

TUTORIAL

ASP

## O que é ASP?

Active Server Pages (ASP) é uma linguagem para geração de páginas HTML dinamicamente.

A partir de um Windows NT Server com o IIS3 ou IIS4 (Internet Information Server) instalado, é possível rodar códigos que geram páginas HTML dinamicamente, e então, mandá-las para o Browser. Assim sendo, sempre que uma página ASP for requisitada pelo Browser, esta página será executada pelo servidor, e a partir daí, será gerada uma página HTML, a qual será enviada para o Browser que a requisitou. É possível então, a partir de uma única rotina, gerar uma página HTML contendo os registros de um banco de dados, cujo o nome começa com a letra A. Ou então gerar uma página HTML contendo todos os registros cuja cidade é São Paulo. Detalhando um pouco mais o exemplo acima, um usuário qualquer entra na página principal da agenda virtual da sua empresa. Nesta página existem links para todas as letras do alfabeto e para todas as cidades listadas na agenda. Esse usuário clica no link na letra A (poderia ser de qualquer outra letra), e então uma página ASP (uma rotina ASP) é acionada no servidor. O servidor por sua vez, vai ler e executar esta página, o que significa que, em primeiro lugar, o servidor abrirá o banco de dados, ai então, fará uma consulta a este banco de dados requisitando todos os registros cujo nome comece com a letra A. Ai então o servidor cria uma página HTML contendo todos esses registros numa tabela.

A linguagem ASP não serve apenas para consultas a banco de dados. Serve também para envio e recebimento de correio eletrônico via páginas HTML, para criação rotinas de propaganda rotativa, para identificação e autenticação de usuários, para leitura de arquivos texto e uma infinidade de outras aplicações.

## VBScript e ASP

A linguagem ASP é, na verdade, uma junção de programação em VBScript e Objetos ActiveX.

VBScript é uma linguagem criada a partir do Visual Basic, mas com algumas limitações, por motivos de segurança.

Objetos ActiveX são objetos, ou melhor, funções prontas, já existentes no servidor. Essas funções captam os parâmetros de entrada dos dados, manipula-os de acordo a sua função e então envia-os para a saída. Um exemplo de ActiveX é o Objeto ASPMail, o qual capta os dados de entrada (nome, email, corpo da mensagem, etc), cria um email com esses dados e depois envia o email.

Uma vez que a linguagem ASP é usada apenas em alguns "pedaços" de páginas HTML, existe um símbolo para diferenciar o que é programação HTML e o que é programação ASP: <% código asp %>

Abaixo um exemplo de página em HTML e ASP:

```
<head>
<title>página em HTML e ASP</title>
</head>
<html>
<p>Olá Mundo!!!</p>
```

```
<%  
' (aspas simples significa observação dentro do código)  
' a linha abaixo tem o mesmo efeito da linha acima  
response.write "<p>Olá Mundo</p>"  
%>  
</html>
```

## Funcionamento, Convenção e Simbologia

Como Funciona o ASP? Qual a simbologia e a convenção usada na programação em ASP?

ASP é uma linguagem que veio para complementar a linguagem HTML. Ela traz para a linguagem HTML todo o poder do acesso a banco de dados, do acesso a arquivos texto, da captação de informações de formulário, da captação de informações sobre o visitante e sobre o servidor, do uso de variáveis, loops. Observe o código abaixo:  
arquivo "olamundo.asp"

```
<html>  
<head><title>Como funciona o ASP?</title></head>  
<body bgcolor="#FFFFFF">  
<p>Ola Mundo!!!</p>  
<%  
response.write "<p>Ola Mundo!!!</p>"  
%>  
<p><%= "Ola Mundo!!!" %></p>  
</body></html>
```

### [clique aqui para rodar o código acima](#)

O código acima exibe a frase Ola Mundo!!! de três modos diferentes. O primeiro, usando apenas programação HTML, o segundo, usando apenas comandos ASP, e o terceiro, é uma mistura de código HTML com ASP. Como se percebe, não é necessário construir a página inteira com códigos ASP. É possível criar páginas onde há a mesclagem das duas programações. A única exigência é que, a partir do momento em que o código da página tenha pelo menos uma linha na linguagem ASP, a terminação do nome da página deve ser .asp. Somente assim o servidor poderá distinguir quais páginas devem ser executadas antes de enviar ao Browser. Abaixo segue a simbologia e as convenções de uso da linguagem ASP.

<% -> início do trecho de código ASP

%> -> final do trecho de código ASP

' (**aspas simples**) -> usada antes de comentários dentro do código

= -> é usado no modo de programação por mesclagem de códigos HTML e ASP.

A linguagem ASP tem uma estrutura muito bem elaborada, uma vez que ela pode ser implementada com os chamados Objetos. A linguagem ASP, além de trazer todo o poder do VBScript para a página HTML, traz ainda o poder dos Componentes ActiveX e dos Objetos. Os componentes ActiveX, simplificando bastante, são bibliotecas que contém vários objetos reunidos. Principalmente objetos que necessitam de instâncias para funcionar, o que significa que, por exemplo, cada acesso a um bando de dados deve ocupar uma instância daquele objeto. Objetos

são add-ons que dão a capacidade de captura e transmissão de variáveis entre as páginas, de criação e manipulação de cookies, sessions e publicidade rotativa nos sites, a capacidade de captura de informações sobre o Browser do usuário e sobre o servidor, de consulta, alteração e adição de dados em Banco de Dados, de envio e recebimento de emails via página WEB, e uma infinidade outras funções. Esses objetos podem ser criados por outras empresas, implementando assim o poder da linguagem ASP.

## O Objeto Response

O objeto Response se refere a todos os dados enviados do servidor para o cliente (usuário - browser), ou seja, qualquer informação enviada do servidor para o browser do usuário se utiliza do objeto Response.

Funções do Objeto Response:

[response.write](#) - permite enviar texto ao browser

[response.redirect](#) - Redireciona o browser para um endereço especificado.

[response.buffer](#) - avisa o browser do usuário que a página só deverá ser mostrada após ser totalmente carregada. É interessante para usar com a função `response.redirect`.

[response.cookies](#) - grava informações no cookie do browser

**response.write** - envia textos ao browser. É interessante na construção de textos que mixam textos prontos e variáveis.

sintaxe: **response.write texto/variável**

```
<html>
<head>
<title>response.asp</title>
</head>
<body bgcolor="#FFFFFF">
<%
response.write "Olá!!! Abaixo uma rotina com o objeto response.write cujo
texto varia de acordo com a variável counter"
for counter=1 to 5000
response.write counter & "->"
next
%>
</body>
</html>
```

[clique aqui para rodar o código acima](#)

**response.redirect** - redireciona o browser do usuário para a página especificada. Esta função só pode ser usada antes do browser divulgar alguma informação na tela.

sintaxe: **response.redirect(URL)**

```
<%
```

'A função `response.redirect` redireciona o browser do cliente para o endereço contido na variável `URLnova`, no caso, `www.aspbrasil.com.br`  
`URLnova = "http://www.aspbrasil.com.br/"`  
`response.redirect(URLnova)`

```
%> <html>  
<head>  
<title>response.asp</title>  
</head>  
<body bgcolor="#FFFFFF">  
</body>  
</html>
```

[clique aqui para rodar o código acima](#)

`response.buffer` - avisa o browser do usuário que a página só deverá ser mostrada após ser totalmente carregada. É interessante para usar com a função `response.redirect`. Observe o exemplo:

sintaxe: `response.buffer = true/false`

```
<%  
response.buffer = true  
%>  
< html>  
< head>  
< title> response.asp< /title>  
</head>  
<body>  
<%
```

`response.write "Olá denovo!!! Abaixo o uso da função response.buffer, a qual, quando tiver o valor true, avisa o browser para mostrar as informações só depois de totalmente carregadas. Isto permite que haja um response.redirect em qualquer parte da página. Mas quando tiver o valor false, o browser mostra as informações assim que elas vão chegando."`

```
response.redirect "http://www.aspbrasil.com.br/"
```

```
%>  
</body>  
</html>
```

[clique aqui para rodar o código acima](#)

`response.cookies` - grava informações no cookie do browser.

```
<html>  
<head><title>cookies</title></head>  
<body>  
<%
```

```
'o comando time insere o horário atual do servidor  
response.cookies("aspbrasil")("horariovisita") = time
```

```
%>
```

</body></html>

## Objeto Request

O objeto Request se refere a todos os dados requisitados pelo servidor ao cliente, ou seja, qualquer informação requisitada pelo servidor ao browser do usuário.

Funções do Objeto Request:

[request.form](#) - recupera valores de variáveis enviadas via formulário pelo método post.

[request.QueryString](#) - recupera valores de variáveis enviadas via formulário pelo método get.

[request.servervariables](#) - recupera informações sobre o browser do usuário e sobre o servidor WEB.

[request.cookies](#) - recupera informações do cookie do browser do usuário.

