MÓDULO

JAVASCRIPT/AJAX

UNIDADE

CRIAÇÃO DE EVENTOS E CÓDIGO PARA PÁGINAS WEB, APLICANDO FUNÇÕES E OBJETOS JAVASCRIPT

Instituto Tecnológico

ÍNDICE

OBJETIVOS

Com esta unidade didática, pretende-se que desenvolva os seguintes objetivos de aprendizagem:

- Saber controlar as funções do objeto window.
- Saber controlar as funções do objeto document.
- Aprender a criar pops e cookies.
- Saber como controlar a execução do código por tempo.
- Conseguir encontrar e interpretar erros de JavaScript.

INTRODUÇÃO

O JavaScript é uma linguagem que possui várias funções próprias que permitem outras funcionalidades muito úteis na programação. Por exemplo, possui sistemas de controlo e execução de código por tempo, criação de cookies, opções de navegação, etc.

Nesta unidade, aprenderá o uso das funções JavaScript mais importantes. O estudo desta unidade é importante, uma vez que muitas das funções do JavaScript permitem criar eventos e código para as páginas web.

1. O OBJETO DOCUMENT

O objeto document contém todos os elementos da página. Com as suas propriedades e funções pode fazer alterações dinâmicas na web sempre que pretender.

As propriedades mais utilizadas deste objeto são:

- **anchors**: um array que contém os links internos existentes no documento.
- **all** (em desuso): um array que contém todos os elementos do documento, sejam de que tipo forem.
- **Applets** (em desuso): um array que contém applets criados na web.
- **images**: um array que contém as imagens da web.
- **forms**: um array com os formulários criados na web.
- **links**: um array com os links externos da web.
- **bgColor** (em desuso): contém a cor de fundo do documento.
- **cookie**: um valor de string que contém os valores dos cookies que possui no documento, separados por ";".
- **domain:** contém o nome do servidor onde a web está hospedada.
- **alinkColor**: contém a cor usada no documento para links ativos.
- **location**: string que contém a localização da web.
- **referrer**: string que contém a URL da página que chamou a página atual.

- **body**: utilizado para se referir ao corpo da web, contido nas tags HTML <body></body>, apenas no modo de leitura.
- **lastModified**: contém a data de modificação do documento.
- **readyState**: estado do objeto; tem quatro valores possíveis:
  - Não carregado.
  - A carregar.
  - Carregamento inacabado, mas interativo.
  - Carga máxima.
- **height**: contém a altura do documento atual, pode ser modificada.
- **width**: contém a largura do documento atual, pode ser modificado.
- **styleSheets**: devolve uma lista das folhas de estilo CSS usadas no documento.
- **parentNode**: devolve o pai ou o elemento anterior ao objeto indicado.

Seguem-se as funções mais utilizadas do objeto document e o seu uso na web.

1.1. DOCUMENT.WRITE

Esta função permite escrever uma string no documento, caso o documento ainda não exista (cria um documento dinâmico). Terá de chamar as funções **document.open()** e **document.close()**, que abrem e fecham um canal de escrita.

A sintaxe é a seguinte:

```
document.write('conteúdo');
```

O conteúdo pode ser um texto indicado entre plicas ou uma variável (neste caso sem utilizar as plicas).

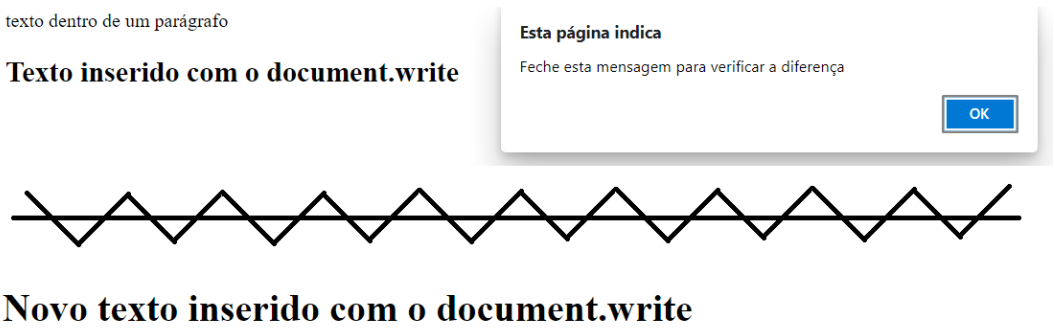
Dependendo da posição onde a função é inserida, terá um ou outro efeito. Por exemplo, no caso de a inserir seguida no código, irá criar mais uma entrada de texto no documento, enquanto se a chamar a partir de uma função, ao carregar o documento, substituirá o conteúdo do documento.

Exemplo 01

Neste exemplo verá o uso da função **document.write** e como a posição onde está inserida a influencia.

Exemplo:

```
<html>

<head>
<title>Exemplo document.write</title>

<script type="text/JavaScript">
function mudar(){
alert('Feche esta mensagem para verificar a diferença');
document.write("<h1>Novo texto inserido como document.write</h1>");
}
</script>

</head>

<body onload="mudar();">
<p>texto dentro de um parágrafo</p>
<script>
document.write("<h2>Texto inserido com odocument.write</h2>");
</script>
</body>

</html>
```



Resultado do exemplo.

