

Aula 20: Cookies

Chegamos ao final de nosso curso! Nesta última aula explicaremos como usar uma propriedade do objeto `Document` que, certamente, você já deve ter encontrado em suas "navegações" na Web, uma propriedade tão importante que merece uma aula só para ela.

Objetivos:

Nesta aula você aprenderá:

- o que são **cookies** e quais suas **limitações**;
- **como usar, ler e armazenar cookies**;
- a usar as **funções** globais **escape** e **unescape**.

Pré-requisitos:

As aulas 4, 13, 14 e 16 são especialmente importantes para o bom entendimento desta aula.

1. Uma Visão Geral dos Cookies

Um problema existente na comunicação entre o cliente e o servidor é que, a princípio, não é possível para o servidor saber se é a **primeira vez** que um cliente solicita uma determinada página ou se ele **já visitou anteriormente** esta e outras páginas do site.

Cada **requisição de página é isolada, não mantendo relação com outras requisições**. Para que seja possível estabelecer uma ligação entre **várias conexões** ao servidor é preciso conhecer o **"estado"** atual do cliente. Estes estados podem ser: **nunca visitou, já visitou** a página que está pedindo ou **já visitou outra página do site**.

Para resolver este problema, um servidor, quando responde a um pedido, pode enviar uma informação que **será armazenada no cliente** solicitante. Numa próxima comunicação deste cliente, esta informação é enviada de volta para o servidor, permitindo que este possa conhecer então o **"estado"** do cliente. Esta informação inclui também um **prazo de validade** após o qual ela é esquecida pelo cliente.

Essa informação de estado que é **trocada** entre o cliente e o servidor é o que **chamamos de cookie**. Por que este nome tão estranho? Apesar de não ter muito sentido, ele tem sido usado historicamente em computação para **designar uma pequena quantidade de dados**, geralmente **privilegiados**.

Um `cookie` nada mais é que um **par do tipo nome=valor** e funciona como **uma variável** que o **servidor guarda no cliente**. O **nome** serve para o servidor indicar qual **informação que deseja recuperar**.

Por exemplo, se o servidor desejasse armazenar o **nome** do usuário que está **visitando o site** poderia guardar na máquina do cliente o cookie **“usuario=maria”**. Sempre que este **cliente fizesse um pedido** a este servidor enviaria também a informação **“usuario=maria”**.

A possibilidade de ter **algum dado do cliente** armazenado de **forma persistente** aumenta em muito a **capacidade das aplicações de redes** baseadas no modelo cliente/servidor. Este mecanismo simples se mostra uma **ferramenta bastante poderosa**, possibilitando o surgimento de aplicações interessantes, como por exemplo: **comércio eletrônico**. Sites de compras podem armazenar agora informação sobre a identificação do usuário, seus pedidos, suas preferências ou os artigos mais selecionados.

Esta **informação pode ser mandada de volta a cada conexão**, livrando o usuário de **fornecer de novo** sua identificação a cada contato. Os sites podem armazenar as **preferências do usuário no cliente**, e podem usá-las toda vez que ele se conectar ao site.

2. Cookies em JavaScript

A informação contida nos `cookies` pode ser manipulada pelas mais variadas tecnologias envolvidas em WEB. A linguagem **JavaScript** também fornece aos programadores este recurso, permitindo a sua manipulação através da **propriedade** `cookie` do objeto `Document`.

Na **tabela 16.1** vimos que o objeto `Document` tem a propriedade `cookie`, mas ela não foi discutida quando tratamos deste objeto. Como já dá para imaginar, esta propriedade permite que possamos **escrever e recuperar** as informações guardadas em um cookie.

O **tipo de dado** guardado nesta propriedade é **sempre uma *string***, mas quando estamos **escrevendo seu valor** devemos interpretá-la de forma diferente de quando o estamos **lendo**.

Quando **atribuímos um valor** a `document.cookie` estamos definindo apenas o valor de um **único** `cookie`.

Já quando **consultamos** `document.cookie` obtemos uma *string* com todos os **cookies visíveis por aquela página**, separados um do outro pelo caracter **ponto e vírgula, " ; "**.

A **sintaxe** que devemos utilizar para **definir o valor** de um `cookie` para posterior recuperação é:

```
document.cookie="nome=valor;EXPIRES=data;  
PATH=caminho;DOMAIN=domínio;SECURE"
```

Na linha acima, as palavras escritas em **maiúsculas** são **palavras reservadas** e devem ser respeitadas. Já as palavras escritas em **minúsculas** são **valores** definidos por quem está criando o `cookie`.