[request.ClientCertificate](#) - mostra informações sobre o certificado de segurança do usuário. Somente é usado em sites com transmissão segura de dados (https)

exemplos:

**request.form** - recupera valores de variáveis enviadas via formulário pelo método post.

sintaxe: **request.form("nomedavariavel")** ou simplesmente

**request("nomedavariavel")**

Abaixo, formulário de cadastramento.

```
<html><head><title>Formulário</title></head>
<body>
<form method="post" action="retrieve.asp">
<div>Nome: </div><input type="text" name="nome">
<div>Endereço: </div><input type="text" name="endereco">
<input type="submit" value="enviar">
</body>
</html>
```

a seguir, a rotina de recuperação destes dados preenchidos (nome e endereço)

```
<html><head><title>Recuperação dos dados</title></head>
<body>
<div>Abaixo, nome e endereço do usuário que preencheu o formulário: </div>
<%
response.write "<div>Nome: </div>" & request.form("nome")
response.write "<div>Endereço: </div>" & request.form("endereco")
%>
</body></html>
```

[clique aqui para rodar a rotina acima](#)

**request.QueryString** - recupera valores de variáveis enviadas via formulário pelo método get.

sintaxe: **request.querystring("nomedavariavel")** ou, também, simplesmente **request("nomedavariavel")**

Abaixo, formulário de cadastramento.

```
<html><head><title>Formulário</title></head>
<body>
<form method="get" action="retrieve.asp">
<div>Nome: </div><input type="text" name="nome">
<div>Endereço: </div><input type="text" name="endereco">
<input type="submit" value="enviar">
</body>
</html>
```

a seguir, a rotina de recuperação destes dados preenchidos (nome e endereço)

```
<html><head><title>Recuperação dos dados</title></head>
<body>
<div>Abaixo, nome e endereço do usuário que preencheu o formulário: </div>
<%
response.write "<div>Nome: </div>" & request.form("nome")
response.write "<div>Endereço: </div>" & request.form("endereco")
%>
</body></html>
```

[clique aqui para rodar a rotina acima](#)

**request.servervariables** - recupera informações sobre o browser do usuário e sobre o servidor WEB.

sintaxe: **request.servervariables("valorsolicitado")**

Abaixo um exemplo de como conseguir alguns valores do Browser do usuário

```
<html><head><title>Server Variables</title></head>
<body>
<%
response.write "<div>" & request.Servervariables("remote_Addr") & "</div>"
response.write "<div>" & request.Servervariables("server_name") & "</div>"
'request.servervariables("remote_addr") busca o IP remoto do usuário
'request.servervariables("server_name") busca o nome ou IP do servidor
%>
</body></html>
```

[clique aqui para rodar a rotina acima](#)

**request.cookies** - recupera informações do cookie do browser do usuário.

sintaxe: **request.cookies("nomedocookie")("parâmetrodocookie")**

Abaixo, rotina que adiciona (response.cookies) e que recupera (request.cookies) informações do cookie

```
<html><head><title>Server Variables</title></head>
<body>
<%
'a função time retorna o horário do servidor
response.cookies("aspbrasil")("horavisita") = time
'acima, rotina de criação de cookie
response.write "<div>" & request.cookies("aspbrasil")("horavisita") & "</div>"
'acima, rotina de recuperação da informação colocada no cookie de nome
aspbrasil
%>
</body></html>
```

**request.ClientCertificate** - mostra informações sobre o certificado de segurança do usuário. Somente é usado em sites com transmissão segura de dados (https).

sintaxe: **request.clientcertificate(parâmetro[sub-parâmetro])**

abaixo, exemplo de recuperação de informações sobre um site seguro.

```
<html><head><title>Server Variables</title></head>
<body>
<%
response.write "<div>" & response.clientcertificate("validfrom") & "</div>"
'retorna a data inicial da validade do certificado
%>
</body></html>
```

## Objeto Server

O objeto Server permite o acesso e a configuração de algumas funções e métodos diretamente ligados ao servidor.

Funções do Objeto Server:

[server.ScriptTimeout](#) - configuração do Timeout de carregamento da página  
[server.HTMLEncode](#) - codifica strings. Usado para codificar strings que contenham caracteres especiais.

[server.URLEncode](#) - transforma strings para o formato QueryString.  
[server.MapPath](#) - mapeia arquivos de acordo com o path físico, a partir do path virtual.

[server.CreateObject](#) - permite abertura de instâncias de objetos vinculados a componentes ActiveX.

exemplos:

**server.ScriptTimeout** - Configura o tempo máximo para execução e carregamento da página. O tempo padrão é de 90 segundos.

sintaxe: `server.ScriptTimeOut = tempo`  
`<html><head><title>Testando ScriptTimeOut</title></head>`  
`<body>`  
`<%`  
`response.write(server.scriptTimeOut & "<BR>")`  
`Server.ScriptTimeOut = 120`  
`response.write(server.ScriptTimeOut & "<br>")`  
`%>`  
`</body></html>`

[clique aqui para rodar a rotina acima](#)

**server.HTMLEncode** - usa-se para codificar textos que contenham caracteres especiais, tais como > ou <. Codificado, o Browser saberá que o caractere faz parte do texto, e então, este caractere não será interpretado com código. É usado também para para codificar caracteres de outros idiomas, de forma que o Browser identifique o idioma, e então exponha os símbolos corretos na tela.

sintaxe: `server.HTMLEncode("string")`

```
<html><head><title>HTML Enconde</title></head>
<body>
<%
response.write(server.HTMLEncode("<<<Olá Mundo>>>"))
response.write(server.HTMLEncode("WELCOME TO MÜNCHEN"))
'os caracteres <, >, e Û são caracteres especiais. O comando HTMLEncode
assegura que esses códigos serão interpretados de forma correta pelo
Browser.
%>
</body>
</html>
```

[clique aqui para rodar a rotina acima](#)

`server.URLEncode` - URLs (Uniform Resource Locators) são usados para requisitar arquivos e informações em servidores WEB. Muitas vezes, os URLs não contém apenas o nome do arquivo, mas também carregam com eles valores de variáveis no formato QueryString. O comando `server.URLEncode` transforma strings de texto em strings no formato QueryString.

sintaxe: `request.URLEncode("string")`

```
<html><head><title>HTML Enconde</title></head>
<body>
<%
response.write(server.URLEncode("test.asp?name= Fernando
Medeiros&idade=25 anos"))
response.write("test.asp?" & server.URLEncode("name= Fernando
Medeiros&idade=25 anos"))
```

```
%>
</body>
</html>
```

[clique aqui para rodar a rotina acima](#)

**server.MapPath** - retorna o path físico do arquivo a partir do path virtual. No caso de arquivos que não tenham homônimos no mesmo domínio, a função `server.MapPath` é capaz de localizar o arquivo no domínio e retornar o path físico.

sintaxe: `server.MapPath("path virtual")`

```
<html><head><title>server.MapPath</title></head>
<body>
<%
response.write(server.mappath("/test.asp"))
'a linha acima retorna o path físico do arquivo test.asp no servidor. No caso,
c:\aspbrasil\test.asp
response.write(server.MapPath("test.asp"))
'a linha acima retorna o mesmo path físico, apesar de não conter o path
virtual, e sim, apenas o nome do arquivo. Tal retorno só é possível se não
existirem dois arquivos com o mesmo nome dentro do mesmo domínio.
%>
</body></html>
```

[clique aqui para rodar a rotina acima](#)

**server.CreateObject** - É usado para gerar instâncias de componentes para uso nos Scripts. Por padrão, cada objeto criado a partir do componente é automaticamente destruído ao acabar o processamento da página.

sintaxe: `Set nomedadoobjeto = Server.CreateObject("iddoobjeto")`

```
<html><head><title>Create Object</title></head>
<body>
<%
Set objtexto = Server.CreateObject("Scripting.FileSystemObject")
'a linha acima criou uma instância de objeto do componente
scripting.filesystemobject, usado para acessar arquivos texto.
Set arquivotexto = objtexto.OpenTextFile(Server.MapPath("test.txt"))
'abre o arquivo texto dentro da instância do objeto Scripting.FileSystemObject
response.write arquivotexto.readline
'a linha acima mostra a primeira linha de texto do arquivo aberto.
arquivotexto.close
'a linha abaixo destruiu imediatamente
%>
</body></html>
```

[clique aqui para rodar a rotina acima](#)

## Componente FileAccess

O componente File Access é um componente usado para acessar arquivos texto. Os arquivos tipo texto, muitas vezes, são usados como banco de dados de acesso sequencial (nem sempre é vantagem se usar bancos de dados com acesso aleatório, como o MSAccess, ...). Assim sendo, foi necessário a criação de um componente que permitisse a leitura e gravação de dados nestes arquivos.

sintaxe:

conexão com o componente: Set fsObj =  
Server.CreateObject("Scripting.FileSystemObject")

criação do arquivo test.txt: Set txtStreamObj =  
fsObj.CreateTextFile(name,[bOverWrite])

abertura do arquivo test.txt para leitura ou gravação: Set txtStreamObj =  
fsObj.OpenTextFile(name,[iomode],[bCreate],[format]])

fsObj, txtStreamObj são variáveis definidas pelo usuário.

name - nome do arquivo texto (colocar o path caso o arquivo se localize em outra pasta que não a do arquivo ASP que possui este código)

bOverWrite - TRUE: caso o arquivo existe, ele é deletado, e um novo é criado.

FALSE: caso o arquivo exista, ele não é substituído.

iomode - FROWRITING, FORREADING: modo de abertura do arquivo selecionado (para gravação, para leitura)

bCreate - TRUE: caso o arquivo selecionado para abertura não exista, ele é criado.

format - TRUE: é criado um arquivo texto usando o formato Unicode. FALSE: é criado um arquivo texto usando o formato ASCII.

Os valores TRUE ou FALSE são determinados de acordo com uma tabela existente num arquivo chamado fobjvbs.inc , o qual deve ser incluído no código da página que contém o código ASP de leitura do arquivo texto.(veja exemplo mais abaixo)

Abaixo, uma tabela dos vários comandos desse componente:

AtEndOfLine	Retorna TRUE caso o ponteiro tenha chegado ao final da linha. Usado apenas em arquivos abertos para leitura.
AtEndOfStream	Retorna TRUE caso o ponteiro tenha chegado ao final do arquivo. Usado apenas em arquivos abertos para leitura.
Column	Retorna o número da coluna que o ponteiro se encontra. A primeira coluna tem numeração 1.
Line	Retorna o número da linha que o ponteiro se encontra. A primeira linha tem numeração 1.
Close	Fecha um arquivo aberto.
Read	Lê um determinado número de caracteres do arquivo.
ReadAll	Lê todo o arquivo. Quando o arquivo for muito grande, usar outros métodos de leitura, como a leitura de linha por linha.
ReadLine	Lê uma linha inteira, ou seja, lê os caracteres que estão entre a posição do ponteiro e o final da linha.

