Lógica de Programação com JavaScript

Novas Tags HTML, Arrays e Estrutura Condicional

Novas Tags HTML

Até o momento, quando precisávamos obter alguma informação do usuário, usávamos o comando **prompt()** do JavaScript. Porém, podemos usar o HTML para criar uma interface com campos (como caixas de texto, botões e checkboxes) que o usuário pode utilizar para enviar informações para o nosso programa. Segue abaixo exemplos de código que utilizamos nas aulas:

```
...

<!-- Cria um campo de texto na interface -->
Nome completo:
<input type="text" id="campoNomeCompleto" />

<!-- Cria um campo de checkbox -->
Sim, eu concordo com os termos de uso
<input type="checkbox" id="campoAceitoTermos" />

<!-- Cria um botão -->
<button>Cadastrar</button>

...
```

Lembrando que usamos o atributo **"id"** para dar um nome à uma tag HTML que permita que o JavaScript a encontre quando ele precisar. No caso do exemplo, para que o JS pegasse o valor do campo **"Nome completo:"** escreveríamos o seguinte comando:

```
...
<script>

/* pega o valor do campo "campoNomeCompleto" e
   guarda na variável "nome" */

var nome = campoNomeCompleto.value;

</script>
```

Sempre que quisermos pegar o valor escrito num campo do tipo texto (type=”text”) usamos o comando **nomeDoCampo.value**, ok?

No caso do campo do tipo checkbox (type=”checkbox”), ele não tem propriamente um valor. Na realidade, queremos saber apenas se ele está marcado ou não. Para saber se o usuário marcou ou não um determinado campo desse tipo, usamos o comando **nomeDoCampoCheckbox.checked**, veja:

```
...
<script>

/* pega o estado do campo “campoAceitoTermos”, ou seja,
   marcado ou desmarcado e guarda na variável “aceito” */

var aceito = campoAceitoTermos.checked;

</script>
```

O comando **checked** do exemplo anterior retorna um valor do tipo *BOLEANO* (em inglês *BOOLEAN*) que é do tipo **TRUE** ou **FALSE** (*verdadeiro* ou *falso*).

Usamos esse tipo de dado quando precisamos guardar informações do tipo “SIM” e “NÃO”, como por exemplo se o cara marcou (SIM = **true**) ou não marcou (NÃO = **false**) o checkbox.

Escrevendo dentro de uma tag HTML

Até o momento, para escrever no documento HTML via JavaScript só utilizávamos o comando document.write(). Porém, há uma forma de escrever dentro de uma tag HTML específica. Ex:

```
...
<h1></h1>

<p></p>
```

No código anterior, criamos duas tags vazias: uma tag **<h1>** e uma tag **<p>**. Se quiséssemos, via JS, escrever algum texto dentro dessas tags já existentes é bem simples. Primeiro, basta adicionar um “id” para cada um deles. Veja:

```
...
<h1 id=”meuTitulo”></h1>

<p id=”meuParag”></p>
```

Em seguida, na tag **<script>**, basta utilizarmos o “id” atribuído às tags para podermos acessá-la e para escrever algo dentro dela, basta usarmos o comando **innerHTML**, confira:

```
...
<h1 id=”meuTitulo”></h1>

<p id=”meuParag”></p>

<script>

/* Escreve, dentro das tags, um texto específico */

meuTitulo.innerHTML = “Um título qualquer”;
meuParag.innerHTML = “Um texto qualquer”;

</script>
```

No exemplo de código anterior, quando a página é executada, o texto *“Um título qualquer”* é inserido na tag **<h1>** através do comando **innerHTML**. O mesmo ocorre com o texto *“Um texto qualquer”* que é inserido na tag **<p>** cujo “id” é “meuParag”.

Arrays

Até o momento conhecemos 3 tipos de dados no JS: dados do tipo texto – que identificamos usando aspas duplas (“ ”), aspas simples (‘ ’) ou, usando Template String, a crase (` `) –, dados do tipo número e dados do tipo boleano ou boolean (true/false).

Um quarto tipo de dado que existe no JavaScript (e em várias outras linguagens de programação também) é o tipo LISTA, também chamado de ARRAY. *Esse tipo permite gravar em memória uma lista de itens onde é possível também acessar cada um dos itens guardados separadamente se quisermos através de um **índice***. Veja:

```
...
<script>
```

```
/* criamos uma lista contendo as siglas dos
   estados da região sudeste */

var regiaoSudeste = [ “SP”, “MG”, “RJ”, “ES” ];

</script>
```

Lembrete: para criar um **array** (ou seja, uma lista de itens), usamos colchetes **[...]**. Assim sinalizamos para o JavaScript que o que queremos guardar na variável é um dado do tipo **array**.

