Introdução ao Javascript

Javascript é uma linguagem de programação utilizada para criar pequenos programinhas encarregados de realizar ações dentro do âmbito de uma página web. Com Javascript podemos criar efeitos especiais nas páginas e definir interatividades com o usuário. O navegador do cliente é o encarregado de interpretar as instruções Javascript e executá-las para realizar estes efeitos e interatividades, de modo que o maior recurso, e talvez o único, com que conta esta linguagem é o próprio navegador.

Javascript é uma linguagem de programação bastante simples e pensada para fazer as coisas com rapidez, às vezes com leveza. Inclusive as pessoas que não tenham uma experiência prévia na programação poderão aprender esta linguagem com facilidade e utilizá-la em toda sua potência com somente um pouco de prática.

Entre as ações típicas que se podem realizar em Javascript temos duas vertentes. Por um lado, os efeitos especiais sobre páginas web, para criar conteúdos dinâmicos e elementos da página que tenham movimento, mudam de cor ou qualquer outro dinamismo. Por outro, javascript nos permite executar instruções como resposta às ações do usuário, com o qual podemos criar páginas interativas.

Javascript é uma linguagem com muitas possibilidades, permite a programação de pequenos scripts, e também de programas maiores, orientados a objetos, com funções, estruturas de dados complexas, etc. Toda esta potência de Javascript se coloca à disposição do programador, que se converte no verdadeiro dono e controlador de cada coisa que ocorre na página.

Diferenças Entre Java e Javascript

Que fique claro que **Javascript não tem nada a ver com Java**, salvo em suas origens. Atualmente são produtos totalmente distintos e não guardam entre si mais relação que a sintaxe idêntica e um pouco mais. Algumas diferenças entre estas duas linguagens são as seguintes:

- **Compilador**. Para programar em Java necessitamos um Kit de desenvolvimento e um compilador. Entretanto, Javascript não é uma linguagem que necessite que seus programas se compilem, senão que estes se interpretem por parte do navegador quando este lê a página.

- **Orientado a objetos**. Java é uma linguagem de programação orientada a objetos, Javascript não é orientado a objetos, isto quer dizer que poderemos programar sem necessidade de criar classes, assim como se realiza nas linguagens de programação estruturada como C ou Pascal.

- **Propósito**. Java é muito mais potente que Javascript, isto é devido a que Java é uma linguagem de propósito geral, com o que se podem fazer aplicações variadas, entretanto, com Javascript somente podemos escrever programas para que se executem em páginas web.

- **Estruturas fortes**. Java é uma linguagem de programação fortemente tipada, isto quer dizer que ao declarar uma variável teremos que indicar seu tipo e não poderá mudar de um tipo a outro automaticamente. Por sua parte, Javascript não tem esta característica, e podemos colocar em uma variável a informação que desejarmos, independentemente do tipo desta. Ademais, poderemos mudar o tipo de informação de uma variável quando quisermos.

- **Outras características**. Como vemos Java é muito mais complexo, mas também, mais potente, robusto e seguro. Têm mais funcionalidades que Javascript e as diferenças que os separam são o suficientemente importantes como para distinguí-los facilmente

Antes de Começar

Previamente para começar a utilizar Javascript podemos ter uma idéia mais concreta das possíveis aplicações desta linguagem assim como as ferramentas que necessitamos para colocarmos mãos a obra.

Usos de Javascript

Vejamos brevemente alguns usos desta linguagem que podemos encontrar na web para termos uma idéia das possibilidades que tem.

Para começar, podemos ver páginas cheia de efeitos super interessantes sobre Javascript, que chegam a assemelhar-se à tecnologia Flash.

Por outro lado, podemos encontrar dentro da Internet muitas aplicações de Javascript muito mais sérias, que fazem com que uma página web se converta em um verdadeiro programa interativo de gestão de qualquer recurso. Também podemos ver exemplos dentro de qualquer página um pouco mais complexa, quando passamos por sites que tenham uma calculadora ou um conversor de divisas, veremos que em muitos casos foi realizado com Javascript.

Na verdade é muito mais habitual encontrar Javascript para realizar efeitos simples sobre páginas web, ou não tão simples, como podem ser rollovers (que muda uma imagem ao passar o mouse por cima), navegadores desdobráveis, abertura de janelas secundárias, etc. Podemos nos atrever a dizer que esta linguagem é realmente útil para estes casos, pois estes típicos efeitos têm a complexidade justa para ser implementados em questão de minutos sem possibilidade de erros.

O que é necessário

Para programar em Javascript necessitamos basicamente do mesmo que para programar páginas web com HTML. Um editor de textos e um navegador compatível com Javascript. Um usuário de Windows possui de saída todo o necessário para poder programar em Javascript, visto que dispõe dentro de sua

instalação típica de sistema operativo, de um editor de textos, o Bloco de notas, e de um navegador: Internet Explorer.

Efeitos Rápidos com Javascript

Antes de continuarmos com a matéria, podemos ver uma série de efeitos rápidos que se podem programar com javascript. Isto, pode nos dar uma idéia mais clara e exata das capacidades e da potência da linguagem.

Abrir uma janela secundária

Primeiro vamos ver que com uma linha de Javascript podemos fazer coisas bastante atrativas. Por exemplo podemos ver como abrir uma janela secundária sem barras de menus que mostre o buscador Google. O código seria o seguinte:

```
<script>
window.open("http://www.google.com","","width=550,height=420,menubar=no")
</script>
```

Para ver o funcionamento basta incrementar esta linha em uma página, entre a tag <head> </head>

Uma mensagem de boas vindas

Podemos mostrar uma caixa de texto emergente ao terminar de carregar o portal de nosso site web, que poderia dar as boas vindas aos visitantes.

```
<script>
window.alert("Bem-vindo ao meu site web. Obrigado...")
</script>
```

Para ver o funcionamento basta incrementar esta linha em uma página, entre a tag <head> </head>

Botão de voltar

Outro exemplo rápido pode ser visto a seguir. Trata-se de um botão para voltar para trás, como o que temos na barra de ferramentas do navegador. Agora veremos uma linha de código que mistura HTML e Javascript para criar este botão que mostra a página anterior no histórico, se é que havia.

```
<input type=button value=Atrás onclick="history.go(-1)">
```

Esta linha deve ser inserida no corpo de uma página, se a mesma contiver um menu de navegação onde você possa ter um histórico dessa navegação, o botão criado, ao ser clicado deverá nos levar à página visitada anteriormente.

