**UNIVERSIDADE FEDERAL DO PIAUÍ**
**CENTRO DE CIÊNCIAS DA NATUREZA**
**DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA**
**BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO**
**LINEU ANTONIO DE LIMA SANTOS**

# I CURSO DE ACTIVE SERVER PAGES DA UFPI

lineulima@yahoo.com

# 1 - INTRODUÇÃO

A Internet é um conjunto de redes de computadores interligados pelo mundo inteiro, que têm em comum um conjunto de protocolos e serviços, de forma que os usuários a ela conectados podem usufruir de serviços de informação e comunicação de alcance mundial tais como: e-mail, servidores Web, ftp, irc, icq etc.

Trata-se da mais bem sucedida aplicação prática do conceito de interoperabilidade, que consiste em conectividade de redes de tecnologias distintas. Isso só foi conseguido graças ao conjunto de protocolos conhecidos como TCP/IP(Transmission Protocol/Internet Protocol).

Mas o que popularizou mesmo a Internet foi a criação da World Wide Web. Trata-se de um serviço para a transmissão multimídia de informações implementado pelo protocolo de aplicação HTTP(Hypertext Transfer Protocol).

Um Cliente HTTP(Browser WEB) se comunica com um servidor HTTP(Servidor WEB) requisitando arquivos. Geralmente esses arquivos estão no formato HTML (Hypertext Markup Language) que pode conter referências para outros arquivos diversos(imagens, sons, vídeos etc). Ao receber o arquivo HTML o cliente verifica cada referência, solicitando ao servidor HTTP os arquivos indicados.

Esse modelo de funcionamento limita bastante o uso da Web uma vez que as páginas HTML têm um conteúdo estático, ou seja, sempre são exibidas da mesma forma e não possibilitam nenhuma interação com o usuário.

Para deixar a Web mais dinâmica e interativa, criou-se o CGI(Common Gateway Interface). Agora podemos ter programas num servidor Web que podem ser requisitados por um cliente Web. O programa é processado e o resultado desse processamento é enviado pelo servidor Web ao cliente, geralmente no formato HTML. É importante percebermos onde está o dinamismo do CGI: o processamento de tais programas pode retornar diferentes resultados, dependendo dos parâmetros informados pelo cliente(interação) ao programa CGI.

Apesar de dar mais “vida” a web, programas CGI possuem uma serie de desvantagens técnicas, sendo a principal delas o fato de tais programas executarem num processo diferente do Web Server. Sendo assim, um servidor web que recebesse várias requisições simultâneas, facilmente se sobrecarregava e parava.

Por isso surgiram, e ainda surgem a cada dia, tecnologias alternativas ao uso do CGI: ISAPI, NISAPI, IDC/HTX, Cold Fusion, Java Server Pages(JSP), Personal Home Page(PHP), Active Server Pages(ASP) etc.

# 2 - ACTIVE SERVER PAGES

ASP é um tecnologia da Microsoft que disponibiliza um conjunto de componentes para o desenvolvimento de páginas Web dinâmicas. Tais páginas consistem em arquivos de extensão *.asp no formato texto(ASCII) que contém combinações de scripts e tags HTML.

Um servidor Web que suporta ASP funciona da seguinte forma:

- Cliente solicita página *.asp
- Servidor abre a página e lê seu conteúdo
    - Se encontra tags HTML, envia direto ao cliente
    - Se encontra comandos de script:
        - Para o envio
        - Processa os comandos

- Envia o resultado HTML ao cliente.

Como todo código de programação existente em páginas Asp é executado no servidor, e este só retorna ao cliente respostas em HTML, aplicações ASP têm seu código fonte totalmente preservado além de poderem ser acessadas por "qualquer" browser existente no mercado.

Entre os recursos que podem ser implementados com ASP, podemos citar:

- Programação com Visual Basic Script e Java Script
- Acesso a banco de dados
- Envio de e-mail

Para utilizar ASP em suas homepages certifique-se que o computador que a hospedará roda Windows NT Server 4.0(ou superior) com o Internet Information Server 3.0(ou superior). Esse último é um programa Servidor Web da Microsoft. Se pretende usar os recursos de acesso a banco de dados, você precisará de um driver de ODBC instalado e funcionando no servidor.

ASP também "funciona" com o MS Personal Web Server(PWS), para Windows NT WorkStation e para Windows 9x, muito embora essa não seja a plataforma mais recomendada. Para os amantes das plataformas Unix/Linux, já existem módulos no mercado que garantem o suporte ao ASP nessa plataforma.

## 3 – ALGUNS SITES BRASILEIROS QUE UTILIZAM ASP

| | |
|---|---|
| **General Motors Center – automóveis GM** | http://www.gmcenter.com.br |
| **Web Motors – venda de automóveis** | http://www.webmotors.com.br |
| **CompareCom – Buscador de preços baixos** | http://www.comparecom.com.br |
| **OneClick – Hospeda Lojas Virtuais** | http://www.oneclick.com.br |
| **SpaSite – Spa Virtual** | http://www.spasite.com.br |
| **NetCasa – classificado de imóveis** | http://www.netcasa.com.br |
| **GuiaSP – Guia da cidade de São Paulo** | http://www.guiasp.com.br |
| **Fulano – Entretenimento** | http://www.fulano.com.br |
| **Gazeta Mercantil – informações financeiras** | http://www.gazetamercantil.com.br |
| **InvestShop – informações financeiras** | http://www.investshop.com.br |
| **Patagon – dicas de investimento** | http://www.patagon.com.br |
| **Arremate – Leilão Virtual** | http://www.arremate.com.br |
| **Usina do Som – música** | http://www.usunadosom.com.br |
| **IBGE** | http://www.ibge.gov.br |
| **Plantão Eletrônico – Delegacia Virtual** | http://www.seguranca.sp.gov.br |
| **Saraiva – livraria virtual** | http://www.saraiva.com.br |
| **Ecovias – Condições das Rodovias** | http://www.ecovias.com.br |
| **TurismoNet – Agência de Turismo** | http://www.turismonet.com.br |
| **ASPBrasil – Tecnologia ASP** | http://www.aspbrasil.com.br |
| **Ipoint – Hospedagem Gratuita ASP** | http://www.ipoint.com.br |

