# 1 Introdución aos portais de información

*Obxectivos:*

- ✓ *Introdución ás páxinas web*
- ✓ *Diferenciar entre portais estáticos e dinámicos*
- ✓ *Comprobar á dificultade á hora de manter os portais estáticos*

Na arañeira mundial podémonos encontrar dous tipos de páxinas, as que se presentan sen máis funcionalidades que as ligazóns e outras coas que podemos interactuar.

## 1.1 Páxinas estáticas ou páxinas dinámicas?

Esta é a primeira pregunta que nos debemos de facer cando nos enfrontamos ante a posibilidade de empezar a deseñar un portal web. A diferenza principal entre ambas estriba na forma da súa construción, xa que en canto ao contido non debería haber diferenza entre ambas tipoloxías.

Unha páxina estática é aquela que non permite interactuar ao usuario con ela, polo que fagamos o que fagamos, premamos os botóns que premamos, actualicemos as veces que queiramos... sempre se manterá igual a páxina. Son relativamente sinxelas á hora de crealas, aínda que resultan moi complexas á hora de mantelas.

A linguaxe de visualización de ambas será unha linguaxe de marcas ou etiquetas que se denomina **HTML**, que é a que nos permite dar formato ao texto e ás imaxes que pensamos visualizar co navegador. Esta linguaxe é a que entenden os navegadores e encárgase de estandarizala o **W$^3$C**.

A dinámica será a que permita que a páxina cambie cando interactuemos con ela, de calquera dos dous modos posíbeis, no cliente ou no servidor. Inclúe linguaxes de programación, aparte da linguaxe de etiquetas, como poden ser linguaxes de script na parte do cliente ou linguaxes de programación de servidor á hora da construción da páxina na parte do servidor.

Co HTML podemos introducir ligazóns, seleccionar o tamaño dos textos ou meter imaxes entre eles, pero será dun xeito prefixado; en ningún caso, a linguaxe de etiquetas permite realizar algo tan sinxelo como sería unha suma ou unha resta ou variar automaticamente o contido dunha das páxinas do portal a partir dunha base de datos.

Por iso, cómpre o emprego doutras linguaxes moito máis versátiles e que terán unha aprendizaxe máis complicada, pero que serán quen de responder de maneira intelixente ás peticións do navegador e que permiten a automatización das tarefas tediosas.

Supoñamos que temos un portal de información adicado á competición de Fórmula 1 de automovilismo. Unha das informacións que poderiamos ter é quen gaña cada carreira ao longo do ano e cantos puntos leva no mundial de pilotos. Esta información pódese conseguir dun xeito periódico tras a disputa de cada carreira e a poderiamos almacenar doadamente nunha base de datos. Se traballamos con páxinas estáticas, teriamos que construír unha páxina independente para cada piloto e para cada gran premio nas que introduciriamos *a man* cada un dos resultados. Tamén, cada gran premio teriámonos que acordar de quitar as páxinas vellas e colgar as novas. Todo isto podería ser facilmente resolto mediante páxinas dinámicas; o que fariamos sería crear un programa (só un) que se encargaría de recoller da base de datos os resultados e de confeccionar as páxinas de cada piloto e de cada gran premio cos datos actualizados cada vez que algún cliente abrise a páxina pedida. Non teriamos fisicamente construídas as páxinas con anterioridade senón que se farían automaticamente cos datos existentes nese momento.

Ben, vamos a empezar vendo como facer páxinas estáticas para ir avanzando cara a construción das páxinas dinámicas, que son as recomendábeis, grazas ao seu fácil mantemento.

## 1.2 Consellos antes de se pór a escribir código HTML

Imos a empezar a traballar e debemos ter en conta que se temos proxectado un portal cun gran número de páxinas o máis habitual será empezar deseñando unha páxina coa estrutura estética, que despois repetiremos a modo de patrón en todas as páxinas.

Para aforrar tempo e erros á hora de facer as páxinas podemos seguir unha serie de consellos útiles:

- Non usar espazos nos nomes dos ficheiros das páxinas ou das imaxes (tampouco usar caracteres como o *ñ* ou as tiles). Sería aconsellábel escribir en minúsculas e cando cambiamos de palabra usar a maiúscula: *nomeDeFicheiro*.
- Ter coidado coas maiúsculas e as minúsculas nos nomes dos ficheiros que usamos. Se as utilizamos equivocadamente o portal pode funcionar en **Windows** (porque tanto lle ten que sexa maiúscula ou minúscula), pero pode ser que deixe de funcionar cando o subamos ao servidor **Linux** (porque estes sistemas si que distinguen entre maiúsculas e minúsculas).
- Traballar sempre cunha soa extensión para os ficheiros que conteñen o código HTML. Non debemos mesturar páxinas con extensión *html* e *htm* (o meu consello é que usedes sempre a extensión *html*).
- Debemos sangrar o código, de forma que queden máis á dereita aquelas etiquetas que estean englobadas dentro doutra.

## 1.3 Que cómpre?

É ben sinxelo, o único que necesitamos para poder facer páxinas web é un editor de textos para escribilas e un navegador para visualizalas.

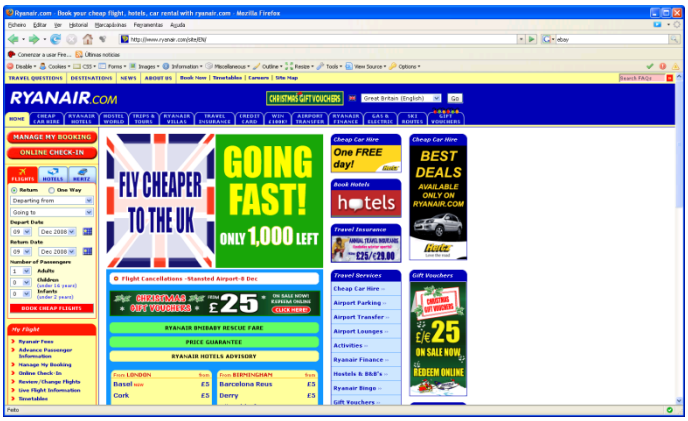
## 1.4 Que navegador escoller?

## 1.5 Funcionalidade ou estética?

Segundo a maioría de estudos **a primeira impresión é a que conta** cando estamos a falar de portais web, polo que as páxinas que os forman xúlganse en a penas uns segundos. Verdadeiramente, os visitantes pódense levar unha boa impresión en décimas de segundo e, tamén, unha mala impresión que faga que a abandonen (probabelmente sen comprobar se a oferta é mellor que a dos competidores) e que non teñan ningunha intención de volver a ela.

Fai tempo que sabemos que a estética é importante. Hai estudos que demostran que as persoas máis atractivas, así como as máis altas, gañan máis cartos por termo medio que as que non están consideradas na mesma clase. Existen estudos análogos que avalían o atractivo visual dunha serie de páxinas web, en escalas que van dende **Moi pouco atractiva** ata **Moi atractiva**. Pois ben, estes estudos concluíron que as decisións que se tomaron durante as dúas primeiras décimas de segundo mantíñanse durante o resto da observación.

Non hai dúbida algunha de que as persoas son moi poco pacientes, e menos na Internet. Nembargante, resulta difícil entender como podemos forxar unha impresión en décimas de segundo cando a meirande parte das páxinas tardan varios segundos en descargar. Polo tanto, **é importante unha velocidade de descarga pequena** para que teñamos opcións de ser atractivos, o que nos obriga a usar imaxes con pouco peso.




Páxinas como a de Ryanair é moi pouco atractiva: utiliza cores chillóns e textos animados en varios puntos da páxina principal; aínda así, é unha das páxinas web máis rendíbeis que existan. O mesmo sucede con centos de páxinas que dificilmente gañarían premios de deseño; algunhas delas parece que foran deseñadas a altas horas da madrugada nunha servilleta de papel... e a saber en que estado por parte dos autores.

Entón, por que se as primeiras impresións son tan importantes estas páxinas teñen éxito na Internet? Pode ser que a resposta sexa que **a estética dunha páxina ven despois da súa funcionalidade**.

Non é aconsellábel pedir a ninguén que xulgue o atractivo visual dunha web mentres non coñezamos a opinión que ten sobor como se realizan as tarefas básicas na páxina. Debemos pensar antes en que queren facer cando visitan a páxina ou en se o poden facer dun xeito sinxelo e rápido. Só cando esas preguntas reciban respostas positivas deberemos centrar as nosas enerxías na apariencia visual.

A pesares de todo, funcionalidade e estética non teñen que estar en conflito, pero... as páxinas web que están a gañar máis cartos céntranse moito máis na funcionalidade que na apariencia visual. Polo tanto, non paga a pena intentar avaliar o atractivo visual mentres a funcionalidade non estea conseguida, pois pódenos levar polo camiño equivocado.

Por se aínda non estamos convencidos, vexámolo dende outro punto de vista: temos en conta cando deseñamos unha páxina web aos posíbeis visitantes? Facemos un deseño que nos resulta a nós atractivo ou tentamos de que lle resulte atractivo aos posíbeis clientes?

Esta claro que a facilidade para manexar elementos tecnolóxicos, como pode ser unha páxina web, que teñen os xoves non é a mesma que teñen outros grupos de persoas (que probabelmente teñan máis cartos e por tanto máis posibilidade de facer negocio con eles) con maior idade ou menos estudos. Algúns elementos gráficos perden validez para este tipo de usuario; aínda que este tipo de usuario recoñece que a estética é algo importante prefire a funcionalidade: quere que se presente a información de maneira clara e agradábel, para levar a cabo as súas actividades.

Por isto temos que ter coñecemento do tipo de usuario ao que vai dirixido o noso traballo; temos que saber cal é a súa prioridade sen esquencer que a estética pode axudar e facer máis amena a navegación polo portal. Debemos dotar as nosas páxinas dunha estética sinxela que brinde funcionalidade ao usuario a partir das súas prioridades.

Entre os datos que podemos ter en conta á hora de establecer prioridades en canto a funcionalidade e a estética poderían estar:

- Os usuarios **non recordan o nome da páxina** totalmente, pero logran chegar a elas polo nome da empresa empregando buscadores
- A **velocidade** de carga da páxina debe ser considerada polo usuario como **non lenta**
- O **tamaño da fonte debe contrastar co fondo** e **ser o suficientemente grande**, pois moitos usuarios teñen problemas visuais
- Na páxina principal non debemos ter un **exceso de información** e **ligazóns**
- Un exceso de publicidade fai que a xente obvie información, pois os usuarios non a distinguen no primeiro vistazo polo que **non debemos unir información con imaxes publicitarias**
- Os usuarios **desconcértanse cando cambiamos a estética do portal dunha páxina a outra**, polo que a debemos manter en todo o sitio
- A xerarquización debe ser correcta, **debemos ir do máis xeral ao máis particular**
- A **linguaxe debe ser apropiada** para os posíbeis usuarios; por moito que nos saibamos que *tkm* é *quérote moito*, non o ten por que saber o posíbel visitante
- A ser posíbel, a páxina **non debe ter posibilidade de deslizamento vertical**, e **nunca o debe ter horizontal**
- **As imaxes** poden ser un bo recurso gráfico, **non deben ser só decorativas**; poden estar relacionadas co menú ou prestar información de interés, senón poden ser vistas como publicidade
- Se hai varios menús nunha páxina, os usuarios **tenden a deixar de lado os que usan menos habitualmente**
- **Non debemos dividir a páxina por medio de marcos**, pois soen desconcertar ao usuario
- **O número de pasos** para chegar ao obxectivo buscado **debe ser o mínimo posíbel**
- Hai que **evitar pór iconas sen texto**, e asegurarse que son o suficientemente grandes e claras para o usuario
- Preferibelmente non usar tecnoloxía **Flash**, xa que se pode lograr coa linguaxe HTML e resulta moito máis accesíbel

## 1.6 A accesibilidade

Se temos en conta que non existe unha porcentaxe alta de xente con problemas de accesibilidade (deficiencias visuais, auditivas... ) aos contornos web, poderiamos pensar que non é necesario que nos preocupemos por este tipo de casos. Pero se o enfocamos a que case o 20% por cento da poboación en Galiza pasa dos 65 anos, xa temos unha porcentaxe considerábel de posíbeis usuarios que poden ter determinados problemas de accesibilidade.

Algúns exemplos de problemas concretos cos que se soen encontrar as persoas maiores sen experiencia previa no uso de ordenadores ou de Internet, e que para outros sectores da poboación tal vez non tiveran especial trascendencia, poderían ser:

- **As ventás emerxentes (popups) desconcértanos**, ao igual que calquera elemento que actúe autonomamente. Tenden a culparse e pensar que fixeron algo mal, e tratan de interactuar na nova ventá en lugar de facelo coa orixinal.
- **Tenden a bloquearse ante unha película de introdución en flash de bucle infinito**, especialmente se a única forma de comezar a interacción é descubrir unha ligazón pouco visíbel que lles leve aos contidos da páxina.
- **A barra de desprazamento desconcértaos**, cústalles intuír que hai máis información nunha páxina que a que vemos nese momento na pantalla e que temos que usar esa barra para visualizala. Se, aínda por riba, empregamos unha barra non estándar, agudizamos o problema.

Aínda así, a pesares de todos estes problemas, os maiores felicítanse por cada pequeno avance, e agradecen encontrar lugares nos que lles resulta fácil interactuar.

**Á idade de 65 anos a metade da poboación terá algún tipo de discapacidade**, independentemente da súa severidade. Para a cuarta parte deses maiores a discapacidade será severa. Cando deseñemos portais web para persoas maiores temos que ter en mente que moitos deles terán algunha desas discapacidades: temos que intentar compensar esa dificultade no acceso á información.

Cada problema específico afecta a determinados aspectos da interacción:

- **Problemas de visión:** as persoas con este tipo de problema terán máis dificultades que o resto cando se encontren con tipos de letra pequenos, pouco contraste entre o texto e o fondo da páxina, imaxes como fondo na páxina...
- **Problemas de psicomotricidade:** afecta á hora de manexar o rato ou o teclado. Os deseños que requiran unha habilidade especial no manexo destes dispositivos seranlles difícilmente manexábeis: menús despregábeis, áreas de ligazón pequenas, accións de teclado combinando varias teclas simultaneamente...
- **Problemas de audición:** se ofrecemos ao usuario algún tipo de información de forma unicamente auditiva limitaremos o acceso a ela ás persoas con dificultades auditivas.
- **Problemas cognitivos:** algunhas persoas maiores poden ter dificultades para entender algúns modelos cognitivos usuales en Internet. Poden ter impedimentos para desenvolverse con soltura no contorno informático en xeral, ou ter problemas de memoria e atención, co que se van a distraer con facilidade.

En relación aos problemas cognitivos de memoria e atención, un estudio da Universidade de California Berkeley conclúe que **os problemas de memoria a curto prazo asociados á idade proveñen, na realidade, dunha falta de capacidade para filtrar as distraccións exteriores, e non de problemas de concentración**.

Polo tanto, aínda que non teñan problemas para concentrarse, si que os poden ter para evitar distraerse fronte a perturbacións externas. De aí que sexa importante que as páxinas que visitan os maiores non conteñan elementos distractivos, tales como animacións innecesarias, información irrelevante, publicidade, ventás emerxentes, música de fondo, reclamos visuais axenos á temática central...

# 2 Linguaxes visuais: JavaScript

*Obxectivos:*

- ✓ *Definir que é JavaScript.*
- ✓ *Identificar as partes que interveñen nunha conexión cliente-servidor.*

## 2.1 Introdución

Inicialmente, crear un documento HTML era crear algo de carácter estático, inmutábel co paso do tempo. Cárgase a páxina e acabouse o conto: non podemos ***interactuar*** con ela.

