JavaScript

LIMA UD6 – Tema 2

IES Plurilingüe Antón Losada Diéguez

Tabla de contenido

JavaScript. JS

1. Modificación dos elementos HTML e dos seus atributos

A maneira máis sinxela de modificar un elemento HTML é a través das propiedades:

- ***innerHTML***: accede ao contido do elemento en cuestión.
- ***outerHTML***: accede ao contido do elemento en cuestión xunto coa etiqueta do elemento.

Vexamos un exemplo de modificación dun parágrafo que contén unha etiqueta ** no seu interior.

```
<!DOCTYPE html>
<html>
    <head>
        <title>UD 6</title>
    </head>
    <body>
        <h1 id="header1">Os tres porquiños</h1>
        <p id="p1"> Soplarei e soplarei e a túa casa <span> derribarei </s
pan> </p>
        <script>
            document.getElementById("p1").innerHTML=" '<span>
Soplarei e soplarei e a túa casa derribarei <span>' ";
            document.getElementById("p1").outerHTML="<p id='p3'> 'Soplarei
 e <span> soplarei e a túa casa derribarei </span>'</p>";
            console.log(document.getElementById("p3").innerHTML);
            console.log(document.getElementById("p3").outerHTML);
        </script>
    </body>
</html>
```

Co uso de `innerHTML` accedemos ao contido desa etiqueta. Se dentro existen outras etiquetas podedes observar na consola que tamén son devoltas.
No caso de `outerHTML` devolve a maiores a propia etiqueta do `<p>`. No exemplo chegamos a modificar o seu propio atributo `id` (algo que non é habitual simplemente é para probar) pasando de ser `p1` a `p3`.

Sen embargo, para modificar un atributo a forma máis recomendable é:

- Propiedade *attribute* en cuestión: é dicir poñendo como atributo do elemento o nome do atributo en cuestión. Por exemplo: *elemento.class* para modificar as clases de dito elemento.
- Método *setAttribute(attribute, value)*: modifica o atributo de nome *attribute* co valor *value*.

```
<script>
    // Cambio do atributo id mediante a propiedade id
    document.getElementById("p1").id="p3";
    console.log(document.getElementById("p3").outerHTML);
    // Cambio do atributo id mediante o método setAttribute
    document.getElementById("p1").setAttribute("id","p3");
    console.log(document.getElementById("p3").outerHTML);
</script>
```

Outra forma de modificar o texto é mediante o método `write()`. Dito método do obxecto `document` permite escribir directamente no fío (*stream*) dun documento. Coa peculiaridade de que si o documento xa estaba cargado previamente, limpará todo o contido do documento e incluirá o texto que se pasou como parámetro. No caso de que a modificación estea incluída no propio documento HTML, simplemente engadirá ese novo texto deixando o resto como estaba como se mostra no seguinte código de exemplo xunto co resultado da figura 1.

```
<!DOCTYPE html>
<html>
    <head>
        <title>UD 6</title>
    </head>
    <body>
        <h1 id="header1">Os tres porquiños</h1>
        <p id="p1" class="paragraph"> Soplarei e soplarei e a túa casa <sp
an> derribarei </span> </p>
        <script>
            document.write("<p> E casiña derribouse </p>");
        </script>
    </body>
</html>
```