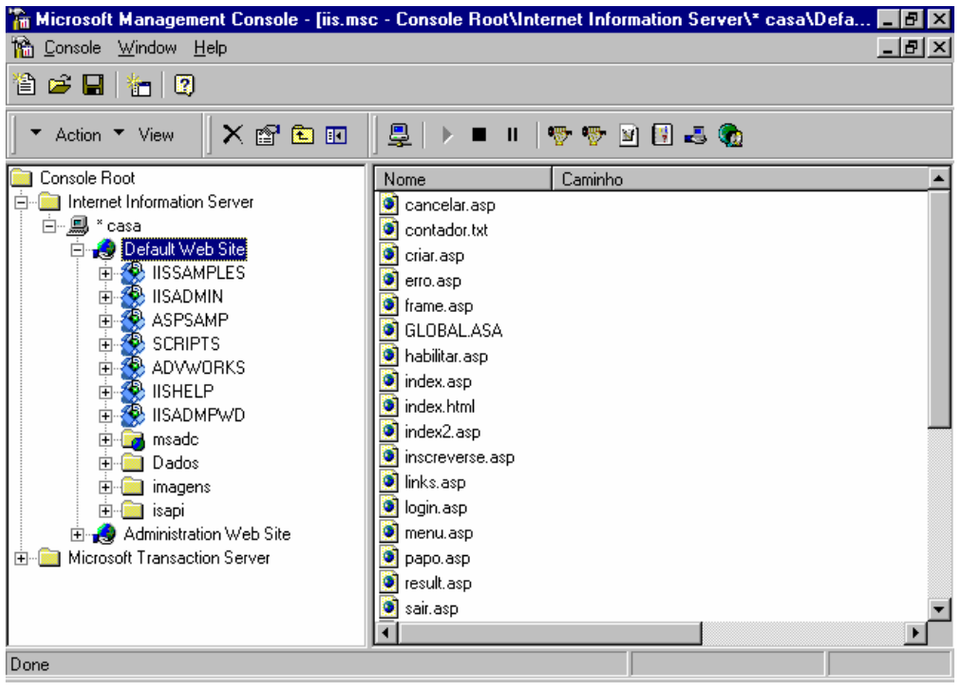
## 4 - INTERNET INFORMATION SERVER

Atualmente na versão 4.0, o IIS é o servidor Web mais recomendada pela Microsoft para desenvolvimento de sites dinâmicos com ASP, pois roda num ambiente

mais seguro e robusto: Windows NT Server. Ele pode ser encontrado no CD-ROOM do Windows NT 4.0 Optional Pack.

Sua instalação é bastante simples, cabendo ao usuário informar apenas quais os componentes do Optional Pack 4.0 deseja instalar e em que diretório serão instalados, bem como informar os diretórios principais para os serviços de publicação WWW e FTP.

A configuração do IIS é feita através do Microsoft Management Console: Iniciar → Programas → Windows NT 4.0 Optional Pack → Microsoft Internet Information Server → Information Server Manager.




A primeira coisa a se fazer é criar um novo Web site. Basta ir no menu Action→New→Site Web, indicar a descrição do novo site, indicar o IP da máquina e o número da porta do servidor Web(80), indicar o caminho para o diretório home do site e, para finalizar, informar as permissões de acesso:

| |
|---|
| Acesso de Leitura |
| Acesso de Script |
| Acesso de Execução |
| Acesso de Gravação |
| Pesquisa em pasta |

Depois de criar o novo site, você pode alterar suas configurações clicando no mesmo com o botão direito do mouse, escolhendo a opção propriedades:

| Opção | Descrição |
|---|---|
| Site da Web | Configuração do IP, portas de conexão, limite de conexões, Ativação do Log |
| Operadores | Designar contas administrativas p/ site |
| Desempenho | Otimização |
| Filtros ISAPI | Adicionar/Remover filtros ISAPI |
| Pasta Base | Configurar permissões de acesso e diretório |

| | |
|---|---|
| | base |
| Erros Personalizados | Personalizar os erros do Servidor Web |
| Pasta de Segurança | Configurar autenticação de usuários, segurança de comunicação e restrições a endereços IP |
| Documentos | Definir arquivos padrões |

O IIS também possui o recurso de Diretórios Virtuais. Eles servem para que os usuários Web possam acessar o seu conteúdo, sem precisar saber sua localização física no disco do servidor. Camuflando a estrutura real do disco, nos prevenimos contra possíveis ataques de hackers e facilitamos a vida dos usuários Web, pois se mudarmos a estrutura interna de armazenamento, o endereço virtual não será afetado. Cada diretório virtual criado possui suas próprias configurações de segurança, permissões de acesso, erros personalizados, documentos padrões etc.

Para criar um diretório virtual, clique com o botão direito do mouse no Web Site onde este deverá se localizar→ New→ Pasta Virtual. Informe então um nome para o diretório virtual, especifique o diretório físico onde estarão as páginas desse diretório, especifique as permissões de acesso de seus usuários. Para que páginas ASP sejam executadas, deve-se pelo menos marcar a permissão de Script. Caso contrário, o servidor Web retornará as página ASP desse diretório aos clientes da mesma forma que estão armazenadas(com todos os comandos de scripts visíveis ao usuário).