Como diferença com os exemplos anteriores, há que destacar que neste caso a instrução Javascript se encontra dentro de um atributo de HTML, onclick, que indica que essa instrução tem de ser executada como resposta ao clicar no botão.

A Linguagem Javascript

Javascript se escreve no documento HTML

O mais importante e básico que podemos destacar neste momento é que a programação de Javascript se realiza dentro do próprio documento HTML. Isto quer dizer que na página se misturam as duas linguagens, e para que estas duas linguagens possam ser misturadas sem problemas temos que incluir alguns delimitadores que separam as etiquetas HTML das instruções Javascript. Estes delimitadores são as etiquetas <SCRIPT> e </SCRIPT>. Todo o código Javascript que colocarmos na página há de ser introduzido entre estas duas etiquetas. Em uma mesma página podemos introduzir vários scripts.

Também se pode escrever Javascript dentro de determinados atributos da página, como o atributo *onclick*. Estes atributos estão relacionados com as ações do usuário e são chamados de manejadores de eventos.

Estas duas maneiras de escrever scripts, tem como diferença principal o momento em que se executam as sentenças.

Maneiras de Executar script.

Existem **duas maneiras de executar scripts** na página. A primeira destas maneiras se trata de execução direta de scripts, a segunda é uma execução como resposta à ação de um usuário. Veremos agora cada uma delas.

Execução direta

É o método mais básico de executar scripts. Neste caso se incluem as instruções dentro da etiqueta <SCRIPT> e </SCRIPT>.

Resposta a um evento

É a outra maneira de executar scripts, mas antes de vê-la devemos falar sobre os eventos. Os eventos são ações que realiza o usuário. Os programas como Javascript estão preparados para apanhar determinadas ações realizadas, neste caso sobre a página, e realizar ações como resposta. Deste modo se podem realizar programas interativos, já que controlamos os movimentos do usuário e respondemos a eles. Existem muitos tipos de eventos distintos, por exemplo, o click do mouse, a seleção de texto da página, entre outros.

As ações que queremos realizar como resposta a um evento devem ser indicadas dentro do mesmo código HTML, mas neste caso se indicam em atributos HTML que se colocam dentro da etiqueta que queremos que responda às ações do usuário. Vimos um exemplo rápido onde já comprovamos que se

quiséssemos que um botão realizasse ações quando se clicasse sobre ele, devíamos indicá-las dentro do atributo onclick do botão.

Comprovamos então, que se pode introduzir código Javascript dentro de determinados atributos das etiquetas HTML. Veremos mais adiante este tipo de execução e os tipos de eventos que existem.

Ocultar scripts em Navegadores Antigos

Nem todos os navegadores da web compreendem Javascript. Nos casos em não se interpretam os scripts, os navegadores assumem que o código destes é o texto da própria página web e como consequência, apresentam os scripts na página web como se tratasse de um texto normal. Para evitar que o texto dos scripts se escreva na página quando os navegadores não os entendem, temos que ocultá-los com comentários HTML. Vejamos com um exemplo como se deve ocultar os scripts.

```
<SCRIPT>
<!--
Código Javascript
//-->
</SCRIPT>
```

Vemos que o inicio do comentário HTML é idêntico a como o conhecemos no HTML, porém o fechamento do comentário apresenta uma particularidade, que começa por duas barras inclinadas. Isto é devido a que o final do comentário contém vários caracteres que o Javascript reconhece como operadores e ao tratar de analisá-los lança uma mensagem de erro de sintaxe. Para que o Javascript não lance uma mensagem de erro se coloca antes do comentário HTML essa barra dupla, que não é mais que um comentário Javascript, que conheceremos mais adiante quando falarmos de sintaxe.

O início do comentário HTML não é necessário comentá-lo com a barra dupla, dado que Javascript entende bem que simplesmente se pretende ocultar o código. Um esclarecimento a este ponto: se colocássemos as duas barras nesta linha, se veriam em navegadores antigos por estar fora dos comentários HTML. Os navegadores antigos não entendem as etiquetas <SCRIPT> , portanto não as interpretam, tal como fazem com qualquer etiqueta que desconhecem.

Sintaxe Javascript

A linguagem **Javascript tem uma sintaxe muito parecida a de Java** por estar baseado nele. Também é muito parecida a da linguagem C, de modo que se você conhece alguma destas duas linguagens poderá manejar com facilidade o código. De qualquer forma, vamos descrever toda a sintaxe com detalhes, para que os novatos não tenham nenhum problema com ela.

Comentários

Um comentário é uma parte de código que não é interpretada pelo navegador e

cuja utilidade é facilitar a leitura ao programador. O programador, a medida que desenvolve o script, vai deixando frases ou palavras soltas, chamadas comentários, que ajudam a ele ou a qualquer outro a ler mais facilmente o script na hora de modificá-lo ou depurá-lo.

Já foi visto anteriormente algum comentário Javascript, mas agora vamos revê-los. Existem dois tipos de comentários na linguagem. Um deles, a barra dupla, serve para comentar uma linha de código. O outro comentário podemos utilizar para comentar várias linhas e se indica com os signos /* para começar o comentário e */ para terminá-lo. Vejamos uns exemplos.

```
<SCRIPT>
//Este é um comentário de uma linha
/*Este comentário pode se expandir
por várias linhas.
As que quiser*/
</SCRIPT>
```

Maiúsculas e minúsculas

Em javascript se deve respeitar as maiúsculas e as minúsculas. Se nos equivocamos ao utilizá-las o navegador responderá com uma mensagem de erro de sintaxe. Por convenção os nomes das coisas se escrevem em minúsculas, salvo que se utilize um nome com mais de uma palavra, pois nesse caso se escreverão com maiúsculas as iniciais das palavras seguintes à primeira. Também se pode utilizar maiúsculas nas iniciais das primeiras palavras em alguns casos, como os nomes das classes, apesar de que já veremos mais adiante quais são estes casos e o que são as classes.