No exemplo anterior, criamos um **array** com **4 itens** (as siglas dos 4 estados da região sudeste do Brasil). Se quiséssemos exibir, todos os valores de uma vez na tela, bastaria usarmos o comando document.write() que já conhecemos bem.

```
...
<script>

/* criamos uma lista contendo as siglas dos
   estados da região sudeste e exibimos na tela */

var regiaoSudeste = [ “SP”, “MG”, “RJ”, “ES” ];
document.write(`
      As siglas dos estados são: ${ regiaoSudeste } <br>
`);

</script>
```

A informação será exibida assim:

As siglas dos estados são: **SP,MG,RJ,ES**

Por padrão, quando pedimos para o JavaScript exibir o conteúdo de uma variável do tipo ARRAY, ele pega todos os itens dela e separa por vírgula. Mas fica tudo grudadinho, como você pode ver!

Se quisermos exibir todos os itens, mas de outra forma (um embaixo do outro, separados por vírgula com com um espaço, separados por barra, etc), podemos usar o comando **join()** para determinar como queremos agrupar os itens do array. Veja os exemplos abaixo:

```
...
<script>
```

```
/* criamos uma lista contendo as siglas dos
   estados da região sudeste e exibimos na tela */

var regiaoSudeste = [ "SP", "MG", "RJ", "ES" ];
document.write(`
    As siglas dos estados são: ${ regiaoSudeste.join(", ") }
    <br>
    As siglas com barras: ${ regiaoSudeste.join(" / ") }
    <br>
    As siglas com hífen: ${ regiaoSudeste.join(" - ") }
`);

</script>
```

O código anterior exibiria as informações assim na tela:

As siglas dos estados são: **SP, MG, RJ, ES**
As siglas com barras: **SP / MG / RJ / ES**
As siglas com hífen: **SP – MG – RJ – ES**

Veja que utilizamos a função **join()** para dizer ao JS como agrupar os itens da nossa lista antes de exibir. Automaticamente, ele sempre separa por vírgulas, mas deixa tudo grudado. No caso agora do nosso 1º exemplo, continuamos separando os itens por vírgula, mas adicionamos ao final um espaço o que fica melhor para visualizarmos.

Nos demais exemplos, pedimos para o JS usar **barras** e **hífen** como separadores, respectivamente.

Ainda fazendo uso do exemplo anterior, se quiséssemos acessar separadamente cada item da nossa lista (exemplo, exibir só a sigla de SP) bastaria usar o **índice** da informação que queremos.

Nossa lista, como dissemos antes, tem 4 itens. O primeiro item (SP) tem índice **0** e os demais seguem a ordem crescente do índice, ou seja:

0 = "SP"
1 = "MG"
2 = "RJ"
3 = "ES"

Lembrete: toda lista (array) sempre começa com índice ZERO. SEMPRE! Isto é, o primeiro item de um array é sempre o item de índice ZERO.

Veja o código abaixo:

```
...
<script>

/* criamos uma lista contendo as siglas dos
   estados da região sudeste e exibimos na tela */

var regiaoSudeste = [ “SP”, “MG”, “RJ”, “ES” ];
document.write(`
    As siglas dos estados são: ${ regiaoSudeste.join(“, ”) }
    <br>
    A sigla de São Paulo é: ${ regiaoSudeste[ 0 ] } <br>
    A sigla de Minas Gerais é: ${ regiaoSudeste[ 1 ] } <br>
    A sigla do Rio é: ${ regiaoSudeste[ 2 ] } <br>
    A sigla do Espírito Santo é: ${ regiaoSudeste[ 3 ] } <br>
`);

</script>
```

O código acima exibe o seguinte:

As siglas dos estados são: **SP, MG, RJ, ES**
A sigla de São Paulo é: **SP**
A sigla de Minas Gerais é: **MG**
A sigla do Rio é: **RJ**
A sigla do Espírito Santo é: **ES**

Perceba que utilizando os índices junto à sintaxe **variavelComArray[índice]** eu acessei cada uma das informações presentes na lista. No caso, **regiaoSudeste[0]** retorna “SP” porque a informação que está no índice 0 é “SP” (ou seja, o primeiro item). O mesmo ocorre com os outros itens quando eu acesso o respectivo índice junto a lista que os contém.