Quando você desejar publicar uma página na Web basta criar o arquivo ASP ou HTML e salvá-lo em algum diretório físico relacionado a algum diretório virtual. Para acessar esse arquivo por um navegados, informe o seguinte URL:

**http://nome_do_servidor/nome_diretorio_virtual/nome_página**

## 5 – ROTINAS DE SCRIPT

Script é um programa escrito numa determinada linguagem de programação que não necessita ser compilado para ser posteriormente executado. Scripts são interpretados, ou seja, seus comandos são lidos em tempo de execução por um Script Engine, processados e seus resultados passados para a saída padrão da aplicação(monitor de vídeo, impressora, sevidor web etc).

Toda a funcionalidade e "Inteligência" de uma página ASP é controlada através de comandos de Script. Teoricamente, o ASP pode utilizar qualquer Script Engine(interpretador), mas na prática a Microsoft só disponibiliza dois:

- Visual Basic Script (VBScript) - default
- MS Java Script(JScript)

Ao escrevemos páginas *.ASP a primeira coisa que devemos fazer é indicar em qual dessas linguagens disponíveis elas serão escritas:

**<% @ LANGUAGE=VBScript %>**

ou

**<SCRIPT LANGUAGE="VBScript" RUNAT=SERVER>...</SCRIPT>**

O Web browser não executará comandos HTML, somente scripts. Mas como ele reconhece um script? Simples, se o Web Server encontrar na página *.ASP a tag **<%** ou **<SCRIPT LANGUAGE="VBScript" RUNAT=SERVER>** ele entende que daquela posição até **%>** ou **</SCRIPT>**existem comandos de scripts a serem executados. Sendo assim, ele os executa e só retornará para o cliente o resultado HTML.

É importante observar que o Web Server só tentará interpretar uma página se a mesma estiver salva com a extensão .asp, caso contrário, o servidor Web enviará a página como se fosse um arquivo de texto normal. Logo, não adianta criar scripts altamente eficientes e esquecer de salvar corretamente o arquivo.

A seguir nossa primeira página ASP. Ela simplesmente retorna a data e hora atual do Servidor Web. Crie o seguinte arquivo chamado now.asp no seu diretório virtual:

Exemplo nº 1 – now.asp

```
<% @ LANGUAGE=VBSCRIPT %>
<HTML><HEAD><TITLE>Curso ASP</TITLE></HEAD>
<BODY>
<%=NOW%>  <!--função do vbscript que retorna hora/data-->
</BODY></HTML>
```

Obs : Tende abrir essa página utilizando o caminho físico da mesma e veja o que acontece quando você tenta visualizar o código fonte. Depois, execute através do URL http://servidor/diretorio_virtual/now.asp e faça o mesmo teste.

## 6 – VISUAL BASIC SCRIPT (VBSCRIPT)

VBScript é uma linguagem criada a partir do Visual Basic, mas com algumas limitações, por motivos de segurança, além de ser interpretada e não compilada. Dentre outros pontos, permite a manipulação de strings, datas, númericos e objetos Active X do servidor. Ela é a linguagem de script mais utilizada para desenvolvimento de páginas ASP. Sendo assim, é de fundamental importância conhecermos seus principais comandos antes de prosseguir com o nosso estudo.

Diferente de linguagens como Delphi ou C++, **VBScript só aceita um comando por linha**. O seguinte código retornaria um erro:

Exemplo 2.1: erro_linha.asp

```
<% @ LANGUAGE=VBSCRIPT %>
<HTML><HEAD><TITLE>Curso ASP</TITLE></HEAD>
<BODY>
<% a = 2 b = a*2 %>
Valor de B = <%=b%>
</BODY></HTML>
```

Executando esse script podemos averiguar que quando a página ASP encontra um erro, o mesmo é bem ilustrado no browser. Isso facilita bastante o trabalho de depuração para os programadores ASP. Mas por outro lado, essa informação de erro não diz muita coisa para o usuário WEB. Temos a opção de tratar os diversos possíveis erros como veremos mais adiante ou simplesmente configurar nosso Web Server para retornar uma mensagem de erro padrão(MS Controle Maneger→ Diretório Virtual→ Propriedades→ Pasta Base→ Configurações→ App Debugging→ Script Error Mensages→ Send Text Error Message To Client→ Digite a Mensagem).

Existem duas formas de consertar esse erro: colocando um comando por linha ou separar os comandos por **:( dois pontos )**

Exemplo 2.2: erro_linha2.asp

```
<% @ LANGUAGE=VBSCRIPT %>
<HTML><HEAD><TITLE>Curso ASP</TITLE></HEAD>
<BODY>
<% a = 2 : b = a*2 %>
Valor de B = <%=b%>
</BODY></HTML>
```

**Um comando não pode existir em mais de uma linha.**

Exemplo 2.3: erro_linha3.asp

```
<% @ LANGUAGE=VBSCRIPT %>
<HTML><HEAD><TITLE>Curso ASP</TITLE></HEAD>
<BODY>
<% texto = “lineu antonio ” +
                “de lima santos”%>
Nome = <%=texto%>
</BODY></HTML>
```

Existem duas formas de se escrever essa página corretamente: colocar o comando numa única linha ou usar o caractere **_ (underline)**.

Exemplo 2.4: erro_linha4.asp

```
<% @ LANGUAGE=VBSCRIPT %>
<HTML><HEAD><TITLE>Curso ASP</TITLE></HEAD>
<BODY>
<% texto = “lineu antonio ” + _
                “de lima santos”%>
Nome = <%=texto%>
</BODY></HTML>
```

## 6.1 VARIÁVEIS

São identificadores alfanuméricos que “apontam” para posições de memória onde existem valores armazenados temporariamente, sendo que estes pode ser alterados durante o processamento de uma aplicação. Não nos interessa saber como esse valor será armazenado na memória, nem onde. Basta apenas sabermos o nome e o tipo do valor armazenado em tal variável.

