Aula 11: Desvios e Laços

Nesta aula explicaremos alguns comandos que podem alterar o fluxo dos seus programas em JavaScript. Você aprenderá a estrutura dos comandos de desvios e laços. Entenderá como funcionam os operadores `break`, `continue` e o operador ternário que mencionamos na aula passada.

Objetivos:
Aprender:
- o desvio condicional `if – else`;
- o que são blocos de comandos;
- como usar o operador condicional ternário;
- o funcionamento do comando `switch`;
- os laços `while`, `for` e `do – while`; e
- os comandos `break` e `continue`.

Pré-requisitos:
As aulas anteriores deste módulo.

1. Mudando o Fluxo do Programa

Você já deve ter notado que um programa na linguagem JavaScript é uma coleção de comandos para manipular variáveis e constantes. Estes comandos vão sendo executados na ordem em que aparecem. É possível mudar esta ordem usando as estruturas ou comandos de controle de fluxo. Este é o assunto desta aula, mas antes precisamos esclarecer uns detalhes sobre como você pode agrupar partes de código na linguagem.

1.1. Compondo Comandos em Blocos

Você já sabe que os comandos são separados por `";"` , o que até é opcional, mas há também uma forma de **combiná-los** em um único bloco, que passa a ter sintaticamente o comportamento de um **único comando**.

Isso é feito, simplesmente, delimitando os comandos que se deseja agrupar por **chaves**, **`{}`** . Esta combinação é muito freqüente em programas na linguagem JavaScript e sua sintaxe é mostrada a seguir:

```
{           comando;
            comando;
            ...
            comando;
}
```

2. Desvio Condicional

Um desvio condicional permite escolher qual comando (ou conjunto de comandos) será ou não executado de acordo com uma condição.

2.1. O Comando `if`

O `if` do JavaScript funciona da mesma maneira que o `if` do Pascal (do C , do Java e da maioria das linguagens) porém, sua sintaxe é um pouco diferente.

Esta sintaxe tem duas formas. A primeira é:

```
if (condição)
        comando;
```

Se você traduzir os termos usados em muitos comandos, (nas linguagens de computador para o português) vai facilmente entender seu funcionamento: neste caso, o **`if...else`**, poderia ser traduzido para **"se"... "se não"**, e o **`if then...else`** de Pascal, seria **"se então"... "senão".**

e a segunda forma é:

```
if (condição)
        comando;
    else
        comando;
```

Na primeira forma, a condição é avaliada. Se o seu resultado for `true`, ou puder ser convertido para `true`, o comando é executado. Se o resultado da condição for `false` ou puder ser convertido para `false`, o comando não é executado. Veja o exemplo:

```
if ( (estado==“RJ”) || (estado==“”) )
     cidade = “Rio de Janeiro”;
```

Na segunda forma, além do `if` é apresentado uma cláusula `else`, que só é executada no caso da condição ser falsa.

Como mencionado na seção 1.1, sempre, em qualquer uma das formas, é possível substituir um comando por um grupo de comandos em um bloco. Exemplificando:

```
if (hora < 12)
{
```

```
     manha = true;
     document.write ("bom dia!");
}
else
{
     manha = false;
      document.write ("boa tarde!");
}
```

A tabela 11.1 mostra um comparação da forma de escrever o `if` em JavaScript e Pascal.

Tabela 11.1 - Comparando `if`

Em JavaScript	Em Pascal

`if (nota >= 7)`
`{`
`  aprovado = true;`
`}`
`else`
`{`
`    aprovado = false;`
`}`
	`if nota >= 7 then`
	`begin`
	`  aprovado := true;`
	`end`
	`else`
	`begin`
	`    aprovado := false;`
	`end;`

Finalmente, se um dos comandos interiores for um outro `if` diz-se que estes são comandos `if aninhados` (`nested if` em Inglês).

```
if (hora < 12)
    if (pais=="Brazil")
        document.write ("bom dia!");
else
    manha = false;
```

No exemplo acima, o `if` mais interno é um comando do `if` mais externo. E neste caso pode não ficar claro, exceto pela endentação, para qual dos `if` associa-se o `else`. Como o interpretador JavaScript não vê a endentação, existe uma regra para estes casos que além de simples é a mesma da maioria das linguagens de programação: o `else` faz parte do `if` **mais próximo.**

Comparando esta regra com o que possivelmente se queria obter do trecho de programa acima, vê-se que o interpretador não iria fazer exatamente o que o programador do último exemplo desejava. Para fazer nestes casos exatamente o que

se deseja, deixando as coisas menos ambíguas, e mais legíveis, a melhor solução é usar **chaves**!

```
if (hora < 12)
{
    if ( (pais==“Brazil”)
        document.write (“bom dia!”);
}
else
    manha = false;
```

Outro comando `if` também pode aparecer como comando ou blocos de comando que segue o `else`, como no exemplo abaixo.

```
if (pais=="Brasil") diz="Oi!";
else
  if (pais=="Brazil") diz="Oi!";
    else
        if (pais=="USA") diz="Hi!";
        else
            diz="Ola!";
```

Não está havendo nada especial neste caso, a não ser que, como no exemplo acima, sempre a mesma variável é testada, o que possivelmente poderia sugerir que o mais eficiente seria o uso de um comando condicional especial: o `switch` (assunto da seção 2.3 a seguir).