Skip	pula um determinado numero de caracteres.
SkipLine	pula uma linha inteira.
Write	grava uma determinada string para o arquivo.
WriteLine	grava uma determinada string para o arquivo e posiciona o ponteiro no início da linha seguinte.
WriteBlankLines	grava uma determinada quantia de linhas brancas ao arquivo.

Veja os exemplos a seguir:

```
<HTML><HEAD><TITLE>Componente File Access</TITLE></HEAD>
<BODY>
<!--#include file="fobjvbs.inc"-->
<%
Set objeto= CreateObject("Scripting.FileSystemObjects")
'a linha acima criou uma instância do componente FileAccess
Set gv = objeto.OpenTextFile("c:\test.txt", ForWriting, True, False)
'a linha acima abre o arquivo C:\test.txt para gravação. Caso ele não exista., é
criado um. (este código tem exatamente o mesmo efeito do método
CREATETEXTFILE.
gv.Writeline("Esta linha foi inserida com o uso do componente FileAccess.")
gv.Close
'a linha acima fecha o arquivo aberto para gravação
objeto=nothing
'a linha acima libera a variável objeto, dessa forma, ela não fica armazenando
valores que não serão mais usados após o fechamento do arquivo.
%>
<p>Pronto. Foi gravada a linha acima</p>
</BODY></HTML>
```

### Componente Browser Capabilities

O componente Browser Capabilities tem como objetivo ser usado para determinar quais são as propriedades que cada Browser (navegador) que está acessando o seu site tem. Desde o nome e versão até a detecção de suporte a JavaScript, Applet Java, Cookies, etc. Abaixo, uma lista contendo as propriedades que podem ser determinadas a partir do componente Browser Capabilites.

ActiveXControls	Especifica quando o browser suporta controles ActiveX
backgroundsounds	Especifica quando o browser tem capacidade de tocar som de fundo (back ground sound)
beta	Especifica quando o browser é versão beta.
browser	Especifica o nome do browser
cookies	Especifica se o browser aceita cookies
frames	Especifica se o browser suporta frames
javaapplets	Especifica se o browser suporta Applets Java (ou,

	em alguns casos, se o suporte a Applets Java está ativo)
javascript	Especifica se o browser suporta JavaScript (ou, em alguns casos, se o suporte a Java Script está ativo)
majorver	Especifica o número mais significativo da Versão (na versão 4.01, por exemplo, é retornado o número 4)
minorver	Especifica os números menos significativos da Versão (no exemplo acima, retorna o número 01)
platform	Especifica em qual sistema operacional o browser está instalado
tables	Especifica se o browser suporta tabelas
vbscript	Especifica quando o browser suporta vbscript (processamento local de vbscript - não há nada a ver com o vbscript usado na programação ASP, o qual é processado no servidor)
version	Retorna o número da versão do browser (exemplo: 4.01)
win16	Especifica se o browser roda num computador com sistema operacional 16 bits (win3.x). Válido somente para Internet Explorer.

Caso alguma dessas especificações não seja definida pelo browser, a string "UNKNOWN" é retornada.

sintax:

```

Set bc = Server.CreateObject("MSWC.BrowserType")
... = bc.propriedade
<html><head>
<TITLE>Componente browser Capabilities</TITLE>
</head><body bgcolor="#FFFFFF">
<% Set bc = Server.CreateObject("MSWC.BrowserType") %>
Browser: <%=bc.browser %><p>
Versão: <%=bc.version%><p>
<% if (bc.frames = TRUE) then %>
Este browser aceita frames<p><p>
<% else %>
É melhor você trocar de Browser. Este não suporta frames...
<% end if %>
<% if (bc.tables = TRUE) then %>
Legal. Este browser aceita tabelas...<p>
<% else %>
É melhor você trocar de Browser. Este não suporta nem tabelas...é bem
velhinho...<p>
<% end if %>
<% if (bc.BackgroundSounds = TRUE)then %>
Este browser permite que haja fundo musical nas páginas<p>
<% else %>

```

Ou este browser é bem antiquinho, ou você realmente não gosta de muito barulho.<p>  
<% end if %>

<% if (bc.vbscript = TRUE) then %>  
Este Browser aceita processamento local de VBScript<p>  
<% else %>  
Este Browser não permite o processamento local de VBScript<p>  
<% end if %>

<% if (bc.javascript = TRUE) then %>  
Este Browser aceita processamento local de JavaScript<p>  
<% else %>  
Este Browser não permite o processamento local de JavaScript<p>  
<%  
end if  
set bc=nothing  
>  
</body></html>

[clique aqui para rodar a rotina acima](#)

### Application/Session Obj.

O objeto Application tem como objetivo armazenar e compartilhar valores, estados e propriedades de um grupo. No caso, valores comuns a todos os visitantes do site, como número total de visitantes no site, quantas pessoas estão visitando o site no momento, etc.

O objeto session tem como objetivo armazenar e compartilhar valores, estados e propriedades de cada visitante, individualmente. São bons exemplos o uso em lojas virtuais, onde a cesta de compras pode ser armazenada dentro de uma session. O objeto Application é ativado quando for feita a primeira visita ao site, e termina quando o servidor for desligado (é permitido gravar todos esses valores em banco de dados, assim não há perda dos valores). Todo o script e códigos de armazenamento somente podem ser escritos em um arquivo chamado global.asa, que deve permanecer na pasta raiz do site. O objeto session é criado para cada visitante, cada vez que um visitante entra no site, e é destruído toda vez que este visitante sai do site. Ao entrar no site, cada visitante recebe um ID (número de identificação), o qual é usado para identificar o usuário e para armazenar os valores, estados e propriedades de cada visitante, individualmente.

O arquivo **global.asa** tem o seguinte formato:

```
<SCRIPT LANGUAGE=VBSCRIPT RUNAT=SERVER>  
Sub Application_OnStart  
End sub  
Sub Application_OnEnd  
End sub
```

```

Sub Session_OnStart
End sub
Sub Session_OnEnd
End sub
</SCRIPT>

```

Nos eventos Application\_OnStart e Application\_OnEnd estão armazenados os valores comuns a todos os visitantes. Nos eventos Session\_OnStart e Session\_OnEnd estão armazenados os códigos, scripts e valores usados nas sessions (valores individuais para cada visitante).

Métodos dos Objetos Application e Session:

Application.Lock -> é usado para bloquear o acesso de outras páginas ao evento Application\_OnStart ou Application\_OnEnd, com excessão da página que requisitou o bloqueio, permitindo assim que somente esta página possa fazer alterações no evento.

Application\_Unlock -> Desbloqueia o acesso de outras páginas.

Esses métodos são usados para que duas páginas não tentem alterar as mesmas propriedades ou valores ao mesmo tempo. Pois isso acarretaria numa perda de consistência dos dados armazenados.

Abaixo, um exemplo de como saber o número de visitantes totais do site e o número de visitantes no site no presente momento.

```

<SCRIPT LANGUAGE=VBSCRIPT RUNAT=SERVER>
Sub Application_OnStart
Application("Totalvisitas") = 0
Application("datainicial") = now
Application("Visitasatuais") = 0
End sub
Sub Application_OnEnd
End sub
Sub Session_OnStart
Application.Lock
Application("Totalvisitas") = Application("Totalvisitas") + 1
Application("Visitasatuais") = Application("Visitasatuais") + 1
Application.Unlock
End sub
Sub Session_OnEnd
Application.Lock
Application("Visitasatuais") = Application("Visitasatuais") -1
Application.Unlock
End sub
</SCRIPT>

```

abaixo, código para exibir esses valores.

```

<HTML><header><title>Application & Session</title></header>
<body>
<%
response.write "Visitas totais desde" & application("datainicial") & ":" &
application("Totalvisitas")

```

```

response.write "Visitantes no site no momento:" & application("Visitasatuais")
%>
</body>
</html>

```

## Componente Database

O componente Database Access é um componente usado para acessar bancos de dados de acesso aleatório, tais como Access, SQLServer, DBF, Oracle, ...

O componente Database Access permite a consulta, inserção, alteração e exclusão de dados nos bancos de dados. Tais operações são, geralmente, feitas através de comandos SQL (Structured Query Language), a qual será exemplificada mais abaixo.

Este tutorial não contém todos os comandos e possibilidades de uso do componente Database Access, uma vez que este é muito grande e complexo, mas os comandos e métodos mais utilizados estão citados neste documento.

sintaxe:

conexão com o componente: **Set Conn =**

**Server.CreateObject("ADODB.Connection")**

abertura de um banco de dados já existente: **Conn.Open "nome da ligação ODBC ou path do BD", "UserID", "Senha"**

execução de comandos SQL: **Set rsQuery = Conn.Execute("string SQL")**

um modo alternativo de se abrir um banco de dados para gravação:

**Set RS = Server.CreateObject("ADODB.RecordSet")**

**RS.Open "tabela", Conn, adOpenKeyset, adLockOptimistic**

Para o funcionamento correto do componente ASP, o arquivo **adovbs.inc** deve ser incluído no código da página que contém o código de manipulação do banco de dados.

Abaixo, uma tabela com alguns dos comandos mais usados desse componente:

EOF	Retorna TRUE caso o ponteiro tenha chegado ao final do arquivo.(depois do último registro)
BOF	Retorna TRUE caso o ponteiro esteja posicionado no início do arquivo.(antes do primeiro registro)
MoveFirst	Posiciona o ponteiro para o início do banco de dados.
MoveLast	Posiciona o ponteiro no final do banco de dados
MoveNext	Move o ponteiro 1 registro adiante no banco de dados.
MovePrevious	Move o ponteiro para o registro anterior no banco de dados.
AddNew	Adiciona um novo registro ao banco de dados. É usado quando, ao invés de usar SQL, abre-se o banco de dados no modo RecordSet.
Update	Salva as alterações feitas no banco de dados, no modo RecordSet.