1.2. DOCUMENT.WRITEIN

Esta função é semelhante à anterior, com a exceção de que no final do texto inserido adiciona uma quebra de linha.

A sintaxe é a seguinte:

```
document.writeln('texto');
```

1.3. DOCUMENT.CREATETEXTNODE

Esta função permite inserir texto dentro de um elemento já criado, por exemplo, inserir um parágrafo depois do texto que já existe dentro do elemento. Para o fazer, utiliza-se com uma função chamada **appendChild**, que permite inserir elementos criados dentro de outros.

A sintaxe é a seguinte:

```
var variavel = document.createTextNode(dados);

objeto.appendChild(variavel);
```

Exemplo 02

Segue-se um exemplo de como criar um texto dentro de um parágrafo já criado ao pressionar o botão.

Exemplo:

```
<html>

<head>

<title>Exemplo createTextNode</title>

<script type="text/JavaScript">
```

```
function adicionarTexto(){

 var variavel = document.createTextNode("texto comcreateTextNode");

 var paragrafo = document.getElementById("p1");

paragrafo.appendChild(variavel);

}

</script>

</head>

<body>

<div style="background-color:#cccccc">

<p id="p1">Linha de texto de documento<br /></p>

</div><br />

<input type="button" name="env" value="Adicionar!"
onClick="adicionarTexto()"/>

</body>

</html>
```

1.4. DOCUMENT.GETELEMENTBYID

Esta função é frequentemente utilizada, pois permite fazer referência a um objeto diretamente pelo seu id, onde quer que ele esteja no código. Permite também aceder a todos os atributos e estilos do elemento.

A sintaxe é a seguinte:

```
document.getElementById(identificador)
```

1.5. DOCUMENT.GETELEMENTSBYNAME

Esta função devolve uma lista dos elementos que possuem o nome indicado como parâmetro na função. A lista é devolvida como um array de "n" posições, onde "n" é o número de elementos encontrados.

A sintaxe é a seguinte:

```
var array = document.getElementsByName(nome);
```

Exemplo 03

Neste exemplo irá criar um formulário na página web com vários campos que têm o mesmo nome, e usar esta função para os percorrer e mostrar o valor que contêm ao premir o botão.

Exemplo:

```
<html>
<head>
<title>Exemplo getElementsByName</title><script>
functionmostrar(){
 var vector = document.getElementsByName("nome");
 for (i=0;i<vector.length;i++){
alert(vector[i].value);
 }
}
</script>
</head>
<body>
<form name="teste">
```

```
<input type="text" name="nome" value="Pedro">

<input type="text" name="nome" value="Maria">

<input type="text" name="nome" value="Joaquim">

<input type="button" name="env" value="Mostrar!"
onClick="mostrar()"/>

</form>

</body>

</html>
```

1.6. DOCUMENT.CREATEELEMENT

Com esta função pode criar elementos dentro do HTML, desde uma div a um radio.

A sintaxe é a seguinte:

```
var novoElemento = document.createElement(nome);
```

Juntamente com a próxima função, verá que é uma ferramenta muito útil para poder compor o conteúdo de uma página de forma dinâmica.

1.7. DOCUMENT.INSERTBEFORE

Permite inserir um elemento antes de outro que está definido e que passa como parâmetro à função. Recebe dois parâmetros: o primeiro é o novo elemento a inserir e o segundo é a referência ao elemento cuja posição é assumida.

A sintaxe é a seguinte:

```
[elementoPai].insertBefore(novoElemento,referenciaElemento)
```

O elemento de referência está contido onde está o "elementoPai".

Exemplo 04

Um exemplo simples de como utilizar a função **insertBefore** para inserir um texto antes de outro texto já existente.

Exemplo:

```
<html>

<head>
<title>Exemplo insertBefore</title>
</head>

<body>
<div>

<span id="segundo">e depois a omelete é feita </span>
</div>

<script type="text/JavaScript">
// criar um novo elemento span
var novo = document.createElement("span");
// criar um texto dentro do novo span
var novoTexto = document.createTextNode("primeiro bate-se os ovos
");
// inserir o novo texto no novo span
novo.appendChild(novoTexto);
var antigo = document.getElementById("segundo");
```

```
var pai = antigo.parentNode;
// por último, usar a função para inserir o texto
pai.insertBefore(novo, antigo);
</script>

</body>

</html>
```

Não existe uma função insertAfter, mas esse efeito pode ser criado através do insertBefore e nextSibling. Esta função devolve o próximo nó indicado.

A sintaxe seria a seguinte:

```
pai.insertBefore(novoElemento, referenciaElemento.nextSibling)
```

1.8. DOCUMENT.CREATEATTRIBUTE E SETATTRIBUTENODE

Estas funções permitem criar atributos não definidos no HTML para os elementos, ou seja, pode criar atributos reais dos elementos, como por exemplo o identificador ou id de uma div, ou criar os novos atributos de que precisar.

