Aula 10: Operadores da Linguagem JavaScript

Nesta aula explicaremos como as expressões e os operadores funcionam em JavaScript. Você aprenderá os diversos tipos de operadores e como eles atuam nas variáveis em que operam. Entenderá como é feita a conversão de tipos implícita ou explicitamente. Será ainda apresentado a algumas funções que auxiliam a interação em seus programas.

Objetivos:
Aprender:
- Operadores: aritméticos, relacionais e lógicos
- Atribuição composta
- Conversão implícita e explícita de tipos, e
- Formas de entrada e saída

Pré-requisito:
A aula 9.

1. Introduzindo Operadores

A linguagem JavaScript oferece uma série de operadores para manipular variáveis e constantes. É possível misturar operandos de tipos diferentes que a linguagem se encarrega de fazer a conversão dos tipos. As tabelas 10.1 a 10.4 mostram os operadores disponíveis em JavaScript (que são semelhantes aos de C, C++ e Java) em comparação com os operadores oferecidos em Pascal.

Tabela 10.1 - Operadores Aritméticos

JavaScript	Pascal	Significado
`+ - * /`	`+ - * /`	Soma, Subtração, Multiplicação e Divisão
`%`	`mod`	Resto da divisão inteira
`-`	`-`	Inverte o sinal do número
`i++,++i`	`i = i + 1`	Incremento
`i--,--i`	`i = i - 1`	Decremento

Os operadores de incremento, `++`, são utilizados para somar 1 ao valor de uma variável. Os operadores de decremento, `--`, são utilizados para subtrair 1 de uma variável.

Se os operadores de incremento e de decremento aparecem no meio de uma expressão, o comportamento destes operadores depende de sua posição em relação ao operando.

Caso, qualquer um deles, aparecer **antes da variável**, é chamado de operador pré-incremental (ou pré-decremental) e **a operação** de incremento (ou decremento**) é realizada antes do cálculo do resto da expressão**, ou seja, o incremento (ou decremento) **altera** o resultado final da expressão.

Quando um desses operadores aparecer **depois da variável**, é chamado de operador pós-incremental (ou pós-decremental) e o **valor original da variável** é utilizado no cálculo da expressão. Somente depois a variável é incrementada (ou decrementada), ou seja, o incremento (ou decremento) **não altera** o resultado final da expressão.

As linhas de código que seguem geram a figura 10.1 e ajudam a entender isso.

```
<script language="javascript">
var i=4, j=4;
x = 2 * ++i;
document.write("<td>1=4<br>x=2*++i<p>");
document.write("x=",x,"<br>i=",i,"</td>");
x = 2 * j++;
document.write("<td>j=5<br>x=2*j++<p>");
document.write("x=",x,"<br>j=",j,"</td>");
</script>
```

Figura 10.1 - Diferenças entre pré-incremento e pós-incremento

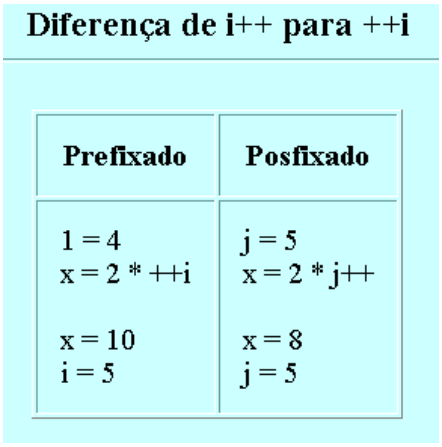


Tabela 10.2 - Operadores de Comparação

JavaScript	Pascal	Significado
> >=	> >=	Maior que, Maior e igual a
< <=	< <=	Menor que, Menor e igual a
==	=	Igual a
!=	<>	Diferente de

Tabela 10.3 - Operadores Lógicos

JavaScript	Pascal	Significado
!	not	Negação
&&	and	E
||	or	OU

A **álgebra booleana** ou lógica é baseada na idéia de que as operações algébricas podem ser expressas através de conceitos que podem ser avaliados como **verdadeiros** ou **falsos**.

