UTN.BA FACULTAD REGIONAL BUENOS AIRES
SECRETARÍA DE EXTENSIÓN UNIVERSITARIA FRBA UTN

Centro de e-Learning

Funciones y Eventos

Temario:

Temario:

1.Eventos

A lo largo del curso pudimos ver que tenemos la posibilidad de programar acciones a realizar en el sitio, pero presentan una limitación: no podemos controlar qué ocurre cuando el usuario acciona de cierta manera. Es decir que van a ir sucediendo a medida que lo declaramos en nuestro archivo JavaScript sin tener interacción por parte del usuario.

Por este motivo, JavaScript nos provee de los ***eventos*** , que nos van a permitir controlar las acciones de los usuarios, y también el comportamiento del navegador en respuesta a esa acción.

Los eventos son interacciones que comienzan y finalizan, como por ejemplo el hacer click en un botón, cuando se carga una imagen o incluso el movimiento del mouse.

JavaScript nos provee de varios tipos de eventos asociados a funciones para poder relacionar dicho evento con las acciones programadas como respuesta. Este tipo de funciones se denominan ***manejador de eventos*** o en inglés ***event handlers.***

Estos ***event handlers*** son una especie de "alarmas" que se encuentran atentos a cuando el usuario ejecuta ese evento, para poder activar el conjunto de acciones que se programó como respuesta.

1.1 Definiendo Eventos

Ahora que ya sabemos lo que es un evento, podemos ver cómo escribirlo en nuestro código. Contamos con dos maneras distintas de definirlas:

- Agregándolos como atributos en las etiquetas del HTML a las que queremos asociarlo.

- Definiéndolos en funciones en nuestro archivo JavaScript.

Si agregamos los atributos en la etiquetas HTML, se nombran colocando la palabra ***"on"*** delante del nombre del evento. Por ejemplo, para el evento de clickear algún elemento se utiliza el evento "click", por lo que el atributo a colocar en el HTML será "onclick".

EJEMPLO

```
<button onclick="funcionAEjecutar()">Clickear</button>
```

Ahora bien, en este ejemplo pudimos ver que programamos que, cuando se haga click en el boton, se ejecute la acción *funcionAEjecutar()*. Esta función está definida en el archivo JavaScript, pero esta no es la única forma de asignar un manejador de eventos a un evento.

También tenemos la posibilidad de programar directamente el accionar en el mismo atributo. Esto nos permite directamente escribir código JavaScript dentro del atributo.

EJEMPLO

```
<button onclick="alert('hiciste click!')">Clickear</button>
```

En este ejemplo, podemos notar que solamente estamos emitiendo un alerta cuando se clickee el botón, pero ¿cómo redactamos si queremos producir un cambio en el DOM ante un evento? Por ejemplo, que el boton cambie de color cuando se lo clickee. Si utilizamos lo aprendido acerca del DOM , dentro del atributo deberíamos llamar al nodo a modificar y, utilizando los métodos aprendidos, realizar el cambio.

EJEMPLO

```
<button onclick="document.getElementsByTagName('button')[0].style.backgroundColor='green'">Clickear</button>
```

Como esto en el HTML requiere de muchas líneas de código (lo que hace que el código quede engorroso y poco legible), JavaScript nos provee, en caso que no queramos definirlo en una función en JavaScript, una variable llamada ***this***. Esta variable hace referencia al elemento HTML en el que se encuentra el evento definido, es decir, en el que estamos colocando el atributo. De esta forma, vamos a estar reduciendo un poco el código y haciendo que el programa sea más visual.

EJEMPLO

```
<button onclick="this.style.backgroundColor='green'">Clickear</button>
```

Por su parte, si utilizamos la opción de definir todo en funciones en el archivo JavaScript, utilizamos el método **addEventListener()** luego del elemento donde se va a producir el evento, que es llamado por medio de las herramientas del **DOM.**

En estos casos, el evento se escribe entre comillas y se pasa como un parámetro del

método addEventListener().