Separação de instruções

As distintas instruções que contém nossos scripts devem ser separadas convenientemente para que o navegador não indique os correspondentes erros de sintaxe. Javascript tem duas maneiras de separar instruções. A primeira é através do caractere ponto e vírgula (;) e a segunda é através de uma quebra de linha.

Por esta razão, as sentenças Javascript não necessitam acabar em ponto e vírgula a não ser que coloquemos duas instruções na mesma linha.

De qualquer forma, não é uma idéia ruim se acostumar a utilizar o ponto e vírgula depois de cada instrução, pois outras linguagens como Java ou C obrigam a utilizá-las e estaremos nos acostumando a realizar uma sintaxe mais parecida à habitual em torno de programações avançadas.

Variáveis em Javascript

Uma variável é um espaço em memória onde se armazena um dado, um espaço onde podemos salvar qualquer tipo de informação que necessitemos para realizar as ações de nossos programas. Por exemplo, se nosso programa realiza somas, será muito normal guardarmos em variáveis as distintas parcelas que

participam na operação e o resultado da soma. O efeito seria algo parecido a isto.

```
parcela1 = 23
parcela2 = 33
soma = parcela1 + parcela2
```

Neste exemplo temos três variáveis, parcela1, parcela2 e soma, onde guardamos o resultado
Os nomes das variáveis devem ser construídos com caracteres alfanuméricos e o caractere sublinhado (_). A parte disso, há uma série de regras adicionais para construir nomes para variáveis. A mais importante é que tem de começar por um caractere alfabético ou sublinhado. Não podemos utilizar caracteres raros como o signo +, um espaço ou um $. Nomes admitidos para as variáveis poderiam ser:

```
Idade
PaísDeNascimento
_nome
```

Também há que evitar utilizar nomes reservados como variáveis, por exemplo não poderemos chamar a nossa variável palavras como return ou for, que já veremos que são utilizadas para estruturas da própria linguagem. Vejamos agora alguns nomes de variáveis que não é permitido utilizar.

```
12meses
seu nome
return
pe%pe
```

Declaração de variáveis

Declarar variáveis consiste em definir e de passo informar ao sistema de que se vai utilizar uma variável. É um costume habitual nas linguagens de programação definir as variáveis que serão utilizadas nos programas e para isso, seguem-se algumas regras restritas. Porém, javascript ultrapassa muitas regras por ser uma linguagem mais livre na hora de programar e um dos casos no qual outorga um pouco de liberdade é na hora de declarar as variáveis, já que não estamos obrigados a fazê-lo, ao contrário do que acontece na maioria das linguagens de programação.

De qualquer forma, é aconselhável declarar as variáveis, além de um bom costume, e para isso Javascript conta com a palavra var. Como é lógico, utiliza-se essa palavra para definir a variável antes de utilizá-la.

```
var operando1
var operando2
```

Também pode-se atribuir um valor à variável quando se está declarando

```
var operando1 = 23
var operando2 = 33
```

Também se permite declarar várias variáveis na mesma linha, sempre que se separem por vírgulas.

```
var operando1,operando2
```

Âmbito das Variáveis

Chama-se âmbito das variáveis ao lugar onde estas estão disponíveis. Em geral, quando declaramos uma variável fazemos com que esteja disponível no lugar onde se foi declarado, isto ocorre em todas as linguagens de programação e como javascript se define dentro de uma página web, **as variáveis que declaremos na página estarão acessíveis dentro dela**. Deste modo, não poderemos acessar às variáveis que tenham sido definidas em outra página. Este é o âmbito mais habitual de uma variável e chamamos a este tipo de variáveis globais à página, mesmo que não seja o único, já que também poderemos declarar variáveis em lugares mais dimensionados.

Variáveis globais

Como dissemos, as variáveis globais são as que estão declaradas no âmbito mais amplo possível, que em Javascript é uma página web. Para declarar uma variável global à página simplesmente faremos em um script, com a palavra *var*.

```
<SCRIPT>
var variávelGlobal
</SCRIPT>
```

As variáveis globais são acessíveis desde qualquer lugar da página, ou seja, a partir do script onde foi declarado e todos os demais scripts da página, incluindo os que manejam os eventos, como o onclick, que já vimos que podia ser incluído dentro de determinadas etiquetas HTML.

Variáveis locais

Também poderemos declarar variáveis em lugares mais dimensionados, como por exemplo, uma função. A estas variáveis chamaremos de locais. Quando se declarem variáveis locais somente poderemos acessá-las dentro do lugar aonde tenha sido declaradas, ou seja, se havíamos declarado em uma função somente poderemos acessá-la quando estivermos nessa função.

As variáveis podem ser locais em uma função, mas também podem ser locais a outros âmbitos, como por exemplo, um loop. Em geral, são âmbitos locais qualquer lugar dimensionado por chaves.

```
<SCRIPT>
function minhaFuncao) (){
   var variavelLocal
}
</SCRIPT>
```

No script anterior declaramos uma variável dentro de uma função, pelo qual esta variável somente terá validade dentro da função. Pode-se ver as chaves para dimensionar o lugar onde está definida essa função ou seu âmbito.

Não há problema em declarar uma variável local com o mesmo nome que uma global, neste caso a variável será visível desde toda a página, exceto no âmbito onde está declarada a variável local já que neste lugar esse nome de variável está ocupado pela local e é ela quem tem validade. Resumindo, em qualquer lugar da página, a variável que terá validade é a global. Menos no âmbito onde está declarada a variável local, que será ela que vai ter validade.

```
<SCRIPT>
var numero = 2
function minhaFuncao (){
   var numero = 19
   document.write(numero) //imprime 19
}
document.write(numero) //imprime 2
</SCRIPT>
```

Um conselho para os principiantes poderia ser não declarar variáveis com os mesmos nomes, para que nunca tenha confusão sobre qual variável é a que tem validade em cada momento.