Veja os exemplos a seguir:

```
<HTML><HEAD><TITLE>Adicionando um novo registro ao banco de dados cadastro</TITLE></HEAD>
```

```
<BODY>
```

```
<!--#include file="adovbs.inc"-->
```

```
<%
```

```
'a variavel abaixo - Connstring - guarda o path físico do banco de dados no servidor. poderia armazenar, no lugar do path, o nome da conexão ODBC, caso esta tenha sido criada (como será visto no próximo exemplo)
```

```
ConnString="DBQ=e:\aspbrasil\teste.mdb;Driver={Microsoft Access Driver (*.mdb)}"
```

```
Set Conn = Server.CreateObject("ADODB.Connection")
```

```
Set RS = Server.CreateObject("ADODB.RecordSet")
```

```
Conn.Open ConnString,"", ""
```

```
RS.Open "cadastro", Conn , adOpenKeyset, adLockOptimistic
```

```
RS.Addnew
```

```
RS("nome") = "João"
```

```
'a linha acima atribui o valor João ao campo nome, nesse novo registro.
```

```
RS("email") = "joao@aspbrasil.com.br"
```

```
'a linha acima atribui o valor joao@aspbrasil.com.br ao campo email do banco de dados.
```

```
RS.update
```

```
'a linha acima confirma a inclusão dos dados. Caso este comando não seja usado, o registro não é inserido.
```

```
RS.Close
```

```
Set Conn = nothing
```

```
Set RS = nothing
```

```
%>
```

```
<p>Pronto. Foi gravado o registro acima.</p>
```

```
</BODY></HTML>
```

```
<HTML><HEAD><TITLE>Adicionando um novo registro ao banco de dados cadastro usando ligação ODBC</TITLE></HEAD>
```

```
<BODY>
```

```
<!--#include file="adovbs.inc"-->
```

```
<%
```

```
'a variavel abaixo - Connstring - guarda o nome da conexão ODBC (há a necessidade de a ligação ODBC já estar criada).
```

```
ConnString="odbcteste"
```

```
Set Conn = Server.CreateObject("ADODB.Connection")
```

```
Set RS = Server.CreateObject("ADODB.RecordSet")
```

```
Conn.Open ConnString,"", ""
```

```
RS.Open "cadastro", Conn , adOpenKeyset, adLockOptimistic
```

```
RS.Addnew
```

```
RS("nome") = "João"
```

```
'a linha acima atribui o valor João ao campo nome, nesse novo registro.
```

```
RS("email") = "joao@aspbrasil.com.br"
```

```
'a linha acima atribui o valor joao@aspbrasil.com.br ao campo email do banco de dados.
```

```
RS.update
```

```
'a linha acima confirma a inclusão dos dados. Caso este comando não seja
```

usado, o registro não é inserido.

RS.Close

Set Conn = nothing

Set RS = nothing

%>

<p>Pronto. Foi gravado o registro acima.</p>

</BODY></HTML>

<HTML><HEAD><TITLE>Fazendo uma consulta SQL e mostrando os dados obtidos em uma tabela</TITLE></HEAD>

<BODY>

<!--#include file="adovbs.inc"-->

<%

'a variavel abaixo - Connstring - guarda o nome da conexão ODBC (há a necessidade de a ligação ODBC já estar criada).

ConnString="odbcteste"

'a variável abaixo guarda a string SQL, usada para fazer a consulta no banco de dados. No caso, pede-se para gerar uma consulta onde apenas os registros onde o campo NOME é igual ao valor JOÃO.

SQLstring = "select \* from cadastro where nome = 'joão' "

Set Conn = Server.CreateObject("ADODB.Connection")

Conn.Open ConnString, "", ""

Set rsQuery = Conn.Execute(SQLstring)

'a linha acima gerou uma consulta chamada rsQuery. todos os registros que satisfazem a SQLstring estão armazenados nesta variável.

%>

<table>

<tr><td>Nome</td><td>email</td></tr>

<%

While not rsQuery.EOF

%>

<tr><td><%=rsQuery("nome").value%></td><td><%=rsQuery("email").value%></td></tr>

<%

rsQuery.Movenext

Wend

%>

</table>

<%

rsQuery.Close

set Conn = nothing

Set rsQuery = nothing

%>

</BODY></HTML>

<HTML><HEAD><TITLE>Fazendo uma alteração via SQL</TITLE></HEAD>

<BODY>

<!--#include file="adovbs.inc"-->

<%

'a variavel abaixo - Connstring - guarda o nome da conexão ODBC (há a necessidade de a ligação ODBC já estar criada).

ConnString="odbcteste"

'a variável abaixo guarda a string SQL, usada para fazer a atualização no banco de dados. No caso, pede-se para mudar o nome e o email do registro de código 5.

```
SQLstring = "update cadastro set nome = 'maria' and  
email='maria@aspbrasil.com.br' "  
Set Conn = Server.CreateObject("ADODB.Connection")  
Conn.Open ConnString,"", ""  
Set rsQuery = Conn.Execute(SQLstring)  
rsQuery.Close  
Set Conn = nothing  
Set rsQuery = nothing  
%>
```

```
<p>Arquivo alterado.</p>  
</BODY></HTML>
```

```
<HTML><HEAD><TITLE>Fazendo uma exclusão de registro via  
SQL</TITLE></HEAD>  
<BODY>
```

```
<!--#include file="adovbs.inc" -->  
<%
```

'a variavel abaixo - Connstring - guarda o nome da conexão ODBC (há a necessidade de a ligação ODBC já estar criada).

```
ConnString="odbcteste"
```

'a variável abaixo guarda a string SQL, usada para fazer a exclusão do registro no banco de dados. No caso, pede-se para excluir o(s) registro(s) onde o código é 5. Por se tratar de um número e não de uma string, o valor do código, no caso 5, não fica entre aspas.

```
SQLstring = "delete * from cadastro where codigo = 5 "  
Set Conn = Server.CreateObject("ADODB.Connection")  
Conn.Open ConnString,"", ""  
Set rsQuery = Conn.Execute(SQLstring)  
Set rsQuery = nothing  
Set Conn = nothing  
%>
```

```
<p>Registro Excluído</p>  
</BODY></HTML>
```

## Server Variables Server Variables

ServerVariables é um conjunto de variáveis de sistema, que podem indicar desde o IP remoto, protocolo, HOST do servidor de acesso do usuário, etc.  
sintaxe:

```
valor = Request.ServerVariables("variável")
```

Abaixo, uma tabela com as principais variáveis desse conjunto.

CONTENT_LENGTH	Retorna o tipo de conteúdo que foi enviado ao servidor.
QUERY_STRING	String que sucede o ? no endereço URL.

REMOTE_ADDR	IP do usuário
REMOTE_HOST	Nome do servidor correspondente ao REMOTE_ADDR (nome do provedor de acesso do usuário)
REQUEST_METHOD	Método usado para transmissão das variáveis de uma página para outra (GET ou POST)
SERVER_NAME	Nome do servidor de hospedagem, como usado no URL (pode ser o IP ou DNS)
URL	endereço URL requisitado (sem a query_string)

Veja os exemplos a seguir:

```
<HTML><HEAD><TITLE>Server Variables </TITLE></HEAD>
<BODY><p>o IP usado pela sua conexão é:
<%=request.servervariables("REMOTE_ADDR")%></p></BODY>
</HTML>
```

[clique aqui para rodar a rotina acima](#)

## Cookies

Cookies são pequenos arquivos no formato txt, gravados nos computadores dos usuários, contendo informações que podem ser recuperadas posteriormente.

sintaxe:

gravação de informações no cookie:

```
response.cookies("nomedocookie")("nomedoparametro") =
"valordoparâmetro"
```

recuperação de informações do cookie: **variável =**

```
request.cookies("nomedocookie")("nomedoparametro")
```

determinação da data de expiração do cookie:

```
Response.Cookies("nomedocookies").Expires = data no formato mm/dd/aa
```

obs: caso não seja determinada uma data de expiração, o cookie será apagado assim que o browser for fechado.

Veja os exemplos a seguir:

```
<%
response.cookies("aspbrasilteste")("data") = now
response.cookies("aspbrasilteste")("nome") = "João"
'as linhas acima criaram o cookie aspbrasil e 2 parâmetros, data e nome.
response.cookies("aspbrasilteste").expires = "6/25/99"
'a linha acima determina uma data de expiração do cookie
```

```
%>
```

```
<HTML><HEAD> <TITLE>Criando um cookie</TITLE>
```

```
</HEAD>
```

```
<BODY>
```

```
<%
```

```
response.write "O cookies aspbrasilteste foi criado em: " &
```

```
request("aspbrasilteste")("data")
```

```
response.write "Quem criou foi: " & request("aspbrasilteste")("nome")
```

%>

</BODY></HTML>

[clique aqui para rodar o código acima](#)

## Objetos Application e Session

O objeto Application foi criado para armazenar propriedades (valores) ligados a um conjunto de usuários. No caso, os visitantes do site, de um modo geral. Como exemplo, podemos citar o número total de visitantes no site a partir de uma determinada data, ou o número de visitantes online no site.

O objeto Session foi criado para armazenar propriedades (valores) ligados a cada visitante, individualmente. Como exemplo, podemos citar o carrinho de compras de um site de comércio online. Uma Session é criada quando o visitante entra no site (cada visitante tem uma session e cada session recebe um ID), e é destruída quando o visitante sai do site (seja por logoff explícito ou por Timeout). Já uma Application é iniciada ao haver o primeiro pedido de acesso ao objeto Application, e é encerrado quando o servidor for desligado.