E se eu tentasse o seguinte comando:

regiaoSudeste[4];

Estou tentando acessar o índice **4** da minha lista **regiaoSudeste**, certo? Esse índice existe? **NÃO!!!** Porque nossa lista só vai até o índice 3 ou seja, (4 itens = índices: 0,1,2 e 3).

Só existiria um índice 4 se nossa lista tivesse 5 itens!

Quando tentamos acessar um índice INEXISTENTE numa lista (array) o JavaScript gera um erro (geralmente exibido no console do navegador) mostrando que estamos tentando encontrar um índice que não existe na nossa lista!

Facinho, não é?

Função split(): Transformando um texto numa lista

A função **split()** é usada quando queremos transformar um texto que tem algum tipo de SEPARADOR em uma lista (array). Vejamos alguns exemplos:

```
...
<script>

/* variáveis contendo textos */

var dataNascimento = “12/04/1991”;
var nomesConvidados = “Andreia;Marcia;Thaís”;

</script>
```

As variáveis acima são do tipo **texto**, pois a informação que elas guardam está entre aspas duplas (“ ”). Sendo assim, não dá pra eu acessar apenas uma parte da informação que ela guarda, como por exemplo, só o mês da data de nascimento ou só o nome “Thaís” da variável nomesConvidados.

Entretanto, podemos perceber que ambas as informações tem um SEPARADOR. A data de nascimento separa as informações com uma barra (“ / ”) e os nomes dos convidados são separados por um ponto-e-vírgula (“;”).

Quando temos uma informação do tipo **texto**, podemos transformá-la numa lista usando a função **split()** *DESDE QUE O TEXTO TENHA UM SEPARADOR QUE SERÁ USADO COMO REFERÊNCIA PARA FAZER O RECORTE DAS INFORMAÇÕES*.

Veja o código abaixo:

```
...
<script>
```

```
/* variáveis contendo textos */

var dataNascimento = “12/04/1991”;
var nomesConvidados = “Andreia;Marcia;Thaís”;

/* transformamos o conteúdo das variáveis de texto em lista (array)
   e armazenamos em novas variáveis */

var pedacosDataNascimento = dataNascimento.split(“/”);
var listaNomesConvidados = nomesConvidados.split(“;”);

document.write(`
      O mês de nascimento é: ${ pedacosDataNascimento[ 1 ] } <br>
      O último convidado é: ${ listaNomesConvidados[ 2 ] } <br>
`);

</script>
```

O código exibirá o seguinte:

O mês de nascimento é: **04**
O último convidado é: **Thaís**

O comando **split()** transforma o texto numa lista usando como referência para CORTAR AS INFORMAÇÕES EM PEDAÇOS um SEPARADOR COMUM. No caso da data de nascimento era a **barra** e no caso dos convidados era o **ponto-e-vírgula**.

Quando o **split()** criou os arrays, armazenou as listas nas suas respectivas variáveis que ficaram estruturadas assim:

Variável: pedacosDataNascimento
Tipo: array (lista)
Conteúdo:
[0] = 12
[1] = 04
[2] = 1991

Variável: listaNomesConvidados
Tipo: array (lista)
Conteúdo:
[0] = Andreia
[1] = Marcia
[2] = Thaís

Ou seja, após *"picotar"* a informação usando o separador informado, o **split()** pega cada pedaço e guarda numa lista, cada um numa posição. Aí é só usar os índices de cada posição para obter as informações separadamente (cada nome de convidado ou cada informação da data de nascimento: dia, mês ou ano).

Blz?

Estrutura Condicional e Operadores Lógicos

As estruturas condicionais nos permitem tomar decisões com base em alguma condição que precisa ser satisfeita. Ex:

> Exiba na tela "Olá, meu jovem!" SE a idade do usuário for menor que 18 anos.

Perceba que a mensagem "Olá, meu jovem!" só será exibida SE uma condição for atendida: a idade do usuário tem que ser menor que 18 anos.