Diferenças entre utilizar var ou não

Como dissemos, em Javascript temos liberdade para declarar ou não as variáveis com a palavra var, mas os efeitos que conseguiremos em cada caso serão distintos. Na verdade, quando utilizamos var, estamos fazendo com que a variável que estamos declarando seja local ao âmbito onde se declara. Por outro lado, se não utilizamos a palavra var para declarar uma variável esta será global a toda a página, seja qual for o âmbito no qual tenha sido declarada.

No caso de uma variável declarada na página web, fora de uma função ou de qualquer outro âmbito mais reduzido, é indiferente se se declara ou não com var, desde um ponto de vista funcional. Isto é devido a que qualquer variável declarada fora do âmbito global a toda a página. A diferença pode ser apreciada em uma função por exemplo, já que se utilizamos var a variável será local à função e se não o utilizamos , a variável será global à página. Esta diferença é fundamental na hora de controlar corretamente o uso das variáveis na página, já que se não o fazemos em uma função poderíamos sobrescrever o valor de uma variável, perdendo o dado que pudesse conter previamente.

```
<SCRIPT>
var numero = 2
function minhaFuncao (){
   numero = 19
   document.write(numero) //imprime 19
}
document.write(numero) //imprime 2
```

```
//chamamos a funcao
minhaFuncao()
document.write(numero) //imprime 19
</SCRIPT>
```

Neste exemplo, temos uma variável global à página chamada numero, que contém um 2. Também temos uma função que utiliza a variável numero sem a ter declarado com var, pelo que a variável numero da funcao será mesma variável global numero declarada fora da função. Em uma situação com esta, ao executar a função se sobrescreverá a variável numero e o dado que havia antes de executar a função se perderá.

Tipos de dados

Em uma variável podemos introduzir vários tipos de informação, por exemplo, texto, números inteiros ou reais, etc. A estas distintas classes de informação conhecemos como tipos de dados. Cada um tem características e usos distintos, vejamos quais são os tipos de dados de Javascript.

Números
Para começar temos o tipo numérico, para salvar números como 9 ou 23.6

Cadeias
O tipo cadeia de caractere salva um texto. Sempre que escrevemos uma cadeia de caracteres devemos utilizar as aspas (").

Boleanos
Também contamos com o tipo boleano, que salva uma informação que pode valer como sim (true) ou não (false).

Na verdade nossas variáveis não estão forçadas a salvar um tipo de dados em concreto e portanto, não especificamos nenhum tipo de dados para uma variável quando a estamos declarando. Podemos introduzir qualquer informação em uma variável de qualquer tipo, inclusive podemos ir mudando o conteúdo de uma variável de um tipo a outro sem nenhum problema. Vamos ver isto com um exemplo.

```
var nome_cidade = "Salvador"
var revisado = true
nome_cidade = 32
revisado = "no"
```

Esta agilidade na hora de atribuir tipos às variáveis pode ser uma vantagem à princípio, sobretudo para pessoas inexperientes, mas a longo prazo pode ser uma fonte de erros já que dependendo do tipo que são as variáveis se comportarão de um modo ou outro e se não controlamos com exatidão o tipo das variáveis podemos encontrar um texto somado a um número. Javascript operará perfeitamente, e devolverá um dado, mas em alguns casos pode ser que não seja o que estávamos esperando. Sendo assim, mesmo que tenhamos liberdade com os tipos, esta mesma liberdade nos faz estar mais atentos a possíveis desajustes

difíceis de detectar ao longo dos programas. Vejamos o que ocorreria no caso de somar letras e números.

```
var parcela1 = 23
var parcela2 = "33"
var soma = parcela1 + parcela2
document.write(soma)
```

Este script nos mostraria na página o texto 2333, que não se corresponde com a soma dos dois números, e sim com sua combinação, um atrás do outro.

Em nossos scripts vamos manejar variáveis de diversas classes de informação, como textos ou números. Cada uma destas classes de informação é um tipo de dados. Javascript distingue entre três tipos de dados e todas as informações que se podem salvar em variáveis vão estar encaixadas em algum destes tipos de dados. Vejamos detalhadamente quais são estes três tipos de dados.

Tipo de dados numérico

Nesta linguagem só existe um tipo de dados numérico, ao contrário do que ocorre na maioria das linguagens mais conhecidas. Todos os números são portanto, do tipo numérico, independentemente da precisão que tenham ou se são números reais ou inteiros. Os números inteiros são números que não têm vírgula, como 3 ou 339. Os números reais são números fracionários, como 2.69 ou 0.25, que também se podem escrever em nota científica, por exemplo, 2.482e12.

Com Javascript também podemos escrever números em outras bases, como a hexadecimal. As bases são sistemas de numeração que utilizam mais ou menos dígitos para escrever os números. Existem três bases com as que podemos trabalhar:

- Base 10, é o sistema que utilizamos habitualmente, o sistema decimal. Qualquer número, por padrão, se entende que está escrito em base 10.
- Base 8, também chamado sistema octal, que utiliza dígitos do 0 ao 7. Para escrever um número em octal basta simplesmente escrever este número precedido de um 0, por exemplo 045.
- Base 16 ou sistema hexadecimal, é o sistema de numeração que utiliza 16 dígitos, os compreendidos entre o 0 e o 9 e as letras da A à F, para os dígitos que faltam. Para escrever um número em hexadecimal devemos escrevê-lo precedido de um zero e um xis, por exemplo, 0x3EF.

Tipo boleano

O tipo boleano, boolean em inglês, serve para salvar ou sim ou um não, ou com outras palavras, um verdadeiro ou falso. Utiliza-se para realizar operações lógicas, geralmente para realizar ações se o conteúdo de uma variável é verdadeiro ou falso.

Se uma variável é verdadeira, então: Executo umas instruções Se não Executo outras

Os dois valores que podem ter as variáveis boleanas são true ou false.

minhaBoleana = true
minhaBoleana = false

Tipo de dados cadeia de caracteres

O último tipo de dados é o que serve para salvar um texto. Javascript só tem um tipo de dados para salvar texto e nele, se podem introduzir qualquer número de caracteres. Um texto pode estar composto de números, letras e qualquer outro tipo de caracteres e signos. Os textos se escrevem entre aspas, duplas ou simples.

meuTexto = "Miguel vai pescar"
meuTexto = '23%%$ Letras & *--*'

Tudo o que se coloca entre aspas, como nos exemplos anteriores é tratado como uma cadeia de caracteres independentemente do que coloquemos no interior das aspas. Por exemplo, em uma variável de texto podemos salvar números e nesse caso temos que ter em conta que as variáveis de tipo texto e as numéricas não são a mesma coisa e que enquanto as de numéricas nos servem para fazer cálculos matemáticos, as de texto não servem.