Evento	Descripción	Elementos para los que está definido
`onblur`	Un elemento pierde el foco	`<button>`, `<input>`, `<label>`, `<select>`, `<textarea>`, `<body>`
`onchange`	Un elemento ha sido modificado	`<input>`, `<select>`, `<textarea>`
`onclick`	Pulsar y soltar el ratón	Todos los elementos
`ondblclick`	Pulsar dos veces seguidas con el ratón	Todos los elementos
`onfocus`	Un elemento obtiene el foco	`<button>`, `<input>`, `<label>`, `<select>`, `<textarea>`, `<body>`
`onkeydown`	Pulsar una tecla y no soltarla	Elementos de formulario y `<body>`
`onkeypress`	Pulsar una tecla	Elementos de formulario y `<body>`
`onkeyup`	Soltar una tecla pulsada	Elementos de formulario y `<body>`
`onload`	Página cargada completamente	`<body>`
`onmousedown`	Pulsar un botón del ratón y no soltarlo	Todos los elementos
`onmousemove`	Mover el ratón	Todos los elementos
`onmouseout`	El ratón "sale" del elemento	Todos los elementos
`onmouseover`	El ratón "entra" en el elemento	Todos los elementos
`onmouseup`	Soltar el botón del ratón	Todos los elementos
`onreset`	Inicializar el formulario	`<form>`
`onresize`	Modificar el tamaño de la ventana	`<body>`
`onselect`	Seleccionar un texto	`<input>`, `<textarea>`
`onsubmit`	Enviar el formulario	`<form>`
`onunload`	Se abandona la página, por ejemplo al cerrar el navegador	`<body>`

EJEMPLO

1.2 Tipos de Eventos

JavaScript pone a nuestra disposición una amplia variedad de eventos. Esto nos permitirá identificar la acción que va a realizar el usuario y. por ende, nos va a permitir definir la respuesta que debe generar el navegador.

En esta sección estaremos definiendo los más utilizados, pero debemos tener en cuenta que la variedad de eventos es más amplia.

Imagen extraída de https://www.arkaitzgarro.com/

1.3 Propagación

Hay situaciones en las que nos vamos a encontrar con elementos padres e hijos que

```
elementoDom.addEventListener("evento",(event)=>{

    //codigo a ejecutar cuando se produzca el evento

    event.stopPropagation()
})
```

poseen eventos asignados a cada uno.

En algunos de estos casos vamos a poder ver que, al activar cualquiera de dichos eventos, se ejecutan tanto los manejadores de eventos asignados al evento del elemento padre, como los asignados al elemento hijo. Es decir que, como una reacción en cadena, se ejecutarán todos los eventos hijos y padres. A esto se le denomina ***propagación*** y ocurre con el uso de algunos eventos.

Para evitar este tipo de situaciones, JavaScript nos provee de una función denominada ***event.stopPropagation()*** , que se coloca al final del manejador de evento correspondiente al evento hijo, antes del cierre de llaves. De esta forma, si se acciona el evento en el hijo, solamente se ejecutará el código definido en el manejador de evento de dicho elemento; y, cuando se accione sobre el elemento padre, solamente se ejecutará el código definido en el manejador de eventos del propio elemento padre.

SINTAXIS

2. Un poco más sobre los Objetos

En la unidad anterior hablamos de un tipo de estructura llamado ***objeto*** que nos permite almacenar propiedades de los datos que definimos y poder guardarlo en una variable.

Estas propiedades se encuentran entre llaves({}), una debajo de la otra, separadas entre sí por medio de una coma (exceptuando la última propiedad). Contienen un nombre de referencia (llamado ***key*** o ***clave***) que indica la propiedad, seguida por dos puntos (:) y el valor de dicha propiedad (llamada ***value*** o ***valor***).

SINTAXIS

```
var/let/const   nombreDelObjeto ={

        nombreDeLaClave : suValor ,
        nombreDeOtraClave : suValor
}
```

Como información importante , debemos saber que los valores pueden ser cualquier tipo de dato: string, boolean, number, array e incluso otros objetos.

Por otro lado, al definir las keys o claves de los objetos, tenemos que tener en cuenta que son strings. Pueden escribirse sin comillas, siempre y cuando sean una única palabra. En caso que queramos colocar más de una palabra como un key necesitamos colocarla entre comillas.

Debemos aclarar que esta última manera de definir las key o claves no es la más utilizada hoy en día.