Essa álgebra foi criada por Boole, no século 19, e, se adapta muito bem à forma digital de processamento dos computadores , que podem guardar o resultado destas operações em apenas 1 bit, geralmente 0 é associado a falso e 1 a verdadeiro.

Os operadores lógicos da tabela 10.3 são tipicamente empregados em operações da **álgebra booleana**. São usados freqüentemente junto aos de comparação para controle do fluxo dos programas. Quando operados com operando booleanos efetuam exatamente as operações esperadas desta álgebra. Assim:

- && , faz a operação de AND, só retornando *true* se ambos forem *true*, caso contrário retornará *false*.
- || , faz a operação de OR, só retornando *false* se ambos forem *false*, e retornando *true* caso contrário.
- operador ! atua invertendo o valor booleano do seu único operando.

Tabela 10.4 - Operador de Atribuição

JavaScript	Pascal	Significado
=	:=	Atribuição de valor a uma variável

O símbolo "=" já foi usado na aula passada para atribuir valores às variáveis. Embora não pensemos nele desta forma, o fato é que tecnicamente "=" é um operador. E, por isso, você pode incluí-lo em operações mais complexas como:

```
(a=b) == 0;
i = j = k = 0;
```

1.1. Operadores quanto ao Número de Operandos

Uma das coisas que distingue os operadores é o **número de operandos** com os quais eles operam. A maioria dos operados em JavaScript, como os das duas primeiras linhas da tabela 10.1, são **operadores binários**, isto é, combinam dois valores, ou em termos mais adequados, **operam com**

dois operandos.

Há também diversos operadores **unários**, isto é, que funcionam modificando apenas um valor. Um destes é o **operador menos**: `-` , que aparece na terceira linha da mesma tabela. Este operador, diferente do **operador subtração** (que por acaso é representado pelo mesmo símbolo), atua invertendo o sinal de apenas um número. A distinção entre quando o símbolo representa o **operador menos** ou o **operador subtração** é feita pela forma como o comando é representado. Por exemplo: `-x` representa o **operador menos**, e `x-y` representa o **operador subtração**.

Finalmente, em termos de número de operandos, há ainda um operador t**ernário**, herdado de C, o operador: `?:`. Este operador será visto nas próximas aulas de nosso curso.

1.2. Operadores quanto aos Tipos de Dados

Outros dois pontos importantes, aos quais se deve ficar atento, são os **tipos de dados** que podem ser combinados pelo operador e o tipo de dado que resulta da operação. Os operadores esperam atuar em tipos de dados específicos. Por exemplo, **todos os operadores da tabela 10.1, com exceção do +, esperam operar com números**: não é possível multiplicar (dividir ou subtrair) trechos de textos!

Assim `"x" * "y"` **não é uma expressão válida** em JavaScript. No entanto, uma característica desta linguagem é **converter expressões** para o tipo apropriado sempre que for possível. Desta forma, a expressão `"5" * "3"` é válida e tem como resultado o número `15` e não a string `"15".`

Além disso, alguns operadores se comportam de forma diferente dependendo do tipo de dado dos seus operandos. Como vimos na aula passada, o operador + também é utilizado em JavaScript para fazer a concatenação de strings. É possível ainda **misturar valores numéricos e strings** numa operação envolvendo o operador +. Neste caso, **os valores numéricos são convertidos para string, e o resultado da operação é a concatenação das duas em uma string**. No exemplo abaixo, a variável `data` recebe o valor `"15 de agosto"` e a variável `som` o valor `"55510":`

```
dia = 15;
data = dia + " de agosto";
x = "555";
som = x + 10;  // resulta "55510"
```

A conversão do valor numérico para string só é válida no

caso do operador +, nos demais casos, a string será convertida para um valor numérico, quando possível (quando não for possível será convertida para o valor numérico especial que vimos na aula passada: `NaN` - não é numérico). No exemplo abaixo a variável `sub` recebe o valor `45`:

```
x = "55";
sub = x - 10;   // resulta 45
```

Finalmente, quanto ao tipo de dados, é importante observar que o operador nem sempre produz como resultado (em termos de linguagens de computação: *retorna*) o mesmo tipo de dado dos seus operandos. Os operadores de comparação da tabela 10.2 operam com diversos tipos de dados, mas o resultado é sempre um valor booleano: `true` ou `false`. Por exemplo, a expressão: `a==3` tem como resultado `true` ou `false` (verdadeiro ou falso) dependendo do valor da variável `a` ser 3 ou não no momento em que foi feita a avaliação. Como veremos na próxima aula, estes operadores são muito usados nas estruturas de controle de fluxo do programa.