2.2. O Operador Condicional Ternário

Um tipo especial de desvio condicional pode ser escrito de uma forma bastante compacta em JavaScript. Quando se utiliza um desvio condicional para determinar qual valor será atribuído a uma variável, é possível substituir o desvio pela utilização do operador condicional ternário (`? :`).

Assim , em situações como:

```
if (x == 20)
    y = 50;
else
    y = 70;
```

Pode-se ao invés do grupo `if...else` usar :

```
y = {(x == 20) ? 50 : 70};
```

Já falamos deste único operador que opera com 3 valores (operandos) na aula passada. Na verdade, cada um dos

operandos tem função diferente no comando. O primeiro, delimitado pelo ponto de interrogação, é uma expressão que deve ser avaliada. Se a avaliação for `true`, o operador terá como resultado o valor do segundo operando, que vai da interrogação até os dois pontos. Se a avaliação for `false`, o operador terá o valor do terceiro operando, que aparece depois dos dois pontos.

Assim: `x>0 ? x : -x`
sempre resultará no valor absoluto de `x`!

A tabela 11.2, para completar seu entendimento deste comando, mostra dois trechos de código equivalentes em JavaScript, o da esquerda utilizando o desvio condicional e o da direita o operador condicional ternário. Observe a tabela e veja se não foi economizado muito espaço com seu uso?

Tabela 11.2 - Comparando `if..else` e `? :`

Desvio condicional	Operador condicional

`if (a > b)`
`{`
`  maior = a;`
`}`
`else`
`{`
`  maior = b;`
`}`
	`maior = (a > b) ? a : b;`

2.3. O Comando `switch`

O comando `if` causa um desvio no fluxo do programa. Pode-se usar múltiplos comandos `if` como o do último exemplo da seção 2.1 para formar múltiplos desvios. No entanto, se todos os desvios dependem de uma mesma variável, o comando `switch`, que surgiu em JavaScript, a partir da versão 1.2, proporciona uma forma mais eficiente de manipular a situação.

O comando `switch` de JavaScript é semelhante ao de Java e C. Depois da palavra-chave `switch` segue **uma expressão** e **blocos de código rotulados** com a palavra-chave `case` seguida de um **valor** e **dois pontos**. Quando for executado, o comando `switch` calcula o valor da expressão, e, então, a compara com **cada valor** entre `case` e `:` . Quando um valor igual é encontrado, passa a executar o bloco de código que

segue os "`:`" . Se nenhum valor igual é encontrado, o fluxo do programa passa a executar a primeira linha de código que segue ao rótulo `default:` , ou se não há rótulo `default`, sai do `switch`, passando a executar a linha depois deste.

A funcionalidade deste comando é melhor compreendida através de exemplos. O código abaixo tem exatamente a mesma função do exemplificado no final da seção 2.1.

```
switch (pais)
{
    case "Brazil":
    case "Brasil":
        diz="Oi!";
        break;
    case "USA":
        diz="Hi!";
        break;
    default:
        diz="Ola!";
}
document.write(diz);
```

Você deve ter notado que apareceu a palavra `break` no final de alguns blocos `case`. O comando `break,` que veremos novamente na seção 4 dessa aula, causa uma mudança do fluxo do programa para o final do `switch`, isto é , passa a executar a linha que segue a " `}` ". Cada `case` no `switch` indica apenas o início do ponto para onde o fluxo do programa passará, eles não especificam o fim. Na ausência do comando `break,` o fluxo passa simplesmente para as linhas seguintes, o que pode até ser útil em algumas raras situações. Mas, em 99% dos casos, é melhor não deixar de finalizar cada `case` com um `break`.

Ponto-flutuante é o formato usado internamente pelo computador para representar números reais, isso é do tipo 3.1415 .

Embora o comando `switch` de JavaScript seja semelhante ao de Java e C, tem 3 diferenças importantes. Em primeiro lugar, nas outras duas linguagens só é possível usar valores inteiros para comparação. Em JavaScript, como você pode ter concluído pelo nosso exemplo, é possível usar valores *string*, inteiros, pontos-flutuantes e booleanos.