Todo o código que se deseja executar ao criar ou destruir uma session, bem como uma Application devem estar contidos no arquivo global.asa, um arquivo texto no formato abaixo demonstrado, que deve ser colocado no diretório raiz do site.

As variáveis do objeto Application e do objeto Session são armazenadas no servidor, mas é necessário que o browser aceite cookies, pois um cookie com o ID da sessão é criado no computador do visitante, para identificá-lo.

Veja o exemplo abaixo:

Listagem do arquivo global.asa

```
<SCRIPT LANGUAGE=VBSCRIPT RUNAT=SERVER>
```

```
Sub Application_OnStart
```

```
Application("totaldeusuarios") = 0
```

```
Application("datainicial") = now
```

```
Application("usuariosonline") = 0
```

```
End Sub
```

```
Sub Application_OnEnd
```

```
End Sub
```

```
Sub Session_OnStart
```

```
Session.Timeout = 20
```

```
Application.Lock
```

```
Application("totaldeusuarios") = Application("totaldeusuarios") + 1
```

```
Application("usuariosonline") = Application("usuariosonline") + 1
```

```
Application.Unlock
```

```
End Sub
```

```
Sub Session_OnEnd
```

```
Application.Lock
```

```
Application("usuariosonline") = Application("usuariosonline") - 1
```

```
Application.Unlock
```

```
End Sub
```

</SCRIPT>

Abaixo, o código de uma página que armazena um valor numa variável de sessão e mostra o total de usuários conectados e o total de visitantes no site.

```
<html><head><title>Application e Session</title></head>
<% Session("variavelqualquer") = "Este valor foi inserido na variável de
sessão de nome variavelqualquer" %>
<body>
<p>Número da Sessão: <%=Session.SessionID%></p>
<p>Existem no momento <%=Application("usuariosonline")%> usuários
conectados.</p>
<p>Total de visitantes no site desde <%=Application("datainicial")%> :
<%=Application("totaldeusuarios")%></p>
<p>Abaixo, a string que foi inserida na variavel variavelqualquer</p>
<p><%=Session("variavelqualquer")%></p>
<% Session.Abandon %>
</body></html>
```

[clique aqui ara rodar a rotina acima](#)

Os comandos Lock e Unlock servem para garantir que somente um visitante estará alterando as variáveis por vez. O comando Lock bloqueia a acesso de outros visitantes às variáveis, e o Unlock desbloqueia. O comando Session.SessionID retorna o valor do ID da sessão. O comando Session.Timeout determina qual o tempo máximo, em minutos, de inatividade do visitante até que a sessão possa ser considerada abandonada. O comando Session.Abandon termina a sessão iniciada explicitamente (não é necessário esperar o vencimento do Timeout para considerar a sessão abandonada).

A procedure Application\_OnStart contém a rotina que será rodada quando o objeto Application for iniciado.

A procedure Application\_OnEnd contém a rotina que será executada quando o objeto Application for terminado (qdo. o servidor for desligado).

A procedure Session\_OnStart contém o código que será rodado quando uma sessão for iniciada.

A procedure Session\_OnEnd contém o código que será rodado quando uma sessão for terminada (por timeout ou logoff explícito).

## Procedures

Algumas vezes, em programação, uma mesma sequência de códigos precisa ser usado constantemente. Ao invés de copiá-los toda vez que for preciso, pode-se usar as Procedures. Procedures são caixas pretas, onde vc entra com alguns dados, a caixa preta processa, e no caso das Functions, retorna outros valores, de acordo com um algoritmo. Existem dois tipos de Procedures, as Subs e as Functions. Observe os

exemplos abaixo:

```
<html><body>
```

```
<%
```

```
Function soma(valor1,valor2)
```

```
If valor1 = "" then valor1 = 0
```

```
If valor2 = "" then valor2 = 0
```

```
soma = valor1 + valor2
```

```
End Function
```

'acima, a função denominada soma foi apresentada

'abaixo, uma chamada à função foi feita. Então a função executa a rotina e retorna um determinado valor, no caso 5.

```
response.write soma(2,3)
```

```
%></body></html>
```

[clique aqui para rodar o código acima](#)

Já uma Sub simplesmente executa uma ação, não retornando valor algum.

```
<html><body>
```

```
<%
```

```
Sub visualizar_nome(nome)
```

```
response.write "O nome do site é: " & nome
```

```
End Sub
```

'acima, a procedure do tipo Sub foi denominada visualizar\_nome

```
response.write "Qual é o site sobre ASP que oferece Tutoriais, Newsgroups e Links a seus visitantes?"
```

```
Call visualizar_nome("ASPBRASIL")
```

```
response.write "<div><a href=www.aspbrasil.com.br>Home</a></div>"
```

'acima, o comando Call faz a chamada à Sub visualizar\_nome. Como pode-se perceber, uma Sub não retorna nenhum valor, e sim executa uma ação.

```
%>
```

```
</body></html>
```

[clique aqui para rodar o código acima](#)

### If...Then...Else ; Select Case

Existem 2 comandos capazes de identificar o conteúdo de uma determinada variável, e de acordo com esse valor, executar uma determinada rotina:

[If...Then...Else](#)

[Select Case](#)

**If...Then...Else**

O comando If...Then...Else possibilita verificar se uma determinada variável está ou não de acordo com um critério de seleção. Observe o exemplo abaixo:

```
<html><body>
<%
variavel_qualquer = hour(now)
If variavel_qualquer < 19 then
response.write "Bom Dia"
Else
response.write "Boa Noite"
End if
'o comando IF...then...Else comparou o valor da variavel variavel_qualquer
com um determinado valor estipulado (no caso, 19). Se o valor for menor que
19, então escreva na tela "Bom Dia". Caso contrário (Else), "Boa Noite"
%></body></html>
```

[clique aqui para rodar o código acima](#)

### Select Case

O comando Select Case permite um maior controle sobre os valores de uma determinada variável. Observe o Exemplo Abaixo:

```
<html><body>
<%
variavel_qualquer = hour(now)
Select Case variavel_qualquer
case 0,1,2,3,4,5,6,7,8,9,10,11,12
response.write "Bom Dia"
case 13,14,15,16,17,18,19
response.write "Boa Tarde"
case 20,21,22,23,24
response.write "Boa Noite"
Case else
response.write "Este relógio está maluco"
End Select
%></body></html>
```

observe que o comando Select Case deve ser fechado com o comando End Select.

[clique aqui para rodar o código acima](#)

### If...Then...Else ; Select Case

Existem 2 comandos capazes de identificar o conteúdo de uma determinada variável, e de acordo com esse valor, executar uma determinada rotina:

[If...Then...Else](#)  
[Select Case](#)

**If...Then...Else**

O comando If...Then...Else possibilita verificar se uma determinada variável está ou não

de acordo com um critério de seleção. Observe o exemplo abaixo:

```
<html><body>
```

```
<%
```

```
variavel_qualquer = hour(now)
```

```
If variavel_qualquer < 19 then
```

```
response.write "Bom Dia"
```

```
Else
```

```
response.write "Boa Noite"
```

```
End if
```

'o comando IF...then...Else comparou o valor da variavel `variavel_qualquer` com um determinado valor estipulado (no caso, 19). Se o valor for menor que 19, então escreva na tela "Bom Dia". Caso contrário (Else), "Boa Noite"

```
%></body></html>
```

[clique aqui para rodar o código acima](#)

## Select Case

O comando Select Case permite um maior controle sobre os valores de uma determinada variavel. Observe o Exemplo Abaixo:

```
<html><body>
```

```
<%
```

```
variavel_qualquer = hour(now)
```

```
Select Case variavel_qualquer
```

```
case is > 0 and qualquer_variavel < 12
```

```
response.write "Bom Dia"
```

```
case os > 12 and qualquer_variavel < 19
```

```
response.write "Boa Tarde"
```

```
case > 19 and qualquer_variavel < 24
```

```
response.write "Boa Noite"
```

```
Case else
```

```
response.write "Este relógio está maluco"
```

```
End Select
```

```
%></body></html>
```

observe que o comando Select Case deve ser fechado com o comando End Select.

[clique aqui para rodar o código acima](#)

## If...Then...Else ; Select Case

Existem 2 comandos capazes de identificar o conteúdo de uma determinada variável, e de acordo com esse valor, executar uma determinada rotina:

[If...Then...Else](#)

[Select Case](#)

## If...Then...Else

O comando If...Then...Else possibilita verificar se uma determinada variável está ou não de acordo com um critério de seleção. Observe o exemplo abaixo:

```
<html><body>
<%
variavel_qualquer = hour(now)
If variavel_qualquer < 19 then
response.write "Bom Dia"
Else
response.write "Boa Noite"
End if
'o comando IF...then...Else comparou o valor da variavel variavel_qualquer
com um determinado valor estipulado (no caso, 19). Se o valor for menor que
19, então escreva na tela "Bom Dia". Caso contrário (Else), "Boa Noite"
%></body></html>
```

[clique aqui para rodar o código acima](#)

## Select Case

O comando Select Case permite um maior controle sobre os valores de uma determinada variável. Observe o Exemplo Abaixo:

```
<html><body>
<%
variavel_qualquer = hour(now)
Select Case variavel_qualquer
case is > 0 and qualquer_variavel < 12
response.write "Bom Dia"
case os > 12 and qualquer_variavel < 19
response.write "Boa Tarde"
case > 19 and qualquer_variavel < 24
response.write "Boa Noite"
Case else
response.write "Este relógio está maluco"
End Select
%></body></html>
```

observe que o comando Select Case deve ser fechado com o comando End Select.

[clique aqui para rodar o código acima](#)

## Rotinas de Loop

Loops são rotinas que devem ser repetidas até que uma determinada condição seja satisfeita. Existem 3 comandos que permitem tal situação:

[Do...Loop](#)  
[For...Next](#)  
[While...Wend](#)

Abaixo seguem as características e exemplos de cada comando.