Para intentar evitar este problema, pois a maioría de usuarios pretenden ter unha especie de diálogo coa páxina que visitan, xorde a interface CGI que, unida aos formularios, comeza a permitir un pouco de interacción entre o ***cliente*** (o que navega) e o ***servidor*** (o que ten as páxinas que queremos visitar). Podemos encher un formulario e mandarllo ao servidor, tendo desta maneira unha vía de comunicación.

Nembargante, para enviar un formulario necesitamos facer unha nova petición ao servidor que, cando a procese, nos enviará (se procede) o resultado.

E se nos esquecemos de encher algún campo?

Cando o servidor procese a información, decatarase de que nos esquecimos de encher algún dos campos, e devolverá unha páxina cunha mensaxe dicindo que faltan campos por completar. Neste caso, teremos que volver a cargar a páxina, encher o formulario, envialo, volver ser analizado polo servidor, e, se esta vez está correcto, devolvernos a páxina que queriamos.

Todo isto supón recargar innecesariamente a rede. A utilidade do JavaScript (parte dinámica no cliente) xorde neste momento: se dalgunha maneira dende o propio cliente existira unha forma de poder comprobar isto antes de enviar a nosa petición ao servidor, estariamos aforrando tempo e non cargariamos a rede.

O seu nome oficial é **ECMAScript** e foi desenvolvido inicialmente por Brendan Eich para Netscape (no Navigator 2.0) e forma parte dos navegadores dende 1996. O desenvolvemento do estándar aínda non concluíu e é mantido na actualidade pola organización ECMA International (European Computer Manufacturers Association International).

Buscando esta interacción co usuario, xorden linguaxes destinadas a ser usadas na rede. Entre elas destacan AJAX (baseado na linguaxe Java) ou a tecnoloxía ActiveX. Ambas presentan o mesmo problema: a aprendizaxe destas opcións supón un esforzo considerábel.

Como solución intermedia usaremos JavaScript, que permite unha curva de aprendizaxe máis suave, o que nos permitirá avanzar dun xeito máis rápido e continuo. ***Trátase dunha linguaxe de tipo script, baseado en obxectos e guiado por eventos***, deseñado especificamente para o desenvolvemento de aplicacións cliente-servidor dentro do ámbito de Internet.

Os programas JavaScript van incrustados nos documentos HMTL, e son os que se encargan de realizar accións no navegador do cliente, como poden ser pedir datos, confirmacións, mostrar mensaxes, crear animacións, comprobar campos...

Como é unha linguaxe de tipo script trátase dunha linguaxe interpretada (fronte aquelas que son compiladas, como C), polo tanto, cando se vai lendo vaise executando polo cliente. Aínda que está baseada en obxectos, só presenta algunhas das súas características debido a limitacións autoimpostas para manter un nivel mínimo de seguridade na rede. Estar guiado por eventos significa que non vamos a ter un programa que se execute secuencialmente de principio a fin cando carguemos a páxina web. O que teremos é que cando no navegador suceda algún evento (por exemplo, un clic feito co rato), entón, se o noso código indica que fagamos algo, pasará. Ese algo será algunha función JavaScript.

O programa que vai interpretar o código JavaScript é o propio navegador, o que significa que se o noso non o soporta (xa sexa por limitacións do mesmo ou porque teñamos desactivada a execución deste tipo de código), non poderemos executar as funcións que programemos. Dende logo, Firefox, Safari, Opera, Chrome ou Explorer sopórtano.

## 2.2 Que cómpre saber?

Para poder seguir a parte relacionada co JavaScript deste manual cómpre ter coñecementos de:

- **HTML (HyperText Markup Language)**: é a linguaxe de etiquetas utilizada para crear documentos que se publican na rede. Se non se coñece esta linguaxe, non ten sentido algún aprender JavaScript.
- **Nocións de programación**: cómpre ter unhas nocións mínimas de programación para poder seguir este manual. Á vez que aprendemos a linguaxe, aprendemos a programar. Para iso, á vez que se mencionan os conceptos, móstrase a súa sintaxe en JavaScript, e téntase de explicar dun xeito detallado mediante pequenos exercicios que nos axuden a mecanizar tarefas de programación.
- **Ferramentas para programar**: chega cun editor de texto. Agora ben, como xa temos visto, é aconsellábel usar algún outro (NotePad++ ou DreamWeaver), pois facilítanos o traballo.
- **Ferramentas para ver os resultados**: JavaScript vai incrustado no código HTML, así que cun navegador que soporte JavaScript, non teremos problemas para poder ver o código. É interesante usar para a corrección os complementos que podemos instalar no navegador FireFox (FireBug ou Web Developer).
- **Follas de estilo**: Se queremos conseguir resultados máis rechamantes, cómpre ter coñecementos sobre follas de estilo (CSS) así como de HTML dinámico (DHTML).

## 2.3 Que podemos facer con JavaScript?

O que podemos facer con JavaScript é:

- **Programar**: é unha ferramenta de programación para os deseñadores web, que polo xeral non son programadores. Case calquera pode incluír anacos de código JavaScript tomados doutras partes e facer que funcionen no seu portal web.
- **Pór texto dinamicamente na páxina HTML**: podemos incluír texto dinamicamente nas nosas páxinas web mediante JavaScript.
- **Reaccionar ante eventos**: pode executarse cando algo sucede, como cando a páxina remata de cargarse ou cando facemos clic co rato sobor dalgún elemento.
- **Ler e escribir código HTML**: pode ler e cambiar o contido dos elementos que conforman a páxina HTML.
- **Validar datos**: úsase para validar os datos recollidos nos formularios antes de envialos ao servidor para evitar colapsos na rede e evitarlle traballo extra ao servidor.
- **Detectar o navegador**: non é de recibo a estas alturas do século XXI ter que usar as posibilidades de JavaScript para controlar que navegador usa o usuario, pero aínda seguen a non cumprir o estándar a maioría deles o que nos provoca código extra para subsanar no posíbel os erros que presentan e lograr que se vexan correctamente as nosas páxinas.
- **Crear cookies**: podemos almacenar información sobor do que visitamos en forma de variables que cargaremos cando volvamos visitar esa páxina.

## 2.4 A sintaxe: variábeis e constantes

### 2.4.1 Xeneralidades

Todas as linguaxes de programación teñen unha serie de normas de escritura (a súa sintaxe), e JavaScript non é unha excepción. Sintacticamente, é unha linguaxe similar a C.

A nivel xenérico segue as seguintes normas:

- **Case Sensitive**: é dicir, distingue as maiúsculas das minúsculas.
- **Comentarios**: **/*** ... ***/** para encerrar un bloque e **//** para comentarios dunha liña.
- **Cambio de liña**: todas as sentencias terminan en **;**
- **Código**: o código agrúpase encerrándoo entre chaves **{** ... **}**.

### 2.4.2 Como incluír código

O primeiro que temos que saber é onde o debemos colocar nun documento HTML. Temos dúas formas: unha, usando unha etiqueta especial **<script>** e outra, como parámetro doutras etiquetas (**<a>**, **<img>**, ...) en resposta a eventos de usuario predefinidos (premer o rato, cargar a páxina...).

O primeiro método consiste en usar a etiqueta **<script>**. Esta etiqueta admite varios atributos, entre os que destacan:

- **type**: indica que linguaxe de script é a que vai a ser programada no bloque; por defecto, **text/javascript**, pero non é a única linguaxe que existe de script (JScript, VBScript, ECMAScript…).

- **src**: permite incluír un ficheiro externo co código JavaScript para que sexa usado no documento.

Debemos ter en conta que haberá quen teña navegadores que non soporten JavaScript e haberá quen si pero non o queira soportar, así que dalgunha maneira temos que prever isto, evitando que todo o código saia pola pantalla do navegador. Para ilo encerramos o código JavaScript entre comentarios, do seguinte xeito para non ter problemas:

```
<script type='text/javaScript'>
<!--
Código JavaScript
//-->
</script>
```

E onde colocamos este código?

Non importa moito onde se poña, nembargante, un bo costume é introducilo na cabeceira do documento HTML, pois así estamos completamente seguros de que cando o documento apareza na pantalla, todo o código JavaScript estará cargado e listo para ser executado se se da o evento apropiado.

```
<html>
<head>
   <script type='text/javaScript'>
   <!--
      document.write("Ola mundo! ;-)");
   //-->
   </script>
</head>
<body>
</body>
</html>
```

A outra forma de introducir código JavaScript nos documentos HTML é en resposta a determinados eventos que pode recibir o documento como consecuencia das accións do usuario que esta vendo a páxina nese momento. Estes eventos están asociados a certas etiquetas HTML. A forma xeral será:

```
<etiqueta nomeEvento='CódigoJavaScript'>
```

Por exemplo, podemos facer que cando o rato se sitúe sobre unha ligazón[1] saia unha xanela cunha mensaxe[2]:

```
<a href='nomeDaPaxinaAQueQueroIr.html' onMouseOver='alert("Por aquí non pasas");'>Zona restrinxida de paso</a>
```

A menos que esteamos usando versións avanzadas da linguaxe, bastará con que poñamos:

```
<script type='text/javaScript'>
```

Se nos queremos asegurar de que un navegador vello non teña problemas co código dunha versión nova, podemos limitar o uso do noso código e facer que ignore o script póndolle para que versión da linguaxe estamos a traballar:

```
<script language='JavaScript 1.x'>
```

### 2.4.3 As variábeis

JavaScript ten a peculiaridade de ser unha linguaxe debilmente tipada, polo que podemos declarar unha variábel como numérica e máis adiante asignarlle unha cadea de texto sen que dea fallos na execución. É dicir, poderemos facer cousas como:

```
var oMeuNumero;
oMeuNumero = 4;
…
oMeuNumero = "Unha cadea de texto";
```

[1] Un posible evento é que o rato pase por unha zona predeterminada do documento HTML, que se programa co atributo **onMouseOver**. É un evento que está asociado, entre outras posibilidades, ás ligazóns.

[2] Hai unha función JavaScript da que falaremos máis adiante, **alert()**. Esta función amosa unha xanela por pantalla coa mensaxe que lle pasemos como argumento entre os parénteses.

Para declarar variábeis só temos que pór a palabra reservada **var** e a continuación a lista de variábeis separadas por comas. Non todos os nomes de variábel son válidos:

- Un nome válido de variábel **non pode ter espazos.**
- Pode estar formado por **números**, **letras** e o carácter subliñado **_**.
- Existen unha serie de **palabras reservadas** que **non se poden usar (if, for, while, break...).**
- **Non poden empezar por un número**: o primeiro carácter do nome da variábel ten que ser unha letra ou _.

```
var A Miña Variable, 123Probando, $Variable, for, while;          // definicións erróneas
var _UnhaVariable, P123robando, _123, forwhile, grtfytdklisdjl;   // definicións correctas
```

Podemos iniciar unha variábel cando a declaramos:

```
var texto = "Isto é unha cadea de texto";
```

### 2.4.4 Ámbito das variábeis

En JavaScript temos dous tipos de ámbitos para as variábeis: **local** e **global**. As variábeis locais serán aquelas que definamos dentro dunha función, mentres que as variábeis globais serán aquelas que definamos fóra das funcións, e poderán ser consultadas e modificadas por calquera das funcións que teñamos no documento HTML.

### 2.4.5 Constantes

As constantes non existen como tales en JavaScript; se queremos usar unha constante no programa, non quedará máis remedio que declarar unha variábel, asignarlle un valor dende o principio do programa, e non o tocar. Isto vai ser o máis parello ás constantes que podemos obter.

### 2.4.6 Tipos de datos

Existen diferentes tipos de datos en JavaScript, entre os que encontramos:

- **Números (enteiros e decimais):** os números englóbanse dentro do tipo **Number**; podemos representar os números enteiros de tres formas distintas: en base 10 (decimal, a de uso común), en base 16 (hexadecimal) ou en base 8 (octal). Para denotar un número hexadecimal farémolo escribindo diante do número 0x (por exemplo, 0xFE0) e un número en base octal farémolo precedéndoo dun 0 (por exemplo, 0666). Os números decimais gárdanse no formato **X.YYYYYYeNN**, onde **e** significa expoñente e **NN** é o número ao que elevar 10 para obter o decimal (por exemplo, o número 666,333 escríbese 6.66333e2, mentres que 0,01234 escríbese 1.234e-3).
- **Lóxicos:** toman os valores lóxicos **true** ou **false** (**1** ou **0**, **verdadeiro** ou **falso**). O resultado de avaliar unha condición será un valor booleano.
- **Cadeas:** en JavaScript, as cadeas están delimitadas por comiñas (dobres ou sinxelas), e poden ter calquera combinación de letras, espazos, números e outros símbolos. Podemos usar os caracteres de escape que existen en C para representar os caracteres invisíbeis: **\b** espazo cara atrás, **\n** salto de liña, **\r** retorno de carro, **\t** tabulador, **** barra invertida (\), **\'** comiñas simples ('), **\"** comiñas dobres (").
- **Obxectos, Nulos e Indefinidos:** unha variábel pode ser tamén un obxecto, e terá propiedades (atributos) e métodos (funcións); os nulos (**null**) e os indefinidos son valores especiais dentro do tipo **Object**. **null** é un valor especial que indica que un obxecto non foi asignado e o valor **undefined** (indefinido) indica que á variábel á que queremos acceder no lle foi asignado ningún valor, e polo tanto permanece indefinida.
- **Funcións:** As funcións tamén son un tipo de dato especial; se declaramos unha variábel de tipo **function**, verdadeiramente o que estamos facendo é declarar unha función.

### 2.4.7 Operadores

Os operadores son símbolos que denotan operacións que podemos realizar entre elementos chamados operandos. Estes operadores xeralmente serán binarios (necesitan dous operandos) e o resultado será operar o primeiro operando co segundo. Por exemplo, **+** é un operador que admite dous operandos e o resultado de facer a operación **a + b** é o resultado de sumar **a** e **b** (ou de concatenalo se se trata de variábeis alfanuméricas).

O operador máis elemental é o operador binario de asignación **=**. Así, se escribimos no código **a = b**, estamos asignando á variábel **a** o contido que ten nese momento a variábel **b**.

```
var a = 4, b = 3;          /* Variábeis globais */
document.write('<p>Valor de a: ' + a + '<br/>Valor de b: ' + b + '</p>');
a = b;
document.write('<p>Valor de a: ' + a + '<br/>Valor de b: ' + b + '</p>');
```

No exemplo podemos ver algúns dos elementos da sintaxe xa vistos con anterioridade, como o uso de comentarios, e tamén o operador **+** usado doutro xeito; cando estamos utilizando cadeas, o operador **+** fai a operación de **concatenación**. Por exemplo:

```
var cadea = "Suma" + " de " + "cadeas";
```

é o mesmo que se escribiramos:

```
var cadea = "Suma de cadeas";
```

### 2.4.8 Operadores aritméticos

Os operadores aritméticos son binarios (necesitan dous operandos), e realizan sobre eles algunha das operacións aritméticas coñecidas. En concreto, temos:

```
suma = a + b;
resta = a - b;
produto = a * b;
cociente = a / b;
modulo = a % b;          // o operador módulo calcula o resto da división enteira

// En JavaScript temos unha versión especial que xunta o operador de asignación con estes:

suma += 3;               // equivale a suma = suma + 3;
resta -= 3;              // equivale a resta = resta - 3;
produto *= 3;            // equivale a produto = produto * 3;
cociente /= 3;           // equivale a cociente = cociente / 3;
modulo %= 3;             // equivale a modulo = modulo % 3;
```

Temos tamén incrementos e decrementos: **++** e **--,**

```
var un, dous;

// non é o mesmo:

dous = 20;
un = dous++;             // resultado: un=20 e dous=21

// que isto:

dous = 20;
un = ++dous;             // resultado: un=21 e dous=21
```

no primeiro caso realízase primeiro a asignación e despois o incremento e no segundo caso realizamos primeiro o incremento e despois a asignación.