O adjetivo **booleano** pode ser usado em referência à teoria do matemático inglês Boole, ou como neste caso, em relação a uma variável que só pode assumir dois valores mutuamente exclusivos: **True** ou **False** (ou Verdadeiro ou Falso, ou ainda 0 e 1).

1.3. Conversão Implícita de Tipos

Da discussão de tipos iniciada na seção 1.2 observa-se que a linguagem tem formas de converter implicitamente os tipos de dados. Essas formas estão **embutidas** nas regras de funcionamento dos operadores, e é importante que a gente aprenda um pouco mais sobre elas.

A regra para os operadores aritméticos da tabela 10.1, a exceção do + , como já foi comentado, é sempre **"se usado com valor não numérico tente convertê-los para números antes".** Por exemplo, os operadores `-, *, /, %` que são binários, se operados em dois valores não numéricos tentam antes convertê-los para numérico e aí sim, subtraí-los, multiplicá-los, dividi-los ou verificar o resto da operação do primeiro operando pelo segundo. Os operadores aritméticos unários de inversão de sinal, `-`, incremento, `++`, e decremento, `--`, tentam fazer o mesmo com seu único operando.

Ainda falando dos operadores aritméticos, se você conhece algo de Pascal e C, é interessante lembrar aqui o que falamos na aula passada sobre os números em JavaScript serem **internamente ponto flutuante** e não inteiros por default como nestas linguagens. Assim, nas operações ligadas à divisão, `/` e `%` , o resultado será real e não inteiro. Por exemplo: `5/2` resultará `2.5` e não `2` como em C ou Pascal. E

`5 % 2` resulta `1`, mas `4.3 % 2.1` resulta `0.1`. O resultado da operação resto da divisão, `%`, em JavaScript pode ser positivo ou negativo: terá o sinal do primeiro operando.

O operador de comparação (tabela 10.2) que testa a igualdade de dois operandos, `==`, e retorna *true* ou *false*, pode operar com operandos de quaisquer tipo de dados e a definição do que é igual depende do tipo. Em JavaScript, **números, strings e booleanos são comparados pelos seus valores**. A operação "igual a" verifica se estes valores são idênticos. Assim, duas variáveis são avaliadas como iguais se seus valores são os mesmos. Este operador também pode trabalhar com **objetos, arrays e funções** (veremos nas próximas aulas o que são estes elementos) e neste caso é usado não os valores mas as referências. O que significa que dois *arrays* nunca serão iguais mesmo que tenham os mesmos elementos, já que são coisas distintas. Se você quiser saber se eles têm os mesmos elementos, deverá verificá-los um por vez e não usar o operador de igualdade.

Dois operandos que não têm o mesmo tipo de dado são comparados pelo operador "igual a" segundo as seguintes regras:

- Se um dos operandos é um número e o outro uma *string*, a *string* é convertida para **número** antes da comparação;
- O booleano *true* é convertido para o valor numérico **1** e o booleano *false* é convertido para o número **0**;
- Qualquer outra combinação de tipos de dados diferentes resulta em **não igual**!

Unicode é uma codificação de caracteres que inclui símbolos da maioria das línguas atualmente escritas no mundo. Os caracteres Unicode são armazenados em 2 bytes de modo que podem armazenar até 65.000 símbolos.

Por exemplo: `"1" == true` resulta igual, pois o booleano `true` é convertido para o número 1 pela regra 2 e, na segunda tentativa, a *string* `"1"` é convertida para o número `1`, pela regra primeira regra anterior.