A sintaxe é a seguinte:

```
var atributo = document.createAttribute(nome);

elemento.setAttribute(atributo);
```

Para saber os valores dos atributos, utiliza-se a função **getAttribute** do objeto do documento.

Exemplo 05

Segue-se um exemplo simples no qual é criado um novo atributo para uma div e com um alert é, posteriormente, exibido o seu valor.

Exemplo:

```
<html>
<head>
<title>Exemplo de criação de atributos</title>
<script type="text/JavaScript">
Function criarAtributo(){
 var elemento = document.getElementById("elemento");
 var atributo = document.createAttribute("meuAtributo");
atributo.nodeValue = "valor do novo atributo";
elemento.setAttributeNode(atributo);
alert(elemento.getAttribute("meuAtributo"));
 }
</script>
</head>
<body onload="criarAtributo();">
<div id="elemento">
 conteúdo do elemento
</div>
</body>
</html>
```

1.9. DOCUMENT.CREATEEVENT

Esta função permite criar eventos de uma forma dinâmica, ou seja, é possível, por exemplo, simular o clique do rato num elemento sem que isso realmente aconteça. Para tal, utilizam-se duas funções, internamente:

- initEvent ou initMouseEvent, cria uma instância do evento a ser lançado. A diferença entre initEvent e initMouseEvent é que o primeiro lança eventos HTML, enquanto o segundo lança eventos do rato, embora em alguns casos possam ambos ser usados (como para eventos click). A função do rato tem muito mais possibilidades.

A sintaxe é a seguinte:

```
event.initMouseEvent (type, canBubble, cancelable, view, detail,
screenX, screenY, clientX, clientY, ctrlKey, altKey, shiftKey,
metaKey, button, relatedTarget);
```

Cada elemento possui a sua função:

- **type**: indica o tipo de evento, por exemplo, para initMouseEvent, click, mousedown, mouseup, mouseover, mousemove.
- **canBubble**: indica se o evento permite um efeito de bolha.
- **cancelable**: indica se o evento permite cancelar ou que a ação padrão seja cancelada.
- **view**: permite passar o objeto window àquilo a que se refere.
- **detail**: conta o número de cliques do rato.
- **screenX**: indica a coordenada no eixo X da posição do evento em relação a toda a janela.
- **screenY**: indica a coordenada do eixo Y da posição do evento em relação a toda a janela.
- **clientX**: indica a coordenada no eixo X da posição do evento em relação à visualização do cliente e do seu navegador.
- **clientY**: indica a coordenada no eixo Y da posição do evento em relação à visualização do cliente e do seu navegador.
- **keyCtrl**: indica se a tecla control (ctrl) do teclado foi pressionada durante o evento.

- **altKey**: indica se a tecla alt do teclado foi pressionada durante o evento.
- **shiftKey**: indica se a tecla shift do teclado foi pressionada durante o evento.
- **button**: indica qual botão do rato está pressionado, correspondendo 0 ao da esquerda, 1 ao central e 2 ao da direita.
- **relatedTarget**: usado apenas nos eventos do tipo mouseover e mouseout, e indica o elemento a que se refere; caso não se pretenda saber o elemento, utiliza-se o null.
- **dispatchEvent** (evento), cria o lançamento do evento a partir do elemento que quiser.

Exemplo 06

Neste exemplo, aprenderá a criar um evento de click do rato numa checkbox.

Exemplo:

```
<html>

<head>

<title> Exemplo de criação de eventos</title>

<script type="text/JavaScript">

Function simularClick() {

 //criar o evento

 var evento = document.createEvent("MouseEvents");

 //dar o tipo e valores que desejar

evento.initMouseEvent("click", true, true, window,0, 0, 0, 0, 0,
false, false, false, false, 0, null);

 //selecionar o elemento sobre o qual se lança o evento criado

 var elemento = document.getElementById("checkbox");

 //lançar o evento

elemento.dispatchEvent(evento);
```

```
}

</script>

</head>

<body>

<input type="checkbox" id="checkbox" />checkbox<br/>

<input type="button" value="criar evento"
onClick="simularClick()"/>

</body>

</html>
```

2. O OBJETO WINDOW

O objeto **window** é o que contém tudo; indica a janela do navegador onde se abriu a página web, e dentro dele está o objeto **document** associado a todos os elementos que contém. Por outras palavras, é o objeto pai de todos os elementos de uma página web.

2.1. PROPRIEDADES

As suas principais propriedades são:

- **closed**, propriedade somente de leitura que devolve "true", se a janela à qual se refere estiver fechada, ou "false", se estiver aberta.

```
var closed = window.closed;
```

- **defaultStatus**, permite alterar e ler o texto da barra de status do navegador.

```
window.defaultStatus = "Olá, bem-vindo";
```

- **frames**, devolve um array composto pelos elementos que estão na web.