Por último, dentro dos operadores aritméticos temos o operador unario **-**, que aplicado diante dunha variábel cambia o signo da mesma.

```
/* Partimos de dúas variábeis 'a' e 'b' e vamos a realizar distintas operacións con elas */

var a = 10, b = 6;
document.write('<p>Valor inicial das variábeis: a = ' + a + ', b = ' + b + '</p>');
document.write('<p>Suma: a + b = ' + (a + b) + '</p>');
document.write('<p>Resta: a - b = ' + (a - b) + '</p>');
document.write('<p>Produto: a * b = ' + (a * b) + '</p>');
document.write('<p>Cociente: a / b = ' + (a / b) + '</p>');
document.write('<p>Módulo: a % b = ' + (a % b) + '</p>');
document.write('<p>PostIncremento: ++ = ' + (a++) + '</p>');

/* A partires deste punto xa temos incrementada a variábel 'a', polo que vale 11 */

document.write('<p>Cambio de signo: -a = ' + (-a) + '</p>');
document.write('<p>Suma compacta: a += b : a = ' + (a += b) + '</p>');
document.write('<p>Módulo compacto: a %= b : a = ' + (a %= b) + '</p>');
```

### 2.4.9 Operadores de Comparación

Os operadores de comparación son binarios e o seu resultado é un valor lóxico. Permítennos expresar se unha relación de igualdade/desigualdade é certa ou falsa dados os operandos. Temos os seguintes:

```
a == b                  // igual
a != b                  // distinto
a > b                   // maior que
a < b                   // menor que
a >= b                  // maior ou igual que
a <= b                  // menor ou igual que
a === b                 // estritamente igual
a !== b                 // estritamente distinto
```

Estes dous últimos comparan o valor e o tipo das variábeis. O primeiro devolve **true** cando os operadores toman o mesmo valor e ademais son do mesmo tipo; o segundo devolve **true** cando os operadores son distintos e de tipo distinto: é dicir, se os operandos teñen distinto valor, pero teñen o mesmo tipo, non son estritamente distintos.

```
/* Partimos de dúas variábeis, 'a' e 'b', e vamos a realizar distintas operacións con elas */

var a = 10, b = 6;
document.write('<p>Valor das variábeis: a = ' + a + ', b = ' + b + '</p>');
document.write('<p>Igualdade: a == b : ' + (a == b) + '</p>');
document.write('<p>Desigualdade: a != b : ' + (a != b) + '</p>');
document.write('<p>Menor que: a < b : ' + (a < b) + '</p>');
document.write('<p>Maior que: a > b : ' + (a > b) + '</p>');
document.write('<p>Menor ou igual que: a <= b : ' + (a <= b) + '</p>');
document.write('<p>Maior ou igual que: a >= b : ' + (a >= b) + '</p>');
a = '6';                // cambiamos o valor e o tipo da variábel 'a'
document.write('<p>Agora a = ' + a + ' (tipo: ' + typeof a + ')</p>');
document.write('<p>Estritamente igual: a === b : ' + (a === b) + '</p>');
a = 6;                  // volvemos a cambiar o tipo da variábel 'a'
document.write('<p>Agora a = ' + a + ' (tipo: ' + typeof a + ')</p>');
document.write('<p>Estritamente igual: a === b : ' + (a === b) + '</p>');
document.write('<p>Estritamente distinto:  a !== b : ' + (a !== b) + '</p>');
a = '6';                // volvemos a cambiar o tipo da variábel 'a'
document.write('<p>Agora a = ' + a + ' (tipo: ' + typeof a + ')</p>');
document.write('<p>Estritamente distinto:  a !== b : ' + (a !== b) + '</p>');
```

### 2.4.10 Operadores Lóxicos

Os operadores lóxicos serven para facer condicións máis simples por medio das regras da lóxica (fundamentalmente usaremos as regras **AND**, **OR** e **NOT**).

```
&&  AND                 // 'e' lóxico
||  OR                  // 'ou' lóxico
!   NOT                 // 'non' lóxico
```

Vannos a permitir expresar condicións complexas das que queremos saber o seu valor de verdade. Por exemplo, se temos tres números, podemos querer saber se un deles é maior que os outros dous, ou se é meirande que algún deles, ou se non é maior que ningún. Usando con tino os parénteses, podemos agrupar como nos conveña as condicións.

```
/* Partimos de tres variábeis, 'a', 'b' e 'c' coas que vamos a realizar distintas operacións */
var a = 10, b = 6, c = 11;
document.write('<p>Valor das variábeis: a = ' + a + ', b = ' + b + ', c = ' + c + '</p>');

/* Comparamos se 'a' é maior que 'b' e á vez maior que 'c' */
document.write('<p>(a > b) && (a > c) : ' + ((a > b) && (a > c)) + '</p>');

/* Comparamos se 'a' é maior que 'b' e tamén menor que 'c' */
document.write('<p>(a > b) && (a < c) : ' + ((a > b) && (a < c)) + '</p>');

/* Miramos se non se cumpre que 'a' sexa menor que 'b' */
document.write('<p>!(a < b) : ' + !(a < b) + '</p>');

/* Miramos se 'a' é maior que 'b' ou se é maior que 'c' */
document.write('<p>(a > b) || (a > c) : ' + ((a > b) || (a > c)) + '</p>');
```

### 2.4.11 Resto de operadores

Existen outro tipo de operadores en JavaScript:

- **new** é un operador que sirve para crear novas instancias dun obxecto vía o seu construtor.
- **delete** é un operador que sirve para destruír unha instancia concreta dun obxecto.

- **void** é un operador que permite avaliar unha expresión sen que devolva ningún valor. Poderiámolo considerar como unha función especial.
- **typeof** é un operador que devolve unha cadea que describe o tipo de dato que corresponde co obxecto (variábel, función…) que se escribe a continuación.

```
document.write('<p>Tipo NaN: ' + (typeof NaN) + '</p>');
document.write('<p>Tipo lóxico: ' + (typeof true) + '</p>');
document.write('<p>Tipo null: ' + (typeof null) + '</p>');
document.write('<p>Tipo decimal: ' + (typeof 45e-3) + '</p>');
document.write('<p>Tipo función: ' + (typeof parseInt) + '</p>');
document.write('<p>Tipo cadea de caracteres: ' + (typeof "Ola mundo") + '</p>');
document.write('<p>Tipo obxecto: ' + (typeof Math) + '</p>');

var a = new Array(1,2);
document.write('<p>a[0]: ' + a[0] + '</p>');
delete a;
document.write('<p>Tipo vector: ' + (typeof a) + '</p>');
```

## 2.5 A sintaxe: estruturas de control

### 2.5.1 Condicionais: if

A estrutura do condicional Se-Entón-Senón sigue esta sintaxe en JavaScript:

```
if(condición)
{
   código necesario
}
else
{
   código alternativo
}
```

A condición de comparación pode ser complexa (usando para ilo operadores lóxicos que unirán as distintas condicións), e podemos ter varios **if** aniñados; o bloque **else** non é obrigatorio. Se o código a executar é unha única sentencia non é necesario encerrala entre chaves, pero é recomendábel para evitar esquecementos se máis tarde temos que reformar o código.

```
var nome, idade;
nome = prompt('Como te chamas?', '');
idade = prompt('Cantos anos tes?', '');
idade = parseInt(idade);
if (idade < 18)
{
   alert('Síntoo, ' + nome + ', con ' + idade + ' anos non podes votar');
}
else
{
   alert('Podes votar; xa es maior de idade, ' + nome);
}
```

Existe outra sintaxe máis compacta para indicar o condicional:

```
[ condición ] ? [ sentenzaIf ] : [ sentenzaElse ] ;
```

que avalía a condición que temos; se o resultado é **true** execútase **sentenzaIf** e se é **false** execútase **sentenzaElse**.

```
(idade>18) ? alert("Podes votar: es maior de idade") : alert("Non podes votar: non tes 18 anos");
```

### 2.5.2 Condicionais: switch

O outro condicional que presenta a linguaxe úsase para non encadear condicións nunha serie de **if** á hora de elixir entre varias opcións.

```
switch(condición)
{
   case caso1: sentenzas; break;
   . . .
   case casoN: sentenzas; break;
   default: sentenzas por defecto; break;
}
```

Por exemplo:

```
var opcion, cor = 'A cor da túa froita preferida é o ';
opcion = prompt('Cal é a túa froita preferida?', '');
switch(opcion.toLowerCase())
{
   case 'laranxa':
   case 'mandarina': cor += 'laranxa'; break;
   case 'pera': cor += 'verde'; break;
   case 'amorodo': cor += 'vermello'; break;
   default: cor = 'non sei de que cor é esa froita'; break;
}
alert(cor);
```

### 2.5.3 Bucles: for

Este bucle presenta o mesmo aspecto que na linguaxe C:

```
for( [Iniciar variábel]; [Condición de saída]; [Actualizar variábel])
{
   Instrucións a repetir
}
```

Ao entrar ao bucle iniciamos a variábel que nos vai servir de referente no bucle, polo que esta serie de sentenzas só se executarán unha vez; despois de feito isto, repetiremos as instrucións existentes no corpo do bucle tantas veces como sexa necesario ata que se cumpra a condición de saída (aínda que non é obrigatorio, nun código correcto debe estar referida á variábel de referencia), tendo en conta que como derradeira instrución do bucle sempre estará a actualización da variábel de referencia. Ningunha das tres expresións son obrigatorias (por iso están entre corchetes); poderiamos escribir código como:

```
for(;;);              // bucle infinito
```

Tende coidado con este tipo de erros (bucles infinitos), pois bloquean o navegador; para desbloquealo, dependendo da versión do navegador, teremos que esperar uns segundos (desbloquéase automaticamente aos 30 segundos executando a mesma instrución sen cambio no xogo de variábeis a usar) ou ben teremos que matar o proceso do navegador.

Para ilustrar o uso do **for**, faremos unha táboa de multiplicar:

```
var i, j;             // Índices para percorrer a táboa
var maximo = 10;
document.write("<table border='2'>");
for(i=1; i<=maximo; i++)
{
   document.write("<tr><th>Táboa de multiplicar do " + i + "</th></tr>");
   for(j=1; j<=10; j++)
   {
      document.write("<tr><td>" + i + ' * ' + j + ' = ' + (i * j) + "</td></tr>");
   }
}
document.write("</table>");
```

### 2.5.4 Bucles: while

Este bucle tamén sigue a sintaxe do **while** da linguaxe C, existindo unha tradución directa entre un bucle **for** e un bucle **while**.

```
[Iniciar variábel];
while( [Condición de saída] )
{
   Instrucións a repetir;
   [Actualizar variábel];
}
```

Así, o exemplo anterior quedaría:

```
var i, j;             // Índices para percorrer a táboa
var maximo = 10;
document.write("<table border='2'>");
i = 1;
while(i<=maximo)
{
   document.write("<tr><th>Táboa de multiplicar do " + i + "</th></tr>");
   j = 1;
   while(j<=10)
```

```
    {
        document.write("<tr><td>" + i + ' * ' + j + ' = ' + (i * j) + "</td></tr>");
        j++;
    }
    i++;
}
document.write("</table>");
```

### 2.5.5 Bucles: do ... while

A sintaxe que presenta tamén é parella á que presenta en C:

```
[Iniciar variábel];
do
{
    Instrucións a repetir;
    [Actualizar variábel];
}
while( [Condición de saída] );
```

Podemos parar os bucles coas instrucións **break** e **continue**; o primeiro salta fora do bucle máis interno, e o segundo salta as instrucións que quedan por executar do bucle e volve á cabeceira do bucle. Existe a posibilidade de usar etiquetas no código (o formato sería **nomeEtiqueta:**), polo que as sentencias para parar os bucles poden facer referencia a unha delas (escribindo **break nomeEtiqueta** ou **continue nomeEtiqueta**).

Este bucle tamén pode ser equivalente aos dous anteriores, tendo en conta que sempre vamos a entrar como mínimo unha vez dentro do corpo do bucle antes de poder avaliar a condición de saída.

### 2.5.6 Bucles: for ... in

En JavaScript tamén dispomos doutro tipo de bucles que produce una iteración a través de todas as propiedades dun obxecto.

```
for(var 'propiedade' in 'obxecto')
{
    Instrucións a repetir
}
```

Por exemplo, este código percorre o **document** e mostra todas as súas propiedades.

```
for(var i in document)
{
    document.write('<p>document.' + i + ' = ' + document[i] + '</p>');
}
```

### 2.5.7 A construción with

Esta construción aforra ter que escribir sempre o nome dun obxecto se o vamos a utilizar moitas veces para acceder aos seus métodos, propiedades... Presenta a seguinte sintaxe:

```
with(obxecto)
{
    Varias sentenzas
}
```

Por exemplo:

```
with(document)
{
    write('Co <b>with</b> podemos evitar ter que escribir <b>document.write</b> ');
    write('cada vez que o usamos; É interesante cando se usa un método moitas veces e moi seguido. ');
    write('Isto só é unha proba, por iso escribo varias liñas en vez dunha soa.');
}
```

### 2.5.8 Funcións

En JavaScript tamén podemos definir funcións (por medio da palabra reservada **function**), ás que lle pasaremos argumentos e que devolverán valores. A estrutura xeral dunha función sería:

```
function nomeDaFuncion(argumento1, ..., argumentoN)
{
   Código da función
   return resultado;
}
```

Podemos chamar ás funcións que nós definamos como resposta dalgún evento ou como parte dun cálculo que esteamos a efectuar, por exemplo:

```
<a HREF='paxina.html' onClick='nomeDaFuncion(arg1,...); '>Calcular a serie de Fibonacci</a>
```

Teñen unha propiedade particular, e é que non hai un número fixo de argumentos; é dicir, podémolas chamar cun número calquera de argumentos e que sexan de calquera tipo. Estes argumentos poden ser accedidos ben polo seu nome, ou ben polo vector de argumentos que ten asociada a función (**nomeDaFuncion.arguments**). Deste xeito podemos saber cantos argumentos ten unha función empregando o atributo **length** do obxecto **Array**; a notación sería **nomeDaFuncion.arguments.length**. Por exemplo:

```
function contar()
{
   var texto;
   texto = 'A función foi chamada con ' + contar.arguments.length + ' argumentos\n\n';
   for(i=0; i<contar.arguments.length; i++)
   {
      texto += 'Argumento ' + i + ': ' + contar.arguments[i] + '\n';
   }
   alert(texto);
}
contar('Desta vez só pasamos un argumento');
contar('Desta vez pasamos ','dous argumentos');
contar('Desta vez pasamos ', 'tres ', 'argumentos');
```

## 2.6 A sintaxe: funcións propias da linguaxe

### 2.6.1 Introdución

JavaScript pode actuar cos elementos dunha páxina web. As funcións que vamos a describir aquí son funcións que non están ligadas aos obxectos do navegador, senón que forman parte da linguaxe. Serán as que permitan converter cadeas de texto a números, avaliar unha expresión, comprobar se se trata dun número ou dunha cadea...

### 2.6.2 parseInt(cadea [, base])

Esta función devolve un enteiro como resultado de converter a cadea que se lle pasa como primeiro argumento. Opcionalmente, podémoslle indicar a base de numeración na que queremos que nos devolva este enteiro (decimal, binaria, hexadecimal, octal...). Se non especificamos a base, devolve o resultado en base decimal.