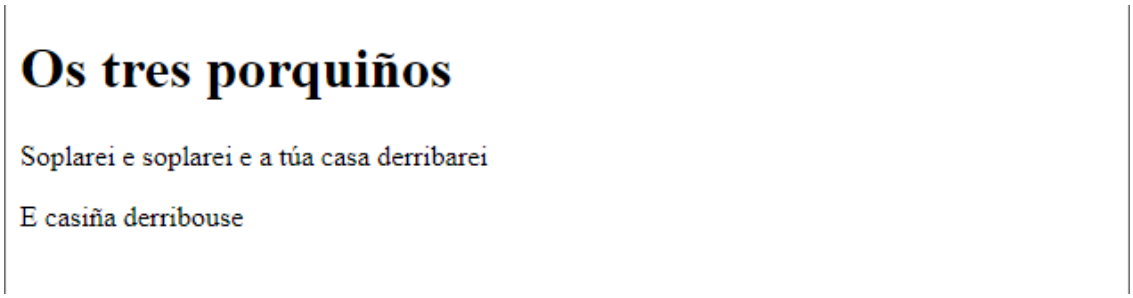

Figura 1: Resultado de engadir o último `<p>` mediante unha chamada a `document.write` dende a etiqueta `<script>` no propio documento HTML

```
<!DOCTYPE html>
<html>
    <head>
        <title>UD 6</title>
        <script src="script01.js" />
    </head>
    <body>
        <h1 id="header1">Os tres porquiños</h1>
        <p id="p1" class="paragraph"> Soplarei e soplarei e a túa casa <sp
an> derribarei </span> </p>
    </body>
```

```
</html>
```

Tendo como `script01.js` a chamada a `document.write()` obtemos o resultado da figura 2:

```
document.write("Quedeime sen conto");
```

Figura 2: Resultado de engadir o último `<p>` mediante unha chamada a `document.write` dende un script externo.

2. Modificación das propiedades CSS

As propiedades CSS poden ser modificadas facilmente mediante JavaScript accedendo ao DOM. O único que precisamos é acceder ao elemento HTML en cuestión e empregamos o obxecto *style* que representa os estilos dese elemento. No obxecto *style* teremos acceso a todas as propiedades CSS. No seguinte exemplo modificamos a cor de letra do *<h1>*

```
<!DOCTYPE html>
<html>
    <head>
        <title>UD 6</title>
    </head>
    <body>
        <h1 id="header1">My Header</h1>
        <p>text text text text text text text text text text text text tex
t text text </p>
        <script>
            document.getElementById("header1").style.color="green";
        </script>
    </body>
</html>
```

Todas as propiedades do obxecto `style` poden ser accedidas dende o elemento seleccionado correspondente. No seguinte enlace de W3Schools podemos comprobar o nome de todas e cada unha das propiedades (https://www.w3schools.com/jsref/dom_obj_style.asp).

Non obstante, se temos que modificar moitas propiedades nun elemento resultará pesado ter que escribir unha liña por cada modificación, co problema a maiores de que cada vez que se fai cada unha desas modificacións é preciso recargar o elemento. A solución a este problema é empregar a propiedade `cssText` do obxecto `style` que nos permitirá modificar varias propiedades CSS en dito texto.

A continuación móstrase un exemplo modificando as propiedades CSS individualmente ou co uso de dita propiedade `cssText`:

```
<script>
    document.getElementById("header1").style.color="green";
    // Método 1
    document.getElementsByTagName("p")[0].style.backgroundColor="lightblue
";
    document.getElementsByTagName("p")[0].style.padding="50px";
    // Método 2
    document.getElementsByTagName("p")[0].style.cssText="background-
color:lightblue; padding:50px";
</script>
```

No caso de traballar con clases CSS podemos destacar as operacións básicas de engadir e eliminar clases CSS dun elemento ou intercambiar entre varias clases. Para iso é imprescindible empregar dúas propiedades:

- ***classList:*** devolve os nomes de clases dun elemento. Trátase dunha propiedade que é de lectura pero pode modificarse a través dos métodos *add()* (engadir unha nova clase), *remove()* (eliminar unha clase), *toggle()* (intercambio entre clases).
- ***className:*** devolve os nomes de clases dun elemento. Esta propiedade pode ser modificada asignando unha cadea cos nomes de clases correspondentes separadas por espazos.

Vexamos un exemplo no que traballamos con ambas propiedades:

```
<!DOCTYPE html>
<html>
    <head>
        <title>UD 6</title>
        <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/boots
trap/4.0.0/css/bootstrap.min.css" integrity="sha384-
Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm" crossori
gin="anonymous">
    </head>
    <body>
        <div id="alerta" class="alert alert-primary m-5" role="alert">
            Exemplo de alerta (primary) con Bootstrap!
        </div>
        <script>
            function eventos(){
                document.getElementById("alerta").addEventListener("click"
,cambiarEstilo,false);
            }
            window.onload= eventos;
            document.getElementById("alerta").classList.remove("m-5");
            document.getElementById("alerta").classList.add("m-5");
            function cambiarEstilo(){
                document.getElementById("alerta").classList.toggle("alert-
danger");
                document.getElementById("alerta").classList.toggle("alert-
primary");
                console.log(document.getElementById("alerta").classList);
            }
            //Empregando a propiedade className
            //document.getElementById("alerta").className="alert alert-
secondary m-2";
        </script>
    </body>
</html>
```