- **history**, devolve a instância de referência do elemento history de uma janela para poder interagir com ela, e para isso tem as seguintes funções:
  - history.back(): o navegador vai para a página anterior guardada na sessão. Semelhante ao botão "voltar" dos navegadores.
  - history.forward(): o navegador vai para a próxima página guardada na sessão. Semelhante ao botão "próximo" do navegador.
  - history.go(número): informa o navegador para estar na página guardada na sessão e na posição que indicar a partir da atual. Por exemplo, colocando um -3, o navegador voltaria atrás três páginas.
- **innerHeight**, altura do conteúdo, não incluindo os menus do navegador, mas contando com os possíveis scrolls.
- **innerWidth**, largura do conteúdo da janela, incluindo os scrolls.
- **outherHeight**, altura do navegador, incluindo a página, os menus e os scrolls.
- **outherWidth**, largura completa do navegador.

Segue-se um exemplo gráfico da diferença:

Exemplo do objetowindow.

- **length**, número de frames ou iframes que a janela contém.
- **opener**, devolve a referência da janela recentemente aberta. Caso a janela não tenha sido aberta a partir de outra, devolve null.
- **parent**, devolve a referência do pai da frame ou da frame atual, por exemplo, se uma página for inserida dentro de uma frame, o pai desta será a janela que contém a frame.
- **navigator**, devolve a referência a um objeto JavaScript predefinido que fornece informações do navegador da web. Os seus possíveis atributos são:
    - navigator.appCodeName: devolve o nome interno do navegador.
    - navigator.appName: devolve o nome oficial do navegador.
    - navigator.appVersion: devolve a versão do navegador.
    - navigator.cookieEnabled: devolve “true”, se o navegador tiver cookies ativados.
    - navigator.language: devolve o código do idioma do navegador (por exemplo, Portugal “pt”, Espanha “es”, França “fr”...).
    - navigator.onLine: devolve “true”, quando o navegador está a ser executado online e não com uma versão da memória.
    - navigator.platform: devolve a plataforma onde o navegador está a ser executado, por exemplo, no Windows XP 32 bits retorna "Win32".
    - navigator.plugins: devolve um array com a lista dos plug-ins instalados no navegador.
    - navigator.product: devolve o nome do produto que é o navegador.
    - navigator.producSub: devolve o número de referência do produto.
    - navigator.userAgent: devolve o agente do navegador usado pelo produto.
    - navigator.vendor: indica o vendedor do produto, por exemplo, no caso do Chrome, devolve “Google Inc.”.
    - navigator.vendorSub: indica a versão do fornecedor (se houver).

2.2. MÉTODOS

Os métodos são utilizados para realizar ações, como uma janela pop-up, um alerta, adicionar elementos a uma lista, removê-los, transformar uma janela, etc.

2.2.1. WINDOW.ADDEVENTLISTENER

Este método permite criar eventos para objetos de forma dinâmica, de modo a que estes estejam à espera de serem iniciados. O método recebe três parâmetros:

- O primeiro é o tipo de evento (click, mouseOver, mouseOut...).
- O segundo é a função que será executada quando o evento for acionado.
- O terceiro recebe "true" ou "false", o que determina se o detetor opera na fase de captura ou na fase de destino e propagação; se for "false", fá-lo nas fases de destino e propagação, enquanto, se for "true", só o faz na fase de captura.

Exemplo 07

Neste exemplo, será atribuído um evento de forma dinâmica a um botão no formulário que iniciará uma função para escrever dentro de um campo de texto.

Exemplo:

```
<html>

<head>

<title>Exemplo de evento dinâmico</title>

<script type="text/JavaScript">

 // função que reinicia o campo do nome

function modificarNome() {

 var campo = document.getElementById("campo");
```

```
campo.value = "Escreva aqui o seu nome";
 }
 // função que insere o evento
function load() {
 var botao = document.getElementById("reset");
botao.addEventListener("click", modificarNome, false);
 }
</script>
</head>
<body onload="load();">
<input type="text" id="campo" name="campo" value="" />
<input type="button" id="reset" name="reset" value="Reset" />
</body>
</html>
```

2.2.2. WINDOW.REMOVEEVENTLISTENER

Este método permite eliminar um evento criado com a função anterior. A sua sintaxe é a seguinte:

```
botao.removeEventListener(“click”,modificarNome, false);
```

2.2.3. WINDOW.CLOSE

Este método permite fechar uma janela aberta (aberta pelo utilizador ou pela própria página web).

2.2.4. WINDOW.OPEN

Este método permite criar novas janelas a partir da janela onde o utilizador está (também chamadas de pop-ups). A sua sintaxe é a seguinte:

```
var novaJanela = window.open(url, nomeJanela,propriedades)
```

O parâmetro url é o endereço da página que pretende abrir; se a página for interna ao website, basta colocar o nome da referida página (ex.: sobre.html), enquanto, se quiser abrir uma página externa, terá de colocar o URL completo (ex: http://www.exemplo.com).

O segundo parâmetro, "nomeJanela", é um nome simbólico dado à janela, que não pode conter espaços e não indica o título da nova janela.

O terceiro parâmetro é opcional e nele pode incluir uma série de parâmetros separados por vírgulas para criar a nova página com um estilo definido. Seguem-se os **parâmetros** que pode incluir (apenas aqueles comuns a todos os navegadores).