A sintaxe descrita acima define **cinco** tipos de informação associada a um `cookie` (os parâmetros ou atributos separados por ponto e vírgula).

O primeiro parâmetro é o `cookie` propriamente dito e é o **único obrigatório**.

Os outros quatro parâmetros controlam a **duração, visibilidade e segurança** do `cookie`. Vejamos, a seguir, o significado de cada um destes parâmetros.

2.1. O Atributo: `nome=valor`

A forma mais simples de criar na máquina do usuário um `cookie` identificado pela *string* `nome` é atribuir a `document.cookie` o par **nome=valor** (Sem especificar os demais parâmetros).

O exemplo a seguir mostra uma linha de código que definiria o `cookie` de nome `usuario` atribuindo a ele o valor `Maria`:

```
document.cookie = "usuario=Maria";
```

Se numa linha posterior aparecer novamente uma referência à propriedade `document.cookie`, desta vez atribuindo o valor “`codigo=123`” como mostrado no exemplo a seguir:

```
document.cookie = "codigo=123";
```

o cookie usuario **não seria destruído, permanecendo armazenado** com o valor Maria, e seria **guardado um novo** cookie de nome `codigo` e valor igual a 123.

No entanto, se algumas linhas adiante novamente for feita uma referência à propriedade `document.cookie`, desta vez atribuindo o valor “`codigo=490`” como mostrado no exemplo a seguir:

```
document.cookie = "codigo=490";
```

o cookie `codigo` seria **mantido**, só que agora com o **novo valor** 490 e **não mais** 123.

As linhas de código a seguir ajudam a entender o processo de **escrita e leitura** da propriedade `cookie`. Visualize o efeito delas em seu *browser*:

```
<html>
<head>
  <title>Criando Cookie simples</title>
</head>

<body bgcolor="tan">
<SCRIPT language="Javascript">
<!--
document.writeln('<B>Valor anterior de ',
                  'document.cookie = </B>''',
                  document.cookie, '"<p>');

// Inclui um valor
document.cookie = "produto=modelo12";
document.writeln('<B>Valor de ',
                  'document.cookie agora = </B>''',
                  document.cookie, '"<p>');

// Inclui mais um valor
document.cookie = "usuario=maria";
document.writeln('<B>Valor de ',
                  'document.cookie recém ',
                  'modificado= </B>''',
                  document.cookie, '"<p>');

// Apaga o primeiro valor
document.cookie = "produto=modelo34";
document.writeln('<B>Valor de ',
                  'document.cookie depois ultima ',
```

```
        'mudanca= </B>"', document.cookie,  
        '"<p>');  
//-->  
</SCRIPT>  
</body>  
</html>
```

Tanto o nome quanto o valor são formados por *strings* que **não podem conter** os caracteres **ponto-e-vírgula, vírgula, espaço em branco ou porcento, %**. Mesmo assim, eventualmente, pode haver a **necessidade** de colocar **tais dados no parâmetro**. Quando isto ocorre, eles devem ser substituídos por uma **seqüência codificada** na forma **%XX**, onde **XX** é o valor **hexadecimal do código ASCII do caracter que se deseja representar**.

Antes que você fique preocupado, imaginando que terá de fazer uma função em JavaScript para converter os caracteres **proibidos para a codificação hexadecimal**, saiba que esta função **já existe na linguagem** e falaremos dela mais adiante neste capítulo.

Por enquanto experimente incluir ";" no valor de algum dos `cookies` do exemplo anterior e **verifique o que vai ocorrer**.

Obs.: No **Firefox** apenas o **ponto-e-vírgula**, não pode ser incluído causando a **finalização** do parâmetro passado.

Nota: há um *bug* no Netscape Navigator versão 1.1 e anteriores.

Só `cookies` cujo atributo `path` seja explicitamente fixo para "/" são armazenados corretamente entre sessões, se eles tiverem ainda um atributo `expires`.

