ArgentinaPrograma
YoPrograMo

JS

por Leonardo Blautzik, Federico Gasior y Lucas Videla

Julio / Diciembre 2021

Ministerio de
Desarrollo Productivo
Argentina

JS (JavaScript)

Es un lenguaje de programación del estilo de “scripting” que es una de las tecnologías principales de la web junto con HTML y CSS. Permite crear contenido dinámico, controlar elementos, y prácticamente cualquier otra cosa que hayan visto en la web

Es multi-paradigma: Soporta estilos de programación imperativos, funcional, orientado a eventos, orientado a propotipos y orientado a objetos

Todos los navegadores web populares tienen incluido un motor de Javascript que permite ejecutarlo desde el dispositivo del usuario

Sintaxis

Considerando que tenemos un título <h1>, el siguiente script de JS va a acceder al DOM para manipular el elemento HTML a través de un evento

```
// Los selectores son similares a los de CSS
const titulo = document.querySelector("h1");

titulo.addEventListener("click", cambiarTitulo);

function cambiarTitulo() {
    let nuevoTitulo = prompt("Ingrese el nuevo título", "Nuevo título
    titulo.textContent = nuevoTitulo;
}
```

Especificaciones

Al igual que el resto de los otros componentes web con los que ya estuvimos trabajando, JS también dispone de especificaciones.

ECMAScript es la especificación de la cual se basa Javascript. La misma define que componentes debería tener y como se tiene que comportar.

ECMA cuenta con distintas versiones conocidas como ES6 (2015), ES7 (2016) entre otras. La mayoría de navegadores populares hoy en día disponen de motores de javascript que respetan y cumplen con prácticamente todas las reglas hasta el ES7. Los motores de Firefox (SpiderMonkey) y Chrome (V8) (actualmente usados por Edge y Opera también) son actualmente los que más soportes a los nuevos estándares introducidos por el EMCA poseen

¿Versiones?

Javascript no dispone de versiones sino que funciona todo un poco distinto

Cada motor de Javascript tiene versiones en las cuales intenta cubrir una serie de especificaciones definidas en alguna versión de ECMAScript

Como realmente es un conocimiento muy específico, y que realmente no nos importa, ya que el desarrollo lo vamos a enfocar a un navegador (o a lenguajes basados en JS como NodeJS), podemos utilizar tablas de compatibilidad de las especificaciones que nos interese para conocer su soporte, como lo hicimos con CSS

Ejemplo para querySelector: https://caniuse.com/?search=querySelector

Entorno

Cada pestaña o ventana de un navegador tiene su propio entorno para correr código. Esto significa que el código de una pestaña es completamente independiente al de otro. Si no fuese así un sitio malicioso podría estar accediendo a datos bancarios, o contraseñas de otros sitios sin niguna restricción.

Hay maneras de comunicarse entre distintas pestañas de manera segura, aunque lo que se suele hacer es utilizar algún tipo de almacenamiento local del navegador, cuyo contexto de trabajo es el dominio del cual es accedido

Orden de ejecución

Javascript se ejecuta de arriba hacia abajo, línea por línea

O eso hace normalmente

En nuestro caso de ejemplo agregamos una función nombrada a un evento, y luego la declaramos. ¿Eso no debería haber dado error? Si hubiésemos cambiado el orden en titulo hubiese dado un error

Orden de ejecución

Esto es principalmente por algo llamado “hoisting”, que explicado de manera MUY sencilla, hace que todas las declaraciones ocurran al principio (sin asignaciones ni otras expresiones). Esto hace que en el caso de la función, podamos enlazarla ya que existe, pero debemos “crearla” antes de utilizarla

Esto es importante ya que si no tienen cuidado y declaran todo de forma prolija y “correcta”, pueden tener comportamientos inesperados

Código interpretado

A diferencia de otros lenguajes (por ejemplo Java), el código es recibido por el navegador en su forma original y lo ejecuta tal cual. Esto hace que a veces las ejecuciones puedan ser más lentas ya que no aprovechan muchas ventajas que tiene la compilación, como las optimizaciones

La mayoría de los navegadores de todas maneras utilizan una técnica de compilación llamada “just-in-time” que mejora el rendimiento de los scripts al ser compilados binariamente mientras el script es ejecutado. Aún así, al código se lo considera interpretado ya que la compilación se realiza en tiempo de ejecución

Formas de incluir JS en un documento

- En un archivo aparte
- En el mismo documento HTML
- Directamente en el elemento (a través de ciertos atributos especiales)
- Inyectándolo (consola, extensiones, etc)

En un archivo aparte

Se pueden incluir uno o varios elementos `<script>` en el `<head>` (o en el `<body>` :( ) que lo que van a hacer es incluir y ejecutar estos scripts en el documento, en el momento que el interprete llegue a esa linea. Lo recomendado es crear un archivo por cada módulo a trabajar

```
<script type="text/javascript" src="script.js"></script>
```

En el mismo documento HTML

En muchos casos puede ocurrir que se necesite de un pequeño script para ejecutarse en un contexto determinado, o sólo en un documento y no vale la pena crear un archivo aparte para este fin. Utilizando nuevamente el elemento `<script>` podemos realizar justamente esto. La diferencia es que ahora tendrá contenido y no un “src”

```
<script type="text/javascript">
/* Scripts */
</script>
```

Directamente en el elemento

Para casos específicos, donde solo un elemento, una única vez va a tener un comportamiento asignado como evento, se puede incluir código JS en un atributo de HTML específico a ese evento. Por ejemplo en un click utilizaremos: “onclick”

```
<button onclick="alert(\"¡Hola mundo!\");">Saludar</button>
```

Carga de scripts

Observemos este caso y recordemos que en HTML, cada línea es procesada apenas es leía, y no leé la siguiente hasta ser procesada

```
<script>
let titulo = document.querySelector("h1");
</script>

<h1>Hola mundo</h1>
```

Esperando al DOM

```
document.addEventListener("DOMContentLoaded", function() {
  /* código */
});
```

Carga asincrónica

Si no queremos que los scripts demoren la carga de la página, podemos utilizar uno de estos dos atributos especiales para prevenirlo, y solucionar la espera de DOM. Ambos harán que los scripts no sean bloqueantes y se ejecuten en un segundo plano

- async: El script se ejecutará tan pronto como la descarga haya terminado. No se conserva el orden original
- defer: El script se ejecutará después de que toda la página haya cargado, y conservará el orden de los elementos

Trabajo en clase

Mayor y menor

- Se dispondrá de un mazo de 10 cartas con valores del 1 al 10
- Se robará una carta y el jugador deberá adivinarla, ingresando un número entre 1 a 10
- Si el número es incorrecto, se le dará una pista si es mayor o menor
- Si el número es correcto, se pasará a la siguiente carta
- Una vez finalizado el mazo deberá indicar cuantos intentos erróneos tuvo. Ese es su puntaje
- Reiniciar juego
- ¡Intenta conseguir el menor puntaje posible!

¡Muchas Gracias!

continuará...

Ministerio de
Desarrollo Productivo
Argentina