Vejamos como poderíamos criar um programa que atendesse a essa receita:

```
...
<script>

/* variável "idade" recebe a idade do usuário */

var idade = prompt("Digite sua idade:");
/* verifica SE a variável "idade" tem um valor menor que 18 */
if ( idade < 18 ) {
    // se tiver, mostra a mensagem
    document.write("Olá, meu jovem!");
}
```

```
</script>
```

Utilizamos a cláusula **if (condição) { }** para verificar se a condição informada é atendida. Se for, o código que estiver entre as chaves do IF será executado.

Agora, digamos que queremos mostrar uma outra mensagem, como por exemplo “Olá, visitante!” para todos os outros usuários cuja idade seja maior ou igual a 18. O que poderíamos fazer?

Bom... poderíamos criar um outro IF para verificar isso, não poderíamos?

Veja:

```
...
<script>

/* variável “idade" recebe a idade do usuário */

var idade = prompt(“Digite sua idade:”);
/* verifica SE a variável “idade” tem um valor menor que 18 */
if ( idade < 18 ) {
    // se tiver, mostra a mensagem
    document.write(“Olá, meu jovem!”);
}
/* verifica SE a variável “idade” tem um valor maior ou igual a 18 */
if ( idade >= 18 ) {
    // se tiver, mostra a mensagem
    document.write(“Olá, visitante!”);
}

</script>
```

Funcionaria! Se o usuário digitasse 16 no prompt, a mensagem que seria exibida é a primeira, isto é: “Olá, meu jovem!”.

Se o usuário digitasse 22, a mensagem seria a segunda: “Olá, visitante!”.

Perceba que o segundo IF nada mais é que o EXATO contrário do primeiro, certo? Ou seja, pensando bem e revisitando nossa lógica ela poderia ser escrita assim: se a idade for menor que 18 eu quero que seja exibido “Olá, meu jovem!”, CASO CONTRÁRIO, ou seja, se a idade for qualquer coisa diferente de um número menor que 18 eu quero que exiba “Olá, visitante!”.

Nesse caso, apesar de não ser INCORRETO o que fizemos, o JS nos dá uma forma mais SIMPLES de criarmos essa estrutura utilizando a cláusula **else**. Veja:

```
...
<script>

/* variável “idade" recebe a idade do usuário */

var idade = prompt(“Digite sua idade:”);
/* verifica SE a variável “idade” tem um valor menor que 18 */
if ( idade < 18 ) {
    // se tiver, mostra a mensagem
    document.write(“Olá, meu jovem!”);
}
/* CASO CONTRÁRIO, ou seja, esse bloco de código só é executado
   se “idade” NÃO for menor que 18 */
else {
    // mostra outra mensagem
    document.write(“Olá, visitante!”);
}

</script>
```

A cláusula **else** realmente significa “caso contrário”, ou seja, se a condição anterior (idade < 18) não foi atendida, então quer dizer que idade NÃO É MENOR QUE 18 e neste caso, o **else** será executado mostrando a mensagem “Olá, visitante!”.

Lembrete: a cláusula **else** sempre nega o **IF** imediatamente anterior a ele, ou seja, ele representa O CONTRÁRIO do último IF executado.

Também podemos usar **else if** para encadear várias condições que tenham alguma correlação, como no exemplo utilizado em sala de aula. Veja:

Plano de Viagens

1) Entre 12 e 17 anos = Plano Teen

2) Entre 18 e 29 anos = Plano Adult

3) 30 anos ou mais = Plano Senior

4) Fora dessas faixas, mostrar: "Não há plano de viagens para sua faixa etária :("

Para essa receita (código) teríamos o seguinte código:

```
...
<script>

/* variável “idade" recebe a idade do usuário */

var idade = prompt(“Digite sua idade:”);
/* verifica SE a variável “idade” está entre 12 e 17 anos */
if ( idade >= 12 && idade <= 17  ) {
    // se tiver, mostra a mensagem
    document.write(“Seu plano ideal é o: Plano Teen! <br>”);
}
/* caso contrário, verifica SE a variável “idade” está
   entre 18 e 29 anos */
else if ( idade >= 18 && idade <= 29  ) {
    // se tiver, mostra a mensagem
    document.write(“Seu plano ideal é o: Plano Adult! <br>”);
}
/* caso contrário, verifica SE a variável “idade” é 30 anos ou mais*/
else if ( idade >= 30 ) {
    // se tiver, mostra a mensagem
    document.write(“Seu plano ideal é o: Plano Senior! <br>”);
}
/* CASO CONTRÁRIO, ou seja, se NENHUMA das condições
anteriores foi atendida */
else {
    // mostramos a mensagem de que não há planos
    document.write(“Não há planos para sua faixa etária :(! <br>”);
}

</script>
```

Se cada **else** nega o **IF** imediatamente anterior a ele, quando encadeamos vários **else if** para avaliar várias condições, o código correspondente a ele só será executado se o ANTERIOR NÃO tiver sido atendido.