- **left**: indica o espaço de separação com a margem esquerda do ecrã.
- **top**: indica o espaço de separação coma margem superior do ecrã.
- **height**: indica a altura da nova janela.
- **width**: indica a largura da nova janela.
- **menubar**: ao passar o valor yes, a nova janela terá a barra de menus do navegador.
- **toolbar**: ao passar o valor yes, a nova janela terá a barra de ferramentas (voltar, avançar, atualizar, parar, ...).

- **location**: ao passar um valor yes, a nova janela terá a barra de link.
- **status**: ao passar um valor yes, a nova janela terá uma barra de status inferior.

Exemplo 08

Neste exemplo serão utilizadas as duas últimas funções do objeto window e será criada uma página com dois botões: o primeiro abrirá uma nova janela e o segundo fechará a nova janela.

Exemplo:

```
<html>

<head>

<title>Exemplo de evento dinâmico</title>

<script type="text/JavaScript">

 //variável para conter a referência da nova janela

 var novaJanela;

function abrirJanela(){

novaJanela = window.open('http://www.google.com',
'teste','width=200,height=200,status=no,toolbar=no,top=300,left=300
');

 }

function fecharJanela() {

novaJanela.close();

 }

</script>

</head>

<body >

<input type="button" onclick="abrirJanela()" value="Abrir"/>

<input type="button" onclick="fecharJanela()" value="Fechar"/>
```

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

2.2.5. WINDOW.CONFIRM

Este método cria um alerta de confirmação, ou seja, mostra no ecrã um alerta com o texto passado no parâmetro e dois botões de opção: "Aceitar" e "Cancelar". Se o utilizador clicar em "Aceitar", devolve o valor “true”; se não, devolve “false”.

Exemplo 09

Neste exemplo, ao carregar a página web, perguntará se o utilizador deseja alterar a cor de fundo do documento para vermelho; se aceitar, mudará a cor.

Exemplo:

```
<html>
<head>
<title>Alerta com opções</title>
</head>
<body>
<script type="text/JavaScript">
if (window.confirm("Quer alterar o fundo para vermelho?")) {
document.bgColor = "#ff0000";
 }
</script>
</body>
</html>
```

2.2.6. WINDOW.MOVETO

Este método permite mover a janela, desde que não seja maximizada, passando os valores das coordenadas.

A sintaxe é a seguinte:

```
window.moveTo(coordenadaHorizontal,coordenadaVertical);
```

2.2.7. WINDOW.PRINT

Este método envia o documento atual para a impressora.

2.3. WINDOW.RESIZETO

Esta função permite mudar o tamanho da janela, de forma dinâmica, passando os novos tamanhos de largura e altura por parâmetro.

```
window.resizeTo(largura,altura)
```

Exemplo 10

Neste exemplo, vai criar uma página com dois botões. Ao clicar no botão "Abrir", abrirá uma nova janela, e ao clicar no botão "Alterar tamanho", alterará o tamanho da janela.

Exemplo:

```
<html>

<head>

<title>Exemplo - Alterar Tamanho</title>

<script type="text/JavaScript">

function mudarTamanho() {

janela.resizeTo(345,200);

 }

function abrir() {

 janela = window.open("", "", "width=100, height=100");

 }

</script>

</head>

<body>

<input type="button" onclick="abrir()" value="Abrir"/>

<input type="button" onclick="mudarTamanho()" value="Alterar
tamanho"/>

</body>

</html>
```

2.3.1. WINDOW.SCROLLTO

Este método permite mover o scroll (desde que exista) da página web até às coordenadas passadas no parâmetro.

```
window.scrollTo(largura,altura)
```

3. COOKIES

Um cookie é um arquivo de texto que é guardado no navegador e armazena informações transmitidas pelo servidor. Ajuda a indicar ao servidor quando são reintroduzidos dados sobre o utilizador. De um modo geral, guarda as preferências do utilizador.

O seu funcionamento é básico: ao conectar à web, verifica-se se o navegador possui cookies para esta página e, caso possua, envia os cabeçalhos dos cookies ao servidor, podendo assim obter os dados.

Um cookie é composto pelos seguintes elementos:

- **Nome e valor**: o primeiro indica como o cookie será chamado e o segundo indica o valor que terá.
- **Data de validade**: pode ser atribuída uma data que indicará até quando o cookie é válido; caso não indique a data, só terá validade enquanto o navegador estiver aberto, depois disso, será apagado; esta data tem um formato específico que pode dar através da função toGMTString da classe date.
- **Domínio:** indica o domínio para o qual o cookie será válido. Deve ter-se em consideração que se trata de URL completos (como, por exemplo, www.exemplo.com), portanto o cookie não será válido para outros domínios dentro da página, como, por exemplo, estudantes.exemplo.com.

- **path,** indica o caminho que o cookie afeta dentro do domínio. Se não for indicado, por defeito toma o caminho raiz "/". Mas se indicar, por exemplo, "/cursos/alunos", o cookie só será válido nas páginas deste caminho.