Em VBScript os nomes de variáveis devem começar obrigatoriamente com uma letra e não podem exceder 255 caracteres. Ao contrário da maioria das linguagens de programação, uma variável do VBScript não necessita ser declarada antes de ser utilizada.

**Dim** Nome_da_Variavel

Exemplo 3.1: var1.asp

```
<% @ LANGUAGE=VBSCRIPT %>
<HTML><HEAD><TITLE>Curso ASP</TITLE></HEAD>
<BODY>
<% Dim v1
   v1 = 100
   v2 = 200
   v3 = 300 %>
V1=<%=v1%><BR>
V2=<%=v2%><BR>
V3=<%=v3%>
</BODY></HTML>
```

**A mesma variável não pode ser declarada mais de uma vez no mesmo escopo do script:** O código a seguir está errado

Exemplo 3.2: var2.asp

```
<% @ LANGUAGE=VBSCRIPT %>
<HTML><HEAD><TITLE>Curso ASP</TITLE></HEAD>
<BODY>
<% Dim v1
   v1 = 100
   v2 = 200
   Dim v1  'Redeclaração da Variável v1
  v1 = 900
   v3 = 300 %>
V1=<%=v1%><BR>
V2=<%=v2%><BR>
V3=<%=v3%>
</BODY></HTML>
```

**Obs:** O "tempo de vida" de uma variável vai desde sua declaração explícita(Dim) ou implícita(sem Dim) até o final do script ou sub-rotina.

Talvez você não perceba isso com exemplos tão simples mas scripts com declarações implícitas de variáveis são mais difíceis de ser entendidos, além de estarmos mais vulneráveis a erros de digitação. Observe o seguinte exemplo e tire suas conclusões.

Exemplo 3.3: var3.asp

```
<% @ LANGUAGE=VBSCRIPT %>
<HTML><HEAD><TITLE>Curso ASP</TITLE></HEAD>
<BODY>
<% teste = "Lineu Antonio de Lima Santos" %>
Nome do Usuário = <%=tste%>
</BODY></HTML>
```

Para evitar esse tipo de erro, podemos utilizar a declaração **Option Explicit.** Ela informa ao interpretador do script que variáveis só poderão ser utilizadas se antes forem declaradas explicitamente.

Exemplo 3.4: var4.asp

```
<% @ LANGUAGE=VBSCRIPT %>
<% Option Explicit %>
<HTML><HEAD><TITLE>Curso ASP</TITLE></HEAD>
<BODY>
<% Dim teste
   teste = "Lineu Antonio de Lima Santos" %>
Nome do Usuário = <%=tste%>
</BODY></HTML>
```

Observe que agora a execução do script retornará um erro, uma vez que a variável **tste** (erro de digitação) não foi explicitamente declarada.

## 6.2 TIPOS DE DADOS

O VBScript contém apenas um tipo de variável chamado de **Variant**. Na verdade, uma variável pode armazenar valores de qualquer tipo. Só que num determinado instante, uma variável possui apenas um tipo implícito. O que determina o subtipo de uma variável é o valor a ela atribuído.

| Subtipos |
|---|
| Integer |
| Long |
| Single |
| Double |
| Date |
| String |
| Boolean |
| Null |
| Empty |
| Object |

Exermplo 4: tipo.asp

```
<% @ Language = VBScript%>
<% Option Explicit %>
<HTML><HEAD><TITLE>Curso ASP</TITLE>
<BODY>
<% Dim A
      A = "Lineu Antonio de Lima Santos" %>
Valor de A como String = <%=A%><BR>
<% A = 2 %>
Valor de A como Inteiro = <%=A%><BR>
<% A=#02/10/1978# %>
Valor de A como Date = <%=A%><BR>
<% A = #06:30:00#  %>
Valor de A como Time = <%=A%><BR>
<% A = 10.20 %>
Valor de A como Real = <%=A%><BR>
<% A = True %>
Valor de A como Booleano = <%=A%><BR>
</BODY>
</HTML>
```

É possível em VBScript declarar uma variável para armazenar mais de um valor: Array. Ao declararmos uma variável array devemos informar seu nome e a quantidade de valores que a mesma pode armazenar:

**Dim Nome_Array(Quantidade)**

Para acessar determinado valor de uma variável array, informamos o nome e a posição de tal valor. Esse índice começa em 0(zero) e vai até o valor especificado na declaração(Quantidade).

Exermplo 5.1: array1.asp

```
<% @ Language = VBScript%>
<HTML><HEAD><TITLE>Curso ASP</TITLE>
<BODY>
  <% Dim MeuArray(3)
     MeuArray(0)=Date
     MeuArray(1)="Lineu Antonio de Lima Santos"
     MeuArray(2)=12.45
     MeuArray(3)=Now
  %>
Posição 1 = <%=MeuArray(0)%><BR>
Posição 2 = <%=MeuArray(1)%><BR>
Posição 3 = <%=MeuArray(2)%><BR>
Posição 4 = <%=MeuArray(3)%>
</BODY>
</HTML>
```

Observe que o valor armazenado em cada elemento de um array pode ser de um subtipo diferente dos demais. Outra observação importante: **arrays têm que ser declarados explicitamente**.

Arrays não são limitados para uma única dimensão. Em VBScript pode-se declarar Arrays de até 60 dimensões.

Exermplo 5.2: array2.asp

```
<% @ Language = VBScript%>
<HTML><HEAD><TITLE>Curso ASP</TITLE>
<BODY>
  <% Dim MeuArray(1,1)
     MeuArray(0,0)= "Lineu Antonio de Lima Santos"
     MeuArray(0,1)=2225240
     MeuArray(1,0)= "Universidade Federal do Piauí"
     MeuArray(1,1)=2172000
%>
<B>Nome</B> = <%=MeuArray(0,0)%> <B>Telefone</B> =
<%=MeuArray(0,1)%><BR>
<B>Nome</B> = <%=MeuArray(1,0)%> <B>Telefone</B> =
<%=MeuArray(1,1)%><BR>
</BODY>
</HTML>
```

Pode-se declarar um array cujo tamanho é alterado durante a interpretação do script. Para tal, basta declarar o array com **Dim** sem informar a quantidade de elementos. Depois deve-se utilizar a declaração **ReDim** para determinar o número de elementos. Caso seja necessário redimensionar o array, utiliza-se novamente a declaração ReDim. Se houver a necessidade de preservar o conteúdo do array a ser redimensionado, utiliza-se a declaração **ReDim Preserve**.