No exemplo mostrado temos unha alerta sobre a cal realizamos as seguintes operacións:

- Eliminar a clase de marxe *m-5*.
- Engadir de novo a marxe *m-5*.
- Evento de clic que permite intercambiar as clases *alert-primary* pola de *alert-danger*.
- O uso da propiedade *className* está comentado. Dito exemplo permite indicar nunha cadea de texto todas as clases separadas por espazos en branco.

3. Eventos

JavaScript naceu coa intencionalidade de engadir interactividade ás páxinas. Cando un usuario fai algo na páxina, prodúcese un *evento* ou cando sucede algo na nosa páxina, por exemplo a súa carga inicial. A programación orientada a eventos está deseñada para que cando se produce un evento se teña que executar a acción correspondente.

A xestión de dito evento pode ser tratada en liña na propia etiqueta na que queremos capturar o evento. Dita práctica non é recomendada porque non estamos independizando a presentación do que é a xestión de JavaScript. Hoxe en día, aínda que este modelo sigue sendo funcional, os navegadores incorporan un modelo de xestión de eventos que proporcionan información sobre cando ocorre un evento.

No seguinte exemplo móstrase como cambiar un `<p>` ao facer clic sobre el facendo uso do evento `onclick` en liña sobre a mesma etiqueta. Cando ese evento é disparado, execútase unha función `cambiarCSS`. Poderiamos non poñer ningunha función e incluír o código no propio `onclick`. O resultado obsérvase nas figuras 2 e 3.

```
<!DOCTYPE html>
<html>
    <head>
        <title>UD 6</title>
    </head>
    <body>
        <p id="p1" onclick="cambiarCSS()"> Fai clic aquí </p>
        <script>
            document.getElementById("p1").style.cssText="background-
color: lightblue; color: blue; font-
size: large; border: blue solid 5px; padding: 30px; margin: 50px";
            function cambiarCSS(){
                document.getElementById("p1").style.cssText="background-
color: lightgreen; color: green; font-
size: large; border: green solid 5px; padding: 30px; margin: 50px";

            }
        </script>
    </body>
</html>
```

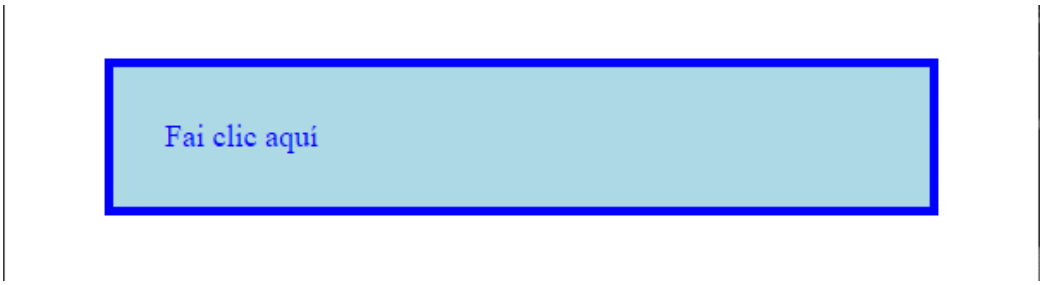

Figura 2: Páxina web antes de facer clic sobre o elemento `<p>`

Figura 3: Resultado da páxina web tras facer clic sobre o elemento `<p>`

Dado outro exemplo de etiqueta <a> co evento `onClick` en liña:

```
<a href ="paxina.html" onclick ="alert('enlace pulsado')">Fai clic aquí</a
>
```

Dita acción de lanzar unha alerta pode ser executada dende unha función:

```
<!DOCTYPE html>
<html>
    <head>
        <title>UD 6</title>
    </head>
    <body>
        <a href ="paxina.html" onClick ="alertar()">Fai clic aquí</a>
        <script>
            function alertar(){
                alert('enlace pulsado');
            }
        </script>
    </body>
</html>
```

O problema que presenta dito exemplo e que ao facer clic no enlace móstrase a alerta e a continuación cargará `paxina.html`. Nese momento desaparecerán da memoria os obxectos que estaban nun principio cando se orixinou o evento. Isto pode ser un problema xa que supoñamos que a función á que chamamos ten que executar varias tarefas, estas terían que facerse antes de cargarse a nova `paxina.html`.