Caracteres de escape em cadeias de texto.

Existe uma série de caracteres especiais que servem para expressar em uma cadeia de texto determinados controles como pode ser uma quebra de linha ou um tabulador. Estes são os caracteres de escape e se escrevem com uma nota especial que começa por uma contra-barra (uma barra inclinada ao contrário da normal '\') e logo se coloca o código do caractere a mostrar.

Um caractere muito comum é a quebra de linha, que se consegue escrevendo \n. Outro caractere muito habitual é colocar umas aspas, pois se colocamos umas aspas sem seu caractere especial nos fechariam as aspas que colocamos para iniciar a cadeia de caracteres. Temos então que introduzir as aspas com \" ou \' (aspas duplas ou simples). Existem outros caracteres de escape, que veremos na tabela abaixo mais resumidos, apesar de que também há que destacar como caractere habitual o que se utiliza para escrever uma contra-barra, para não confundi-la com o início de um caractere de escape, que é a dupla contra-barra \\.

Tabela com todos os caracteres de escape

Quebra de linha: \n
Aspas simples: \'
Aspas dupla: \"
Tabulador: \t
Enter: \r
Avance de página: \f

Retroceder espaço: \b
Contra-barra: \\

Alguns destes caracteres provavelmente não os chegará a utilizar nunca, pois sua função é um pouco rara e também, às vezes pouco clara.

Operadores

Ao desenvolver programas em qualquer linguagem se utilizam os operadores. Estes servem para fazer os cálculos e operações necessárias para realizar seus objetivos. Um programa que não realiza operações somente se pode limitar a fazer sempre o mesmo. É o resultado destas operações que faz com que um programa varie seu comportamento segundo os dados que obtenha. Existem operações mais simples ou mais complexas, que se podem realizar com operandos de distintos tipos de dados, como números ou textos.

Exemplos de uso de operadores

Antes de começar a numerar os distintos tipos de operadores vamos ver dois exemplos destes para ajudar a termos uma idéia mais exata do que são. No primeiro exemplo, vamos realizar uma soma utilizando o operador soma.

```
3 + 5
```

Esta é uma expressão muito básica que não tem muito sentido por si só. Faz a soma entre os dois operadores número 3 e 5, porém, não serve muito porque não se faz nada com o resultado. Normalmente combinam-se mais de um operador para criar expressões mais úteis. A expressão seguinte é uma combinação entre dois operadores, um realiza uma operação matemática e o outro serve para salvar o resultado.

```
minhaVariavel = 23 * 5
```

No exemplo anterior, o operador * se utiliza para realizar uma multiplicação e o operador = se utiliza para atribuir o resultado em uma variável, de modo que salvamos o valor para seu posterior uso.

Os operadores podem ser classificados segundo o tipo de ações que realizam. A seguir veremos cada um destes grupos de operadores e descreveremos a função de cada um.

Operadores aritméticos

São os utilizados para a realização de operações matemáticas simples como a soma, diferença ou multiplicação. Em javascript são os seguintes:

+ Soma de dois valores
- Diferença de dois valores, também se pode utilizar para mudar o sinal de um número se o utilizamos com um só operando -23
* Multiplicação de dois valores

/ Divisão de dois valores
% O resto da divisão de dois números (3%2 devolveria 1, o resto de dividir 3 entre 2)
++ Incremento em uma unidade, se utiliza com um só operando
-- Decremento em uma unidade, utilizado com um só operando

Exemplos
preço = 128 //introduzo um 128 na variável preço
unidades = 10 //outra atribuição, logo veremos operadores de atribuição
fatura = preço * unidades //multiplico preço por unidades, obtenho o valor fatura
resto = fatura % 3 //obtenho o resto de dividir a variável fatura por 3
preço++ //incrementa em uma unidade o preço (agora vale 129)

Operadores de atribuição

Servem para atribuir valores às variáveis, já utilizamos em exemplos anteriores o operador de atribuição =, mas existem outros operadores deste tipo, que provém da linguagem C e que muitos dos leitores já conhecerão.

= Atribuição. Atribui a parte da direita do igual à parte da esquerda. À direita se colocam os valores finais e à esquerda geralmente se coloca uma variável onde queremos salvar o dado.
+= Atribuição com soma. Realiza a soma da parte da direita com a da esquerda e salva o resultado na parte da esquerda.
-= Atribuição com diferença
*= Atribuição da multiplicação
/= Atribuição da divisão
%= Se obtém o resto e se atribui

Exemplos
poupança = 7000 //atribui um 7000 à variável poupança
poupança += 3500 //incrementa em 3500 a variável poupança, agora vale 10500
poupança /= 2 //divide entre 2 minha poupança, agora ficam 5250

Operadores de cadeias

As cadeias de caracteres, ou variáveis de texto, também têm seus próprios operadores para realizar ações típicas sobre cadeias. Apesar do javascript ter somente um operador para cadeias se podem realizar outras ações com uma série de funções pré-definidas na linguagem.

+ Concilia duas cadeias, pega a segunda cadeia a seguir da primeira.

Exemplo
cadeia1 = "ola"
cadeia2 = "mundo"
cadeiaConciliada = cadeia1 + cadeia2 //cadeia conciliada vale "olamundo"

Um detalhe importante que pode ser visto neste caso, é que o operador + serve para dois usos distintos, se seus operandos são números, os soma, mas se se trata de cadeias, as concilia. Isto ocorre em geral com todos os operadores que se repetem na linguagem, javascript é suficientemente esperto para entender que tipo de operação realizar mediante uma comprovação dos tipos que estão implicados nela.