No primeiro ponto da unidade foram explicadas as propriedades do objeto **document**. No entanto, faltou referir a seguinte propriedade:

3.1. DOCUMENT.COOKIE

Com esta propriedade é possível criar, ler e apagar cookies de JavaScript. Seguem-se as explicações dessas mesmas opções.

Criar cookie

Basta fazer uma chamada para o atributo **document.cookie** com os valores vistos anteriormente.

```
document.cookie = "nombe=Fernando; path=/"
```

No exemplo anterior, vê um cookie sem uma data de validade atribuída. Segue-se um exemplo com uma data de validade:

```
document.cookie = "nombe=Ana; expires Tue, 12 Jan 2011 12:23:00
GMT; path=/"
```

Esta linha de código cria um cookie que expira em 12 de janeiro de 2011. Observe que a data indicada possui um formato específico, baseado no sistema GMT.

Em JavaScript existe uma função da classe date, chamada **toGMTString()**, que devolve a data tal como deve ser inserida. Exemplo:

```
var data = newDate();

var dataExp = data.toGMTString();
```

Pode criar tantos cookies quantos necessitar.

Ler

Para ler um cookie deve percorrer a string que compõe cada document.cookie, procurar o cookie de que precisa e obter o valor que divide essa string. Segue-se um exemplo de desta função:

```
functionlerCookie(nome){

// criar uma variável com o texto inicial que pretende que a cookie
leia

var inicioCookie = nome + "=";

// usar a função split para criar um array com todos os pedaços das
cookies

var cookies = document.cookie.split(';');

// criar um for para percorrer o array

for (i = 0;i<cookies.length; i++){

peca = cookies[i];

// passar esta peça por um loop para eliminar os espaços à frente

while (peca.charAt(0) == ' ') peca = peca.substring(1,peca.length)

// verificar se a peça do array contém a variável criada com a
cookie referida

if (peca.indexOf(inicioCookie) == 0){

 // devolver apenas uma parte do array, o valor do cookie

return peca.substring(inicioCookie.length,trozo.length);

}

}
```

Apagar

Não é possível excluir uma cookie do JavaScript diretamente. Para o eliminar, deve recriá-lo com uma data de validade anterior à data atual e, desse modo, o navegador detetará que ele expirou e irá excluí-lo.

Exemplo 11

A seguir, encontra um exemplo completo de como trabalhar com cookies. Pretende-se criar um objeto, chamado "gestorCookie", que terá três métodos: um para cada evento de cookies (criar, ler e apagar). O método de criação recebe o nome, o valor e o número de dias que desejar que o cookie esteja ativo, enquanto os outros dois métodos recebem apenas o nome do cookie. De seguida, insira três botões: um para criar um cookie, outro para o ler e o último para o excluir.

Este exemplo requer um servidor web para a sua verificação. No caso do Windows pode instalar o IIS (Internet InformationService) que vem no CD de instalação, ou um servidor Apache (mamp, appServ ou similar), para poder fazer o teste.

Exemplo:

```
<html>

<head>

<title>Exemplo cookies</title>

<script type="text/JavaScript">

 var gestorCookie = {

 criar: function(nome, valor, dias) {

if (dias) {

 var data = newDate();

data.setTime(data.getTime()+(dias*24*60*60*1000));

 var expira = "; expira="+data.toGMTString();
```

```
}else var expira = "";

document.cookie = nome+"="+valor+expira+"; path=/";

alert('Criada')

 },

 ler: function(nome) {

 var inicioCookie = nome + "=";

 var cookies = document.cookie.split(';');

for(var i=0;i <cookies.length;i++) {

var peca = cookies[i];

while (peca.charAt(0)==' ') peca = peca.substring(1,peca.length);

if (peca.indexOf(inicioCookie) == 0) return peca.substring

 (inicioCookie.length,peca.length);

 }

 return null;

 },

 apagar: function(nome) {

gestorCookie.criar(nome,"",-1);

alert('Apagada')

 }

};

</script>

</head>

<body>

 Criar Cookie<br /> Nome <input type="text" name="nome" id="nome"
/><br /> Valor <input type="text" name="valor" id="valor" /><br />

 dias <input type="text" name="dias" id="dias" /><br />

<input type="button"
```

```
onclick="gestorCookie.criar( document.getElementById(nome).value,
document.getElementById('valor').value,
document.getElementById('dias').value)"

value="Criar" /><br /><br />

 Ler Cookie<br />

 Cookie a Ler :<input type="text" name="valordev" id="valordev"
/><br />

<input type="button"
onclick="alert(gestorCookie.leer(document.getElementById('valordev'
).value))"

value="Ler" /><br /><br />

 Apagar Cookie<br />

 Cookie a apagar :<input type="text" name="apagar" id="apagar"
/><br />

<input type="button"
onclick="gestorCookie.apagar(document.getElementById(apagar).value)
"value="Ler" /><br /><br />

</body>

</html>
```

Este exemplo deve ser testado com um servidor web; não funciona simplesmente abrir o HTML num navegador, porque o gerenciamento de cookies não é válido.