Por exemplo, vamos imaginar que o usuário tenha digitado 13 no prompt que pede a idade. Sendo o valor digitado 13 o plano que será exibido é o

plano TEEN. Como a primeira condição foi satisfeita e a próxima possui um ELSE, o JS já sabe que não precisa entrar nela, porque o ELSE indica que aquele comando só deve ser considerado (avaliado) se o anterior NÃO foi atendido. Assim, o JS nem verifica mais os outros blocos contendo ELSE pois sabe que uma das condições do bloco já foi satisfeita, logo, ele não precisa ficar perdendo tempo avaliando as demais.

Se não usássemos o **else if** aqui mas apenas um **if** embaixo do outro, como no exemplo abaixo:

```
...
<script>

/* variável “idade" recebe a idade do usuário */

var idade = prompt(“Digite sua idade:”);
/* verifica SE a variável “idade” está entre 12 e 17 anos */
if ( idade >= 12 && idade <= 17  ) {
    // se tiver, mostra a mensagem
    document.write(“Seu plano ideal é o: Plano Teen! <br>”);
}
/* verifica SE a variável “idade” está entre 18 e 29 anos */
if ( idade >= 18 && idade <= 29  ) {
    // se tiver, mostra a mensagem
    document.write(“Seu plano ideal é o: Plano Adult! <br>”);
}
/* verifica SE a variável “idade” é 30 anos ou mais*/
if ( idade >= 30 ) {
    // se tiver, mostra a mensagem
    document.write(“Seu plano ideal é o: Plano Senior! <br>”);
}
/* CASO CONTRÁRIO, ou seja, se A CONDIÇÃO ANTERIOR NÃO foi
atendida */
else {
    // mostramos a mensagem de que não há planos
    document.write(“Não há planos para sua faixa etária :(! <br>”);
}

</script>
```

Usando o mesmo exemplo com idade igual a 13, aqui seriam exibidas duas mensagens:

> Seu plano ideal é o: Plano Teen!
> Não há planos para sua faixa etária :(!

Por quê?

O **else** nega o **if** IMEDIATAMENTE anterior a ele, lembra? Como aqui o **else** só aparece no final, ele SÓ será executado se a condição do if anterior a ele não for atendida. Se a idade é de 13 anos temos a seguinte situação:

1. O JavaScript encontra a primeira instrução **if** e a executa, pois idade é maior que 12 e menor que 17, logo ele mostra a mensagem: "Seu plano ideal é o: Plano Teen!"

2. Como a próxima instrução NÃO TEM ELSE, mesmo sabendo que a primeira já foi atendida ele ainda assim a verifica. Como não atende as condições (pois 13 não é maior que 18), o Plano Adult não é exibido. Se tivéssemos usado **else if** aqui, ele NÃO a verificaria, já que o **else** indica sempre que o JS só deve executar o que estiver ali SE a condição anterior não foi atendida, e como a anterior FOI atendida, não tem porquê ele verificar essa.

3. Finalmente o JS chega ao último **if** que verifica se a idade é maior ou igual a 30 anos. Lembre-se, aqui não tem **else if**, é só **if**. Ele executa e verifica que 13 não é maior que 30. Em seguida, ele encontra um **else** no final (que está associado só ao **if** anterior). Bom, como 13 não é maior que 30 e temos um **else**, logo o código do **else** será executado mostrando: "Não há plano para sua faixa etária :(!".

Por isso, em casos como o do exemplo anterior onde TEMOS VÁRIAS CONDIÇÕES ONDE UMA CONDIÇÃO SEMPRE ANULA A ANTERIOR, devemos usar o **else if** para encadear as anulações e verificações para que o código seja executado corretamente.

Operadores Lógicos: são símbolos que ajudam a indicar ao JS como ele deve avaliar uma condição. Veja abaixo a lista com os principais operadores lógicos:

&& **= E** : Indica a UNIÃO entre as condições. Ou seja, todas as condições unidades por ele devem ser atendidas.