### **Do...Loop**

O comando Do...Loop executa uma determinada rotina até que a condição se torne verdadeira. Observe o exemplo abaixo:

```
<html><body>  
<%  
x = 0  
Do Until x=10  
x = x + 1  
Loop  
response.write x  
>%></body></html>
```

para interromper o Loop, usa-se o comando **Exit Do**

[clique aqui para rodar o código acima](#)

### For...Next

O comando For...Next executa uma determinada rotina até que o contador (uma variável) atinja o valor determinado. Observe o exemplo:

```
<html><body>  
<%  
For i=1 to 50  
response.write i  
next  
>%></body></html>
```

observe que não é necessário nenhuma rotina de incrementação de valores em i. A cada ciclo, o valor de i é acrescido de uma unidade, automaticamente. para interromper o Loop, usa-se o comando **Exit For**

[clique aqui para rodar o código acima](#)

### While...Wend

O comando While...Wend executa uma determinada rotina até que a condição imposta seja alcançada. Observe o exemplo dado:

```
<html><body>  
<%  
q = "SELECT * FROM cadastro where cidade = 'São Paulo'"  
connstring = "DBQ=c:\teste.mdb;Driver={Microsoft Access Driver (*.mdb)}"
```

```
Set Conexao = Server.CreateObject("ADODB.Connection")
Conexao.Open connstring, "", ""
Set tabela = Conexao.Execute (q)
While not tabela.EOF
response.write tabela("nome").value
wend
%></body></html>
```

## Rotinas de Loop

Loops são rotinas que devem ser repetidas até que uma determinada condição seja satisfeita. Existem 3 comandos que permitem tal situação:

[Do...Loop](#)  
[For...Next](#)  
[While...Wend](#)

Abaixo seguem as características e exemplos de cada comando.

### Do...Loop

O comando Do...Loop executa uma determinada rotina até que a condição se torne verdadeira. Observe o exemplo abaixo:

```
<html><body>
<%
x = 0
Do Until x=10
x = x + 1
Loop
response.write x
%></body></html>
```

para interromper o Loop, usa-se o comando **Exit Do**

[clique aqui para rodar o código acima](#)

### For...Next

O comando For...Next executa uma determinada rotina até que o contador (uma variável) atinja o valor determinado. Observe o exemplo:

```
<html><body>
<%
For i=1 to 50
response.write i
next
%></body></html>
```

observe que não é necessário nenhuma rotina de incrementação de valores em i. A cada ciclo, o valor de i é acrescido de uma unidade, automaticamente.

para interromper o Loop, usa-se o comando **Exit For**

[clique aqui para rodar o código acima](#)

## While...Wend

O comando While...Wend executa uma determinada rotina até que a condição imposta seja alcançada. Observe o exemplo dado:

```
<html><body>
<%
q = "SELECT * FROM cadastro where cidade = 'São Paulo'"
connstring = "DBQ=c:\teste.mdb;Driver={Microsoft Access Driver (*.mdb)}"
Set Conexao = Server.CreateObject("ADODB.Connection")
Conexao.Open connstring, "", ""
Set tabela = Conexao.Execute (q)
While not tabela.EOF
response.write tabela("nome").value
wend
%></body></html>
```

## Rotinas de Loop

Loops são rotinas que devem ser repetidas até que uma determinada condição seja satisfeita. Existem 3 comandos que permitem tal situação:

[Do...Loop](#)  
[For...Next](#)  
[While...Wend](#)

Abaixo seguem as características e exemplos de cada comando.

## Do...Loop

O comando Do...Loop executa uma determinada rotina até que a condição se torne verdadeira. Observe o exemplo abaixo:

```
<html><body>
<%
x = 0
Do Until x=10
x = x + 1
Loop
response.write x
%></body></html>
```

para interromper o Loop, usa-se o comando **Exit Do**

[clique aqui para rodar o código acima](#)

For...Next

O comando For...Next executa uma determinada rotina até que o contador (uma variável) atinja o valor determinado. Observe o exemplo:

```
<html><body>  
<%  
For i=1 to 50  
response.write i  
next  
>%></body></html>
```

observe que não é necessário nenhuma rotina de incrementação de valores em i. A cada ciclo, o valor de i é acrescido de uma unidade, automaticamente.

para interromper o Loop, usa-se o comando **Exit For**

[clique aqui para rodar o código acima](#)

### While...Wend

O comando While...Wend executa uma determinada rotina até que a condição imposta seja alcançada. Observe o exemplo dado:

```
<html><body>  
<%  
q = "SELECT * FROM cadastro where cidade = 'São Paulo'"  
connstring = "DBQ=c:\teste.mdb;Driver={Microsoft Access Driver (*.mdb)}"  
Set Conexao = Server.CreateObject("ADODB.Connection")  
Conexao.Open connstring, "", ""  
Set tabela = Conexao.Execute (q)  
While not tabela.EOF  
response.write tabela("nome").value  
wend  
>%></body></html>
```

### Tipos de Dados

O VBScript contém apenas um tipo de variável, denominado Variant. O tipo variant pode armazenar qualquer tipo de dado, e de acordo com o tipo de dados que é armazenado, é possível classifica-lo de acordo com os subtipos de dados, abaixo relacionados:

Subtipo	Descrição
Empty	Variável que contém 0 para valores numéricos e "" (string vazia) para strings.
Null	Variável que não contém dado algum.
Boolean	Contém True ou False
Byte	Números inteiros entre 0 e 255

<b>Integer</b>	Números inteiros no intervalo de -32,768 a 32,767.
<b>Long</b>	Números inteiros no intervalo de -2,147,483,648 a 2,147,483,647.
<b>Single</b>	Números com ponto flutuante de precisão simples na faixa de -3.402823E38 a -1.401298E-45 para números negativos e 1.401298E-45 a 3.402823E38 para números positivos.
<b>Double</b>	Números com ponto flutuante de dupla precisão na faixa de -1.79769313486232E308 a -4.94065645841247E-324 para números negativos e 4.94065645841247E-324 a 1.79769313486232E308 para números positivos.
<b>Date (Time)</b>	Dados no formato de Data (data e tempo) na faixa de 1 de janeiro de 100 a 31 de dezembro de 999. (January 1, 100 a December 31, 9999).
<b>String</b>	Contém dados no formato de string, que podem ter até aproximadamente 2 bilhões de caracteres de tamanho.

PS: é possível a conversão de dados de um tipo para outro, mas para isso, os dados devem ser compatíveis com o subtipo desejado, ou seja, a String "23/5/99" pode ser convertida para o subtipo Date, e vice-versa. Mas a String "ASPBRASIL" não pode.

É importante a conversão de tipos de dados uma vez que o modo como os dados serão manipulados dependem do seu subtipo.

### Convertendo Dados

Verificando e Convertendo Tipos de Dados A linguagem VBScript contém algumas funções de verificação e conversão de tipos de dados importantes para a melhor manipulação dos dados.

As funções de verificação são importantes na hora de detectar se os dados contidos numa variável são compatíveis com o subtipo para o qual se deseja converter estes dados.

As funções de conversão fazem a conversão de dados de um subtipo para outro, mas para isso, é necessário que esses dados sejam compatíveis com o subtipo que se deseja obter.

#### Funções de Verificação

[IsArray](#) - [IsDate](#) - [IsEmpty](#) - [IsNull](#) - [IsNumeric](#)

#### Funções de Conversão

[CBool](#) - [CByte](#) - [CDate](#) - [CDBl](#) - [CInt](#) - [CLng](#) - [CStr](#) - [CSng](#)

**IsArray** - retorna True caso a variável seja um array, caso contrário, retorna False.

sintaxe: IsArray(nomedavariavel)

```
<%  
Dim aspbrasil  
Dim Arraybrasil(5)  
response.write IsArray(aspbrasil) 'retorna false  
response.write IsArray(Arraybrasil) 'retorna true  
%>
```

**IsDate** - retorna True caso o valor da variável possa ser convertido em data, caso contrário, retorna False.

sintaxe: IsDate(nomedavariavel)

```
<%  
aspbrasil = "23/5/99"  
aspbrasil2 = "ASPBRASIL"  
response.write IsDate(aspbrasil) 'retorna true  
response.write IsDate(aspbrasil2) 'retorna false  
%>
```

**IsEmpty** - retorna True caso a variável contenha o valor 0 ou "", caso contrário, retorna False.

sintaxe: IsEmpty(nomedavariavel)

```
<%  
aspbrasil = ""  
aspbrasil2 = 0  
aspbrasil3 = "ASPBRASIL"  
response.write IsEmpty(aspbrasil) 'retorna true  
response.write IsEmpty(aspbrasil2) 'retorna true  
response.write IsEmpty(aspbrasil3) 'retorna false  
%>
```

**IsNull** - retorna True caso a variável não contenha dados válidos.

sintaxe: IsNull(nomedavariavel)

```
<%  
aspbrasil = "23/5/99"  
response.write IsNull(aspbrasil) 'retorna false  
%>
```

**IsNumeric** - retorna True caso o valor da variável possa ser convertido para algum tipo de dados numéricos.

\* A conversão de tipo de dados numéricos entre si, ou seja, de um número Double para o formato Sng pode acarretar a perda de dados, ou mesmo, podem ser incompatíveis, uma vez que existem faixas de atuação para cada tipo de dado.

sintaxe: IsNumeric(nomedavariavel)

```
<%  
aspbrasil = "23"  
aspbrasil2 = 56  
response.write IsNumeric(aspbrasil) 'retorna true  
response.write IsNumeric(aspbrasil2) 'retorna true  
%>
```