A segunda diferença tem a ver também com tipos de dados. Nas outras linguagens, que são altamente *tipadas*, todos os rótulos dos cases devem ser do mesmo tipo de dado. Em JavaScript não, cada `case` pode ser de um tipo diferente.

A terceira diferença é que os rótulos não precisam ser

constantes, e podem ser expressões compostas, desde que possam ser avaliadas durante a etapa de interpretação do código JavaScript (ou usando termos de informática em *tempo de interpretação*). Assim, são permitidos para rótulos:

```
case 0:
case 60*60*24:
case "Alo"+"gente":
case Number.POSITIVE_INFINITY:
```

3. Laços

Os laços permitem manter o fluxo do programa restrito a uma região até a ocorrência de alguma condição.

3.1. O Comando `while`

Permite repetir um bloco de comandos enquanto uma condição for verdadeira. É semelhante ao `while` do Pascal, apenas com uma sintaxe diferente. A tabela 11.3 mostra uma comparação entre JavaScript e Pascal.

Tabela 11.3 - Comparando `while`

Em Javascript	Em Pascal

`i = 0;`
`while (i < 20)`
`{`
`  i++;`
`}`
	`i := 0;`
	`while i < 20 do`
	`begin`
	`  i := i + 1;`
	`end;`

Forma geral:

```
while (condição)
{
    comandos;
    // executados enquanto
    // a condição for verdadeira
}
```

O `while` funciona primeiro avaliando a expressão, se ela for `false` o fluxo passa para o próximo comando do programa; mas se ela for `true`, o grupo de comandos do interior do `while` é executado e, no final, a expressão é avaliada novamente. E este processo se repete indefinidamente, até a expressão ser avaliada como falsa. A linha seguinte causaria um laço infinito!

Interação. (ou inter-ação) se refere à possibilidade de cada entrada provocar uma resposta. permitindo uma comunicação entre o usuário e o computador.

Iterativo é um processo que se repete. O trecho repetido é chamado de **iteração**.

Assim **interação** e **iteração** embora sejam palavras muito parecidas significam coisas bem diferentes!

```
while(true)
    document.write("sou o maior");
```

Usualmente não se deseja que a operação seja exatamente a mesma, de modo que geralmente uma ou mais variáveis geralmente são modificadas a cada iteração. Como no exemplo abaixo:

```
var i = 0;
while (i < 5) {
     document.write (i + "<BR>");
     i++;
}
```

É importante lembrar, finalmente, que se a expressão inicialmente não for avaliada como verdade, nunca o laço será executado.

3.2. Comando `do/while`

O comando `do/while`, que surgiu em JavaScript a partir da versão 1.2, tem comportamento idêntico ao comando de mesmo nome das demais linguagens: proporciona uma execução inicial do bloco de comandos do laço antes da comparação.

O comportamento deste é idêntico ao comando anterior, a não ser pelo fato de que o laço é executado pelo menos uma vez, já que a comparação só se realiza no final. Sua sintaxe é:

```
do
{
    comandos;
    // executados a primeira vez e
    // enquanto a condição for verdadeira
} while (condição);
```

O exemplo abaixo tem o mesmo comportamento do similar da seção anterior :

```
var i = 0;
do {
     document.write (i + "<BR>");
     i++;
} while (i < 5);
```

3.3. Comando `for`

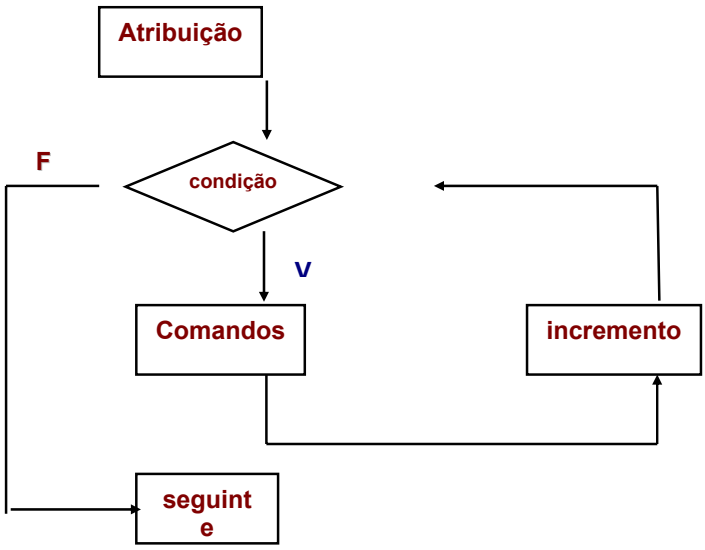
Permite repetir um bloco de comandos enquanto uma condição for verdadeira. Esta condição é controlada por uma variável, cuja inicialização e modificação (usualmente um incremento) estão previstas na própria sintaxe do comando. A sintaxe geral é mostrada a seguir:

```
for (atribuição; condição; incremento)
{
   comandos;// executados
            //enquanto a condição for
            // verdadeira
}
```

Ele unifica os comandos mais comuns em laços. A maioria dos laços tem uma variável que é inicializada antes da primeira iteração, que é testada antes de cada iteração e dependendo do resultado do teste faz com que o fluxo permaneça ou saia do laço. No final de cada iteração, a variável é modificada e o teste executado novamente para verificar se o fluxo continua no laço ou segue para a linha seguinte.