Este exemplo mostra a secuencia de execución na xestión de eventos cando un evento provoca a execución dun script e tamén provoca a acción por defecto para ese obxecto (no caso do noso exemplo será abrir a nova páxina). Este caso provoca unha secuencia de execución:

- O script execútase primeiro.
- A acción por defecto execútase despois.

En ocasións pode ser interesante bloquear ou evitar que se execute a acción por defecto. Por exemplo, no caso anterior poderiamos evitar que conectara coa nova páxina, engadindo un *return false*. Cando programamos un xestor de eventos, dito xestor pode devolver *true* ou *false*. No caso de devolver *false,* quere dicir que non se execute a acción por defecto.

```
<a href ="paxina.html" onclick ="alert('enlace pulsado'); return
false">Fai clic aquí</a>
```

No exemplo anterior, ao pulsar o enlace fará saltar a alerta e cando se peche executará a instrución de `return false` que indicará ao navegador que non execute a acción por defecto (neste caso abrir `paxina.html`).

Este modelo de xestión de eventos en liña estendeuse dende atributos de elementos a propiedades dos propios elementos (por exemplo: `elemento.onclick = acción`). Sendo aceptado por exemplo o seguinte código:

```
<!DOCTYPE html>
<html>
    <head>
        <title>UD 6</title>
    </head>
    <body>
        <a id="enlace1" href ="paxina.html">Fai clic aquí</a>
        <script>
            document.getElementById("enlace1").onclick=alertar;
            function alertar(){
                alert('enlace pulsado');
            }
        </script>
    </body>
</html>
```

É importante resaltar que non se poñen os parénteses ao chamar a función *alertar.* Se poñemos os parénteses, a función sería executada e o seu resultado asignado a propiedade `onClick` do elemento que non é o que se buscaba. Ao poñelo sen parénteses considérase evento e a función soamente se executa cando se dispara o evento.

Esta nova forma de xestión de eventos non foi estandarizada polo World Wide Web Consortium (W3C) pero segue sendo válida na actualidade. Supón xa un avance porque a asignación de eventos a un obxecto pódese facer dende JavaScript, polo tanto contribúe a separación da presentación de toda a xestión de JavaScript.

Ademais, con esta metodoloxía xa podemos facer lanzar un evento de forma manual chamando ao método correspondente e automaticamente se executará a acción asociada. No caso do exemplo que estamos a tratar, se chamamos a función `onClick()` fará lanzar o evento:

```
<script>
    document.getElementById("enlace1").onclick=alertar;
    function alertar(){
        alert('enlace pulsado');
    }
    document.getElementById("enlace1").onclick();
</script>
```

Existe a posibilidade de realizar a chamada a máis dunha función directamente na propiedade do evento, é o que se coñece como evento multifunción. O único que temos que facer é declarar unha `function()` coas chamadas correspondentes como se observa no exemplo seguinte (recordando escribir punto e coma ao final da chamada).

```
<script>
    document.getElementById("enlace1").onclick=function (){
        alert('enlace pulsado (aviso1) ');
        alert('enlace pulsado (aviso2)');
    };
</script>
```

O W3C puxo especial interese nos problemas presentados polos modelos tradicionais de rexistro de eventos e ofreceu unha maneira sinxela de rexistrar sobre un obxecto determinado os eventos que precisásemos. A clave para poder facer todo isto foi o método `addEventListener()` que conta con tres parámetros:

- Tipo do evento.
- Función a executar cando se dispara o evento.
- Valor booleano de verdadeiro ou falso que indica o momento no que se teñen que executar as accións asociadas ao evento (será analizado máis adiante).