Debemos ter coidado, pois se a cadea non se pode converter a un enteiro, a función devolve o maior enteiro que puidera construír a partir da cadea. Se o primeiro carácter da cadea non é un número, a función devolve **NaN** (Not a Number).

```
a = parseInt("12345");         // En 'a' almacenamos o valor 12345, e poderemos facer cousas como a++
a = parseInt("12345",10);      // Mesma instrución que a anterior, pero especificando a base de numeración
a = parseInt("123cuatro5");    // En 'a' almacenamos o valor 123
a = parseInt("a1234");         // En 'a' almacenamos o valor NaN

// Se usamos outras bases podemos obter resultados como os seguintes:

a = parseInt("FF",16);         // a = 255 (expresado en base decimal)
a = parseInt("GG",16);         // a = NaN
```

### 2.6.3 parseFloat(cadea)

Esta función devolve un real como resultado de converter a cadea que se lle pasa como argumento. Se a cadea non se pode converter a un real, a función devolve o maior real que puidera construír a partir dela. Se o primeiro carácter non pode ser convertido nun número, a función devolve **NaN**.

```
a = parseFloat("1.2345");      // En 'a' almacenamos o valor 1.2345
a = parseFloat("1.2345e-1");   // En 'a' almacenamos o valor 0.12345
a = parseFloat("1.2e-1er45"); // En 'a' almacenamos o valor 0.12
a = parseFloat("e");           // a = NaN
a = parseFloat("e-1");         // a = NaN
```

```
a = parseFloat(".e-1");        // a = NaN
a = parseFloat(".0e-1");       // a = 0
a = parseFloat("1.e-1");       // a = 0.1
```

### 2.6.4 escape(cadea)

Esta función permite tratar con cadeas que conteñen códigos de escape, que se utilizan para dar formato á saída do texto. Devolve a secuencia de códigos de escape de cada un dos carácteres da cadea que lle pasamos como argumento. Estes códigos serán devoltos co formato **%NN**, onde **NN** é o código de escape en hexadecimal do carácter. Hai algúns carácteres que non dan lugar a este código de escape, senón que o resultado de escape sobre eles volven a ser eles. Exemplos:

```
escape('$');                   // devolve %24
escape("a B c_1_%á");          // devolve a%20B%20c%20_1_%25%E1
```

### 2.6.5 unescape(cadea)

Devolve a cadea formada polos carácteres que corresponden ao código que especificamos. Os códigos débense especificar no mesmo formato co que a función **escape** os proporciona: **%NN**, onde **NN** é o código en hexadecimal.

```
unescape('outro%20exemplo');   // devolve 'outro exemplo'
```

### 2.6.6 isNaN(valor)

Esta función indica se un argumento **NON** é numérico, devolvendo **falso** cando é un número e **verdadeiro** cando non o é.

```
isNaN("Ola mundo");            // devolve verdadeiro
isNaN("091");                  // devolve falso
```

### 2.6.7 eval(expresión)

Esta función devolve o resultado de avaliar a expresión pasada como parámetro.

```
eval("2+3");                   // devolve 5
eval(2+3);                     // tamén devolve 5
eval("Ola");                   // devolve un erro xa que a variábel 'Ola' non está definida
```

A función **eval** espera unha expresión que poida ser convertida en anacos máis pequenos, números e operadores, para podela avaliar. Esta función considera como variábel todo o que non sexa un número ou un operador. Como non temos definida unha variábel global que se chame **Ola**, o navegador devolve un erro.

Vamos a supor que si temos definida a variábel global **Ola**. Supoñamos que na cabeceira do documento, dentro da etiqueta <**script**>, definímola como o texto dun saúdo.

```
var Ola = "Ola a todos";
eval('1 + Ola');
```

como **Ola** non é un número, pensa que é unha variábel e a busca, topándose que é unha cadea de texto. O resultado desta operación (**+**) será unha cadea, pois entende a operación como unha concatenación, interpretando que **1** e o contido de **Ola** son cadeas, dando como resultado **1Ola a todos.** Nembargante, se pomos outra operación, como podería ser a multiplicación, devolvería **NaN**.

## 2.7 Introdución aos obxectos

### 2.7.1 Introdución

A programación estruturada presenta convencións á hora de escribir algoritmos: hai tipos básicos de variábeis, estruturas de control e, a partires destes elementos, construímos estruturas máis complexas para poder dar solución a todo tipo de problemas. Todo ilo, dende un punto de vista máis ou menos secuencial, e tendo ás funcións e procedementos como principais protagonistas do desenvolvemento dun programa, sen estar interrelacionados cos datos que manexan.

Coa **P**rogramación **O**rientada a **O**bxectos (POO en galego ou castelán e OOP en inglés) este concepto cambia radicalmente, e o desenvolvemento dos programas céntrase, como indica o seu nome, nos obxectos.

### 2.7.2 Que é un obxecto?

A resposta é que un obxecto é un contedor que consta de dúas partes: **atributos** e **métodos**. Eses atributos van ser as características de cada instancia dun obxecto (é dicir, datos) e os métodos serán propiedades que ten ese obxecto (é dicir, funcións que fan cálculos ou modifican os atributos dos obxectos).

A diferenza coa programación estruturada está en que esas funcións non están separadas dos datos, senón que forman parte da mesma estrutura: o obxecto.

Supoñamos que queremos facer un programa que debuxe figuras xeométricas. O normal sería que non o fixeramos en modo texto, pois sería máis agradábel á vista que tivera elementos como xanelas, botóns, menús, barras de desprazamento... Todo isto é posíbel levalo a cabo coa programación estruturada, pero… non sería máis doado ter un obxecto **xanela**? Este obxecto podería ter os seguintes atributos: coordenadas da figura xeométrica que se vai a debuxar, estilo do debuxo... e podería ter uns métodos que, lendo eses datos, debuxara a xanela, cambiara o seu tamaño, pecháraa...

A vantaxe que tería isto é que xa podemos crear o número de xanelas que queiramos no noso programa, xa que as funcións asociadas terían en conta as variábeis dese obxecto e non llas teriamos que pasar como argumentos.

Igual que coa xanela, poderiamos crear o obxecto **botón**, que podería ter como atributos as coordenadas, se está premido ou non, se o rato está sobor del... e uns métodos que o debuxaran, o debuxaran premido, detectaran se se premera ese botón, fixeran algo ao premelo...

O relevante de todo isto é que estes métodos pertencentes a un obxecto proporcionan unha **interface** para manexalo: non cómpre saber como está feito un obxecto para podelo usar. Cando creamos un obxecto, en realidade estamos creando un *molde*, que recibe o nome de **clase**, no que especificamos todo o que podemos facer cos obxectos que utilicen ese *molde*.

A realidade é que creamos unha clase, e cando a queremos usar creamos instancias da clase, e estas instancias son as que imos a usar e as que chamaremos obxectos. Por exemplo, no caso da xanela, poderíamos empregar o seguinte pseudocódigo:

```
CLASE Xanela
ATRIBUTOS
x0, y0, x1, y1: ENTEIROS
MÉTODOS
inicializarCoordenadas()
debuxarXanela()
moverXanela()
FIN CLASE Xanela
```

E cando queiramos ter unha (ou máis) xanelas, declarariámolas (o que facemos é instancialas) como calquera outro tipo de variábel:

```
Xanela xanela1, xanela2;
```

Cando accedemos aos atributos que ten cada unha destas xanelas (valores que non teñen por que ser iguais para cada unha delas) ou cando utilizamos os métodos de cada unha delas, habería que pór:

```
xanela1.x0 = 3;              // obxecto.atributo
xanela2.debuxarXanela();     // obxecto.método(argumentos)
```

Cando traballamos con **POO** sempre se fala de acceso ás partes dun obxecto (públicas, privadas e protexidas), pero como JavaScript é máis simple, non ten distinción de accesos. Nembargante, a forma de iniciar as coordenadas da xanela máis correcta non sería como no exemplo, senón que se debería encargar de ilo unha función especial chamada **construtor** (esta función debe empezar sempre por maiúscula, para diferenciala do resto), á que lle pasaremos como argumentos os valores de inicialización. En JavaScript, o construtor é a función coa que crearemos o molde da clase.

### 2.7.3 Como son en JavaScript?

JavaScript non é unha linguaxe orientado a obxectos: está baseada nuns obxectos (os obxectos do navegador e outros propios) dos que podemos usar unhas funcións que xa están definidas e cambiar os valores das variábeis, sempre accedendo ás partes do obxecto mediante a notación da linguaxe. Tamén dispomos dunha notación en forma de vector para acceder a cada unha das variábeis do obxecto:

```
obxecto['variábel']
```

Ademais, permite crear obxectos, pero dun xeito moi sinxelo, dado que non ten unha das características esenciais da **POO**: a **herdanza**. Crear as nosas clases para obxectos será tan sinxelo como facer:

```
function OMeuObxecto(argumento1, ..., argumentoN)
{
    this.variabel1 = argumento1;
    ...
    this.variabelN = argumentoN;

    this.funcion1 = funcion1;
    ...
    this.funcionN = funcionN;
}
```

Polo tanto, **OMeuObxecto** é o nome do construtor da clase; **this** é un obxecto especial, que dirixe o obxecto actual que está sendo definido na declaración. Trátase dunha referencia ao obxecto actual, que chamamos **instancia**. Ao definir os métodos só pomos o nome, non os seus argumentos. A implementación de cada un destes métodos da clase farémola do mesmo xeito que a declaración dunha función normal:

```
function aMiñaFuncion([argumento1], ..., [argumentoN])
{
    // O que teña que facer a función
}
```

Cando os creamos, temos que usar un operador especial, **new**, seguido do construtor do obxecto, ao que lle pasamos como argumentos os argumentos cos que definimos a clase. Así,

```
var instancia1 = new OMeuObxecto(dato1, ..., datoN);
```

crearía unha instancia do obxecto **OMeuObxecto** que recibiría o nome de **instancia1**. Ao tela creada xa podemos usar os seus métodos ou modificar ou usar convenientemente os seus atributos.

Non hai lugar a confusión en canto a se o navegador sabe a que clase pertence o obxecto, pois non pode haber dúas clases que teñan unha función que se chame igual e faga cousas distintas; se nos despistamos e definimos dúas veces a mesma función para clases distintas, como non hai nada que distinga esas funcións, o navegador elixe tomar a última delas que se teña definido co mesmo nome. Témonos que asegurar de non usar o mesmo nome para métodos de dúas clases distintas, xa que JavaScript non incorpora **polimorfismo**.

Como exemplo de declaración de clases e instanciación de obxectos, vamos crear unha clase para obxectos de tipo **círculo** ao que lle pasamos o **nome** e o **radio** cando o iniciamos, e que teña métodos que nos calculen a súa **área** e a súa **lonxitude**, mostrándoos por pantalla.

A chamada que faremos en forma de evento no código HTML sería:

```
<body onLoad="xeometria();">
```

e o código JavaScript sería:

```
var PI = 3.141592;

/* Primeiro creamos a clase Circulo, á que lle pasamos o radio como parámetro ao instanciar */
function Circulo(nome, radio)
{
    this.nome = nome;
    this.radio = radio;
    this.area = area;
    this.perimetro = perimetro;
    this.imprimir = imprimir;
}

/* Definimos os métodos asociados aos obxectos de tipo Circulo */
function area()
{
    var area = PI * this.radio * this.radio;
    this.imprimir("A área do círculo " + this.nome + " é: " + area);
}

function perimetro()
{
    var perimetro = 2 * PI * this.radio;
    this.imprimir("O perímetro do círculo " + this.nome + " é: " + perimetro);
}

function imprimir(texto)
```

```
{
   alert(texto);
}

/* Xogo de probas para comprobar que o programa funciona */
function xeometria()
{
   var circulo = new Circulo("circulo pequeno", 1);
   circulo.area();
   circulo.perimetro();
}
```

## 2.8 Os obxectos da linguaxe

JavaScript proporciona unha libraría básica de obxectos que podemos instanciar e cos que podemos traballar directamente. Unha primeira clasificación do modelo de obxectos dividiríaos en dous grandes grupos: os relacionados co navegador (que poderemos usar e modificar tanto no ámbito HTML como no ámbito JavaScript) e os relacionados coa estrutura da linguaxe (que só poderemos usar e modificar no ámbito JavaScript).

Estes obxectos da linguaxe son: **String**, **Array**, **Math**, **Date**, **Boolean**, **Number** e **Function**.

### 2.8.1 Obxectos da linguaxe: String

Este obxecto permite manipular as cadeas, co gallo de que traballar con elas sexa máis sinxelo. Cando asignamos unha cadea a unha variábel, JavaScript está creando un obxecto de tipo **String** que é o que nos permite facer as manipulacións.

| String | |
|---|---|
| **ATRIBUTO** | **USO** |
| `length` | Valor numérico que indica a lonxitude en caracteres da cadea dada |
| `prototype` | Permite asignar novas propiedades ao obxecto |
| **MÉTODO** | **USO** |
| `charAt(índice)` | Devolve o carácter situado na posición especificada por ***índice*** |
| `indexOf(cadeaBuscada,índice)` | Devolve a posición da primeira aparición de ***cadeaBuscada*** dentro da cadea actual, a partires da posición dada por ***índice***. Este último argumento é opcional e, se o omitimos, a busca comeza polo primeiro carácter da cadea |
| `lastIndexOf(cadeaBuscada,índice)` | Devolve a posición da última aparición de ***cadeaBuscada*** dentro da cadea actual, a partires da posición dada por ***índice***, e buscando cara atrás. Este último argumento é opcional e, se se omite, a busca comeza polo último carácter da cadea |
| `link(URL)` | Convirte a cadea nun vínculo asignando ao atributo ***HREF*** o valor de ***URL*** |
| `split(separador)` | Parte a cadea nun ***array*** de caracteres. Se o carácter separador non se encontra, devolve un array cun só elemento que coincide coa cadea orixinal |
| `substring(primeiroIndice,segundoIndice)` | Devolve a subcadea que comeza na posición ***primeiroIndice + 1*** e que finaliza na posición ***segundoIndice***. Se ***primeiroIndice*** é maior que ***segundoIndice***, empeza por ***segundoIndice + 1*** e termina en ***primeiroIndice*** |
| `toLowerCase()` | Devolve a cadea en minúsculas |
| `toUpperCase()` | Devolve a cadea en maiúsculas |

```
function probarMetodos()
{
   var cadea = "Ola, compañeiros do mundo da informática", i;
   var cadeaAuxiliar = new Array();
   cadeaAuxiliar = cadea.split("o");
   with(document)
   {
      write("<p>A cadea a tratar é: " + cadea + "</p>");
      write("<p>Lonxitude da cadea: " + cadea.length + "</p>");
      write("<p>O 5º carácter é: " + cadea.charAt(5) + "</p>");
      write("<p>A subcadea <b>mun</b> está na posición: " + cadea.indexOf("mun") + "</p>");
      write("<p>O primeiro <b>a</b> está, empezando a contar de atrás a diante na posición: " + cadea.lastIndexOf("a") +
"</p>");
      write("<p>Convertida en ligazón: " + cadea.link("outraPaxina.html") + "</p>");
      write("<p>En minúsculas: " + cadea.toLowerCase() + "</p>");
      write("<p>En maiúsculas: " + cadea.toUpperCase() + "</p>");
      write("<p>Subcadea entre os caracteres 5 e 10: " + cadea.substring(4,10) + "</p>");
      write("<p>Subcadea entre os caracteres 20 e 5: " + cadea.substring(20,4) + "</p>");
      write("<p>Subcadeas resultantes de separar polo <b>o</b>:</p>");
      for(i=0; i < cadeaAuxiliar.length; i++)
      {
         write("<p>" + cadeaAuxiliar[i] + "</p>");
      }
   }
}
```

### 2.8.2 Obxectos da linguaxe: Array

Este obxecto vainos dar a facilidade de construír vectores que poden conter calquera tipo básico (non teñen por que ser todos do mesmo), e poderemos modificar a lonxitude do mesmo de forma dinámica sempre que engadamos un novo elemento. Para poder usar un obxecto **Array**, teremos que crealo mediante o seu construtor:

```
vector = new Array(5);
```