2.2. O Atributo: EXPIRES=DATE

Já falamos na seção anterior que basta definir o primeiro parâmetro (`nome=valor`) para criar um `cookie`. Porém, talvez você tenha notado que se abrir a página do exemplo e sair do navegador depois, quando abrir novamente o mesmo documento, os `cookies` que você definiu **anteriormente não aparecem mais definidos**. Isso ocorre pois, quando nada é dito ao contrário, um `cookie` somente é **armazenado durante a sessão do navegador em que foi criado**, ou seja, enquanto o *browser* estiver sendo executado.

Como muito provavelmente não é isso que você deseja, existe o **parâmetro opcional Expires**. Este atributo recebe como valor uma **data num formato pré-determinado**, com a **validade do cookie**. Uma vez alcançada esta data, o **navegador automaticamente apaga o cookie**.

A data deve ser formatada como: Wdy, DD-Mon-YYYY
HH:MM:SS GMT, onde:

- Wdy é o dia da semana em inglês.
- DD é o dia do mês.
- Mon é o nome do mês em Inglês.
- YYYY é o ano com 4 dígitos.
- HH é a hora.
- MM é o número de minutos.
- SS é o número de segundos.
- O único tempo legal é GMT.
- O **separador** entre os elementos da data deve ser **hífen**.

Antes de ficar preocupado imaginando que vai ter de fazer uma função para converter uma data para este formato, saiba que ela **já existe**. É um **método da classe** Date chamado toGMTstring().

2.3. O Atributo: PATH=CAMINHO

O atributo de path (caminho em inglês) é usado para especificar o subconjunto de URLs em um domínio para o qual o cookie é **válido**. Se você definir um valor para path, qualquer **página do mesmo domínio que tenha o mesmo caminho** (ou seja, esteja no mesmo diretório ou em um subdiretório) **poderá acessá-lo**.

Este atributo também é **opcional** e caso não tenha sido especificado será **assumido o caminho do documento** que criou o cookie. Isso é, ele estará **acessível à página que o criou e às outras que estiverem no mesmo diretório ou em algum subdiretório deste**. Normalmente, é o que se deseja.

Por exemplo, se o cookie foi criado pela página:

```
http://virtual.com.br/dir/index.htm
```

Ele será por *default* visível também na página (localizada no mesmo diretório):

```
http://virtual.com.br/dir/secretario.htm
```

E também na página (localizada num subdiretório):

```
http://virtual.com.br/dir/finan/index.htm
```

Mas não na página (não faz parte do caminho da página que criou o cookie):

```
http://virtual.com.br/index.htm
```

Se o path for definido como "/", o cookie será visível a todas as páginas do servidor `http://virtual.com.br`, pois este caminho é a raiz **da hierarquia de diretórios** (o diretório inicial), e todas as páginas do mesmo domínio estão em **subdiretórios** da raiz.

A forma de **execução da verificação** é a seguinte: quando um cookie já passou na comparação de domínio, então o componente *pathname* da URL é comparado com o atributo de caminho, e **se há um “casamento”**, o cookie é considerado **válido** e é enviado junto com o pedido de URL.

Quando forem enviados cookies a um servidor, todos os cookies com um caminho **mais específico** deveriam ser enviados antes de cookies com caminho **menos específico**. Por exemplo, um cookie "nome1=valor1" com um caminho que inicia em "/" deveria ser enviado depois de um cookie "nome1=valor2" com um caminho que inicia de **"/bar"**, se ambos devem ser enviados.

2.4. O Atributo: DOMAIN=DOMINIO

O valor *default* de domínio é o nome do *host* do servidor que contém a página que criou o cookie. Isto é, por padrão são acessíveis **somente às páginas do mesmo servidor** Web que as definiu.

Grandes *sites* podem, no entanto, querer compartilhar cookies entre seus servidores. Por exemplo, é o caso do servidor de: `pedidos.lojavirtual.com.br` precisar ler cookies de `catalogo.lojavirtual.com.br`. Neste caso o atributo `domain` deve ser definido como "lojavirtual.com.br" e o atributo `path` como "/", na página que criou o cookie em `catalogo.lojavirtual.com.br`, de modo que ele ficará disponível a todos os servidores do domínio `lojavirtual.com.br`. **Nunca** podem ser definidos domínios diferentes do domínio do servidor inicial.

A forma como isso é executado é: quando for procurando na **lista** de cookies **válidos**, uma comparação dos atributos de

domínio do `cookie` é feita com o **nome do domínio** na Internet do *host* no qual a URL foi buscada. Se há uma "identidade da terminação", então o `cookie` passará pela comparação do caminho, *path*, para ver se deverá ser enviado. "Identidade da terminação" significa que o atributo do domínio é **comparado com a terminação** para ter completamente qualificado o nome de domínio do *host*.