Seguindo o exemplo que estamos traballando neste apartado, ao facer clic no noso enlace saltaría o evento da alerta.

```
<script>
    document.getElementById("enlace1").addEventListener('click',alertar,fa
lse);
    function alertar(){
        alert('enlace pulsado (aviso1) ');
    }
</script>
```

A principal vantaxe que presenta este método é poder engadir tantos eventos como se precisen

.

```
<script>
    document.getElementById("enlace1").addEventListener('click',metodo1,fa
lse);
    document.getElementById("enlace1").addEventListener('click',metodo2,fa
lse);
    document.getElementById("enlace1").addEventListener('click',metodo3,fa
lse);
</script>
```

W3C non especifica a orde de disparo dos eventos polo que non sabemos cal das tres funcións se disparará primeiro.

Ademais tamén podemos empregar unha función anónima dentro da propia función de `addEventListener`.

```
<script>
    document.getElementById("enlace1").addEventListener('click',function()
{
        this.style.color="red";
    },false);
</script>
```

3.1. Tipos de Eventos:

Son moitos os eventos que poden ser xestionados polos navegadores web. Na seguinte táboa enuméranse os máis empregados:

Evento	Disparador do evento
onblur	O elemento perde o foco
onchange	O contido dun campo cambia.
onclick	Faise clic co rato sobre o elemento.
ondblclick	Faise dobre clic co rato sobre o elemento.
onerror	Hai algún erro ao cargar a páxina ou unha imaxe.
onfocus	Un elemento ten o foco.
onkeydown	Unha tecla do teclado é presionada.
onkeypress	Unha tecla do teclado é presionada ou mantense presionada.
onkeyup	Unha tecla do teclado presionada é soltada.
onload	Unha páxina ou imaxe terminaron de cargarse.
onmousedown	Un botón do rato é presionado.
onmousemove	O cursor do rato móvese cara ese elemento.
onmouseout	O cursor do rato móvese fora dese elemento.
onmouseover	O cursor do rato móvese sobre ese elemento.
onmouseup	Libérase un botón do rato.
onresize	O tamaño da ventá é modificado.
onselect	Selecciónase o texto.
onunload	O usuario abandona unha páxina.

Unha listaxe completa pode ser consultada na documentación de W3C (https://www.w3schools.com/jsref/dom_obj_event.asp).

Destacar entre ditos eventos os que nos permite executar algo cando a páxina está cargada por completo. O evento dispárase ao final do proceso de carga do documento. Neste punto, a árbore DOM está creada e todos os obxectos do documento son DOM, e todas as imaxes cargáronse por completo.

```
<script>
    let clicksRealizados = 0;
    let cambioEstilo=0;
    window.onload= function(){
        // CÓDIGO A EXECUTAR CANDO XA ESTÁ CARGADA A PÁXINA
    }
</script>
```

3.1.1. Orden de disparo dos eventos

Tendo en conta que na árbore DOM os elementos teñen uns antecesores e uns sucesores poden xurdir problemas cando temos o mesmo evento para o pai e o fillo (por exemplo cando se fai clic) e pódenos xurdir a pregunta ¿cal se executa primeiro?

W3C define un modelo no que se distinguen dúas fases:

- **Fase de captura**: que se lanza ata chegar ao elemento destino.
- **Fase de propagación**: na que se recorren os elementos dende abaixo ata arriba seguindo o modelo DOM.

Empregando o método `addEventListener` dispomos do terceiro elemento que pode ser `true` ou `false` en función de cando queremos que se rexistre o evento: fase de captura (`true`) ou fase de propagación (`false`).

A mellor forma de velo é na práctica co seguinte exemplo (figura 4) no que temos dous `<div>` con xestión de ambos eventos de clic.