Um caso que seria confuso é o uso do operador + quando se realiza a operação com operadores texto e numéricos misturados. Neste caso javascript assume que se deseja realizar uma conciliação e trata aos dois operandos como se tratasse de cadeias de caracteres, inclusive se a cadeia de texto que temos for um número. Isto veremos mais facilmente com o seguinte exemplo.

```
meuNumero = 23
minhaCadeia1 = "pedro"
minhaCadeia2 = "456"
resultado1 = meuNumero + minhaCadena1 //resultado1 vale "23pedro"
resultado2 = meuNumero + minhaCadeia2 //resultado2 vale "23456"
minhaCadeia2 += meuNumero //minhaCadena2 agora vale "45623"
```

Como podemos ver, também no caso do operador +=, se estamos tratando com cadeias de texto e números misturados, tratará aos dois operadores como se fossem cadeias.

Operadores lógicos

Estes operadores servem para realizar operações lógicas, que são aquelas que dão como resultado um verdadeiro ou um falso, e se utilizam para tomar decisões em nossos scripts. Ao invés de trabalhar com números, para realizar este tipo de operações se utilizam operandos boleanos, que conhecemos anteriormente, que são o verdadeiro (true) e o falso (false). Os operadores lógicos relacionam os operandos boleanos para dar como resultado outro operando boleano, tal como podemos ver no seguinte exemplo.

Se tenho fome e tenho comida, então irei comer

Nosso programa javascript utilizaria neste exemplo um operando boleano para tomar uma decisão. Primeiro irá ver se tenho fome, se é certo (true) irá ver se disponho de comida. Se os dois são certos, poderá comer. No caso de que não tenha comida ou de que não tenha fome não comeria, assim como se não tenho fome nem comida. O operando em questão é o operando Y, que valerá verdadeiro (true) no caso de que os dois operandos sejam verdadeiros.

! Operador Não ou negação. Se é true passa a false e vice-versa.
&& Operador E, se são os dois verdadeiros vale verdadeiro.
|| Operador Or, vale verdadeiro se pelo menos um deles for verdadeiro.

Exemplo

```
meuBoleano = true
meuBoleano = !meuBoleano //meuBoleano agora vale false
```

tenhofome = true
tenhoComida = true
comoComida = tenhoFome && tenhoComida

Operadores condicionais

Servem para realizar expressões condicionais mais complexas que desejarmos. Estas expressões se utilizam para tomar decisões em função da comparação de vários elementos, por exemplo, se um número é maior que outro ou se são iguais. Os operadores condicionais se utilizam nas expressões condicionais para tomar decisões. Como estas expressões condicionais serão objeto de estudo mais adiante será melhor descrever os operadores condicionais mais adiante. De qualquer forma, aqui podemos ver a tabela de operadores condicionais.

== Comprova se dois números são iguais
!= Comprova se dois números são distintos
> Maior que, devolve true se o primeiro operador for maior que o segundo
< Menor que, é true quando o elemento da esquerda for menor que o da direita
>= Maior igual.
<= Menor igual

Controle de Tipos

Vimos para determinados operadores que é importante o tipo de dados que estão manejando, visto que se os dados são de um tipo irão realizar operações distintas que se são de outro.

Assim, quando utilizávamos o operador +, se se tratava de números, os somava, mas se se tratava de cadeias de caracteres, os conciliava. Vemos então, que o tipo dos dados que estamos utilizando sim que importa e que teremos que estar pendentes a este detalhe se quisermos que nossas operações se realizem tal como esperávamos.

Para comprovar o tipo de um dado se pode utilizar outro operador que está disponível a partir de javascript 1.1, o operador typeof, que devolve uma cadeia de texto que descreve o tipo do operador que estamos comprovando.

```
var boleano = true
var numerico = 22
var numerico_flutuante = 13.56
var texto = "meu texto"
var data = new Date()
document.write("<br>O tipo de boleano é: " + typeof boleano)
document.write("<br>O tipo de numerico é: " + typeof numerico)
document.write("<br>O tipo de numerico_flutuante é: " + typeof
numerico_flutuante)
document.write("<br>O tipo de texto é: " + typeof texto)
document.write("<br>O tipo de data é: " + typeof data)
```

Este script dará como resultado o seguinte:

O tipo de boleano é: boolean
O tipo de numerico é: number
O tipo de numerico_flutuante é: number
O tipo de texto é: string
O tipo de data é: object

Neste exemplo podemos ver que se imprime na página os distintos tipos das variáveis. Estes podem ser os seguintes:

- boolean, para os dados boleanos. (True ou false)
- number, para os numéricos.
- string, para as cadeias de caracteres.
- object, para os objetos.

Queremos destacar apenas mais dois detalhes:

1) Os números, já tendo ou não parte decimal, são sempre do tipo de dados numéricos.

2) Uma das variáveis é um pouco mais complexa, é a variável data que é um objeto da classe Date(), que se utiliza para o manejo de datas nos scripts.

Estruturas de controle

Os scripts vistos até agora foram tremendamente simples e lineares: iam-se executando as sentenças simples uma atrás da outra desde o princípio até o fim. Entretanto, isto não tem porque ser sempre assim, nos programas geralmente necessitaremos fazer coisas distintas, dependendo do estado de nossas variáveis realizar um mesmo processo muitas vezes sem escrever a mesma linha de código uma e outra vez.

Para realizar coisas mais complexas em nossos scripts se utilizam as estruturas de controle. Utilizando-as podemos realizar tomadas de decisões e loops.

Tomada de decisões

Servem para realizar umas ações ou outras em função do estado das variáveis. Ou seja, tomar decisões para executar umas instruções ou outras dependendo do que esteja ocorrendo neste instante em nossos programas.

Por exemplo, dependendo se o usuário que entra em nossa página for maior de idade ou não, podemos lhe permitir ou não ver os conteúdos de nossa página.

Se idade é maior que 18 então:
  Deixo-lhe ver o conteúdo para adultos
Se não
  Mando-lhe fora da página

Em javascript podemos tomar decisões utilizando dois enunciados distintos.