No exemplo, teremos creada unha variábel **vector** que conterá 5 elementos, numerados do 0 ao 4. Para acceder a cada elemento individual usaremos a notación **vector[i]**, onde **i** variará entre **0** e **N-1**, sendo **N** o número de elementos que lle pasamos ao construtor. Tamén poderiamos inicializar o vector á vez que o declaramos, pasándolle os valores que queiramos directamente ao construtor:

```
vector = new Array(21,"o anterior é un Number e o seguinte un Boolean, eu son un String",true);
```

Polo tanto, se pomos un argumento cando chamamos ao construtor, este será o número de elementos do vector (aos que lle deberemos asignar valores posteriormente), e se pomos máis dun, será a forma de iniciar o vector con tantos elementos como argumentos reciba o construtor. Temos que ter coidado coa inicialización dos vectores multidimensionais (como podería ser unha matriz), pois o seguinte código:

```
vector = new Array(2,3);
```

**NON** é unha matriz con 2 filas e 3 columnas, senón que é un vector que ten como valor do primeiro elemento 2 e como valor do segundo elemento 3. Para crear un vector bidimensional (unha matriz bidimensional é unha construción bastante frecuente), temos que crear un vector coas filas que queremos e, despois, cada elemento do vector iníciase cun vector coas columnas requiridas. Por exemplo, se queremos crear unha matriz con 4 filas e 5 columnas, chegará escribir:

```
matriz = new Array(4);
for(i = 0; i < 4; i++)
{
    matriz[i] = new Array(5);
}
```

Para nos referir ao elemento que ocupa a posición **(i,j)**, escribiremos **matriz[i][j]**.

| Array | |
|---|---|
| **ATRIBUTO** | **USO** |
| `length` | Valor numérico que indica o número de elementos do vector |
| `prototype` | Permite asignar novas propiedades ao obxecto |
| **MÉTODO** | **USO** |
| `join(separador)` | Une os elementos das cadeas de caracteres de cada elemento dun vector nun ***String***, separando cada cadea polo separador especificado |
| `reverse()` | Invirte a orde dos elementos do vector |
| `sort()` | Ordena os elementos do vector seguindo a orde lexicográfica |

```
function vectores()
{
   var matriz = new Array(2);
   var vector = new Array("Can", "Gato", "Bolboreta", "Vaca");
   matriz[0] = new Array(3);
   matriz[1] = new Array(2);
   matriz[0][0] = 0;
   matriz[0][1] = 1;
   matriz[0][2] = 2;
   matriz[1][0] = 3;
   matriz[1][1] = 4;
   matriz[1][2] = 5;

   document.write("<p>1ª fila da matriz: " + matriz[0][0] + " " + matriz[0][1] + " " + matriz[0][2] + "</p>");
   document.write("<p>2ª fila da matriz: " + matriz[1][0] + " " + matriz[1][1] + " " + matriz[1][2] + "</p>");
   document.write("<p>Antes de ordenar: " + vector.join(',') + "</p>");
   document.write("<p>Ordenados: " + vector.sort() + "</p>");
   document.write("<p>Ordenados en orde inversa: " + vector.sort().reverse());
}
```

### 2.8.3 Obxectos da linguaxe: Math

Este obxecto utilízase para poder realizar cálculos nos scripts. Os seus atributos non se poden modificar, só se poden consultar. Estes atributos son constantes matemáticas de uso frecuente nalgunhas tarefas, por eso é lóxico que só se poida consultar o seu valor pero non modificalo.

Para poder usar algún dos atributos ou dos métodos deste obxecto, loxicamente, teremos que antepor a palabra **Math**; por exemplo, teremos que escribir **Math.sin(Math.PI)**, se queremos calcular o seno de **PI**.

| Math | |
|---|---|
| **ATRIBUTO** | **USO** |
| `E` | Número ***e***, base dos logaritmos naturais (neperianos) |
| `PI` | Número ***PI*** |
| **MÉTODO** | **USO** |
| `abs(número)` | Función valor absoluto |
| `ceil(número)` | Devolve o enteiro superior ao redondear ***número*** |
| `cos(número)` | Devolve o coseno de ***número*** (que debe estar en radiáns) ou ***NaN*** |
| `floor(número)` | Devolve o enteiro inferior ao redondear ***número*** |
| `log(número)` | Devolve o logaritmo neperiano de ***número*** |
| `max(x,y)` | Devolve o máximo de ***x*** e ***y*** |
| `min(x,y)` | Devolve o mínimo de ***x*** e ***y*** |
| `pow(base,exp)` | Calcula a potencia $base^{exp}$ |
| `random()` | Devolve un número pseudoaleatorio entre ***0*** e ***1*** |
| `round(número)` | Redondea ***número*** ao enteiro máis cercano |
| `sin(número)` | Devolve o seno de ***número*** (que debe estar en radiáns) ou ***NaN*** |
| `sqrt(número)` | Devolve a raíz cadrada de ***número*** |
| `tan(número)` | Devolve a tanxente de ***número*** (que debe estar en radiáns) ou ***NaN*** |

### 2.8.4 Obxectos da linguaxe: Date

Permite facer manipulacións coas datas: pór datas, consultalas... sempre tendo en conta que JavaScript manexa:

- as **datas** en milisegundos
- os **meses** de Xaneiro a Decembro están representados por un enteiro que varía entre 0 e 11 (é dicir, o mes 0 é Xaneiro, o mes 1 é Febreiro, … , o mes 11 é Decembro)
- os **días da semana** ordénanse co formato anglosaxón, de Domingo a Sábado, e están representados por un enteiro que varía entre 0 e 6 (o día 0 é o Domingo, o día 1 é o Luns...)
- os **anos** ponse con catro díxitos
- as **horas** no formato **HH:MM:SS**

Podemos crear un obxecto **Date** baleiro, ou podémolo crear dándolle unha data concreta. Se non lle damos unha data concreta, crearase coa data correspondente ao momento no que se crea.

```
var data = new Date(ano, mes);
var data = new Date(ano, mes, dia);
var data = new Date(ano, mes, dia, horas);
var data = new Date(ano, mes, dia, horas, minutos);
var data = new Date(ano, mes, dia, horas, minutos, segundos);
```

Recordade que cada variábel ten que estar definida con anterioridade e que para que non dea problemas a inicialización do obxecto **Date** temos que enviar datos coherentes: o día non poden ser o 33, nin tampouco o 31 no mes de Xuño. Todos os valores que pasamos ao constructor teñen que ser enteiros.

| Date |
|---|

| MÉTODO | USO |
|---|---|
| `getDate()` | Devolve o día do mes actual coma un enteiro entre ***1*** e ***31*** |
| `getDay()` | Devolve o día da semana actual coma un enteiro entre ***0*** e ***6*** |
| `getHours()` | Devolve a hora do día actual coma un enteiro entre ***0*** e ***23*** |
| `getMinutes()` | Devolve os minutos da hora actual coma un enteiro entre ***0*** e ***59*** |
| `getMonth()` | Devolve o mes do ano actual coma un enteiro entre ***0*** e ***11*** |
| `getSeconds()` | Devolve os segundos do minuto actual coma un enteiro entre ***0*** e ***59*** |
| `getTime()` | Devolve o tempo transcorrido en milisegundos dende o 1 de Xaneiro de 1970 ata o intre actual |
| `getYear()` | Devolve o ano actual coma enteiro |
| `setDate(díaMes)` | Pon o día do mes actual no obxecto ***Date*** que estamos usando |
| `setDay(díaSemana)` | Pon o día da semana actual no obxecto ***Date*** que estamos usando |
| `setHours(horas)` | Pon a hora do día actual no obxecto ***Date*** que estamos usando |
| `setMinutes(minutos)` | Pon os minutos da hora actual no obxecto ***Date*** que estamos usando |
| `setMonth(mes)` | Pon o mes do ano actual no obxecto ***Date*** que estamos usando |
| `setSeconds(segundos)` | Pon os segundos do minuto actual no obxecto ***Date*** que estamos usando |
| `setTime(milisegundos)` | Pon a data que dista dos milisegundos que lle pasemos do 1 de Xaneiro de 1970 no obxecto ***Date*** que estamos usando |
| `setYear(año)` | Pon o ano actual no obxecto ***Date*** que estamos usando |
| `toGMTString()` | Devolve unha cadea que usa as convencións de Internet coa zona horaria GMT |

### 2.8.5 Obxectos da linguaxe: Number

Este obxecto representa os números. Podémoslle asignar a unha variábel un número, ou darlle un valor, mediante o constructor **Number**:

```
numero = new Number(4.5);      // iniciamos numero co valor 4.5
```

Se non lle pasamos ningún valor ao constructor, a variábel iníciase co valor 0.

| Number |
|---|

| ATRIBUTO | USO |
|---|---|
| `E` | Número ***e***, base dos logaritmos naturais (neperianos) |
| `PI` | Número ***PI*** |
| `MAX_VALUE` | Valor máximo que se pode manexar cos tipos numéricos |
| `MIN_VALUE` | Valor mínimo que se pode manexar cos tipos numéricos |
| `NaN` | Representación dun dato que non é un número |
| `NEGATIVE_INFINITY` | Valor a partir do que hai desbordamento negativo (underflow) |
| `POSITIVE_INFINITY` | Valor a partir do que hai desbordamento positivo (overflow) |
| **MÉTODO** | **USO** |
| `abs(número)` | Función valor absoluto |
| `ceil(número)` | Devolve o enteiro superior ao redondear ***número*** |
| `cos(número)` | Devolve o coseno de ***número*** (que debe estar en radiáns) ou ***NaN*** |
| `floor(número)` | Devolve o enteiro inferior ao redondear ***número*** |
| `log(número)` | Devolve o logaritmo neperiano de ***número*** |
| `max(x,y)` | Devolve o máximo de ***x*** e ***y*** |
| `min(x,y)` | Devolve o mínimo de ***x*** e ***y*** |
| `pow(base,exp)` | Calcula a potencia $base^{exp}$ |
| `random()` | Devolve un número pseudoaleatorio entre ***0*** e ***1*** |
| `round(número)` | Redondea ***número*** ao enteiro máis cercano |
| `sin(número)` | Devolve o seno de ***número*** (que debe estar en radiáns) ou ***NaN*** |
| `sqrt(número)` | Devolve a raíz cadrada de ***número*** |
| `tan(número)` | Devolve a tanxente de ***número*** (que debe estar en radiáns) ou ***NaN*** |

Para consultar o valor dos atributos **MAX_VALUE**, **MIN_VALUE**, **NaN**, **NEGATIVE_INFINITY** e **POSITIVE_INFINITY** non podemos facer:

```
a = new Number();
alert(a.MAX_VALUE);
```

porque JavaScript devolverá **`undefined`**; temos que o facer directamente sobre **Number**, é dicir, teremos que consultar os valores **`Number.MAX_VALUE`**, **`Number.MIN_VALUE`**, ...

### 2.8.6 Obxectos da linguaxe: Boolean

Este obxecto permítenos crear un tipo de dato que só permite dous valores: **certo** ou **falso**, tomando os valores **true** ou **false**.

```
valor = new Boolean();                      // asigna false a valor
valor = new Boolean(0);                     // asigna false a valor
valor = new Boolean("");                    // asigna false a valor
valor = new Boolean(false);                 // asigna false a valor
valor = new Boolean(numero_distinto_de_0);  // asigna true a valor
valor = new Boolean(true);                  // asigna true a valor
```

### 2.8.7 Obxectos da linguaxe: Function

É un obxecto que xa temos usado con anterioridade, cando estudiamos a declaración das funcións. Proporciona o atributo **arguments** que é un **Array** cos parámetros que pasamos á función cando a chamamos. Polo feito de ser un **Array**, conta con todos os atributos e métodos de este tipo de obxecto.

### 2.8.8 Consideracións finais

Estes obxectos propios da linguaxe son (salvo **Math** e **Date**), precisamente, os **tipos de datos** cos que conta JavaScript. Verdadeiramente, o que sucede cando crea unha variábel é que en canto sabe de que tipo é, **crea** un obxecto dese tipo para esa variábel; por iso, se temos unha variábel que é unha cadea, automaticamente podemos usar os métodos do obxecto **String**. En canto lle cambiamos o tipo (por exemplo, metemos na variábel un número), destrúe o obxecto inicial e crea outro

do novo tipo (se é un número, entón sería un obxecto de tipo **Number**), e xa podemos usar os atributos e métodos dese obxecto.

## 2.9 Os obxectos do navegador

### 2.9.1 Introdución

Neste capítulo vamos a estudiar a xerarquía que presentan os obxectos do navegador, atendendo a unha relación **contedor - contido** que temos entre eles.

Segundo esta xerarquía, podemos entender o obxecto **`area`** coma un obxecto dentro do obxecto **document** que a súa vez está dentro do obxecto **window**.

```
window.document.area
```

Por exemplo, se queremos facer referenza a un campo dun formulario, teremos que escribir

```
xanela.documento.formulario.campo
```

onde **xanela** é o nome do obxecto **window**, **documento** o do obxecto **document**, **formulario** o do obxecto **forms** (é un vector) e **campo** é o nome do obxecto **input**. Na meirande parte dos casos podemos ignorar a referenza á ventá actual, pero cómpre usala cando estamos utilizando múltiples ventás ou marcos.

Tamén podemos utilizar a notación vectorial para nos referir aos obxectos (normalmente cómpre facelo cando estes obxectos non teñen nome). O exemplo seguinte fai referenza ao segundo elemento do primeiro formulario do documento activo; este elemento será o segundo que creamos secuencialmente no código HTML.

```
document.forms[0].elements[1];
```

### 2.9.2 Métodos destacados

| Window | |
|---|---|
| **MÉTODO** | **USO** |
| **`alert(mensaxe)`** | Mostra a **mensaxe** nunha caixa de diálogo |
| `confirm(mensaxe)` | Mostra unha caixa de diálogo coa **mensaxe** e dous botóns, un de **Aceptar** e outro de **Cancelar**. Devolve ***true*** se prememos **Aceptar** e ***false*** se prememos **Cancelar** |
| **`prompt(mensaxe [,resposta])`** | Mostra unha caixa de diálogo que contén unha caixa de texto na que poderemos escribir unha resposta á **mensaxe**. O parámetro **resposta** é opcional, e mostrará a resposta por defecto indicada cando se abra a caixa de diálogo. Este método devolve unha cadea de caracteres coa resposta introducida |

### 2.9.3 Eventos en JavaScript

Un evento non ven sendo máis que algo que acontece para o que queremos definir unha resposta. Por exemplo, cando prememos o rato. Nas primeiras versións de JavaScript existía un conxunto mínimo de eventos para poder dar interactividade ás páxinas web. A partir da versión 1.2 isto cambiou para dar lugar a un modelo máis sofisticado.

Existen etiquetas HTML que poden xerar eventos. Por exemplo, se temos unha ligazón podemos pasar sobor dela ou podemos premer nela co rato. A estas posíbeis situacións déronselle nomes e asociáronse ás etiquetas que os podían xerar de xeito que chamaran a funcións JavaScript previamente implementadas.