Figura 4: Páxina web con dous `<div>` con diferentes cores de fondo sobre os que se xestionan ambos eventos de clic

```
<!DOCTYPE html>
<html>
    <head>
        <title>UD 6</title>
    </head>
    <body>
        <div id="div1">
            Elemento PAI
            <div id="div2">
                Elemento FILLO
            </div>
        </div>
        <script>
```

```
            document.getElementById("div1").style.cssText="padding: 20px;
background-color: lightblue; margin: 20px";
            document.getElementById("div2").style.cssText="padding: 20px;
background-color: lightgreen; margin: 20px";
            document.getElementById("div1").addEventListener('click',cambi
arCSSPai, true);
            document.getElementById("div2").addEventListener('click',cambi
arCSSFillo, false);
            function cambiarCSSPai(){
                document.getElementById("div1").style.backgroundColor="ora
nge";
                console.log("Executouse o evento PAI");
            }
            function cambiarCSSFillo(){
                document.getElementById("div2").style.backgroundColor="vio
let";
                console.log("Executouse o evento FILLO");
            }
        </script>
    </body>
</html>
```

O rexistro do evento fillo está en fase de propagación e o rexistro do evento pai en fase de captura. Imaxinemos que facemos clic sobre o elemento fillo. Os pasos a seguir polo navegador serán os seguintes:

- O evento clic comeza na fase de captura.
- O xestor de eventos comproba se hai algún ancestro dese elemento na árbore DOM que teña programado o evento de *click* na fase de captura (*true*).
- O xestor de eventos encontra o elemento pai que ten o terceiro parámetro a *true* e polo tanto executarase primeiro o método *cambiarCSSPai()*.
- O xestor de eventos volverá ao elemento fillo e iniciará a fase de propagación e executará *cambiarCSSFillo()* dado que está configurado para executarse nesa fase (*false* como terceiro parámetro do *addEventListener*).
- O xestor comprobará se existe algún elemento pai configurado para lanzar ese mesmo evento en fase de propagación (que non é o caso) e finalizará.
- Para deter a propagación do evento na fase de propagación, disponse do método *stopPropagation()*. Na fase de captura é imposible deter a propagación.

Podemos observar a través da consola que o que se executa primeiro tal e como detallamos é o pai e despois o fillo. É preciso comprobar as catro combinacións (*false-false*, *false-true*, *true-false*, *true-true*) para comprender ben o funcionamento ao facer clic sobre o elemento fillo. O resultado sempre nos dará o cambio de cor no elemento fillo e no elemento pai como se mostra na figura 5 pero o interese está en saber cal se executa primeiro o cal podemos ver a través da consola.

Figura 5: Resultado do noso exemplo ao facer clic sobre o elemento fillo

No caso de querer cancelar o evento si este é cancelable, sen deter o resto do funcionamento do evento, existe o método `preventDefault`. A chamada a `preventDefault` en calquera momento durante a execución, cancela o evento, o que significa que calquera acción por defecto que poida producirse como resultado deste evento, non sucederá.

Se o que desexamos é eliminar un evento dun elemento, empregaremos o método `removeEventListener()` que presenta a mesma sintaxe que o `addEventListener()` incluíndo tres parámetros: (1) evento, (2) función que se dispara e (3) `false` ou `true` se estaba rexistrado na fase de propagación ou captura.

```
<script>
    elemento.removeEventListener('evento', función, false|true)
</script>
```

3.1.2. Eventos Drag&Drop

Drag and Drop supón levar un elemento dende A ata B arrastrándoo. Pode ser aplicado a calquera clase de elementos como contedores, parágrafos, títulos, imaxes, etc.

Para facer drag and drop son necesarios dous elementos:

- Un elemento arrastrable no que teremos que engadir os atributos *draggable='true'* e asociar o evento *ondragstart=funcionxxx()*.
- Un elemento no que se poida soltar no seu interior no que como mínimo hai que asociar dous eventos: *ondragover* e *ondrop*.

Os enlaces e as imaxes son arrastrables por defecto, é dicir, non faría falta poñer o atributo *draggable*. Non obstante, poñelo permite unha maior lexibilidade de código.

Existen moitos eventos que se usan e poden ocorrer nas diferentes etapas dunha operación de arrastrar e soltar. Debemos distinguir entre os eventos disparados no elemento arrastrable e os eventos disparados no destino:

Evento no elemento arrastrable	Disparador do evento
`ondragstart`	Ocorre cando o usuario comeza a arrastrar o elemento.
`ondrag`	Ocorre cando se arrastra o elemento.
`ondragend`	Ocorre cando o usuario termina de arrastrar o elemento.