EJEMPLO

```
let alumno={
    nombre:"Juan",
    apellido:"Perez",
    edad:32

}
```

```
let alumno={
    "nombre del alumno":"Juan",
    "apellido del alumno ":"Perez",
    "edad del alumno ":32

}
```

Al momento de acceder a estas propiedades, JavaScript nos ofrece dos formas de hacerlo:

- **nombreDelObjeto.nombreDeLaClave**

Esta es la forma más utilizada hoy en día, donde se le ingresa el nombre del ***objeto*** y un punto, seguido de la ***key*** que indica la propiedad que desea obtener. Este pedido nos da como resultado el valor de la propiedad guardada (es decir el ***valor*** o ***value***).

EJEMPLO

```
let alumno={
    nombre:"Juan",
    apellido:"Perez",
    edad:32

}

console.log ( alumno.nombre) // nos devuelve por consola "Juan"
```

- **nombreDelObjeto[nombreDeLaClave]**

Esta forma es la menos utilizada hoy en día, donde se coloca el nombre del objeto y entre corchetes la key o clave de la propiedad que estamos ubicando, dándonos como resultado el valor de la propiedad guardada.

EJEMPLO

```
let alumno={
    nombre:"Juan",
    apellido:"Perez",
    edad:32

}

console.log ( alumno[nombre]) // nos devuelve por consola "Juan"
```

Según sea la forma en la que accedemos a las propiedades de los objetos, podemos ver que, a diferencia de los array, va a ser indistinto el orden en el que se encuentren las propiedades. Solamente va a importar que cada key esté asociado a su valor

correspondiente.

EJEMPLO

Esta misma sintaxis nos va a servir tanto para agregar propiedades al objeto como para reemplazar el valor de las key. Es decir que, en caso que queramos agregar una propiedad como modificar el valor de una propiedad ya existente en el objeto, solamente deberemos agregarle a esta sintaxis el igual y el valor a modificar o agregar.

EJEMPLO

```
let alumno={
    nombre:"Juan",
    apellido:"Perez",
    edad:32

}


alumno.apellido ="Gomez"
/* el objeto ahora es
let alumno={
    nombre:"Juan",
    apellido:"Gomez",
    edad:32

}
*/
```

```
let alumno={
    nombre:"Juan",
    apellido:"Perez",
    edad:32

}

alumno.secundarioCompleto =true

/* el objeto ahora es
let alumno={
    nombre:"Juan",
    apellido:"Gomez",
    edad:32,
    secundarioCompleto: true
}
*/
```

En caso que necesitemos eliminar alguna propiedad del objeto en algún momento de nuestro archivo, JavaScript nos provee de la palabra reservada ***delete,*** que se coloca delante de la expresión que hace referencia a esa propiedad.

SINTAXIS

delete *nombreDelObjeto.nombreDeLaClave*

delete *nombreDelObjeto[nombreDeLaClave]*

EJEMPLO

3. En resumen

```
let alumno={
    nombre:"Juan",
    apellido:"Perez",
    edad:32

}

delete alumno.edad

/* el objeto ahora es:

let alumno={
    nombre:"Juan",
    apellido:"Gomez",

}
*/
```

```
let alumno={
    nombre:"Juan",
    apellido:"Perez",
    edad:32

}

delete alumno[apellido]

/* el objeto ahora es:

let alumno={
    nombre:"Juan",
    edad:32
}
*/
```

En esta unidad aprendimos cómo poder programar eventos que nos permitan ejecutar acciones según distintos sucesos que se vayan dando en el sitio.

También aprendimos que algunos de los eventos van a poder ser provocados por el usuario, lo que le agrega una mayor interacción y dinamismo al sitio.

Pudimos ver las distintas formas de definirlos y los tipos de eventos que nos proporciona JavaScript.

Por último, retomamos la explicación de los objetos, la posibilidad que nos ofrecen de definir propiedades de datos o elementos, su uso y cómo acceder al valor de dichas propiedades.

4 .Bibliografía utilizada y sugerida

https://desarrolloweb.com/articulos/1235.php
https://www.arkaitzgarro.com/javascript/capitulo-15.html
https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Working_with_Objects
https://developer.mozilla.org/es/docs/Web/Events
https://www.w3schools.com/js/js_events.asp