Cando creamos un manexador para o evento estamos asignando ao obxecto unha propiedade que toma o nome dese manexador, e esa propiedade é a que nos permite acceder ao manexador do evento do obxecto, permitíndonos cambiar o código asociado ao evento do obxecto. Por exemplo, se queremos crear unha páxina na que pase algo ao premer nun botón e que pase outra cousa cando volvamos a premer implica que de algún modo temos que ser capaces de cambiar o código asociado ao evento para que faga cousas distintas cada vez. No ficheiro HTML teriamos:

```
<html>
<head>
   <script language="JavaScript" src="cambiar.js"></script>
</head>
<body>
   <form name="formulario">
      <input type="button" name="boton" value="Prémeme" onClick="botonPremido();">
   </form>
</body>
```

```
</html>
```

E no ficheiro **cambiar.js** que contería o código JavaScript:

```
var nome;
var vez = 0;

function botonPremido()
{
   if (vez==0)
   {
      nome = prompt("Como te chamas?", "Nome");
      vez++;
   }
   else
   {
      alert("Ola " + nome + ", encantado de volver a verte :-)");
   }
}
```

Se quixeramos poderiamos asignar outra función ao manexador do evento sen mais que pór **document.formulario.boton.onclick = outraFuncion;** dentro do código co que traballamos.

**EVENTOS**

| MÉTODOS | O EVENTO OCORRE CANDO... |
|---|---|
| **onabort** | a carga dunha imaxe é interrompida |
| onblur | o elemento perde o foco |
| **onchange** | o contido dun campo cambia |
| **onclick** | o rato preme nun obxecto |
| **ondblclick** | o rato preme dúas veces nun obxecto |
| **onerror** | un erro acontece cando cargamos un ficheiro ou uha imaxe |
| **onfocus** | un elemento consigue o foco |
| **onkeydown** | prememos unha tecla |
| **onkeypress** | prememos ou mantemos premida unha tecla |
| **onkeyup** | soltamos unha tecla |
| **onload** | unha páxina ou unha imaxe se acaba de cargar |
| **onmousedown** | prememos calquera botón do rato |
| **onmousemove** | movemos o rato |
| **onmouseout** | movemos o rato fóra dun elemento |
| **onmouseover** | pasamos por riba dun elemento co rato |
| **onmouseup** | soltamos un botón do rato |
| **onreset** | prememos o botón de reinicio |
| **onresize** | cambiamos o tamaño dunha xanela ou dun marco |
| **onselect** | seleccionamos texto |
| **onsubmit** | prememos o botón de envío dun formulario |
| **onunload** | o usuario sae da páxina |
| **ATRIBUTOS** | **DESCRICIÓN** |
| **altKey** | devolve se a tecla ***Alt*** estaba premida ou non cando se xera o evento |
| **button** | devolve que botón do rato estaba premido cando se xera o evento |
| **clientX** | devolve a coordenada horizontal do rato cando se xera o evento |
| **clientY** | devolve a coordenada vertical do rato cando se xera o evento |
| **ctrlKey** | devolve se a tecla ***Ctrl*** estaba premida ou non cando se xera o evento |
| **metaKey** | devolve se unha ***metatecla*** estaba premida ou non cando se xera o evento |
| **relatedTarget** | devolve o elemento relacionado co elemento que xerou o evento |
| **screenX** | devolve a coordenada horizontal do rato cando se xera o evento |
| **screenY** | devolve a coordenada vertical do rato cando se xera o evento |
| **shiftKey** | devolve se a tecla ***Maiúsculas*** estaba premida ou non cando se xera o evento |

O evento **onerror** dáse cando hai un erro de sintaxe JavaScript non cando hai un erro semántico que provoca unha resposta inesperada do navegador. Por exemplo, se tentamos cargar unha páxina que non existe, non se dará este evento.

Se o manexador devolve **false**, a acción por defecto do obxecto cancélase do seguinte xeito:

- Os **botóns** non teñen acción por defecto: non pasa nada.
- **Radio button** e **checkbox**: non se marcan.
- Botón **submit**: non se envía o formulario.
- Botón **reset**: non se borra o formulario.

## 2.10 Exemplos resoltos

Na actualidade estes sinxelos exemplos están xa desenvolvidos como funcións e non cómpre programalos, pero seguen a ser fonte de ideas á hora de resolver outros enunciados.

### 2.10.1 Exemplo 1: Efecto roll-over

Denomínase roll-over ao efecto de facer que unha imaxe cambie por outra cando pasa por riba dela o rato. A idea para conseguir este efecto é cambiar o atributo **src** da imaxe polo da nova cando pasa o rato por riba dela, e volver a colocar a inicial cando o rato rematou de pasar. Inicialmente foi un grave problema pois a etiqueta **<img>** nas súas orixes non admintía os eventos **onMouseOver** e **onMouseOut** polo que algo tan sinxelo como **<img onMouseOver="CambiarImaxe(); return true;">** non era posíbel.

A solución que se usaba era embeber a etiqueta **<img>** noutra que si admitise estes eventos como podía ser a áncora **<a>** e darlle á propia imaxe o atributo **border="0"** para que non aparentara ser unha ligazón (aínda que internamente si que o era). A maiores, precargábanse ambas imaxes para lograr un efecto fluído.

```
<html>
<head>
   <title>-:: EXEMPLO 1: Efecto RollOver ::-</title>
   <script language="JavaScript" src="rollover.js"></script>
</head>
<body onLoad="precargarImaxes();">
   <a href="#" onMouseOver="rollOver(true);" onMouseOut="rollOver(false);">
      <img src="correcto.png" id="imaxe" name="imaxe" border="0">
   </a>
</body>
</html>
```

```
//ficheiro rollover.js
function rollOver(estado)
{
   if(!estado) { document.imaxe.src = "correcto.png"; }
   else { document.imaxe.src = "incorrecto.png"; }
}

function precargarImaxes()
{
   imaxe = new Image();
   imaxe.src = "incorrecto.png";
}
```

Xa hai moito tempo que intercambiar imaxes é tan sinxelo como facer:

```
<img src="correcto.png" onMouseOver="this.src='incorrecto.png'" onMouseOut="this.src='correcto.png'">
```

### 2.10.2 Exemplo 2: Validar formularios

Está claro que podemos construír formularios para recoller información, pero... como comprobamos que eses datos son correctos?. É o momento de ver como validar os datos introducidos nun formulario. Faise para non recargar ao servidor de peticións: cando enviamos un formulario, se os datos non son correctos, o servidor terá que nos reenviar a solicitude do formulario pedíndonos os datos correctos.

```
<html>
   <head>
      <title>-:: EXEMPLO 2: Validar formulario ::-</title>
      <link rel='stylesheet' type='text/css' href='formulario.css'/>
      <script language='javascript' src='formulario.js'></script>
   </head>
```

```
<body class='folla'>
   <div class='verde'><p class='titulo'>CREAR USUARIO</p></div>
   <div class='branco'>
      <form action='irAOutraPaxina.html' enctype='text/plain' method='get' name='formulario' id='formulario'>
      <div class='fila'>
         <p class='texto'><sup>*</sup>Nome: </p>
         <p class='campo'><input id='nome' name='nome' type='text'/></p>
         <p class='texto'><sup>*</sup>Alcume: </p>
         <p class='campo'><input id='alcume' name='alcume' type='text'/></p>
      </div>
      <div class='fila'>
         <div><p class='texto'><sup>*</sup>1º Apelido: </p></div>
         <div><p class='campo'><input id='apelido1' name='apelido1' type='text'/></p></div>
         <div><p class='texto'><sup>*</sup>Contrasinal: </p></div>
         <div><p class='campo'><input id='contrasinal' name='contrasinal' type='password'/></p></div>
      </div>
      <div class='fila'>
         <div><p class='texto'>2º Apelido: </p></div>
         <div><p class='campo'><input id='apelido2' name='apelido2' type='text'/></p></div>
         <div><p class='texto'><sup>*</sup>Repite o contrasinal: </p></div>
         <div><p class='campo'><input id='repetir' name='repetir' type='password'/></p></div>
      </div>
      <div class='fila'>
         <div><p class='texto'><sup>*</sup>Correo electrónico: </p></div>
         <div><p class='campo'><input id='email' name='email' type='text'/></p></div>
         <div><p class='texto'>Páxina web: </p></div>
         <div><p class='campo'><input id='web' name='web' type='text'/></p></div>
      </div>
      <div class='fila'>
         <div><p class='texto'>Data: </p></div>
         <div><p class='campo'>
               <select name='dia' id='dia'>
                  <option value='1'>1</option>
                  <option value='2'>2</option>
                  <option value='3'>3</option>
                  <option value='4'>4</option>
                  <option value='5'>5</option>
                  <option value='6'>6</option>
                  <option value='7'>7</option>
                  <option value='8'>8</option>
                  <option value='9'>9</option>
                  <option value='10'>10</option>
                  <option value='11'>11</option>
                  <option value='12'>12</option>
                  <option value='13'>13</option>
                  <option value='14'>14</option>
                  <option value='15'>15</option>
                  <option value='16'>16</option>
                  <option value='17'>17</option>
                  <option value='18'>18</option>
                  <option value='19'>19</option>
                  <option value='20'>20</option>
                  <option value='21'>21</option>
                  <option value='22'>22</option>
                  <option value='23'>23</option>
                  <option value='24'>24</option>
                  <option value='25'>25</option>
                  <option value='26'>26</option>
                  <option value='27'selected='selected'>27</option>
                  <option value='28'>28</option>
                  <option value='29'>29</option>
                  <option value='30'>30</option>
                  <option value='31'>31</option>
               </select>
               de
               <select name='mes' id='mes'>
                  <option value='1'>Xaneiro</option>
                  <option value='2' selected='selected'>Febreiro</option>
                  <option value='3'>Marzo</option>
                  <option value='4'>Abril</option>
                  <option value='5'>Maio</option>
                  <option value='6'>Xuño</option>
                  <option value='7'>Xullo</option>
                  <option value='8'>Agosto</option>
                  <option value='9'>Setembro</option>
                  <option value='10'>Outubro</option>
                  <option value='11'>Novembro</option>
                  <option value='12'>Decembro</option>
               </select>
               de
               <select name='ano' id='ano'>
                  <option value='2007'>2007</option>
                  <option value='2008' selected='selected'>2008</option>
                  <option value='2009'>2009</option>
                  <option value='2010'>2010</option>
               </select>
            </p>
```

```
            </div>
            <div><p class='texto'>Sexo: </p></div>
            <div><p class='campo'><input type='radio' id='home' name='sexo' value='home' checked='true'>Home
                                  <input type='radio' id='muller' name='sexo' value='muller'>Muller</p></div>
         </div>
         <div class='fila'>
            <p class='textoBoton'>
               <input id='borrar' name='borrar' type='reset' value='Borrar'/>
               <input id='enviar' name='enviar' type='submit' value='Enviar' onClick='return validar(formulario);'/>
            </p>
         </div>
         </form>
         <div class="nota">
            <span class="negra">NOTA:</span> Os campos marcados con * son obrigatorios, e non serás dado de alta se os
deixas baleiros. O resto de campos son optativos e non cómpre enchelos
         </div>
      </div>
   </body>
</html>
```

```
/* ficheiro formulario.css */
.folla
{
height:100%;
width:100%;
margin:0px 0px 0px 0px;
}
.verde
{
float:left;
height:20%;
width:100%;
background-color:green;
}
.branco
{
float:left;
height:80%;
width:100%;
background-color:white;
}
.fila
{
float:left;
width:100%;
}
.texto
{
float:left;
text-align:right;
width:25%;
font-size:large;
font-family:Verdana, Arial, Helvetica, sans-serif;
font-weight:bold;
vertical-align:middle;
}
.titulo
{
text-align:center;
width:100%;
font-size:x-large;
font-family:Verdana, Arial, Helvetica, sans-serif;
font-weight:bold;
vertical-align:middle;
}
.campo
{
float:left;
text-align:left;
width:25%;
font-size:large;
font-family:Verdana, Arial, Helvetica, sans-serif;
font-weight:normal;
}
.textoBoton
{
float:left;
text-align:center;
width:100%;
font-size:large;
font-family:Verdana, Arial, Helvetica, sans-serif;
font-weight:bold;
vertical-align:middle;
}
.nota {
```

```
font-family:verdana,sans-serif;
text-align:center;
font-size:x-small;
}
.negra {
font-weight:bold;
}
```

Agora temos que analizar as necesidades deste formulario:

- como temos marcadas algunhas celas coma introdución obrigatoria de datos, eses campos non poden quedar baleiros polo que haberá que comprobar que a propiedade **value** deses obxectos sexa distinta da cadea baleira.
- o enderezo de correo electrónico debe ter polo menos unha @ e non pode ser o primeiro carácter. Á vez podemos comprobar que non existan outros caracteres erróneos no enderezo.
- os contrasinais non poden ser a cadea baleira e ademais deben coincidir.
- debemos comprobar que a data de nacemento é válida.

```
//ficheiro formulario.js
function validar(formulario)
{
   nome = formulario.nome.value;
   apelido1 = formulario.apelido1.value;
   email = formulario.email.value;
   alcume = formulario.alcume.value;
   contrasinal = formulario.contrasinal.value;
   repetir = formulario.repetir.value;
   dia = formulario.dia.value;
   mes = formulario.mes.value;
   ano = formulario.ano.value;

   return ( validarNonBaleiro(nome,"nome") && validarNonBaleiro(apelido1,"1° apelido") && validarNonBaleiro(email,"correo
electrónico") && validarNonBaleiro(alcume,"alcume") && validarNonBaleiro(contrasinal,"contrasinal") &&
validarNonBaleiro(repetir,"duplicado do contrasinal") && validarEmail(email) && validarContrasinal(contrasinal,repetir)
&& validarData(dia,mes,ano));
}

function validarNonBaleiro(campo, texto)
{
   if (campo == "") {  alert("O campo que contén o " + texto + " está baleiro"); return false; }
   else { return true; }
}

function validarEmail(email)
{
   if(email.indexOf('@',0) <= 0 || email.indexOf(' ',0) != -1 || email.indexOf('/',0) != -1 ||
      email.indexOf(';',0) != -1 || email.indexOf('<',0) != -1 || email.indexOf('>',0) != -1 ||
      email.indexOf('*',0) != -1 || email.indexOf('|',0) != -1 || email.indexOf('`',0) != -1 ||
      email.indexOf('&',0) != -1 || email.indexOf('$',0) != -1 || email.indexOf('!',0) != -1 ||
      email.indexOf('"',0) != -1 || email.indexOf(':',0) != -1 )
      { alert("Enderezo de correo electrónico non válido: " + email); return false; }
   else { return true; }
}

function validarContrasinal(contrasinal, repetir)
{
   if ( contrasinal != repetir ) { alert("Os contrasinais non coinciden"); return false; }
   else { return true; }
}

function validarData(dia,mes,ano)
{
   if((mes == 4 || mes == 6 || mes == 9 || mes == 11) && dia == 31) { alert("A data é incorrecta"); return false; }
   else { if(mes == 2 && (dia > 29 || (dia == 29 && (ano % 400 != 0) && ((ano % 4 != 0) || (ano % 100 == 0)))))
          { alert("A data é incorrecta"); return false; }
          else { return true; }
        }
}
```

## 2.10.3 Exemplo 3: contador

Consiste nun efecto moi sinxelo usado para anunciar cantos días faltan para que suceda algún acontecemento. Para poder facer este efecto, debemos manexar o obxecto **Date**. Declararemos unha variable que conteña a data do acontecemento e outra coa data actual. A resta será o tempo que falta para a data sinalada, que teremos que pasar a días, pois o ordenador traballa en milisegundos.

```
<html>
<head>
   <title>-:: EXEMPLO 3: Contador para un acontecemento ::-</title>
```

```
    <script language="JavaScript" src="contador.js"></script>
</head>
<body onLoad="contador();">
</body>
</html>
```

```
// ficheiro contador.js
function contador()
{
    var data, hoxe, faltan;
    data = new Date(2010, 3, 27);     // Por exemplo o 27 de Marzo de 2010
    hoxe = new Date();
    faltan = (data - hoxe)/86400000;  // 86400000 milisegundos = 1000 milisegundos x 60 segundos x 60 minutos x 24 horas
    alert("Faltan " + Math.round(faltan) + " días para o exame de Ciclo Superior :-)");
}
```