4. INTERVALOS DE TEMPO

Para lidar com intervalos de tempo, o JavaScript usa duas funções: **setTimeout** e **setInterval**. Estas funções permitem definir um espaço de tempo entre a invocação da mesma e a execução do código. Servem, por exemplo, para criar um relógio e controlar a aparência dos elementos num horário específico.

4.1. SETTIMEOUT

Esta função permite definir o tempo exato (em milissegundos) antes de executar a função/código que indicado.

A sintaxe é a seguinte:

```
var temporizador = setTimeout(função, milissegundos);
```

Exemplo 12

Este exemplo cria uma página que não faz nada ao ser aberta, mas que após cinco segundos mostra um alerta com a mensagem "Olá!".

Exemplo:

```
<html>

<head>

<title>setTimeout</title>
<script type="text/JavaScript">
function mensagem(){
alert('Olá!');
 }
 var temporizador = setTimeout('mensagem()',5000);
</script>

</head>

<body >

Espere a mensagem

</body>

</html>
```

Se quiser eliminar um temporizador, poderá fazê-lo graças à função **clearTimeout**, passando-lhe o identificador do temporizador. No exemplo anterior seria:

```
clearTimeout(temporizador);
```

4.2. SETINTERVAL

Esta função permite executar uma tarefa num loop infinito a cada determinado intervalo de tempo. Esse intervalo de tempo entre a execução e a execução do código é passado por parâmetro em milissegundos.

A sintaxe é a seguinte:

```
var temporizador = setInterval(função, milissegundos);
```

Neste caso, a função ou código será executado num loop infinito, a menos que o cronómetro seja reduzido a zero; servir-se-á da função **clearInterval**, para a qual passa o temporizador a ser eliminado.

```
clearInterval(referenciaTemporizador);
```

Exemplo 13

Neste exemplo, utilizará a função **setInterval** para mudar a cor de uma caixa aleatoriamente a cada 3 segundos, podendo parar a mudança de cor, a qualquer momento, com um botão.

Exemplo:

```
<html>

<html>

<head>

<title>setInterval</title>

<script type="text/JavaScript">

 var temporizador;

function mudar() {
```

```
 temporizador = setInterval(rodaCores, 2000);

 }

function parar() {

    clearInterval(temporizador);

 }

function rodaCores() {

    document.getElementById('caixa').style.backgroundColor =
getRandomColor();

 }

function getRandomColor() {

     var possivel = '0123456789ABCDEF';

     var color = '#';

     for (var i = 0; i < 6; i++) {

           color += possivel[Math.floor(Math.random() * 16)];

     }

     return color;

    }

</script>

</head>

<body >

Caixa de cores<br/>

<div id="caixa" style="width:100px;height:100px;background-
color:#ff0000"> </div>

<br/><br/>

<button onclick="mudar()">Rodar cores</button>

<button onclick="parar()">Parar</button>

</body>
```

```
</html>
```

5. INTERPRETANDO ERROS JAVASCRIPT

Um dos principais problemas com o JavaScript é que, por ser uma linguagem interpretada por navegadores, depende deles para controlar os erros. Felizmente, o Mozilla Firefox e o Google Chrome incorporam uma consola JavaScript no navegador, que pode visitar para rastrear as páginas JavaScript e ver erros que elas geram.

Para controlar esses erros de JavaScript, há um evento do objeto Windows, chamado onerror, que é disparado sempre que um erro de sequência de JavaScript aparece, o que significa que podemos carregar uma página sem que apareça um erro até que algum evento seja lançado e passe a sequência de execução por meio desse código.

Esta função onerror cria três parâmetros: a mensagem de erro; o URL que gerou o erro (caso seja um arquivo externo); e a linha em que ocorreu o erro.

Um manipulador de erros pode ser definido da seguinte maneira:

```
<script language="JavaScript">

window.onerror= manipulador;

function manipulador(mensagem,url,lin) {

 var texto="Erro:\n" + mensagem + "\n" +"(página: " + url + ". lin
" + lin + ")";

alert(texto);
```

```
}
</script>
```

Se usar este manipulador ao criar a página, deverá lembrar-se de remover o manipulador de erros quando terminar de depurar o código, porque, se não o fizer, os erros serão mostrados aos utilizadores (se forem gerados).

Mesmo assim, continuará com um problema: depois de surgir um erro em JavaScript, a sua execução para. Para resolver este problema existe uma frase, designada try... catch, que permite executar um código JavaScript e, se aparecer um erro, tratá-lo como pretender sem interromper a execução do JavaScript. Segue-se um exemplo para ilustrar o uso desta função:

```
try {
funcao(valor); // irá gerar um erro porque a função não existe
} catch(err) {
alert("Ocorreu um erro:" + err.description);
}
```

Dentro do try pode inserir todo o código JavaScript que pretender.

CONCLUSÃO

Os objetos document e window têm atributos e métodos que são muito úteis ao criar páginas web interativas. Graças a estes objetos poderá aceder aos dados do utilizador, controlar a navegação, criar janelas, trabalhar com cookies, etc.

AUTOAVALIAÇÃO

1. **O que é que devolve a propriedade anchors do objeto document?**

   a) Nada, pois essa propriedade não existe.

   b) Um array com os links internos da web.

   c) Um array com os links externos da web.

   d) Um array com as imagens da web.