A figura 11.1 ilustra este processo:

Figura 11.1 - Estrutura funcional do `for`




```
for (i = 0; i < 5; i++)
     document.write (i + "<BR>");
```

A tabela 11.4 mostra uma comparação entre JavaScript e Pascal.

Tabela 11.4 - Comparando `for`

Em Javascript	Em Pascal

`for (i = 1; i <= 20; i++)`
`{`
`  soma += i;`
`}`
	`for i := 0 to 20`
	`do`
	`begin`
	`  soma := soma`
	`+ i;`
	`end;`

Cada uma das partes não precisa ser exatamente única, mais de uma variável pode ser inicializada, comparada, modificada por vez, desde que separadas por " , " . Também estes campos não precisam ter estes significados. O importante é que a atribuição é executada apenas uma vez, antes do início da execução do laço. A seguir, é feito o teste da condição. Se esta for falsa, o laço não é executado. Se for verdadeira, os comandos do interior do laço são executados, seguidos da execução do modificador e volta-se ao teste da condição.

```
document.write ("<h3>Tabela de Fatoriais
    e Contagem decrescente de 10</h3>");
for(i=1,j=10,fat=1;
    i<10;
    i++, j--, fat*=i)
    document.write(i+"!="+fat + j"<br>");
```

Na verdade, a vírgula, " , " , é um `operador` de JavaScript usado para combinar expressões. Ele avalia o argumento à sua esquerda, depois avalia o argumento da sua direita sucessivamente. Assim, a linha:

```
i=1,j=10,fat=1;
```

é equivalente às linhas:

```
i=1;
j=10;
fat=1;
```

Este operador é geralmente usado apenas nos laços `for`, como mostrado acima.

4. Comandos `break` e `continue`

O comando `break` permite a interrupção de um laço antes que a condição seja satisfeita. Este comando é utilizado após um desvio condicional que testa uma segunda condição para o fim do laço. No exemplo a seguir, o laço é executado 9

vezes, a menos que o usuário selecione cancelar na janela aberta pela função `confirm`:

```
for (i = 1; i < 10; i++)
{
    if ( !confirm("i = " + i +
        "\nDê um clique em Cancelar") )
    {
        break;
    }
}
alert("O laço terminou\n i = " + i);
```

O comando `continue` interrompe a iteração atual do laço, passando imediatamente à próxima iteração. Este comando também é utilizado após um desvio condicional. No exemplo a seguir, o número de vezes que o `write` será executado é determinado pelo número de vezes que o usuário selecionar `cancelar` na janela aberta pela função `confirm`. Se o usuário cancelar 4 vezes, o `write` será executado apenas 5 vezes:

```
var i = 1;
while (i < 10)
{
    i++;
    if (!confirm("i = " + i +
        "\n Clique cancelar para executar
         um continue"))
    {
        continue;
    }
    document.write("iteração " + i);
}
alert("O laço terminou\ni = " + i);
```

O comando `break` usado desta maneira é válido apenas no interior de laços ou com `switch`. O comando `continue` pode ser apenas usado no corpo de `laços`. Em outros casos causa erro de sintaxe.

Exercícios:

1. Utilize a estrutura do exemplo atividade da aula 9 para testar todos os operadores vistos nesta aula, trocando primeiro o comando `"for` com vírgulas`",` por um `"for"` que opera apenas uma variável por seção. Depois substitua este `for` por cada um dos outros comandos que permitem

mudar o fluxo do programa. Visualize em cada caso os resultados que terá.

2. Inclua em cada um dos casos do exercício anterior o cálculo de fatoriais dos números de 1 a 10 (se você não lembra como é feito, releia o exemplo apresentado no final da na seção 3.3. Comando `for`).

Resumo:

Nesta aula você aprendeu a usar os operadores da linguagem JavaScript que permitem desviar ou prender em um laço o fluxo de execução dos programas. Conheceu o operador condicional ternário e os comandos `break` e `continue`.

Auto-avaliação:

Você concluiu com facilidade os exercícios e entendeu bem os diversos comandos novos? Se algum ponto não ficou muito claro releia-o antes da próxima aula! Na qual você verá uma das coisas mais importantes para um programador em qualquer linguagem: como reaproveitar um trecho de código, ou, em palavras mais adequadas, como usar funções.