuando encuentra la palabra reservada "var", lo que hace es mover la declaración de la variable al inicio (ésto se conoce como **hoisting**). En otras palabras, *Javascript traduce (internamente)* el código que vimos, al siguiente:

```
var mivariable;
console.log(mivariable)
mivariable = 'hola';
```

Como todas las variables a las que no se les asigna nada se crean con el valor *undefined,* lo que se termina imprimiendo en la consola es *undefined*.

¿Qué es IIFE y para qué sirve?

IIFE significa **Immediately Invoke Function Expression** y básicamente lo que hace es crear una función y ejecutarla al mismo tiempo.

Existen dos formas de crear una función:

```
function miFuncion() {}
```

Esta forma es llamada *function statement.*

Y luego, la forma:

```
var miFuncion = function() {}
```

Esta otra forma se llama **Function Expression** (FE).

Si a ésta forma le sacamos la asignación a la variable miFuncion y la ejecutamos, tenemos lo siguiente:

```
(function() {
   /* tu codigo */
})();
```

Esto es lo que se conoce como **IIFE,** ya que es una **Function Expression invocada/ejecutada inmediatamente**.

Esto se hace para crear variables que están sólo dentro del contexto de esa IIFE ya que, las variables que se crean dentro de una función (si no hacemos nada raro), sólo tienen el scope de esa función.

Conclusiones

Las entrevistas suelen ser un proceso incómodo y extenso. Mientras más preparados estemos mejor nos irá, pero lo importante es tomarlo como un proceso de aprendizaje.

Si en una entrevista no te fue tan bien como pensabas, es fundamental que adquieras experiencia y aprendas un poquito más para la siguiente oportunidad. ¡No te desanimes! ¡Uno no puede saberlo todo! Pero lo que sí podés hacer es tener una actitud positiva y aprender de tus errores.

De esa forma, a la larga, ¡vas a pasar las entrevistas que vos te propongas!.

Recursos

Si te quedaste con ganas de saber más de *Javascript*, visitá mi blog:

http://www.gustavodohara.com

Si tenés alguna consulta podés mandarme un email a:

gustavo@gustavodohara.com