O operador que testa a diferença, `!=` , é de fato a combinação do operador de negação `!` com o operador `==`. Assim, seu comportamento quanto ao tipo de dados é o equivalente ao dos dois operadores.

Os demais operadores de comparação da tabela 10.2 também sempre retornam valores *true* ou *false*. Embora possam operar com qualquer tipo de dado, a comparação internamente é apenas feita com *strings* ou números. Assim, se algum dos operandos não é *string* ou número, passa a ser convertido para um destes valores antes da comparação ser feita. Se depois da conversão ambos são números, a comparação é feita com seus valores numéricos. Caso ambos forem convertidos para *strings* serão comparados de acordo

com a ordem alfabética. Se um deles é *string* e o outro número, o operador tenta converter a *string* para número e fazer a comparação. Caso um deles não puder ser convertido para número ou *string,* o resultado da comparação será sempre *false*.

Latin-1 é uma forma de codificação de caracteres que usa 8 bits (1 byte), usada na Europa Ocidental, e padronizada pela ISO 8859-1.

No entanto, a ordem alfabética usada na comparação de *strings* é a da codificação *Unicode* (ou dos subconjuntos *ASCII* ou *Latin-1* em implementações não internacionalizadas). E, nesta codificação, as maiúsculas vêm antes das minúsculas, o que significa que o código destas é sempre maior que o daquelas. Assim "Ar" < "ar" será sempre verdade!

1.4. Atribuição Composta com Operação

A linguagem JavaScript, a exemplo de sua antecessora C, permite a escrita simplificada de expressões do tipo "a = a + b", na qual uma variável recebe o valor de uma expressão em que ela própria aparece. Esta expressão pode ser escrita como: "a += b". A tabela abaixo mostra algumas das combinações que podem ser realizadas:

Tabela 10. 5 - Operadores de atribuição e operação

Expressão	Simplificação
`a = a + b`	`a += b`
`a = a - b`	`a -= b`
`a = a * b`	`a *= b`
`a = a / b`	`a /= b`
`a = a % b`	`a %= b`

1.5. Precedência de Operadores

A precedência de operadores controla a ordem em que cada operação é feita quando mais de um operador aparece em uma expressão. A tabela a seguir mostra a ordem em que são avaliadas as expressões. Para alterar a precedência, é necessário usar parênteses.

Tabela 10.6 - Ordem de avaliação das expressões

1º	`! - ++ --`
2º	`* / %`
3º	`+ -`
4º	`< <= > >=`
5º	`== !=`
6º	`&&`
7º	`||`
8º	`?:`
9º	`= += -= *= /= %=`

Isso significa que o operador `*` tendo precedência maior que `+` faz com que, em uma expressão, a multiplicação seja efetuada primeiro que a adição. O operador de atribuição, =, tendo a precedência mais baixa faz com que o resultado só seja atribuído à variável do lado esquerdo em uma expressão, depois de completa toda a avaliação do lado direito do sinal de `=`.

Assim, o resultado da expressão abaixo será `7`:

```
w = 1 + 2 * 3;
```

Se você quiser forçar que a adição seja efetuada primeiro deve usar parênteses. Por exemplo: para que `w` seja `9`, a expressão deve ser escrita:

```
w = (1 + 2) * 3;
```

Em resumo, se na prática você tiver alguma dúvida sobre a precedência de operadores, a coisa mais simples a fazer é usar parênteses, para assim ter certeza de que a ordem da operação está bem explicitada.

2. Conversão Explícita de Tipos

Como já mencionamos, JavaScript é uma linguagem não "tipada", ou, talvez, expressando com mais correção, uma linguagem que é "tipada" dinamicamente. Isso significa que você não precisa ***declarar*** o tipo de um dado de uma variável antes de usá-lo (embora isso interfira no escopo do dado, como falaremos em outra oportunidade). Essa forma de tratar as variáveis permite que elas tenham uma flexibilidade e simplicidade que é muito desejada em uma linguagem de *scripts* (ao contrário das linguagens de programação voltadas para a elaboração de grandes programas e sistemas).