Exermplo 5.3: array3.asp

```
<% @ Language = VBScript%>
<HTML><HEAD><TITLE>Curso ASP</TITLE>
<BODY>
Sem Preserve <BR>
  <% Dim MeuArray()
     ReDim MeuArray(1)
     MeuArray(0)= "Lineu Antonio de Lima Santos"
     MeuArray(1)=12.95
     ReDim MeuArray(2)
     MeuArray(2)=9090
   %>

<B>Posição 0</B> = <%=MeuArray(0)%><BR>
<B>Posição 1</B> = <%=MeuArray(1)%><BR>
<B>Posição 2</B> = <%=MeuArray(2)%><BR>
<HR>
Com Preserve <BR>
  <% ReDim MeuArray(1)
     MeuArray(0)= "Lineu Antonio de Lima Santos"
     MeuArray(1)=12.95
     ReDim Preserve MeuArray(2)
     MeuArray(2)=0909
  %>
<B>Posição 0</B> = <%=MeuArray(0)%><BR>
<B>Posição 1</B> = <%=MeuArray(1)%><BR>
<B>Posição 2</B> = <%=MeuArray(2)%>
</BODY>
</HTML>
```

## 6.3 CONSTANTES

Uma constante representa um valor fixo através de um identificador alfanumérico. Sua diferença para variáveis é que seu valor uma vez definido, não pode ser modificado. Para definir uma função em VBScript utiliza-se a declaração Const:

**Const Nome_Constante = Valor_Constante**

Exemplo 6 : constante.asp

```
<% @ LANGUAGE=VBSCRIPT %>
<HTML><HEAD><TITLE>Curso ASP</TITLE></HEAD>
<BODY>
<% Const nome = "Lineu"
   Const data = #02/10/1978# %>
<B>Nome: </B><%=nome%><BR>
<B>Nascido em :</B><%=data%><BR>
</BODY></HTML>
```

## 6.4 OPERADORES

De nada adiantaria termos valores armazenados em nossas variáveis de memória, se não pudéssemos fazer cálculos, comparações ou qualquer outra operação com eles. Em VBScript temos um conjunto de símbolos alfanuméricos para efetuar tais operações:

| Operador | Descrição |
|---|---|
| = | Atribuição / Igual |
| <> | Diferente |
| < | Menor que |
| <= | Menor ou igual que |
| > | Maior que |
| >= | Maior ou igual que |
| + | Soma numérica/ Concatenação de Strings |
| - | Subtração ou negativo Numérico |
| * | Multiplicação |
| / | Divisão |
| \ | Efetua a divisão entre dois números e retorna um número inteiro |
| Mod | Retorna o Resto de uma divisão entre inteiros |
| ^ | Exponenciação |
| & | Concatenação de Strings |
| Is | Comparação de Igualdade entre dois Objetos |

Pelo exposto, não fica difícil perceber que as variáveis serão manipuladas de acordo com o seus subtipos. Os valores envolvidos numa mesma operação devem ser do mesmo subtipo ou de subtipos compatíveis.

Exemplo 7.1 : operador1.asp

```
<% @ LANGUAGE=VBSCRIPT %>
<% Option Explicit %>
<HTML><HEAD><TITLE>Curso ASP</TITLE></HEAD>
<BODY>
<% Dim a,b
     a = 1
     b = "2"
%>
A+B = <%=a+b%><HR>
<% a="Número de Variáveis"%>
A+B = <%=a+b%>
</BODY></HTML>
```

Esse exemplo funciona corretamente pois pela regra de compatibilidade a soma de um número inteiro e uma "string numérica" resulta realmente na soma numérica dos dois. Seria lógico pensar que o contrário estaria correto também, ou seja,

que a soma de uma número e uma string alfanumérica resultaria numa concatenação. Verifique o próximo exemplo e tire suas conclusões:

Exemplo 7.2 : operador2.asp

```
<% @ LANGUAGE=VBSCRIPT %>
<% Option Explicit %>
<HTML><HEAD><TITLE>Curso ASP</TITLE></HEAD>
<BODY>
<%  Dim a,b
        a = 1
        b = "Alfanumérica"  %>
A + B = <%=a+b%>
</BODY></HTML>
```

Podem ocorrer confusões de outros tipos:

Exemplo 7.3 : operador3.asp

```
<% @ LANGUAGE=VBSCRIPT %>
<% Option Explicit %>
<HTML><HEAD><TITLE>Curso ASP</TITLE></HEAD>
<BODY>
<%  Dim a,b
        a = "1"
        b = "2"  %>
A - B = <%=a-b%><BR>
A + B = <%=a+b%>
</BODY></HTML>
```

Para resolver esses problemas, poderíamos ser mais cautelosos na escrita de nossos programas efetuando exaustivos testes. Outra solução menos “dolorosa” seria utilizarmos conversões de tipos explícitas nos nossos programas.

| Função de Conversão | Descrição |
|---|---|
| CStr | Converte uma expressão para o subtipo String |
| Cint | Tenta converter uma expressão para o subtipo Integer |
| CLng | Tenta converter uma expressão para o subtipo Long |
| Cbool | Tenta Converter para Booleano |
| Cbyte | Tenta converter para o subtipo Byte |
| Cdate | Tenta converter para o subtipo Date |
| CDbl | Tenta converter para o subtipo Double |
| CSng | Tenta converter para o subtipo Single |