Assim, um atributo do domínio "lojavirtual.com.br" deve "casar" com os nomes de *host* "catalogo.lojavirtual.com.br" como também com "pedidos.lojavirtual.com.br".

Só *hosts* dentro do **domínio especificado** podem definir um `cookie` para um domínio. Domínios têm que ter pelo menos dois (2) ou três (3) pontos para prevenir comparações com domínios da forma: ".com", ".edu", e ".gov.br".

Relembrando domínio é um nome que serve para localizar e identificar conjuntos de computadores existentes na Internet, por exemplo: novahost.com.br.

O nome de domínio é o que vem depois de `www` no endereço de um site. O nome foi criado com o objetivo de facilitar a memorização dos endereços de computadores na Internet. Sem ele, teríamos que memorizar uma sequência grande de números.

Vimos isso na Aula 1!

Se o domínio estiver dentro de um dos **sete domínios de nível especial** só requer **dois pontos** apenas. Qualquer outro domínio requer **três pontos pelo menos**.

Os **sete domínios especiais** iniciais da WWW eram: "COM", "EDU", "NET", "ORG", "GOV", "MIL", e "INT".

Atualmente também se encontram domínios com as seguintes extensões: `aero`, `arpa`, `coop`, `.info`, `jobs`, `.biz`, `travel`

Em <http://www.iana.org/domains/root/db/> você encontra uma lista atual de todos os domínios reservados.

2.5. O Atributo: SECURE

O último atributo é um *booleano*, que especifica como os valores do `cookie` são transmitidos na rede.

Se `secure` **não é especificado**, o `cookie` é enviado mesmo nos canais inseguros, ou seja, é transmitido **normalmente em qualquer conexão HTTP**.

Se um `cookie` está especificado com `true` em `secure`, **só será transmitido se os canais de comunicação com o *host* forem seguros**, isto é se o *browser* e o servidor forem conectados via um protocolo seguro. Atualmente, isto significa que `cookies` seguros só serão enviados a servidores HTTPS (HTTP sobre SSL).

3. As Funções `escape` e `unescape`

As funções globais `escape(s)` e `unescape(s)`, **codificam ou decodificam** uma *string* `s` para transmissão.

O valor de retorno de `escape(s)` é uma cópia de `s` codificada, não sendo alterada a *string* `s` original.

A estrutura de codificação usada é:

Se algum caracter de `s` for

- um **espaço em branco**;
- um **símbolo de pontuação**;
- um caracter com **acento**; ou
- um caracter **diferente de letras ou números ASCII**

então este caracter será convertido para a forma `%XX`, onde `XX` são **dois dígitos em hexadecimal** que representam o código Latin-1 (ISO 8859-1) do caracter.

Por exemplo:

`"!"`, é codificado com `33` em Latin-1, que é o hexadecimal `21` ($2 \times 16 + 1 = 33$).

Espaço em branco é codificado como `%20`. Assim o resultado de:

```
escape("Bom dia!");  
será a string codificada: Bom%20dia%21
```

A utilidade desta função é **assegurar** que a *string* seja **portável** em todos os computadores e **transmissível** através de **toda a rede**, independentemente de como é utilizada em cada lugar (língua e país).

`unescape(s)` também é uma **função embutida** em JavaScript, e não um método de qualquer objeto.

Ela decodifica uma string codificada por `escape(s)`, retornando a *string* obtida através de **busca e substituição** de seqüências de caracteres na forma `%XX`, onde `XX` são dois dígitos em hexadecimal. Cada seqüência achada deste tipo é substituída pelo caracter Latin-1 cujo código corresponde a estes dígitos.

Assim,
`unescape("Bom%20dia%21")`
retornará
`"Bom dia!"`.

Vimos hexadecimal na aula 4!

O objeto `Date` foi visto na aula 14.

Como você deve lembrar a chamada a `Date()` sem argumentos cria um objeto do tipo `Date` definido com a hora e data do instante da chamada ao construtor.

Veja na mesma aula 14 como funcionam os demais métodos utilizados no exemplo ao lado.

4. Armazenando Cookies

Vamos fixar melhor o que já vimos, revendo os procedimentos para criação de um `cookie`.