Evento no elemento destino	Disparador do evento
`ondragenter`	Ocorre cando o elemento arrastrado entra ao destino de colocación.
`ondragover`	Ocorre cando o elemento arrastrado está sobre o destino de colocación.
`ondragleave`	Ocorre cando o elemento arrastrado abandona o destino de colocación.
`ondrop`	Ocorre cando o elemento arrastrado se solta no destino de colocación.

Ao arrastrar un elemento, o evento `ondrag` dispárase cada 350 milisegundos.
No seguinte código mostramos un exemplo de *drag and drop*:

```
<!DOCTYPE html>
<html>
    <head>
        <title>UD 6</title>
        <style>
            .droptarget {
                margin: 15px;
                padding: 50px;
                border: 1px solid #aaaaaa;
            }
            .textStyle{
                background-color: lightgreen;
                padding: 10px;
            }
        </style>
    </head>
    <body>
        <p>Arrastra o elemento p entre os dous rectángulos:</p>
        <div class="droptarget" ondrop="drop(event)" ondragover="allowDrop
(event)">
            <p class="textStyle" ondragstart="dragStart(event)" ondrag="dr
agging(event)" draggable="true" id="dragtarget">Arrástrame!</p>
        </div>
        <div class="droptarget" ondrop="drop(event)" ondragover="allowDrop
(event)"></div>
        <p id="demo"></p>
        <script>
            function dragStart(event) {
                event.dataTransfer.setData("Text", event.target.id);
            }
            function dragging(event) {
                document.getElementById("demo").innerHTML = "O elemento p
está sendo arrastrado";
            }
            function allowDrop(event) {
                event.preventDefault();
            }
            function drop(event) {
                event.preventDefault();
                var data = event.dataTransfer.getData("Text");
                event.target.appendChild(document.getElementById(data));
                document.getElementById("demo").innerHTML = "O elemento p
foi soltado";
            }
        </script>
```

```
    </body>
</html>
```

Dando como resultado unha páxina web na que o parágrafo `<p>` pode ser arrastrado e soltado entre ámbolos dous `<div>` (figura 6).

A propiedade `dataTransfer` do evento é o centro de toda a actividade da función de arrastrar e soltar. Contén os datos que se envían na acción de arrastre. Establécese no evento `dragstart` e se procesa no evento `drop`. Ao activar `event.dataTransfer.setData(format, data)` establécese o contido do obxecto de tipo MIME e transmítese a carga de datos en forma de argumentos. `DataTransfer` incorpora tamén unha serie de propiedades para ofrecer sinais visuais ao usuario durante o proceso de arrastre.