Exemplo 8.1 : converte1.asp

```
<% @ LANGUAGE=VBSCRIPT %><% Option Explicit %>
<HTML><HEAD><TITLE>Curso ASP</TITLE></HEAD><BODY>
<%  Dim a,b
      a = "1" :    b = "2"  %>
A - B = <%=Cint(a)-Cint(b)%><BR>A + B = <%=Cint(a)+Cint(b)%>
</BODY></HTML>
```

exceção:

Exemplo 8.2 : converte2.asp

```
<% @ LANGUAGE=VBSCRIPT %>
<% Option Explicit %>
<HTML><HEAD><TITLE>Curso ASP</TITLE></HEAD><BODY>
<%  Dim a,b
   a = 1.1
   a = CDate(a) %>
<%=a%><BR>
<% a = CDbl(#02/10/2000#) %>
<%=a%>
</BODY></HTML>
```

O erro não ocorreu pois na verdade um subtipo Date é implementado como um número real. Ele armazena a quantidade de dias desde 31/12/1899. Valores a esquerda do decimal representam a data e a direita o horário. Sendo assim, operações aritméticas também podem ser feitas em cima desse subtipo sem problema algum.

Exemplo 8.3 : converte3.asp

```
<% @ LANGUAGE=VBSCRIPT %>;
<% Option Explicit %>
<HTML><HEAD><TITLE>Curso ASP</TITLE></HEAD><BODY>
<%  Dim Datan,hoje
   Datan = #02/10/1978#
   Hoje = Date%>
Eu tenho <%=CLng(Hoje-DataN)%> dias de vida<BR>
Farei 10000 dias de vida em <%=Cdate(Datan+10000)%>
</BODY></HTML>
```

O próximo exemplo faz uso do operador (^) exponenciação:

Exemplo 9.1 : exp1.asp

```
<% @ LANGUAGE=VBSCRIPT %>
<% Option Explicit %>
<HTML><HEAD><TITLE>Curso ASP</TITLE></HEAD><BODY>
<%  Dim Resultado
      Resultado = 2^2^3  %>
2^2^3 = <%=Resultado%>
</BODY></HTML>
```

Esse script da a falsa impressão de estarmos calculando o valor da expressão da direita para a esquerda, ou seja, Resultado = 2^**2^3** = **2^ 8** = 256. Mas o que percebemos foi que o resultado gerado é igual a 64, ou seja, Resultado = **2^2**^3 = **4^3**=64. Para evitar esse tipo de confusão com relação à ordem de cálculo de uma expressão devemos utilizar parênteses. A sub expressão que estiver dentro de parênteses têm prioridade de cálculo em cima das demais.

Exemplo 9.2 : exp2.asp

```
<% @ LANGUAGE=VBSCRIPT %>
<% Option Explicit %>
<HTML><HEAD><TITLE>Curso ASP</TITLE></HEAD><BODY>
<%  Dim Resultado
      Resultado = 2^(2^3)  %>
2^(2^3) = <%=Resultado%>
</BODY></HTML>
```

## 6.5. SUBROTINAS

VBScript possui dois tipos de subrotinas: Sub e Function. Um Sub é um conjunto de comandos associados a um identificador alfanumérico. Uma Function possui a mesma definição só que além de executar os comandos a ela associados, pode gerar um valor como resultado. Essas subrotinas podem ainda receber algum(s) valor(s) como parâmetro. Veja a seguir suas respectivas sintaxes:

```
Sub NomeDoProcedimento([Parâmetro(s)])
   Comando(s)
End Sub

Function NomeDaFunção([Parâmetro(s)])
   Comando(s)
End Function
```

Dentro da Function devemos implementar um mecanismo para passar o valor calculado para fora da function. Basta atribuir tal valor ao identificador da Function.

Na verdade, o identificador da Function é uma expressão pois retorna um valor. Para verificarmos o valor retornado por uma Function, colocamos a mesma no lado direito de uma atribuição:

Exemplo 10.1: sub1.asp

```
<% @ LANGUAGE=VBSCRIPT %>
<HTML><HEAD><TITLE>Curso ASP</TITLE></HEAD><BODY>
<% Function Soma(a,b)
        Dim Resultado
        Resultado = a + b
        Soma = Resultado
  end Function

  Dim c,d,e
  e = Soma(10,20) %>
Soma(10,20)= <%=e%><BR>
Soma(100,200)=<%=Soma(100,200)%>
</BODY></HTML>
```

Esse exemplo serve para ilustrar as vantagens de se trabalhar com subrotinas : programa fica mais estruturado e dependendo do número de comandos de uma subrotina, existe uma economia de linhas de código.

Cabem aqui algumas observações. A variável declarada dentro da Function é dita variável local da Subrotina, pois seu valor não pode ser "enxergado" fora da mesma. Por outro lado, as variáveis c,d,e podem ser "vistas" pela Function pois estas são variáveis de escopo global.

Exemplo 10.2: sub2.asp

```
<HTML><HEAD><TITLE>Curso ASP</TITLE></HEAD><BODY>
<% Dim A,B
     A = "Variável Global A"
     B = "Variável Global B"
    MudaB  %>
Valor de A = <%=A%><BR>
Valor de B = <%=B%>

<% Sub MudaB()
       Dim A
        A = "Variável A no SUB"
        B = "Variável B no SUB"
     End Sub
 %>
</BODY></HTML>
```

Observe que o código para o SUB(ou Function) não precisa ser escrito antes de sua chamada.

## 6.6. INCLUDE FILES

Essa é mais uma das formas que existe para poupar trabalho dos programadores economizando linhas de código. A idéia é criar um arquivo texto de qualquer extensão que contenha um conjunto de subrotinas. Essas, estarão disponíveis a qualquer página asp que faça referência a esse arquivo.