### 2.10.4 Exemplo 4: xerador de táboas

A partir dun formulario imos xerar o código HTML necesario para crear unha táboa cos datos que introducimos no formulario. En primeiro lugar vamos a decidir que datos solicitar: número de filas e de columnas, ancho da táboa, tipo de borde e permitiremos a escolla entre pixels e porcentaxe.

```
<html>
<head>
    <title>-:: EXEMPLO 4: Xerador de código HTML para táboas ::-</title>
    <link rel='stylesheet' type='text/css' href='xerador.css'/>
    <script language="JavaScript" src="xerador.js"></script>
</head>
<body class='folla'>
    <div class='verde'><p class='titulo'>XERADOR DE CÓDIGO HTML PARA TÁBOAS</p></div>
    <div class='branco'>
        <form name='formulario' id='formulario'>
        <div class='fila'>
            <p class='texto'>Número de filas: </p>
            <p class='campo'><input id='numFilas' name='numFilas' type='text' size='2'/></p>
            <p class='texto'>Número de columnas: </p>
            <p class='campo'><input id='numColumnas' name='numColumnas' type='text' id='2'/></p>
        </div>
        <div class='fila'>
            <p class='texto'>Ancho da táboa: </p>
            <p class='campo'><input id='ancho' name='ancho' type='text' size='4'/>
                        <input type='radio' id='pixel' name='tipo' value='0' checked='true'>Pixels
                        <input type='radio' id='porcentaxe' name='tipo' value='1'>Porcentaxe</p>
            <p class='texto'>Tipo de borde: </p>
            <p class='campo'><select name='borde' id='borde'>
                            <option value='thin'>Estreito</option>
                            <option value='medium' selected='selected'>Normal</option>
                            <option value='dotted'>Punteado</option>
                            <option value='double'>Dobre</option>
                            <option value='solid'>Groso</option>
                            <option value='none'>Sen borde</option>
                        </select></p>
        </div>
        <div class='fila'>
            <p class='textoBoton'>
                <input id='borrar' name='borrar' type='reset' value='Borrar datos'/>
                <input id='crear' name='crear' type='button' value='Crear táboa' onClick='return crearTaboa();'/>
            </p>
        </div>
        <div class='fila'>
            <p id='resultado' name='resultado'></p>
        </div>
        </form>
    </div>
</body>
</html>
```

```
/* ficheiro xerador.css */
.folla
{
height:100%;
width:100%;
margin:0px 0px 0px 0px;
}
.verde
{
float:left;
height:20%;
width:100%;
```

```
background-color:green;
}
.branco
{
float:left;
height:80%;
width:100%;
background-color:white;
}
.fila
{
float:left;
width:100%;
}
.texto
{
float:left;
text-align:right;
width:25%;
font-size:large;
font-family:Verdana, Arial, Helvetica, sans-serif;
font-weight:bold;
vertical-align:middle;
}
.titulo
{
text-align:center;
width:100%;
font-size:x-large;
font-family:Verdana, Arial, Helvetica, sans-serif;
font-weight:bold;
vertical-align:middle;
}
.campo
{
float:left;
text-align:left;
width:25%;
font-size:large;
font-family:Verdana, Arial, Helvetica, sans-serif;
font-weight:normal;
}
.textoBoton
{
float:left;
text-align:center;
width:100%;
font-size:large;
font-family:Verdana, Arial, Helvetica, sans-serif;
font-weight:bold;
vertical-align:middle;
}
```

```
//ficheiro xerador.js
function crearTaboa()
{
    numFilas = parseInt(numFilas.value);
    numColumnas = parseInt(numColumnas.value);
    ancho = parseInt(ancho.value);
    borde = borde.value;

    if(porcentaxe.checked) { tipo = '%'; }
    else { tipo = ''; }

    taboa = '<table border="' + borde + '" width="' + ancho + tipo + '">\n';

    for(i=1;i<=numFilas;i++)
    {
        taboa += '<tr>\n';
        for(j=1;j<=numColumnas;j++)
        {
            taboa += '<td> </td>\n';
        }
        taboa += '</tr>\n';
    }

    taboa += '</table>\n';
    resultado.textContent = taboa;
}
```

## 2.10.5 Exemplo 5: marcar todas as celas de verificación

Vamos a marcar todas as celas de verificación que hai nun formulario dentro da páxina.

```
<html>
<head>
   <title>-:: EXEMPLO 5: Marcar todas as celas de verificación ::-</title>
   <link rel='stylesheet' type='text/css' href='marcar.css'/>
   <script language="JavaScript" src="marcar.js"></script>
</head>
<body class='folla'>
   <div class='verde'><p class='titulo'>MARCAR CELAS DE VERIFICACIÓN</p></div>
   <div class='branco'>
      <form name='formulario' id='formulario'>
      <div class='fila'>
         <p class='texto'>Que cores te gustan?: </p>
         <p class='campo'><input type='checkbox' id='corAzul' name='corAzul' value='azul'>Azul
                          <input type='checkbox' id='corAmarelo' name='corAmarelo' value='amarelo'>Amarelo
                          <input type='checkbox' id='corVermello' name='corVermello' value='vermello'>Vermello
                          <input type='checkbox' id='corVerde' name='corVerde' value='verde'>Verde
                          <input type='checkbox' id='corLaranxa' name='corLaranxa' value='laranxa'>Laranxa
                          <input type='checkbox' id='corNegro' name='corNegro' value='negro'>Negro
                          <input type='checkbox' id='corBranco' name='corBranco' value='branco'>Branco</p>
      </div>
      <div class='fila'>
         <p class='texto'>Que deportes te gustan?: </p>
         <p class='campo'><input type='checkbox' id='depBalonman' name='depBalonman' value='balonman'>Balonmán
                          <input type='checkbox' id='depBasket' name='depBasket' value='basket'>Baloncesto
                          <input type='checkbox' id='depFutbol' name='depFutbol' value='futbol'>Fútbol
                          <input type='checkbox' id='depNatacion' name='depNatacion' value='natacion'>Natación</p>
      </div>
      <div class='fila'>
         <p class='textoBoton'>
            <input id='cores' name='cores' type='button' value='Elixir todas as cores' onClick='marcar("cor");'/>
            <input id='deportes' name='deportes' type='button' value='Elixir todos os deportes'
onClick='marcar("dep");'/>
         </p>
      </div>
   </form>
   </div>
</body>
</html>
```

```
/* ficheiro marcar.css */
.folla
{
height:100%;
width:100%;
margin:0px 0px 0px 0px;
}
.verde
{
float:left;
height:20%;
width:100%;
background-color:green;
}
.branco
{
float:left;
height:80%;
width:100%;
background-color:white;
}
.fila
{
float:left;
width:100%;
}
.texto
{
float:left;
text-align:right;
width:25%;
font-size:large;
font-family:Verdana, Arial, Helvetica, sans-serif;
font-weight:bold;
vertical-align:middle;
}
.titulo
{
text-align:center;
width:100%;
font-size:x-large;
font-family:Verdana, Arial, Helvetica, sans-serif;
font-weight:bold;
vertical-align:middle;
}
.campo
```

```
{
float:left;
text-align:left;
width:75%;
font-size:large;
font-family:Verdana, Arial, Helvetica, sans-serif;
font-weight:normal;
}
.textoBoton
{
float:left;
text-align:center;
width:100%;
font-size:large;
font-family:Verdana, Arial, Helvetica, sans-serif;
font-weight:bold;
vertical-align:middle;
}
```

```
//ficheiro marcar.js
function marcar(que)
{
   for (i=0;i<document.formulario.elements.length;i++)
   {
       if(document.formulario.elements[i].type == "checkbox" && document.formulario.elements[i].id.substring(0,3) == que)
       {
          document.formulario.elements[i].checked = true;
       }
   }
}
```

### 2.10.6 Exemplo 6: Enmarcar unha imaxe

Vamos a enmarcar unha imaxe cando pasamos por riba dela co rato. Para que isto funcione temos que combinar dous elementos: un estilo CSS e unha función que sexa quen de cambiar ese estilo. O estilo CSS será o encargado de pór o borde á imaxe.

```
<html>
<head>
   <title>-:: EXEMPLO 6: Pór un marco a unha imaxe ::-</title>
   <link rel='stylesheet' type='text/css' href='enmarcar.css'/>
   <script language="JavaScript" src="enmarcar.js"></script>
</head>
<body>
   <img id="imaxe" name="imaxe" src="correcto.png" class="imaxeBorde" onMouseOver="porBorde(this,'maroon');"
onMouseOut="quitarBorde(this);">
</body>
</html>
```

```
/* ficheiro enmarcar.css */
.imaxeBorde
{
border:0px none white;
}
```

```
// ficheiro enmarcar.js
function porBorde(imaxe, cor)
{
   imaxe.style.border = '10px solid' + cor;
}
function quitarBorde(imaxe)
{
   imaxe.style.border = '';
}
```

Este efecto non é só aplicábel a imaxes, podémolo aplicar a calquera elemento que queiramos, sempre que teña o borde entre as súas propiedades CSS.

## 3 Etiquetas HTML/XHTML e DTDs válidos

A seguinte táboa mostra todas as etiquetas HTML/XHTML, e indica en que DTD aparece cada una delas.

| Etiqueta | HTML 4.01 / XHTML 1.0 | | | XHTML 1.1 |
|---|---|---|---|---|
| | Transitional | Strict | Frameset | |
| <a> | Si | Si | Si | Si |
| <abbr> | Si | Si | Si | Si |
| <acronym> | Si | Si | Si | Si |
| <address> | Si | Si | Si | Si |
| <applet> | Si | Non | Si | Non |
| <area /> | Si | Si | Si | Non |
| <b> | Si | Si | Si | Si |
| <base /> | Si | Si | Si | Si |
| <basefont /> | Si | Non | Si | Non |
| <bdo> | Si | Si | Si | Non |
| <big> | Si | Si | Si | Si |
| <blockquote> | Si | Si | Si | Si |
| <body> | Si | Si | Si | Si |
| <br /> | Si | Si | Si | Si |
| <button> | Si | Si | Si | Si |
| <caption> | Si | Si | Si | Si |
|  | Si | Non | Si | Non |
| <cite> | Si | Si | Si | Si |
| <code> | Si | Si | Si | Si |
| <col /> | Si | Si | Si | Non |
| <colgroup> | Si | Si | Si | Non |
| <dd> | Si | Si | Si | Si |
| <del> | Si | Si | Si | Non |
| <dfn> | Si | Si | Si | Si |
| <dir> | Si | Non | Si | Non |
|  | Si | Si | Si | Si |
| <dl> | Si | Si | Si | Si |
| <dt> | Si | Si | Si | Si |
|  | Si | Si | Si | Si |
| <fieldset> | Si | Si | Si | Si |
| <font> | Si | Non | Si | Non |
| <form> | Si | Si | Si | Si |
| <frame /> | Non | Non | Si | Non |
| <frameset> | Non | Non | Si | Non |
| <h1> to <h6> | Si | Si | Si | Si |
| <head> | Si | Si | Si | Si |
| <hr /> | Si | Si | Si | Si |
| <html> | Si | Si | Si | Si |
| <i> | Si | Si | Si | Si |
| <iframe> | Si | Non | Si | Non |
| <img /> | Si | Si | Si | Si |
| <input /> | Si | Si | Si | Si |
| <ins> | Si | Si | Si | Non |
| <isindex> | Si | Non | Si | Non |
| <kbd> | Si | Si | Si | Si |
| <label> | Si | Si | Si | Si |
| <legend> | Si | Si | Si | Si |
| <li> | Si | Si | Si | Si |
| <link /> | Si | Si | Si | Si |
| <map> | Si | Si | Si | Non |
| <menu> | Si | Non | Si | Non |
| <meta /> | Si | Si | Si | Si |
| <Nonframes> | Si | Non | Si | Non |
| <Nonscript> | Si | Si | Si | Si |
| <object> | Si | Si | Si | Si |

| | | | | |
|---|---|---|---|---|
| <ol> | Si | Si | Si | Si |
| <optgroup> | Si | Si | Si | Si |
| <option> | Si | Si | Si | Si |
| <p> | Si | Si | Si | Si |
| <param /> | Si | Si | Si | Si |
| <pre> | Si | Si | Si | Si |
| <q> | Si | Si | Si | Si |
| <s> | Si | Non | Si | Non |
| <samp> | Si | Si | Si | Si |
| <script> | Si | Si | Si | Si |
| <select> | Si | Si | Si | Si |
| <small> | Si | Si | Si | Si |
|  | Si | Si | Si | Si |
| <strike> | Si | Non | Si | Non |
|  | Si | Si | Si | Si |
| <style> | Si | Si | Si | Si |
| <sub> | Si | Si | Si | Si |
| <sup> | Si | Si | Si | Si |
| <table> | Si | Si | Si | Si |
| <tbody> | Si | Si | Si | Non |
| <td> | Si | Si | Si | Si |
| <textarea> | Si | Si | Si | Si |
| <tfoot> | Si | Si | Si | Non |
| <th> | Si | Si | Si | Si |
| <thead> | Si | Si | Si | Non |
| <title> | Si | Si | Si | Si |
| <tr> | Si | Si | Si | Si |
| <tt> | Si | Si | Si | Si |
| <u> | Si | Non | Si | Non |
| <ul> | Si | Si | Si | Si |
| <var> | Si | Si | Si | Si |

## 4 Atributos CSS

| Bordes | | | |
|---|---|---|---|
| **Propiedade** | **Valor por defecto** | **Descrición** | **Sintaxe en JavaScript** |
| border | | Indica todas as propiedades do borde nunha soa declaración, pola seguinte orde: *border-width*, *border-style* e *border-color* | *obxecto*.style.border="3px solid blue" |
| border-bottom | | Indica todas as propiedades do borde inferior nunha soa declaración, pola seguinte orde: *border-bottom-width*, *border-bottom-style* e *border-bottom-color* | *obxecto*.style.borderBottom="3px solid blue" |
| border-bottom-color | | Indica a cor do borde inferior | *obxecto*.style.borderBottomColor="blue" |
| border-bottom-style | | Indica o estilo do borde inferior | *obxecto*.style.borderBottomStyle="dotted" |
| border-bottom-width | medium | Indica o ancho do borde inferior | *obxecto*.style.borderBottomWidth="thick" |
| border-color | | Indica a cor dos catro bordes: arriba, dereita, abaixo e esquerda | *obxecto*.style.borderColor="#FF0000 blue" |
| border-left | | Indica todas as propiedades do borde esquerdo nunha soa declaración, pola seguinte orde: *border-left-width*, *border-left-style* e *border-left-color* | *obxecto*.style.borderLeft="3px solid blue" |
| border-left-color | | Indica a cor do borde esquerdo | *obxecto*.style.borderLeftColor="blue" |
| border-left-style | | Indica o estilo do borde esquerdo | *obxecto*.style.borderLeftStyle="dotted" |
| border-left-width | medium | Indica o ancho do borde esquerdo | *obxecto*.style.borderLeftWidth="thick" |
| border-right | | Indica todas as propiedades do borde dereito nunha soa declaración, pola seguinte orde: *border-right-width*, *border-right-style* e *border-right-color* | *obxecto*.style.borderRight="3px solid blue" |
| border-right-color | | Indica a cor do borde dereito | *obxecto*.style.borderRightColor="blue" |
| border-right-style | | Indica o estilo do borde dereito | *obxecto*.style.borderRightStyle="dotted" |
| border-right-width | medium | Indica o ancho do borde dereito | *obxecto*.style.borderRightWidth="thick" |
| border-style | | Indica o estilo dos catro bordes: arriba, dereita, abaixo e esquerda | *obxecto*.style.borderStyle="dotted double" |
| border-top | | Indica todas as propiedades do borde superior nunha soa declaración, pola seguinte orde: *border-top-width*, *border-top-style* e *border-top-color* | *obxecto*.style.borderTop="3px solid blue" |
| border-top-color | | Indica a cor do borde superior | *obxecto*.style.borderTopColor="blue" |
| border-top-style | | Indica o estilo do borde superior | *obxecto*.style.borderTopStyle="dotted" |
| border-top-width | medium | Indica o ancho do borde superior | *obxecto*.style.borderTopWidth="thick" |
| border-width | medium | Indica o ancho dos catro bordes: arriba, dereita, abaixo e esquerda; se só indicamos dous o primeiro aplícase aos bordes de arriba e de abaixo e o segundo aos da esquerda e dereita e se só indicamos un aplícase aos catro bordes | *obxecto*.style.borderWidth="thin thick medium 10px" |
| outline | invert none medium | Indica as propiedades da liña exterior (por fora dos bordes) nunha soa declaración, pola seguinte orde: *outline-color*, *outline-style* e *outline-width* | *obxecto*.style.outline="#0000FF dotted thin" |
| outline-color | invert | Indica a cor da liña exterior | *obxecto*.style.outlineColor="#00FF00" |
| outline-style | none | Indica o estilo da liña exterior | *obxecto*.style.outlineStyle="dotted" |
| outline-width | medium | Indica o ancho da liña exterior | *obxecto*.style.outlineWidth="thin" |

| Contidos e contadores xerados | | | |
|---|---|---|---|
| **Propiedade** | **Valor por defecto** | **Descrición** | **Sintaxe en JavaScript** |
| content | normal | Úsase cos pseudo-elementos **:before** e **:after** para inserir contido xerado a posteriori | *obxecto*.style.content="url(beep.wav)" |
| counter-increment | none | Incrementa un ou máis contadores | *obxecto*.style.counterIncrement="subsection" |
| counter-reset | none | Crea ou reinicia un ou máis contadores | *obxecto*.style.counterReset="subsection" |
| quotes | | Indica como deben ser as comiñas dependendo do nivel no que nos atopemos | *obxecto*.style.quotes="none" |