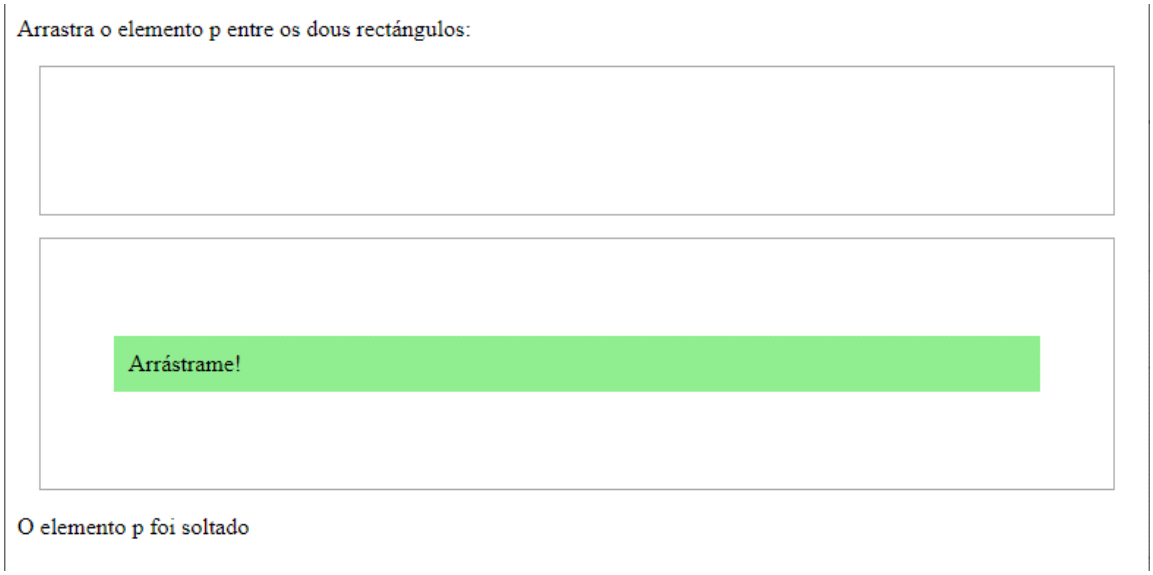

Figura 6: Resultado de arrastrar o parágrafo `<p>` dende o primeiro `<div>` e soltalo no segundo

`DataTransfer` ten varias propiedades que poden ser empregadas para controlar a resposta de cada elemento de destino da operación de arrastre a un determinado tipo de datos.
A funcionalidade de arrastrar e soltar facilita a interacción do usuario co navegador. Entre as aplicacións habituais podemos indicar as seguintes:

- Permitir ao usuario arrastrar os arquivos a subir a un servidor.
- Permitir ao usuario realizar seleccións arrastrando. Por exemplo: elixir un coche e escoller a cor arrastrando un recadro coa cor correspondente.
- Aplicacións de comercio electrónico nas que o usuario arrastra os produtos a súa cesta da compra.
- Xogos nos que o usuario arrastra e solta elementos.
- Poder ordenar elementos arrastrando e soltando.
- Efecto de ventás na que o usuario simula na web o despregamento de novos elementos arrastrando e soltando.

4. Animacións

As animacións a través de JavaScript realízanse programando cambios graduais no estilo do elemento sobre o que realizamos a animación. Os cambios son lanzados mediante un temporizador (*timer*). Cando os intervalos do temporizador son moi pequenos a animación parece continua. É similar a como vemos un vídeo que non deixa de ser unha secuencia de moitos fotogramas nun intervalo de tempo curto dando a sensación de continuidade.

Un dos métodos principais que intervén neste proceso é `setInterval(función, tempo)` que permite chamar a "función" ou avaliar a expresión pasada como primeiro parámetro. O segundo parámetro especifica o intervalo de tempo en milisegundos.
Vexamos un exemplo de animación no que un recadro `<div>` se move de esquerda a dereita e de dereita a esquerda (figura 7):

```
<!DOCTYPE html>
<html>
    <head>
        <title>UD 6</title>
        <style>
            #contedor {
                width: 500px;
                height: 50px;
                position: relative;
                background-color: lightblue;
            }
            #animacion {
                width: 50px;
                height: 50px;
                position: absolute;
                background-color: green;
            }
        </style>
    </head>
    <body>
        <p>
            <button onclick="animacion()">Activar animación!</button>
            <button onclick="pararAnimacion()">Parar animación!</button>
        </p>
        <div id ="contedor">
            <div id ="animacion"></div>
        </div>
        <script>
            var id;
            function animacion() {
                var elem = document.getElementById("animacion");
                var direction=1;
                var pos = 0;
                id = setInterval(frame, 5);
                function frame() {
                    if (pos > 450) direction=-1;
                    if (pos < 0) direction=1;
                    pos+=direction;
                    elem.style.left = pos + "px";
                }
            }
            function pararAnimacion() {
                clearInterval(id);
```

```
            }
        </script>
    </body>
</html>
```

Figura 7: Captura da web executándose a animación na que se move o recadro verde

Ao facer clic sobre o botón "Iniciar animación!" lánzase a execución do método `animacion()` encargado de mover o `<div>` verde en intervalos de 5 milisegundos. Os movementos son de píxel en píxel, e a dirección de movemento depende de se chegamos ao comezo (posición 0 en *x*) ou ao final (posición 450 en *x*). Existe tamén un método de `pararAnimación()` que permite parar o temporizador e polo tanto evitar que se volva a executar o método `animacion()`.