Existem duas formas de referenciar tal arquivo numa página ASP:

<!-- #INCLUDE VIRTUAL=”Path_Virtual/Nome_Arquivo” --> ou

<!-- #INCLUDE FILE=”Path_FÍSICO/Nome_Arquivo” -->

Exemplo 11 : subrotinas.inc

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

  Function Dias(DiaI,DiaF)
     Dias = CDate(DiaF)-CDate(DiaI)
  End Function

  Function Horas(DiaI,DiaF)
      Horas = Dias(DiaI,DiaF)*24
  End Function

</SCRIPT>
```

Exemplo 11: includefiles.asp

```
<% @ LANGUAGE=VBSCRIPT %>
<!-- #INCLUDE VIRTUAL="curso/subrotinas.inc" -->
<HTML><BODY>
   Eu tenho <%=Dias(#02/10/1978#,Date)%> dias de vida !<BR>
   Ou seja, mais ou menos <%=Horas(#02/10/1978#,Date)%> horas de
vida!
</BODY></HTML>
```

## 6.6 FUNÇÕES

| Função | Descrição |
|---|---|
| Abs(valor_numérico) | Retorna o módulo de um número |
| Fix (valor_numérico) | Retorna a parte inteira de um número |
| Int(valor_numérico) | Retorna a parte inteira de um número. Quando o valor numérico é menor que zero, é retornado o valor imediatamente menor.<br>Ex: Fix(-1,4) = -2<br>Int(1.4) = 1 |
| Sin(valor_numérico) | Retorna o seno de um determinado angulo (radianos) |
| Cos (valor_numérico) | Retorna o coseno de um determinado angulo (radiando) |
| Tan (valor_numérico) | Retorna a tangente de um determinado angulo (radiando) |
| Atn (valor_numérico) | Retorna o arco tangente(radiando) de um |

| | |
|---|---|
| | determinado valor |
| Log (valor_numérico) | Retorna o Logaritmo Neperiano de um número |
| Exp (valor_numérico) | Retorna e^valor_numérico |
| Sqr (valor_numérico) | Retorna a raiz quadrada de um valor numérico maior ou igual a zero |
| Date | Retorna a Data atual |
| Time | Retorna a Hora atual |
| Now | Retorna a Data/Hora atual |
| Day (valor_data) | Retorna o dia de uma determinada data |
| Month (valor_data) | Retorna o mês de uma determinada data |
| Year (valor_data) | Retorna o ano de uma determinada data |
| Weekday (valor_data) | Retorna o dia da semana no formato numérico de uma determinada data |
| Hour (tempo) | Retorna a hora de uma determinada expressão de tempo |
| Minute (tempo) | Retorna os minutos de uma determinada expressão de tempo |
| Second (tempo) | Retorna os segundos de uma determinada expressão de tempo |
| TimeSerial(hora,minuto,segundo) | Retorna uma expressão de tempo |
| DateSerial(ano,mes,dia) | Retorna caracteres são de data |
| Asc (caractere) | Retorna o número correspondente na tabela ASCII do caractere informado |
| Chr (valor inteiro) | Retorna o caractere ASCII correspondente ao valor inteiro informado |
| Lcase (string) | Converte todos os caracteres de uma string para minúsculas |
| Ucase(String) | Converte todos os caracteres de uma string para maiúscolas |
| Len (String) | Retorna o número de caracteres de uma determinada string |
| InStr (pos_inicial,string,substring) | Retorna a posição do primeiro caractere de uma substring numa determinada string. Pos_inicial indica a posição da string onde devemos começar a busca.<br>Ex : InStr(1,”Lineu”,”eu”) = 4<br>InStr(3,”Lineu”,”eu”) = 4 |
| Mid (string,pos_inicial,Tamanho) | Retorna uma substring a partir de uma string informada , bastando apenas especificarmos qual a posição inicial da substring e seu tamanho.<br>Ex : Mid (“Lineu”,4,2) = eu |
| Left (string,Tamanho) | Corta uma string a partir do lado esquerdo. Tamanho é o número de caracteres da nova string<br>Ex: Left (“Lineu”,3) = Lin |
| Right(string,Tamanho) | Corta uma string a partir do lado direito. |

| | |
|---|---|
| | Tamanho é o número de caracteres da nova string<br>Ex: Right (“Lineu”,2) = eu |
| Ltrim (String) | Retira caracteres de espaços que possam existir no lado esquerdo de uma string<br>Ex : Ltrim(“ Lineu S ”) = “Lineu S “ |
| Rtrim (String) | Retira caracteres de espaços que possam existir no lado direito de uma string<br>Ex : Ltrim(“ Lineu S ”) = “ Lineu S“ |
| Trim (String) | Retira caracteres de espaço do inicio e do fim de uma string<br>Ex : Trim(“ Lineu S ”) = “Lineu S” |
| FormatCurrency(Valor Numérico) | Formata um valor numérico para o padrão moeda configurado no computador servidor. |
| FormatNumber(Valor Numérico) | Formata um valor numérico para o padrão numérico configurado no computador servidor. |

Exemplo 12.1 : funcoes1.asp

```
<% @ Language = VBScript%><% Option Explicit%>
<HTML><HEAD><TITLE>Curso ASP</TITLE><BODY>
  <% Dim s
     S = "  Lineu  "
     S = Trim(S)  %>
 Caracteres da palavra <%=UCase(s)%> :<BR>
 1 - <%=Mid(s,1,1)%><BR> 2 - <%=Mid(s,2,1)%><BR>
 3 - <%=Mid(s,3,1)%><BR> 4 - <%=Mid(s,4,1)%><BR>
 5 - <%=Mid(s,5,1)%><BR>
</BODY></HTML>
```

Verifique no próximo exemplo a ocorrência de erros:

Exemplo 12.2 : funcoes2.asp