Esta flexibilidade está associada à conversão automática de tipos que JavaScript realiza nas operações (como já

comentamos, detalhadamente, nas seções anteriores desta aula). Mas, algumas vezes, é importante que esta conversão seja feita por você na forma que realmente você deseja. A linguagem lhe oferece algumas funções para isso. A seguir comentaremos duas funções predefinidas para conversão explícita de tipos e outras funções auxiliares na avaliação de expressões.

2.1. Função `parseInt`

Sintaxe: `parseInt(str)` ou `parseInt(str,base);`

Descrição: Converte a *string* `str` para um número inteiro. Opcionalmente, pode-se indicar a base em que deve ser interpretado o número contido na *string*. Se o parâmetro base não for especificado, assume-se a base 10.

Exemplos:

```
num = "3A";
x = parseInt(num);
y = parseInt(num,16);
```

2.2. Função `parseFloat`

Sintaxe: `parseFloat (str);`

Descrição: Converte a *string* `str` num número real.

Exemplos:

```
num = "3.4";
x = parseFloat (num);
```

2.3. Função `eval`

Sintaxe: `eval (str);`

Descrição: Efetua a avaliação da expressão contida na *string* `str`.

Exemplos:

```
expr = "x*2+5";
result = eval(expr);
```

2.4. Função `isNaN`

Sintaxe: `isNaN (valor);`

Descrição: Retorna "true" se o valor não for numérico.

Exemplos:

```
x = prompt("Entre um numero:","");
if (isNaN(x))  {  ...  }
```

3. Algumas Funções para Entrada e Saída

Antes de poder fazer programas mais complexos em JavaScript é conveniente conhecer algumas funções que permitem realizar entrada e saída de dados. Começaremos por duas que inclusive já usamos nos exercícios da aula passada e depois veremos outras também muito úteis que usaremos nos próximos exercícios.

3.1. Função `document.write`

Sintaxe: `document.write(string);`

Descrição: Escreve uma *string* na página em exibição pelo navegador.

Exemplo: `document.write("<H1>Esta é minha página</H1>");`

3.2. Função `alert`

Sintaxe: `alert(aviso);`

Descrição: Abre uma janela para exibir um aviso.

Exemplo: `alert("Você digitou um caracter inválido !");`

3.3. Função `prompt`

Sintaxe: `prompt(mensagem);`

Descrição: Abre uma janela para entrada de uma linha de

texto, exibindo a mensagem passada como parâmetro. A função produz como resultado (*retorna*) o texto digitado pelo usuário, que deve ser atribuído a uma variável.

Exemplo:

```
nome = prompt("Digite o seu nome");
```

3.4. Função `confirm`

Sintaxe: `confirm(mensagem);`

Descrição: Abre uma janela para exibir uma pergunta para o usuário. A função retorna verdadeiro ou falso de acordo com a resposta.

Exemplo: `if (confirm("Você quer mesmo sair da página ?"))return;`

Por enquanto não se preocupe muito com o `if(...) return` que apareceu neste último exemplo. Ele serve para estimular sua curiosidade para o conteúdo da próxima aula.

Os resultados de 3.2 a 3.4 são aberturas de janelas, como as mostradas nas figuras que seguem o exemplo atividade. Algumas têm apenas um botão além do texto, mas outras requerem uma entrada de *string* pelo usuário. Note que os textos mostrados nestes diálogos não são HTML e sim textos comuns. As únicas exceções serão os espaços, mudanças de linha (\n) e os outros caracteres de pontuação comentados na aula anterior. Ajustar o texto que aparecerá na forma desejada pode exigir algumas tentativas.