- IF
- SWITCH

Loops

Os loops se utilizam para realizar certas ações repetidamente. São muito utilizados em todos os níveis na programção. Com um loop podemos por exemplo, imprimir em uma página os números de 1 ao 100 sem a necessidade de escrever cem vezes a instrução a imprimir.

```
Desde o 1 até o 100
  imprimir o número atual
```

Em javascript existem vários tipos de loops, cada um está indicado para um tipo de repetição distinto e são os seguintes:

- FOR
- WHILE
- DO WHILE

Como já assinalamos as estruturas de controle são muito importantes em Javascript e em qualquer linguagem de programação

Funções em Javascript

Agora veremos um assunto muito importante, sobretudo para os que nunca programaram e estão dando seus primeiros passos no mundo da programação com Javascript, já que veremos um conceito novo, o de função, e os usos que têm. Para os que já conhecem o conceito de função também será um capítulo útil, pois também veremos a sintaxe e o funcionamento das funções em Javascript.

O que é uma função

Na hora de fazer um programa levemente grande existem determinados processos que se podem conceber de forma independente, e que são mais simples de resolver que o problema inteiro. Ademais, estes costumam ser realizados repetidas vezes ao longo da execução do programa. Estes processos podem se agrupar em uma função, definida para que não tenhamos que repetir uma vez ou outra esse código em nossos scripts, e sim, simplesmente chamamos a função, e ela se encarrega de fazer tudo o que deve.

Portanto, podemos ver uma função como uma série de instruções que englobamos dentro de um mesmo processo. Este processo poderá ser executado somente ao ser chamado. Por exemplo, em uma página web pode haver uma função para mudar a cor de fundo e de qualquer ponto da página poderíamos chamá-la para que nos mude a cor quando desejarmos.

As funções utilizam-se constantemente, não só as que você escreve como também as que já estão definidas no sistema, pois todas as linguagens de programação têm um montão de funções para realizar processos habituais como, por exemplo, obter

a hora, imprimir uma mensagem na tela ou converter variáveis de um tipo a outro. Já vimos alguma função em nossos simples exemplos anteriores quando fazíamos um document.write() na verdade estávamos chamando à função write() associada ao documento da página que escreve um texto na página. Nos capítulos de funções vamos ver primeiro como realizar nossas próprias funções e como chamá-las.

Como se escreve uma função

Uma função deve-se definir com uma sintaxe especial que vamos conhecer a seguir:

```
function nomefuncao (){
  instruções da função
  ...
}
```

Primeiro escreve-se a palavra função, reservada para este uso. Seguidamente se escreve o nome da função, que como os nomes de variáveis podem ter números, letras e algum caractere adicional como um hífen abaixo. A seguir se colocam entre chaves as diferentes instruções da função. As chaves no caso das funções não são opcionais, ademais é útil colocá-las sempre como se vê no exemplo, para que seja visto facilmente a estrutura de instruções que engloba a função.

Vejamos um exemplo de função para escrever na página uma mensagem de boas vindas dentro de etiquetas <H1> para que fique mais ressaltado.

```
function escreverBoasvindas(){
  document.write("<H1>Olá a todos</H1>")
}
```

Simplesmente escreve na página um texto, é uma função tão simples que o exemplo não expressa suficientemente o conceito de função, mais já veremos outras mais complexas. As etiquetas H1 não se escrevem na página, e sim são interpretadas como o significado da mesma, neste caso que escrevemos uma cabeçalho de nível 1. Como estamos escrevendo em uma página web, ao colocar etiquetas HTML se interpretam como o que são.

Como chamar a uma função

Quando se chamam às funções: Para executar uma função temos que chamá-la em qualquer parte da página, com isso conseguiremos que se executem todas as instruções que tem a função entre as duas chaves. Para executar a função utilizamos seu nome seguido dos parênteses.

```
NomeDaFuncao()
```

Onde Colocamos as Funções

À princípio, podemos colocar as funções em qualquer parte da página, é claro que sempre entre etiquetas <SCRIPT>. Não obstante, existe uma limitação na hora de

colocá-la em relação aos lugares de onde for chamada. O mais normal é colocar a função antes de qualquer chamada à mesma e assim, certamente não iremos nos enganar.

Teoricamente, a função deve-se definir no bloco <SCRIPT> onde esteja a chamada à função, embora seja indiferente se a chamada se encontrar antes ou depois da função, dentro do mesmo bloco <SCRIPT>.

```
<SCRIPT>
minhaFuncao()
function minhaFuncao(){
   //faço algo...
   document.write("Isto está bem")
}
</SCRIPT>
```

Este exemplo funciona corretamente porque a função está declarada no mesmo bloco que sua chamada.

Também é válido que a função se encontre em um bloco <SCRIPT> anterior ao bloco onde está a chamada.

```
<HTML>
<HEAD>
  <TITLE>MINHA PÁGINA</TITLE>
<SCRIPT>
function minhaFuncao(){
   //faço algo...
   document.write("Isto está bem")
}
</SCRIPT>
</HEAD>
<BODY>

<SCRIPT>
minhaFuncao()
</SCRIPT>

</BODY>
</HTML>
```

Vemos um código completo sobre como poderia ser uma página web onde as funções estão no cabeçalho. Um lugar muito bom para colocá-las, porque se supõem que no cabeçalho ainda não vão utilizar e sempre poderemos desfrutar deles no corpo porque certamente já foram declarados.

Este último em compensação seria um erro:

O que será um erro é uma chamada a uma função que se encontra declarada em um bloco <SCRIPT> posterior.

```
<SCRIPT>
minhaFuncao()
</SCRIPT>

<SCRIPT>
function minhaFuncao(){
   //faço algo...
   document.write("Isto está bem")
}
</SCRIPT>
```

Parâmetros das Funções

As estruturas que vimos anteriormente sobre funções não são as únicas que devemos aprender para manejá-las em toda a sua potência. As funções também têm uma entrada e uma saída, que se podem utilizar para receber e devolver dados.

Parâmetros

Os parâmetros se usam para mandar valores à função, com os quais ela trabalhará para realizar as ações. São os valores de entrada que recebem uma função. Por exemplo, uma função que realizasse uma soma de dois números teria como parâmetros esses dois números. Os dois números são a entrada, assim como a saída seria o resultado.