2. **Qual é a propriedade de document que contém o URL da página atual?**

   a) urlActual.

   b) urlString.

   c) urlNavegator.

   d) referrer.

3. **O que significa o quarto valor na propriedade readyState?**

   a) Carregamento.

   b) Não carregado.

   c) Carga total.

   d) Carregamento inacabado, mas interativo.

4. Qual é a função ou o método de document utilizado para escrever?

a) document.print.

b) response.write.

c) writeInDocument.

d) document.write.

5. Qual é a função do document que permite adicionar um pedaço de texto a um parágrafo já existente?

a) appendChild.

b) insertText.

c) InserAfterP.

d) InsertAfter.

6. Quais são as funções do seletor do document que permitem referir a um objeto pelo seu id ou pelo seu name?

a) getElementById e getElementsByName.

b) getElementById e getNameObject.

c) getObjectId e getObjectNAme.

d) Não é possível referir um objeto apenas pelo nome.

7. Qual é a função que devolve o próximo elemento ao qual é aplicada a função?

a) nextNode.

b) nextSibling.

c) nodeNext.

d) nodeAfter.

8. Qual é a função do dispatchEvent que todos os elementos HTML têm?

a) Saber se tem um evento atribuído.

b) Executar um evento criado dinamicamente.

c) Criar um evento para um elemento.

d) Não existe esse evento.

9. O que é que a propriedade closed do objeto window devolve?

a) true, se a janela estiver aberta.

b) true, se a janela estiver fechada.

c) true, se a janela possuir menu para fechar, maximizar e minimizar.

d) true, caso a janela não possua menu para fechar, maximizar e minimizar.

10. Qual é a propriedade do objeto window que permite alterar o texto da barra de status da janela?

a) setStatus.

b) status.value.

c) defaultStatus.

d) windowStatus.

11. Qual é a diferença entre as propriedades innerHeight e outherHeight do objetowindow?

a) innerHeight devolve a altura da janela incluindo os menus do navegador, enquanto outherHeight não.

b) outherHeight devolve a altura da janela incluindo os menus do navegador, enquanto innerHeight não.

c) innerHeight fornece a altura em píxeis e outherHeight em pontos.

d) Não diferem em nada.

12. Qual é a propriedade da window que devolve a referência da janela atual que abriu?

a) referrer.

b) parent.

c) opener.

d) after.

13. Qual é a propriedade utilizada para saber se o navegador tem os cookies ativados?

a) window.cookies.

b) navigator.cookiesPermis.

c) navigator.enabledCookies.

d) navigator.cookieEnabled.

14. Qual é a propriedade do objeto window que permite saber o número de frames que a janela contém?

a) length.

b) frames.

c) numFrames.

d) frames.length.

15. Qual é a função do objeto window utilizado para criar pop-ups ou novas janelas?

a) popup.

b) newWindow.

c) open.

d) opener.

16. Qual é a propriedade do objeto window que mostra uma mensagem com duas opções para escolher entre "Aceitar" e "Cancelar"?

a) window.alertOption.

b) window.mesage.

c) window.confirm.

d) window.select.

17. Qual é a propriedade do objeto window que permite modificar o tamanho da janela?

a) scrollTo.

b) moveTo.

c) changeSize.

d) resizeTo.

18. Qual é a função utilizada ao criar um cookie com uma data dinâmica que obtemos do JavaScript, para formatar a data?

a) setFormat.

b) dateFormat.

c) toString.

d) toGMTString.

19. Qual é a diferença entre setTimeout e setInterval?

a) setTimeout serve para remover um cronómetro, enquanto setInterval serve para configurá-lo.

b) setTimeout executa um código uma vez no período indicado e setInterval fá-lo infinitas vezes num intervalo de tempo predefinido.

c) setTimeout difere no formato em que o tempo que deve decorrer é enviado.

d) Ambos fazem o mesmo, mas em contextos diferentes.

20. As funções setTimeout e setInterval são definidas em:

a) Milissegundos.

b) Segundos.

c) Minutos.

d) A unidade que for mais oportuna.

SOLUÇÕES

1.	b	**2.**	d	**3.**	c	**4.**	d	**5.**	a
6.	a	**7.**	b	**8.**	b	**9.**	b	**10.**	c
11.	b	**12.**	c	**13.**	d	**14.**	a	**15.**	c
16.	c	**17.**	d	**18.**	d	**19.**	b	**20.**	a

PROPOSTAS DE DESENVOLVIMENTO DO ESTUDO

Para mais informações sobre os servidores web consulte os seguintes websites:

- https://developer.mozilla.org/pt-BR/docs/Learn/Common_questions/o_que_e_um_web_server
- https://www.apache.org/

BIBLIOGRAFIA

- MDN Web Docs (2021). "Referências JavaScript". Disponível em:

  https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference?retiredLocale=pt-PT. Consultado a 09 de fevereiro de 2021.

- VV. AA. (2010). *JavaScript*. Madrid: AnayaMultimedia.