**CBool** - retorna True ou False (0 ou 1), de acordo com a expressão analisada ou com o valor da variável.

sintaxe: CBool(nomedavariavel ou expressão)

```
<%  
aspbrasil = 5  
aspbrasil2 = 5  
aspbrasil3 = 0  
response.write CBool(aspbrasil = aspbrasil2) 'retorna true  
response.write CBoll(aspbrasil3) 'retorna false  
%>
```

**CByte** - converte a expressão dada em um dado do subtipo Byte.

sintaxe: CByte(nomedavariavel ou expressão)

```
<%  
aspbrasil = "23"  
aspbrasil2 = 34.67  
response.write IsArray(aspbrasil) 'retorna 23  
response.write IsArray(aspbrasil2) 'retorna 35, pois há um arredondamento  
%>
```

**CDate** - converte a expressão dada em um dado do subtipo Date.

sintaxe: CDate(nomedavariavel)

```
<%  
aspbrasil = "23/5/99"  
response.write CDate(aspbrasil) 'convertendo strings de data e tempo para o subtipo  
Date, é possível fazer operações matemáticas com essas datas, somando dias,  
meses, segundos, anos...  
%>
```

**Cdbl** - converte a expressão dada em um dado do subtipo Double.

sintaxe: Cdbl(nomedavariavel ou expressão)

```
<%  
aspbrasil2 = 34.6767786543E56
```

```
response.write CDbl(aspbrasil2)
%>
```

**CInt** - converte a expressão dada em um dado do subtipo Integer. A diferença para o subtipo Byte é a faixa de abrangência.

sintaxe: CInt(nomedavariavel ou expressão)

```
<%
aspbrasil = "23"
aspbrasil2 = 34.67
response.write CInt(aspbrasil) 'retorna 23
response.write CInt(aspbrasil2) 'retorna 35, pois há um arredondamento
%>
```

**CLng** - converte a expressão dada em um dado do subtipo Long. A diferença para o subtipo byte é a faixa de abrangência.

sintaxe: CByte(nomedavariavel ou expressão)

```
<%
aspbrasil = "23"
aspbrasil2 = 34.6
response.write CLng(aspbrasil) 'retorna 23
response.write CLng(aspbrasil2) 'retorna 35, pois há um arredondamento
%>
```

**CSng** - converte a expressão dada em um dado do subtipo Single.

sintaxe: CInt(nomedavariavel ou expressão)

```
<%
aspbrasil2 = 88734.679999
response.write CInt(aspbrasil2)
%>
```

**CStr**- converte a expressão dada em um dado do subtipo String. Qualquer outro tipo de dado, dos listados acima, pode ser convertido em String.

sintaxe: CStr(nomedavariavel ou expressão)

```
<%
aspbrasil2 = 34.67
response.write CStr(aspbrasil2) 'retorna "23"
%>
```

**Manipulando Números**  
**Manipulando Datas**

O VBScript contém várias funções utilizadas para a manipulação de dados numéricos. Abaixo uma lista das principais funções. Os operadores básicos (+, -, \*, /, ^) estão descritos no tutorial sobre Operadores Básicos do VBScript.

[Abs](#) - [Atn](#) - [Cos](#) - [Exp](#) - [Fix](#) - [Int](#) - [Log](#) - [Sin](#) - [Sqr](#) - [Tan](#)

**Abs** - retorna o módulo do valor de entrada

sintaxe: Abs(número)

```
<%  
aspbrasil = -78  
response.write Abs(aspbrasil)  
%>
```

**Atn** - retorna o valor do arcotangente do valor entrado (resposta em radianos)

sintaxe: Atn(número)

```
<%  
aspbrasil = 1  
response.write Atn(aspbrasil)  
%>
```

**Cos** - retorna o valor do cosseno do angulo entrado (entrar angulo em radianos)

sintaxe: Cos(angulo)

```
<%  
aspbrasil = 3,1416  
response.write Cos(aspbrasil)  
%>
```

**Exp** - retorna o valor de e (euler) elevado ao valor de entrada ( $e^x$ )

sintaxe: Exp(numero)

```
<%  
aspbrasil = 3  
response.write Exp(aspbrasil)  
%>
```

**Fix** - retorna a parte inteira de um número.

sintaxe: Fix(número)

```
<%  
aspbrasil = 78.778  
response.write Fix(aspbrasil)  
%>
```

**Int** - retorna a parte inteira de um número. Caso o número seja negativo, é retornado o valor imediatamente menor.

sintaxe: Int(número)

```
<%  
aspbrasil = -78,389  
response.write Int(aspbrasil)  
%>
```

**Log** - retorna o valor do Log do número de entrada na base e.

sintaxe: Log(número)

```
<%  
aspbrasil = 4  
response.write Log(aspbrasil)  
%>
```

**Sin** - retorna o valor do seno de um angulo entrado em radianos.

sintaxe: Sin(angulo)

```
<%  
aspbrasil = 3,1416  
response.write Sin(aspbrasil)  
%>
```

**Sqr** - retorna o valor da raiz quadrada do número de entrada.(valor deve ser maior ou igual a zero)

sintaxe: Sqr(numero)

```
<%  
aspbrasil = 4  
response.write Sqr(aspbrasil)  
%>
```

**Tan** - retorna o valor da tangente do angulo pedido (entrar angulo em radianos)

sintaxe: Tan(angulo)

```
<%  
aspbrasil = 2  
response.write Tan(aspbrasil)  
%>
```

**Manipulando Datas**  
**Manipulando Datas**

O VBScript possibilita a obtenção e manipulação de dados no formato Data e Tempo facilmente. Abaixo, uma lista das principais funções relacionadas a esse tópico.

[CDate](#) - [Date](#) - [DateSerial](#) - [Day](#) - [Hour](#) - [IsDate](#) - [Minute](#) - [Month](#) - [Now](#) - [Second](#) - [Time](#) - [TimeSerial](#) - [TimeValue](#) - [Weekday](#) - [Year](#)

**CDate** - converte a expressão dada em um dado do subtipo Date.

sintaxe: CDate(nomedavariavel)

```
<%  
aspbrasil = "23/5/99"  
response.write CDate(aspbrasil) 'convertendo strings de data e tempo para o subtipo  
Date, é possível fazer operações matemáticas com essas datas, somando dias,  
meses, segundos, anos...  
%>
```

**Date** - retorna a data do sistema (a data que consta no relógio do servidor).

sintaxe: Date

```
<%  
response.write Date  
%>
```

**DateSerial** - retorna um valor do subtipo data para um determinado valor de ano, mes e dia, ou seja, entrando esses 3 valores respectivamente, a função DateSerial retorna a data respectiva no formato Date.

sintaxe: DateSerial(year, month, day)

```
<%  
response.write DateSerial(99,12,23)  
%>
```

**Day** - retorna um número entre 1 e 31, representando o dia de uma determinada data.

sintaxe: Day(data)

```
<%  
response.write Day(now)  
%>
```

**Hour** - retorna um número entre 0 e 23 representando a hora de uma determinada expressão de tempo.

sintaxe: Date

```
<%  
response.write Date  
%>
```

**IsDate** - retorna True caso o valor da variável possa ser convertido em data, caso contrário, retorna False.

sintaxe: IsDate(nomedavariavel)

```
<%  
aspbrasil = "23/5/99"  
aspbrasil2 = "ASPBRASIL"  
response.write IsDate(aspbrasil) 'retorna true  
response.write IsDate(aspbrasil2) 'retorna false  
%>
```

**Minute** - retorna um número entre 0 e 59 representando os minutos de uma determinada expressão de tempo.

sintaxe: Minute(tempo)

```
<%  
response.write Minute(now)  
%>
```

**Month** - retorna um número entre 1 e 12 representando o mês de uma determinada data.

sintaxe: Month(data)

```
<%  
response.write Month(now)  
%>
```

**Now** - retorna a data e o horário do relógio do servidor.

sintaxe: Now

```
<%  
response.write Now  
%>
```

**Second** - retorna um número entre 0 e 59 representando os segundos do sistema (o segundo que consta no relógio do servidor).

sintaxe: Second(tempo)

```
<%  
response.write Second(now)  
%>
```

**Time** - retorna o horário do relógio do sistema.

sintaxe: Time

```
<%  
response.write Time  
%>
```

**TimeSerial** - retorna uma expressão do tipo Date contendo o tempo para uma dada hora, minuto e segundo.

sintaxe: TimeSerial(hora,minuto,segundo)

```
<%  
response.write TimeSerial(22,11,33)  
%>
```

**Weekday** - retorna o dia da semana de uma determinada data.

sintaxe: Weekday(date)

```
<%  
response.write weekday(now)  
%>
```

**Year** - retorna o ano de uma determinada data.

sintaxe: Year(data)

```
<%  
response.write Year(now)  
%>
```

## Manipulando Strings

O VBScript disponibiliza algumas funções utilizadas na manipulação de Strings. Abaixo, as principais funções, que entre outras coisas, permitem saber o número de caracteres de uma string, encontrar determinado caracter dentro de uma string, comparar duas strings, etc.