Ex:

```
...
<script>

var animal1 = "Gato";
var animal2 = "Leão";
/* condicional com operador &&*/
if ( animal1 == "Gato" && animal2 == "Leão"  ) {
    // se tiver, mostra a mensagem
    document.write("Os animais são: FELINOS! <br>");
}

</script>
```

No exemplo anterior a mensagem "Os animais são: FELINOS!" só será exibida se, E SOMENTE SE as variáveis **animal1** for exatamente igual a "Gato" e **animal2** for exatamente igual a "Leão". Caso **animal1** fosse "Gato" e **animal2** fosse "Onça" a mensagem não seria exibida pois ela NÃO ATENDE a ambas as condições estabelecidas.

|| **= OU** : Indica que SE QUALQUER uma das condições agrupadas por esse operador for atendida, o código será executado.

Ex:

```
...
<script>

var animal1 = "Gato";
var animal2 = "Arara";
/* condicional com operador || */
if ( animal1 == "Gato" || animal2 == "Leão"  ) {
    // se tiver, mostra a mensagem
```

```
        document.write("Os animais são: FELINOS! <br>");
    }

    </script>
```

Neste exemplo, a mensagem "Os animais são: FELINOS!" será exibida SE AO MENOS UMA DAS CONDIÇÕES FOR ATENDIDA. Caso **animal1** fosse "Gato" e **animal2** fosse "Onça" a mensagem seria exibida, pois **animal1** atendeu a uma das condições. O mesmo aconteceria se **animal1** fosse "Papagaio" e **animal2** fosse "Leão". Apesar de **animal1** não atender a primeira condição, **animal2** atendeu à segunda e, portanto, a mensagem será exibida. O bloco só não será executado, se AMBAS NÃO FOREM ATENDIDAS, como por exemplo se **animal1** fosse "Arara" e **animal2** fosse "Cachorro".

! = **NÃO**: Indica que o JS deve verificar se o conteúdo da expressão é FALSE.

Ex:

```
    ...
    <script>

    var ehMenorDeIdade = false;
    /* condicional com operador ! (NÃO) */
    if ( !ehMenorDeIdade ) {
        // se for falsa, mostra a mensagem
        document.write("Você tem autorização para tirar CNH!");
    }

    </script>
```

Neste exemplo, a mensagem "Você tem autorização para tirar CNH!" só será exibida se o resultado da condição for **false**. Como o conteúdo da variável **ehMenorDeIdade** é **false**, então quer dizer que, hipoteticamente aqui, a pessoa NÃO É MENOR DE IDADE, certo? E é exatamente isso que o **if** está verificando: *SE NÃO É MENOR DE IDADE (!ehMenorDeIdade), MOSTRE A MENSAGEM "Você tem autorização para tirar CNH!".*

Simples, né?

Poderíamos inverter a condição e perguntar SE ELA É VERDADEIRA (true), para isso, basta usarmos o nome da variável direto, que no caso do exemplo anterior ficaria:

```
...
<script>

var ehMenorDeIdade = true;
/* condicional sem operador lógico */
if ( ehMenorDeIdade ) {
    // se for verdadeira, mostra a mensagem
    document.write("Você NÃO tem autorização para tirar CNH!");
}

</script>
```

Removemos o operador NÃO (!) e trocamos o valor da variável para **true**. Também adaptamos a mensagem que será exibida. Agora o **if** (sem o operador de negação), verifica se a variável ehMenorDeIdade é **true** (verdadeira) e se for, a mensagem "Você NÃO tem autorização para tirar CNH!" será exibida. Neste caso, como ela É IGUAL a **true**, a mensagem aparecerá na tela.

Uma terceira maneira de refazer esses exemplos seria utilizando o operador de comparação == **(IGUAL)** para comparar EXPLICITAMENTE se a variável CONTÉM O VALOR INDICADO. Veja:

```
...
<script>

var ehMenorDeIdade = false;
/* condicional com operador == (IGUAL) */
if ( ehMenorDeIdade == false ) {
    // se for IGUAL a false, mostra a mensagem
    document.write("Você tem autorização para tirar CNH!");
}

</script>
```

Exemplo com o TRUE:

```
...
<script>
```

```
var ehMenorDeIdade = true;
/* condicional com operador == (IGUAL) */
if ( ehMenorDeIdade == true ) {
    // se for IGUAL a true, mostra a mensagem
    document.write("Você NÃO tem autorização para tirar CNH!");
}
</script>
```

É isso!!! 😊