Vejamos um exemplo anterior no qual criávamos uma função para mostrar uma mensagem de boas vindas à página web, mas que agora passaremos um parâmetro que vai conter o nome da pessoa a qual se vai saudar.

```
function escreverBoasvindas(nome){
   document.write("<H1>Ola " + nome + "</H1>")
}
```

Como podemos ver no exemplo, para definir na função um parâmetro temos que por o nome da variável que vai armazenar o dado que passarmos. Essa variável, que neste caso se chama nome, terá como valor o dado que passarmos a esta função quando a chamarmos, alé disso, a variável terá vida durante a execução da função e deixará de existir quando a função terminar sua execução.

Para chamar a uma função que tem parâmetros coloca-se entre parêntesis o valor do parâmetro. Para chamar à função do exemplo haveria que escrever:

```
escreverBoasvindas("Alberto Garcia")
```

Ao chamar à função assim, o parâmetro nome toma como valor "Alberto Garcia" e ao escrever a saudação na tela escreverá "Olá Alberto Garcia" entre etiquetas <H1>.

Os parâmetros podem escrever qualquer tipo de dados, numérico, textual, boleano ou um objeto. Realmente não especificamos o tipo do parâmetro, por isso devemos ter um cuidado especial ao definir as ações que realizamos dentro da função e ao passar valores à função para assegurarmos que tudo é coerente com os tipos de nossas variáveis ou parâmetros.

Múltiplos parâmetros

Uma função pode receber tantos parâmetros quanto quisermos e para expressá-lo colocam-se os parâmetros separados por vírgulas dentro dos parênteses. Vejamos rapidamente a sintaxe para que a função de antes receba dois parâmetros, primeiro, o nome a quem saudar e segundo, a cor do texto.

```
function escreverBoasvindas(nome,corTexto){
   document.write("<FONT color=" + corTexto + ">)
   document.write("<H1>Olá " + nome + "</H1>")
   document.write("</FONT>")
}
```

Chamaríamos à função com esta sintaxe. Entre parênteses colocaremos os valores dos parâmetros.

```
var meuNome = "Pedro"
var minhaCor = "red"
escreverBoasvindas(meuNome,minhaCor)
```

Coloquei entre parênteses, duas variáveis no lugar de dois textos entre aspas. Quando colocamos variáveis entre os parâmetros na verdade o que estamos passando à função são os valores que contém as variáveis e não as mesmas variáveis.

Parâmetros passam-se por valor

Para seguir a linha do uso de parâmetros em nossos programas Javascript, temos que indicar que os parâmetros das funções se passam por valor. Isto quer dizer que mesmo que modifiquemos um parâmetro em uma função a variável original que havíamos passado não mudará seu valor. Pode-se ver facilmente com um exemplo.

```
function passoPorValor(meuParametro){
   meuParametro = 32
   document.write("mudei o valor a 32")
}
var minhaVariavel = 5
passoPorValor(minhaVariavel)
document.write ("o valor da variavel e: " + minhaVariavel)
```

No exemplo, temos uma função que recebe um parâmetro, e que modifica o valor do parâmetro atribuindo-lhe o valor 32. Também temos uma variável, que iniciamos a 5 e posteriormente chamamos à função passando esta variável como parâmetro. Como dentro da função modificamos o valor do parâmetro poderia

acontecer da variável original mudasse de valor, mas como os parâmetros não modificam o valor original das variáveis, esta não muda de valor. Deste modo, ao imprimir na tela o valor de minhaVariavel se imprimirá o número 5, que é o valor original da variável, no lugar de 32 que era o valor que havíamos atualizado o parâmetro.

Em javascript somente se podem passar as variáveis por valor.

Valores de Retorno

As funções também podem retornar valores, de modo que ao executar a função poderá se realizar ações e dar um valor como saída. Por exemplo, uma função que calcula o quadrado de um número terá como entrada -tal como vimos- a esse número e como saída terá o valor resultante de encontrar o quadrado desse número. Uma função que devolva o dia da semana teria como saída em dia da semana.

Vejamos un exemplo de função que calcula a média de dois números. A função receberá os dois números e retornará o valor da média.

```
function media(valor1,valor2){
   var resultado
   resultado = (valor1 + valor2) / 2
   return resultado
}
```

Para especificar o valor que retornará a função se utiliza a palavra return seguida do valor que se deseja devolver. Neste caso se devolve o conteúdo da variável resultado, que contém a média dos dois números.

Para receber os valores que devolve uma função se coloca o operador de atribuição =. Para ilustrar isto, pode-se ver este exemplo, que chamará à função média() e salvará o resultado da média em uma variável para logo, imprimi-la na página.

```
var minhaMedia
minhaMedia = media(12,8)
document.write (minhaMedia)
```

Múltiplos return

Em uma mesma função podemos colocar mais de um return. Logicamente só vamos poder retornar uma coisa, mas dependendo do que tenha acontecido na função poderá ser de um tipo ou de outro, com uns dados ou outros.

Nesta função podemos ver um exemplo de utilização de múltiplos return. Trata-se de uma função que devolve um 0 se o parâmetro recebido era par e o valor do parâmetro se este era ímpar.

```
function multiploReturn(numero){
   var resto = numero % 2
```

```
    if (resto == 0)
      return 0
    else
      return numero
}
```

Para averiguar se um número é par encontramos o resto da divisão ao dividi-lo entre 2. Se o resto é zero é porque era par e devolvemos um 0, em caso contrário - o número é ímpar- devolvemos o parâmetro recebido.

Âmbito das variáveis em funções

Dentro das funções podemos declarar variáveis, inclusive os parâmetros são como variáveis que se declaram no cabeçalho da função e que se iniciam ao chamar a função. Todas as variáveis declaradas em uma função são locais a essa função, ou seja, somente terão validade durante a execução da função.

Podemos declarar variáveis em funções que tenham o mesmo nome que uma variável global à página. Então, dentro da função a variável que terá validade é a variável local e fora da função terá validade a variável global à página.

Em troca, se não declaramos as variáveis nas funções se entenderá por javascript que estamos fazendo referência a uma variável global à página, de modo que se não está criada, a variável a cria, mas sempre global à página no lugar de local à função.

Veja alguns exemplos de efeitos com javascript em:

http://www.cultura.ufpa.br/dicas/htm/htm-scrip.htm