[Asc](#) - [Chr](#) - [CStr](#) - [InStr](#) - [Lcase](#) - [Left](#) - [Len](#) - [LTrim](#) - [Mid](#) - [Right](#) - [RTrim](#) - [Trim](#) - [UCase](#)

**Asc** - Retorna o caracter (pela tabela ANSI) correspondente à primeira letra da string de entrada.

sintaxe: Asc(string)

```
<%  
aspbrasil = "aspbrasil"
```

```
response.write Asc(aspbrasil)
%>
```

**Chr** -Retorna o caracter correspondente a um determinado codigo (numerico) da tabela ANSI.

```
sintaxe:Chr(codigo)
<%
response.write Chr(34)
%>
```

**CStr** - Converte uma expressão para o formato String. Caso uma variável seja um número, ela será convertida numa string que represente tal número. Se for uma data, essa data será convertida numa string que represente tal data.

```
sintaxe: CStr(expressão)
```

```
<%
aspbrasil = 458
response.write CStr(aspbrasil) 'retorna a string, sequencia de caracteres "458", e
não o número 458.
%>
```

**InStr** - Retorna a posição da primeira ocorrência de uma string dentro da outra. É possível delimitar a partir de qual caracter será feita tal procura.

```
sintaxe:InStr(posicaooinicial, string, stringdecomparação)
```

```
<%
aspbrasil = "aspbrasil, site sobre ASP em Português."
response.write Instr(4,aspbrasil,"ASP") '4 é a posição a partir da qual a procura será
feita.
%>
```

**LCase** - Converte todos os caracteres de uma string para minúsculas.

```
sintaxe: LCase(string)
```

```
<%
aspbrasil = "ASPBRASIL"
response.write LCase(aspbrasil) 'deve retornar ASPBRASIL
%>
```

**Left** - Retorna um determinado número de caracteres a partir do lado esquerdo da string.

```
sintaxe: Left(string, numerodecaracteres)
```

```
<%
aspbrasil = "aspbrasil, site sobre ASP."
```

```
response.write Left(aspbrasil,4) 'deve retornar "aspb"  
<%>
```

**Len** - Retorna o número de caracteres numa string (contando espaços em branco), ou, no caso de variáveis numéricas, ou no formato Date, retorna a quantidade de bytes necessários para armazenar aquela expressão.

sintaxe: Asc(string/expressão)

```
<%  
aspbrasil = "aspbrasil"  
response.write Len(aspbrasil) 'retorna 9  
<%>
```

**LTrim** - Retorna a cópia da string de entrada, mas os espaços no começo do lado esquerdo.

sintaxe: LTrim(string)

```
<%  
aspbrasil = " <--Trim-> "  
response.write Asc(aspbrasil) 'retorna "<--Trim-> "  
<%>
```

**Mid** - Função usada para "cortar" uma string. Delimita-se a posição inicial, a string e a quantidade de caracteres a partir da posição inicial que devem ser "capturados".

sintaxe: Mid(string, posicao inicial, tamanho)

```
<%  
aspbrasil = "aspbrasil"  
response.write Mid(aspbrasil,1,4) 'deve retornar "aspb"  
<%>
```

**Right** - Retorna um determinado número de caracteres a partir do lado direito da string.

sintaxe: Right(string, numerodecaracteres)

```
<%  
aspbrasil = "aspbrasil"  
response.write Right(aspbrasil,3) 'deve retornar "sil"  
<%>
```

**RTrim** - Retorna uma cópia da string de entrada, mas sem os espaços no final da string.

sintaxe: RTrim(string)

```
<%  
aspbrasil = "aspbrasil "
```

```
response.write Asc(aspbrasil) 'deve retornar "aspbrasil"
%>
```

**Trim** - Retorna uma cópia da string de entrada, mas sem os espaços no início e no final da string.

sintaxe: Trim(string)

```
<%
aspbrasil = " aspbrasil "
response.write Asc(aspbrasil) 'deve retornar "aspbrasil"
%>
```

**UCase** - Converte toda a string para MAIÚSCULAS.

sintaxe: UCase(string)

```
<%
aspbrasil = "aspbrasil"
response.write Asc(aspbrasil) 'deve retornar "ASPBRASIL"
%>
```

## Operadores

Existem alguns caracteres e funções que permitem a ligação de uma ou mais expressões, de vários modos diferentes. Tais funções e caracteres são chamados Operadores. Abaixo, uma lista dos mais importantes:

[+](#) [-](#) [\\*](#) [/](#) [\](#) [and](#) [&](#) [Eqv](#) [^](#) [Imp](#) [Mod](#) [Not](#) [Or](#) [Xor](#)

**+** (**mais**)- Usado para somar dois valores numéricos

sintaxe: resultado = expressão1 + expressão2

```
<%
aspbrasil = 5 + 6
response.write (aspbrasil)
%>
```

**-**(**menos**) -Faz a subtração entre dois números ou indica valor negativo numa expressão numérica.

sintaxe:resultado = número1 - número2

```
<%
response.write (12 - 6)
%>
```

**\*** (**multiplicar**) - Usado para multiplicar dois valores numéricos

sintaxe: resultado = expressão1 \* expressão2

```
<%  
aspbrasil = 5 * 6  
response.write (aspbrasil)  
%>
```

**/(dividir)** - Usado para dividir um valor por outro.  
sintaxe: resultado = numerador/denominador

```
<%  
aspbrasil = 31/7  
response.write (aspbrasil)  
%>
```

**\** - Usado para dividir um valor por outro, e retornar um valor inteiro.  
sintaxe: resultado = numerador/denominador

```
<%  
aspbrasil = 31/7  
response.write (aspbrasil)  
%>
```

**and** - Usado para "validar" um conjunto de duas expressões lógicas. Se, e somente se as duas expressões tiverem como resultado TRUE, é retornado TRUE. Caso contrário, é retornado FALSE.

sintaxe: resultado = expressão1 and expressão2

```
<%  
aspbrasil = (7>4) and (4<8)  
response.write (aspbrasil)  
%>
```

**&** - Usado para somar duas cadeias de caracteres (strings).

sintaxe: resultado = string1 & string2

```
<%  
aspbrasil = "ASPBRASIL." & "Site sobre ASP na lingua portuguesa."  
response.write (aspbrasil)  
%>
```

**Eqv** - Usado para "validar" um conjunto de expressões lógicas de acordo com a seguinte tabela:

Expressão1	Expressão2	Resultado
true	true	true
true	false	false
false	true	false

false	false	true
-------	-------	------

sintaxe: resultado = expressão1 Eqv expressão2

```
<%
aspbrasil = (6<4) Eqv (8>9)
response.write (aspbrasil)
%>
```

**^** - Usado para "elevar" um número a outro.

sintaxe: resultado = número1^número2

```
<%
aspbrasil = 6^2
response.write (aspbrasil)
%>
```

**Imp** - Usado para "validar" um conjunto de expressões lógicas (Implicação Lógica) de acordo com a tabela:

Expressão1	Expressão2	Resultado
true	true	true
true	false	false
true	null	null
false	true	true
false	false	true
false	null	true
null	true	true
null	false	null
null	null	null

sintaxe: resultado = expressão1 Imp expressão2

```
<%
aspbrasil = (9<8) Imp (7>4)
response.write (aspbrasil)
%>
```

**Mod** - Usado para dividir dois números e retornar apenas o resto.

sintaxe: resultado = número1 Mod número2

```
<%
aspbrasil = 7 Mod 3
response.write (aspbrasil)
%>
```

**Not** - Usado para inverter, ou negar o resultado de uma comparação, como segue na tabela:

Expressão 1	Resultado
true	false
false	true
null	null

sintaxe: resultado = Not expressão

```
<%  
aspbrasil = Not (7>6)  
response.write (aspbrasil)  
%>
```

**Or** - Usado para "validar" um conjunto de expressões lógicas de acordo com a tabela:

Expressão1	Expressão2	Resultado
true	true	true
true	false	true
true	null	true
false	true	true
false	false	false
false	null	null
null	true	true
null	false	null
null	null	null

sintaxe: resultado = expressão1 Imp expressão2

```
<%  
aspbrasil = (9<8) or (7>4)  
response.write (aspbrasil)  
%>
```

**Xor** - Usado para "validar" um conjunto de expressões lógicas de acordo com a seguinte tabela:

Expressão1	Expressão2	Resultado
true	true	false
true	false	true
false	true	true
false	false	false

sintaxe: resultado = expressão1 Xor expressão2

```
<%  
aspbrasil = (6<4) Xor (8>9)  
response.write (aspbrasil)  
%>
```

## Demais Funções do VBScript

Abaixo, algumas importantes funções do VBScript que não se encaixaram no restante dos tópicos.

[LBound](#) - [Rnd/Randomize](#) - [Sgn](#) - [UBound](#) - [VarType](#)

**LBound** - Retorna o menor índice de uma determinada dimensão num array.

sintaxe: resultado = LBound(array,dimensão) 'a dimensão é opcional.

```
<%  
dim aspbrasil(8)  
aspbrasil(2)="Tigre"  
aspbrasil(4)="Onça"  
response.write LBound(aspbrasil) 'deve retornar 2  
%>
```

**Rnd** - Retorna um número randômico entre 0 e 1.

sintaxe:

```
Randomize  
Rnd (numero)
```

```
<%  
Randomize  
aspbrasil = Rnd  
response.write aspbrasil  
%>
```

**Sgn** - Retorna a natureza de um determinado número (maior, menor ou igual a zero), como visto abaixo:

```
maior que zero (numero>0) 'retorna 1  
igual a zero (numero=0) 'retorna 0  
menor que zero (numero<0) 'retorna -1
```

sintaxe: resultado = Sgn(numero)

```
<%  
aspbrasil = Sgn(-67)
```

```
response.write aspbrasil  
%>
```

**UBound** - Retorna o maior índice de uma determinada dimensão num array.

sintaxe: resultado = UBound(array,dimensão) 'a dimensão é opcional.

```
<%  
aspbrasil(2)="Tigre"  
aspbrasil(4)="Onça"  
response.write UBound(aspbrasil) 'deve retornar 4  
%>
```

**Vartype** - Retorna um valor, de acordo com a tabela abaixo, indicando o subtipo da variável.

sintaxe: resultado = VarType(variável)  
vel)

0	vazia (não inicializada)
1	null (nula)
2	inteiro (int)
3	inteiro longo (lng)
4	ponto flutuante de precisão simples (Sng)
5	ponto flutuante de dupla precisão (Dbl)
6	moeda (currency)
7	data (date)
8	String
9	objeto OLE

```
<%  
aspbrasil="aspbrasil"  
aspbrasil2 = 34  
response.write vartype(aspbrasil) 'deve retornar 8  
response.write vartype(aspbrasil2) 'deve retornar 3  
%>
```