Para associar o valor de um `cookie` ao documento corrente, simplesmente se define a propriedade `cookie` com uma *string* que tenha a forma `nome=valor`.

Por exemplo, as linhas abaixo dão ao nome `versao` o valor 2:

```
document.cookie="versao=2".
```

Assim, da próxima vez que se ler a propriedade `cookie`, o par `nome/valor` que foi armazenado será incluído na lista de `cookies` do documento.

Os valores dos cookies não podem incluir: `;`, `"`, `"`, `"` ou espaços em branco. Por isso pode ser necessário usar a função `escape(s)` para **codificar os valores antes de armazená-los** e a função correlata `unescape(s)` para **decodificá-los** quando forem lidos.

Assim, por exemplo, se fosse querido atribuir ao nome `versao` a data da última modificação do documento, seria possível usar:

```
document.cookie="versao="
    +escape(document.lastModified);
```

Desta maneira, porém, a **duração** do `cookie` seria a da **sessão atual**, sendo perdido quando o usuário saísse do *browser*.

Para criar um `cookie` mais **durável** deve-se incluir um valor adequado ao atributo `expires`: `expires=data`.

A data definida deve ser especificada na **forma do método** `toGMTString()` do objeto `Date`. Assim, por exemplo, para fazer um `cookie` **durar 3 meses**, você poderia usar as linhas de código que seguem:

```
//constroi-se a variavel com a data atual
var duraMes= new Date();
//ao valor do mes atual soma-se 3 meses
if (duraMes.getMonth()+3>12)
    duraMes.setMonth(duraMes.getMonth()+3)
else
{
    duraMes.setMonth(duraMes.getMonth()-9);
```

```
duraMes.setFullYear(duraMes.getFullYear+1);
}
document.cookie="versao=" +
    escape(document.lastModified) +
    ";expires=" + duraMes.toGMTString();
```

De maneira semelhante você pode definir os **outros atributos** (path, domain e secure) do cookie. Ou seja você irá **adicionando strings com os nomes:** ";path=valor_do_caminho;domain=valor_do_dominio,secure=false", onde, obviamente os valores valor_do_caminho e valor_do_dominio serão substituídos pelos conteúdos apropriados.

Para **mudar** os valores de um cookie, ele deve ser definido **novamente usando o mesmo nome com um novo valor**. O caminho e nome têm que “**casar**” **exatamente** para que o novo cookie substitua o antigo. Esta exigência torna difícil para **qualquer um que não o tenha criado, mudá-lo**.

Para **apagar** um cookie, ele deve ser **definido novamente**, com uma data de **expiração já passada**. Para isso, deve-se usar o mesmo nome, com um valor qualquer. O caminho e nome têm que "casar" **exatamente** para que o cookie expirado substitua o válido. Esta exigência torna difícil que o cookie **seja apagado por qualquer um que não o tenha criado**.

O *browser* não irá necessariamente apagá-lo de **imediato**, assim ele pode permanecer nos arquivos de cookie do *browser* depois da data de expiração.

Um cliente também pode apagar um cookie antes de sua data de vencimento chegar, se o número de cookies **exceder seus limites internos**.

5. Limitações

Os dados dos cookies são **transmitidos automaticamente entre browsers e servidores**. Sempre se destinam ao armazenamento de **pequeno volume de informação** e nunca para uma comunicação ou transferência freqüente de dados, de modo que devem ser usados moderadamente.

Há **limitações no número** de `cookies` que um cliente pode armazenar e receber.

Estes limites são:

- 300 `cookies` totais;
- 4 kilobytes por `cookie`;
- 20 `cookies` por servidor ou domínio.

Para este último limite, note que o *host* e os domínios são tratados como entidades separadas e tem cada um, uma limitação de 20 `cookies` para cada, e não combinadas.

Os servidores não esperam que os clientes excedam estes limites. Quando o limite de 300 `cookie` ou o de 20 `cookie` por limite de servidor é excedido, os clientes deveriam apagar `cookies` usados menos recentemente.

Quando um `cookie` maior que 4 kilobytes é encontrado, este `cookie` deve ser cortado para se ajustar, mas o nome deve permanecer intacto contanto que seu tamanho seja menor que 4 kilobytes.