Exemplo Atividade:

As linhas de código que seguem mostram um programa JavaScript, embutido em uma página HTML, que usa todas as funções descritas na última seção.

```
<HTML>
  <HEAD>
    <TITLE>Curso de
    Constru&ccedil;&atilde;o de
    P&aacute;ginas WEB- Mod. 2
    </TITLE>
    <script language="Javascript">
    var nome=prompt("Qual seu nome?","");
    </script>
  </HEAD>
```

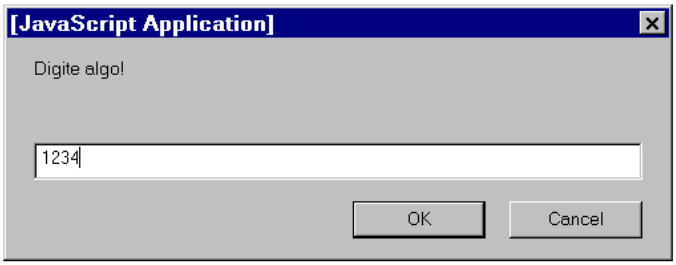
```
  <BODY>
    <H2 align=center >
    Segundo Exemplo de JavaScript
    </H2>
    <script language="Javascript">
    document.write("<h3>Bem-vindo"
    +nome+"!</h3>");
    n=prompt("Digite algo!"," ");
    alert("Iremos providenciar a
    avaliacao \n\n\n\t Espere um pouco!"+

    "\n_____________________________\n\
    n\n Paciencia!");
    var mensagem='voce digitou: \t'+n;
    if(confirm(mensagem))
    document.write("<h3>Acertei!</h3>");
    </script>
  </BODY>
</HTML>
```

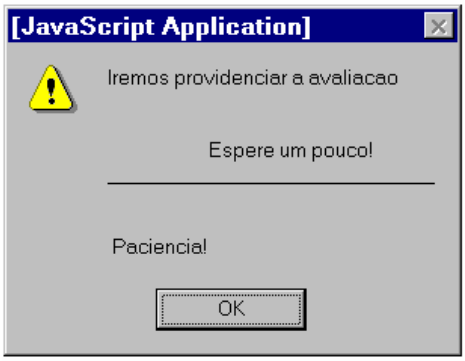
Bata estas linhas e carregue-as em uma página Web. Quando você carregar esta página em um navegador que interpreta JavaScript verá as telas mostradas a seguir. Observe que a palavra "JavaScript" aparecerá em todas as telas. Neste exemplo você deve ter entendido tudo o que bateu, exceto o `if(...)` que veremos posteriormente.

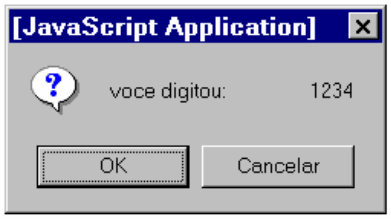
Figura 10.2 - Formas de diálogo geradas por:
`prompt(), aler()t e confirm()`

Segundo Exemplo de JavaScript

Bem-vindo Carlos !







Depois, olhando para o resultado obtido na página, tente mudar todos os avisos que aparecem nas janelas antes de passar aos exercícios.

Exercícios:

1. Utilize a estrutura do exemplo atividade da aula anterior para avaliar expressões aritméticas que combinem o maior número possível de operadores. Depois avalie expressões onde há conversão implícita de tipo de dados. Observe no exercício todos os detalhes comentados na seção 1. Finalmente, utilize de alguma forma no seu desenvolvimento cada uma das funções apresentadas na seção 2. Visualize em cada caso os resultados que terá.

2. Misture cada uma das operações que você deve ter feito no exercício anterior com as formas de interação do exemplo atividade deste capítulo. Ou seja, use `prompt`, `alert` e `confirm`, para incluir variáveis fornecidas pelo usuário nas avaliações das expressões.

Resumo:

Nesta aula, você aprendeu sobre os operadores da linguagem JavaScript. Conheceu a forma implícita de conversão de tipos e algumas formas novas de interação com o usuário. Fez uso delas escrevendo páginas com "interações dinâmicas". nas quais testou seus novos conhecimentos.

Auto-avaliação:

Você concluiu com facilidade os exercícios? Se não, sabe o que deve fazer, não é? Releia a aula e refaça os exercícios até conseguir responder afirmativamente a questão anterior. Depois disso estará preparado para a próxima aula, onde veremos como mudar o "fluxo" da execução de um programa!