**Dimensións**

| Propiedade | Valor por defecto | Descrición | Sintaxe en JavaScript |
|---|---|---|---|
| height | auto | Indica o alto do elemento sen incluír as marxes, os bordes nin os ocos entre elementos | *obxecto*.style.height="50px" |
| max-height | none | Indica o alto máximo do elemento sen incluír as marxes, os bordes nin os ocos entre elementos | *obxecto*.style.maxHeight="10px" |
| max-width | none | Indica o ancho máximo do elemento sen incluír as marxes, os bordes nin os ocos entre elementos | *obxecto*.style.maxWidth="100px" |
| min-height | 0 | Indica o alto mínimo do elemento sen incluír as marxes, os bordes nin os ocos entre elementos | *obxecto*.style.minHeight="10px" |
| min-width | 0 | Indica o ancho mínimo do elemento sen incluír as marxes, os bordes nin os ocos entre elementos | *obxecto*.style.minWidth="100px" |
| width | auto | Indica o ancho do elemento sen incluír as marxes, os bordes nin os ocos entre elementos | *obxecto*.style.width="50px" |

**Fondo**

| Propiedade | Valor por defecto | Descrición | Sintaxe en JavaScript |
|---|---|---|---|
| background | | Indica todas as propiedades do fondo nunha soa declaración, pola seguinte orde: *background-color*, *background-image*, *background-repeat*, *background-attachment* e *background-position*. Podemos non indicar algunha delas sen maiores atrancos para o navegador | *obxecto*.style.background="white url(*imaxe*) repeat-y" |
| background-attachment | scroll | Indica se a imaxe do fondo se despraza ou non co resto da páxina. | *obxecto*.style.backgroundAttachment="fixed" |
| background-color | transparent | Indica a cor do fondo dun elemento | *obxecto*.style.backgroundColor="#00FF00" |
| background-image | none | Indica a imaxe do fondo dun elemento | *obxecto*.style.backgroundImage="url(*imaxe*)" |
| background-position | 0% 0% | Indica onde colocamos a imaxe do fondo. Se só indicamos un valor, o outro asúmese como ***center*** | *obxecto*.style.backgroundPosition="center" |
| background-repeat | repeat | Indica como se repite a imaxe do fondo | *obxecto*.style.backgroundRepeat="repeat-y" |

**Fontes**

| Propiedade | Valor por defecto | Descrición | Sintaxe en JavaScript |
|---|---|---|---|
| font | | Indica as propiedades da fonte nunha soa declaración, pola seguinte orde: *font-style*, *font-variant*, *font-weight*, *font-size/line-height* e *font-family*. | *obxecto*.style.font="italic small-caps bold 12px arial,sans-serif" |
| font-family | | Indica a fonte dun elemento | *obxecto*.style.fontFamily="arial,sans-serif" |
| font-size | medium | Indica o tamaño da fonte | *obxecto*.style.fontSize="larger" |
| font-style | normal | Indica o estilo da fonte | *obxecto*.style.fontStyle="italic" |
| font-variant | normal | Indica se as minúsculas do texto se mostran en maiúsculas máis pequenas | *obxecto*.style.fontVariant="small-caps" |
| font-weight | normal | Indica o ancho da fonte | *obxecto*.style.fontWeight="900" |

**Impresión**

| Propiedade | Valor por defecto | Descrición | Sintaxe en JavaScript |
|---|---|---|---|
| orphans | | Indica o número mínimo de liñas que non poden quedar soas ao final dunha páxina ao paxinar un elemento | |
| page-break-after | auto | Indica se paxinamos automaticamente despois dun elemento | *obxecto*.style.pageBreakAfter="always" |
| page-break-before | auto | Indica se paxinamos automaticamente antes dun elemento | *obxecto*.style.pageBreakBefore="always" |
| page-break-inside | auto | Indica se paxinamos automaticamente no medio dun elemento | *obxecto*.style.pageBreakInside="avoid" |
| widows | | Indica o múmero mínimo de liñas que non poden quedar soas ao principio dunha páxina ao paxinar un elemento | |

### Listas

| Propiedade | Valor por defecto | Descrición | Sintaxe en JavaScript |
|---|---|---|---|
| list-style | disc outside none | Indica as propiedades dunha lista nunha soa declaración, pola seguinte orde: *list-style-type, list-style-position* e *list-style-image* | *obxecto*.style.listStyle="decimal inside" |
| list-style-image | none | Indica unha imaxe como icona da lista de elementos | *obxecto*.style.listStyleImage="url('*imaxe*')" |
| list-style-position | outside | Indica se sangramos ou non a icona da lista de elementos | *obxecto*.style.listStylePosition="inside" |
| list-style-type | disc | Indica o tipo de icona da lista de elementos | *obxecto*.style.listStyleType="square" |

### Marxes

| Propiedade | Valor por defecto | Descrición | Sintaxe en JavaScript |
|---|---|---|---|
| margin | 0 | Indica as propiedades da marxe nunha soa declaración, pola seguinte orde: arriba, dereita, abaixo e esquerda, permitindo valores negativos. Se só indicamos dous o primeiro aplícase aos bordes de arriba e de abaixo e o segundo aos da esquerda e dereita e se só indicamos un aplícase aos catro bordes | *object*.style.margin="10px 5px" |
| margin-bottom | 0 | Indica o tamaño da marxe inferior | *obxecto*.style.marginBottom="10px" |
| margin-left | 0 | Indica o tamaño da marxe esquerda | *obxecto*.style.marginLeft="10px" |
| margin-right | 0 | Indica o tamaño da marxe dereita | *obxecto*.style.marginRight="10px" |
| margin-top | 0 | Indica o tamaño da marxe superior | *obxecto*.style.marginTop="10px" |

### Posicionamento

| Propiedade | Valor por defecto | Descrición | Sintaxe en JavaScript |
|---|---|---|---|
| bottom | auto | Nos elementos posicionados dun modo absoluto, indica a distancia dende o fondo do elemento ao fondo do contedor e nos elementos posicionados dun xeito relativo a distancia dende o fondo do elemento á posición considerada normal. Se a posición é estática, **position:static**, esta propiedade non ten efecto sobre o elemento | *obxecto*.style.bottom="50px" |
| clear | none | Indica en que lados dun elemento non podo haber flotabilidade doutros elementos | *obxecto*.style.clear="left" |
| clip | auto | Permite indicar as dimensións dun elemento posicionado de forma absoluta de tal xeito que se mostra na figura creada. Esta propiedade non ten efecto se temos a propiedade **overflow:visible** | *obxecto*.style.clip="rect(0px,50px,50px,0px)" |
| cursor | auto | Indica o tipo de cursor que se mostra | *obxecto*.style.cursor="crosshair" |
| display | inline | Indica o tipo de caixa que xerará o elemento | *obxecto*.style.display="inline" |
| float | none | Indica se un obxecto flota ou non con relación aos demais; os obxectos posicionados dun xeito absoluto non se ven afectados por esta propiedade | *obxecto*.style.cssFloat="left" |
| left | auto | Actúa como **bottom** aplicado á marxe esquerda | *obxecto*.style.left="50px" |
| overflow | visible | Indica que sucede se o contido dun elemento rebasa os límites do contedor | *obxecto*.style.overflow="scroll" |
| position | static | Indica como posicionar un elemento | *obxecto*.style.position="absolute" |
| right | auto | Actúa como **bottom** aplicado á marxe dereita | *obxecto*.style.right="50px" |
| top | auto | Actúa como **bottom** aplicado á marxe superior | *obxecto*.style.top="50px" |
| visibility | visible | Indica se un elemento é visible ou non | *obxecto*.style.visibility="hidden" |
| z-index | auto | Indica a orde na pía de visualización do elemento; un elemento cun valor maior está sempre por diante dos elementos con valores menores | *obxecto*.style.zIndex="1" |

**Separadores**

| Propiedade | Valor por defecto | Descrición | Sintaxe en JavaScript |
|---|---|---|---|
| padding | 0 | Indica as propiedades de separación dende os bordes dun elemento ao contido nunha soa declaración, pola seguinte orde: arriba, dereita, abaixo e esquerda. Se só indicamos dous o primeiro aplícase aos bordes de arriba e de abaixo e o segundo aos da esquerda e dereita e se só indicamos un aplícase aos catro bordes | *obxecto*.style.padding="10px 5px" |
| padding-bottom | 0 | Indica a separación dende abaixo | *obxecto*.style.paddingBottom="2cm" |
| padding-left | 0 | Indica a separación dende a esquerda | *obxecto*.style.paddingLeft="2cm" |
| padding-right | 0 | Indica a separación dende a dereita | *obxecto*.style.paddingRight="2cm" |
| padding-top | 0 | Indica a separación dende arriba | *obxecto*.style.paddingTop="2cm" |

**Táboas**

| Propiedade | Valor por defecto | Descrición | Sintaxe en JavaScript |
|---|---|---|---|
| border-collapse | separate | Especifica se os bordes da táboa se xuntan para aparentar un borde sinxelo ou bordes separados para cada elemento | *obxecto*.style.borderCollapse="collapse" |
| border-spacing | | Especifica a distancia entre os bordes de celas adxacentes: o primeiro valor indica a separación horizontal e o segundo a vertical; se só pomos unha medida aplícase tanto en horizontal como en vertical | *obxecto*.style.borderSpacing="15px 20px" |
| caption-side | top | Indica o lugar onde se coloca o título da táboa | *obxecto*.style.captionSide="bottom" |
| empty-cells | show | Indica se se deben mostrar os bordes e o fondo das celas baleiras | *obxecto*.style.emptyCells="hide" |
| table-layout | auto | Indica o deseño que debe ser usado para debuxar a táboa | *obxecto*.style.tableLayout="fixed" |

**Texto**

| Propiedade | Valor por defecto | Descrición | Sintaxe en JavaScript |
|---|---|---|---|
| color | | Indica a cor do texto | *obxecto*.style.color="#FF0000" |
| direction | | Indica o sentido da escritura: de esquerda a dereita ou de dereita a esquerda | *obxecto*.style.direction="rtl" |
| letter-spacing | normal | Incrementa ou decrementa o espazo entre os caracteres do texto | *obxecto*.style.letterSpacing="3px" |
| line-height | normal | Indica o alto da liña | *obxecto*.style.lineHeight="2" |
| text-align | | Indica o aliñamento horizontal do texto | *obxecto*.style.textAlign="right" |
| text-decoration | none | Indica que elementos decorativos se lle engaden ao texto | *obxecto*.style.textDecoration="overline" |
| text-indent | | Indica a sangría da primeira liña dun bloque de texto permitindo valores negativos | *obxecto*.style.textIndent="50px" |
| text-transform | none | Controla a capitalización do texto | *obxecto*.style.textTransform="uppercase" |
| vertical-align | baseline | Indica o aliñamento vertical do elemento | *obxecto*.style.verticalAlign="bottom" |
| white-space | normal | Especifica como se manexan os espazos consecutivos | *obxecto*.style.whiteSpace="pre" |
| word-spacing | normal | Modifica o espazo entre as palabras dun texto permitindo valores negativos | *obxecto*.style.wordSpacing="10px" |

**Pseudoclases**

| Propiedade | Descrición | Sintaxe |
|---|---|---|
| :link | Engade estilos ás ligazóns que non foron visitadas | Orde que deben ter na folla de estilos para que funcionen correctamente:<br>a:link {color:#FF0000}<br>a:visited {color:#00FF00}<br>a:hover {color:#FF00FF}<br>a:active {color:#0000FF} |
| :visited | Engade estilos ás ligazóns que xa foron visitadas | |
| :hover | Engade estilos ao elemento sobre o que pasamos co rato | |
| :active | Engade estilos ao elemento que está activo | |
| :after | Engade contido despois dun elemento | |
| :before | Engade contido antes dun elemento | |
| :first-child | Engade estilos ao elemento que é o primeiro fillo doutro elemento | |
| :first-letter | Engade estilos ao primeiro carácter dun texto | |
| :first-line | Engade estilos á primeira liña dun texto | |
| :focus | Engade estilos ao elemento no que vamos introducir datos | |
| :lang | Engade estilos aos elementos que teñen asignado o idioma especificado | |

## 5 Referencias

Páxina oficial das escolas de W3C

**http://www.w3schools.com**

Usabilidade para os maiores

**http://www.nosolousabilidad.com/articulos/usabilidad_mayores.htm**

Porqué empregar complementos para Firefox

**http://mir.aculo.us/stuff/orcreatehappyusers.pdf**

Titoriais con exemplos prácticos de JavaScript, CSS, HTML e deseño web.

**http://www.howtocreate.co.uk**

Titoriais de Javascript e CSS. Moi bos recursos sobre CSS.

**http://www.quirksmode.org/resources.html**

Repositorio de scripts, manuais, descarga dos código fonte dos scripts, buscador, ...

**http://www.hotscripts.com**

Máquinas virtuais

**http://www.virtualbox.org**
**http://www.vmware.com/es**
**http://wiki.qemu.org**
**http://linux-vserver.org**
**http://www.microsoft.com/windows/virtual-pc**

## 6 Bibliografía

**[DE04] DesarrolloWeb**
*Manual de HTML*
**DesarrolloWeb.com**

**[HT04] HTML Code Tutorial**
*The Idocs Guide to HTML*
**http://www.htmlcodetutorial.com/**

**[PA00] Panéele, Eduard**
*Cree su propia página web con HTML 4.0*
Salvat, 2000

**[RO00] Rodríguez, Daniel**
*HTML*
**Daniel Rodríguez**

**[RO04] Rodríguez Herrera, Daniel**
*HTML en castellano: Curso de HTML 4.0*
**http://www.programacion.net/html/tutorial/curso/**