Em geral, a mais restritiva das limitações acima é de 20 `cookies` por servidor. De modo que não é uma boa idéia usar `cookies` **separados para cada variável** que se deseja armazenar. Deve-se, ao **contrário tentar armazenar diversas variáveis em uma única denominação**.

Podem ser emitidos `cookies` **múltiplos** em uma única resposta do servidor. Instâncias com mesmo caminho e nome se sobre-escreverão (a mais recente tomando o lugar da outra instância, precedente). Instâncias com mesmo `path`, mas nomes diferentes serão adicionadas.

Fixando o `path` com um valor de alto-nível, não anula outras de caminho mais específicos. Se há múltiplos "casamentos" para um determinado nome do `cookie`, mas com caminhos separados, todos os "casamentos" serão enviados.

Estes métodos de `String` podem ser vistos na aula 13, tabela 13.1.

6. Lendo Cookies

O valor lido de um `cookie` pode não ser exatamente o mesmo escrito. Quando você usa a propriedade `cookie` em uma expressão da linguagem JavaScript, o valor retornado é uma *string* que **contém todos os cookies visíveis pelo documento corrente**.

Esta *string* é uma lista de pares `nome=valor` separados por ";", onde `nome` corresponde ao nome de um `cookie` e `valor` é uma *string*. Não são incluídos os atributos do `cookie`.

Para determinar o valor de um nome específico de interesse é necessário usar os métodos do objeto `String`:

- `String.indexOf()`;
- `String.substring()` ou
- `String.split()`

Estes métodos ajudam a **separar** a *string* que retornou em cada um dos `cookies` individuais.

Uma vez que o valor tenha sido extraído da propriedade `cookie`, ele deve ser **interpretado dependendo da forma como foi criado**. Por exemplo, se foi usado `escape()` na codificação, deve ser usado `unescape()` na decodificação.

A linha a seguir mostra como pode ser lida a propriedade `cookie`:

```
var todosOsCookies=document.cookie;
```

Nesta propriedade são retornados todos os `cookies` do documento. Depois disso dever-se **procurar** os `cookies` de interesse pelos seus nomes.

Suponha que você esteja interessado no de nome "versao", a forma de chegar até seu valor pode ser :

```
var pos=todosOsCookies.indexOf("versao=");
//obtem indice do inicio do cookie de
//nome "versao"
//se encontrar o cookie extrai seu valor
if(posicao!=-1)
{
  var inicio=posicao+7
  //inicio do valor
  var fim=todosOsCookies.indexOf(";", inicio);
  //obtem o indice final do valor
  if(fim== -1) fim=todosOsCookies.lenght
  //le a string com o conteudo do valor
  var valor=todosOsCookies.substring(inicio, fim);
```

```
//decodifica o valor
valor=unescape(valor);
//agora é possível usar o valor do cookie
}
```

Desta maneira é possível **obter o valor** de qualquer `cookie` desejado. É importante notar que os demais atributos do `cookie` **nunca podem ser lidos**. A *string* retornada quando a lemos `document.cookie` não contém nenhum dado dos atributos. Estas propriedades apenas podem ser definidas mas nunca lidas.

Exercícios:

1. Utilizando a idéia do exemplo apresentado no final da seção 4 e as linhas de código da seção 2.1. crie um `cookie` que dure por uma semana e visualize-o sempre que carregar a página.
2. Misture os métodos apropriados do objeto `String` para ler o valor de um `cookie` específico. Uma boa dica é usar as linhas de código da seção 6. Depois visualize seu resultado em um navegador.
3. Inclua no exercício 1 a possibilidade de remover o `cookie` de uma semana que foi criado.

Resumo:

Nesta aula você aprendeu a ler e escrever `cookies`. Uma importante característica dos navegadores, por isso merecedora de uma aula inteira! Agora sim você está pronto para ser um eficiente programador em JavaScript! Parabéns! Mas lembre-se: **nada está parado e especialmente em computação tudo sempre evolui**. Neste assunto aproveite sempre que possível para aprender mais um pouco. A própria WWW lhe fornece enormes possibilidades de novos aprendizados. Sempre que encontrar uma página interessante, veja seu código fonte, tente entendê-lo e aproveite para seguir as boas idéias!

Auto-avaliação:

Voltamos a dizer que a complexidade do assunto neste final do curso não deve assustá-lo. Não desanime e volte sempre a reler os pontos que ficaram um pouco obscuros. Boa sorte!