```
<% @ Language = VBScript%>
<% Option Explicit%>
<HTML><HEAD><TITLE>Curso ASP</TITLE>
<BODY>
  <% Dim dia,mes,ano
     dia = 2
     mes = 31
     ano = 2000  %>
<%=DateSerial(ano,mes,dia) %>
<BR><%=CDate(dia & "/" & mes & "/" & ano)%>
</BODY>
</HTML>
```

Exemplo 12.3 : funcoes3.asp

```
<% @ Language = VBScript%>
<% Option Explicit%>
<HTML><HEAD><TITLE>Curso ASP</TITLE>
<BODY>
Eu gostaria de ganhar <%=FormatCurrency(50000)%> por mês!<BR>
Hoje eu ganho <%=FormatNumber(35/3)%> por dia...
</BODY></HTML>
```

## 6.7 ESTRUTURAS DE CONTROLE DE DECISÃO

São estruturas embutidas numa linguagem de programação que nos permite executar determinado conjunto de comandos de acordo com uma determinada condição.

O primeiro comando do VBScript que se encaixa nessa descrição é o IF:

```
If <Condição> Then
   Comando(s)
End if
```

Esse comando testa a Condição, se mesma for verdadeira, o bloco de comandos entre If e End If será executado.

Exemplo 13.1 : if1.asp

```
<% @ Language = VBScript%>
<HTML><HEAD><TITLE>Curso ASP</TITLE></HEAD>
<BODY>
 <%  if WeekDay(Date)=1 then  %>
     Hoje é Domingo
 <%  end if  %>
 <%  if WeekDay(Date)<>1 then  %>
     Hoje não é Domingo
 <%  end if  %>
 </BODY></HTML>
```

O exemplo anterior apesar de correto poderia ser escrito de uma forma mais eficiente. Para tal, devemos utilizar o comando If associado a Else:

```
If <Condição> Then
   Comandos_1
Else
   Comandos_2
End if
```

Quando o interpretador encontra esse comando testa a condição, se for verdadeira, executa o bloco de Comandos_1. Se a condição for falsa(0), o interpretador pula para Else e executa o bloco de comandos_2.

Exemplo 13.2 : if2.asp

```
<% @ Language = VBScript%>
<HTML>
<HEAD>
<TITLE>Curso ASP</TITLE>
</HEAD>
<BODY>
 <%  if WeekDay(Date)=1 then  %>
      Hoje é Domingo
 <%  else  %>
      Hoje não é Domingo
 <%  end if  %>
 </BODY>
</HTML>
```

Como dá pra perceber, há aqui uma economia de processador uma vez que o teste da condição agora só é realizado uma vez.

Podemos também ter vários If..Then...Else aninhados, sendo que cada clausula Else estará ligada ao If .. Then imediatamente anterior.

Exemplo 13.3 : if3.asp

```
<% @ Language = VBScript%>
<HTML><HEAD><TITLE>Curso ASP</TITLE></HEAD><BODY>
 <%  if WeekDay(Date)=1 then  %>
         Hoje é Domingo
 <%  else
          if WeekDay(Date)=7 then  %>
               Hoje é Sábado
<%     else       %>
               Hoje é dia de trabalhar e estudar
<%      end if
       end if  %>
</BODY>
</HTML>
```

No exemplo anterior temos um **End IF** para cada **If .. then .. Else**. Isso dificulta bastante a leitura do código por parte do programador. Para melhorar a nossa vida, o VBScript possui uma estrutura derivada da anterior: o **If Then ElseIf**. A diferença dessa estrutura para a anterior é que agora só precisamos de um **endif** ao final de todos os comandos. Sintaxe:

```
If <Condição> Then
   Comandos_1
ElseIF <Condição>
   Comandos_2
```

**Else**
**Comando_3**
**End if**

Exemplo 13.4 : if4.asp

```
<% @ Language = VBScript%>
<HTML><HEAD><TITLE>Curso ASP</TITLE>
<BODY>
 <%  if WeekDay(Date)=1 then  %>
            Hoje é Domingo
 <%  elseif WeekDay(Date)=7 then  %>
            Hoje é Sábado
<%   else      %>
            Hoje é dia de trabalhar e estudar
<%   end if %>
</BODY></HTML>
```

Ainda existe uma estrutura alternativa ao If..Then..Else, mais flexível e mais elegante: o Select Case. Sintaxe:

**Select Case Expressao**
**Case Condição**
**Comandos_1**
**Case Condição 2**
**Comandos_2**
**Case Else**
**Comandos_3**
**End Select**

O resultado da expressão será comparado com uma série de condições, até encontrar uma que case com ele. Quando isso ocorre, os comandos que estiverem associados à condição serão executados. Se nenhuma das condições for satisfeita e houver a cláusula Case Else então os comandos associados a ela serão executados.

Exemplo 14 : SelCase.asp

```
<% @LANGUAGE=VBSCRIPT%>
<HTML>
<HEAD><TITLE>Curso de ASP</TITLE></HEAD>
<BODY>

<%      Dim DiaS

        Select Case WeekDay(Date)
                Case 1
                     DiaS = “Domingo”
                Case 2
```

```
                    DiaS = “Segunda-feira”
                Case 3
                    DiaS = “Terça-feira”
                Case 4
                    DiaS = “Quarta-feira”
                Case 5
                    DiaS = “Quinta-feira”
                Case 6
                   DiaS = “Sexta-feira”
               Case Else
                   DiaS = “Sábado”
          End Select   %>
Hoje é <%=DiaS%>
</BODY>
</HTML>
```

## 6.8 ESTRUTURAS DE CONTROLE DE REPETIÇÃO

É possível repetir um bloco de instruções dentro de um programa escrito é VBScript. Para isso existem sete tipos diferentes de estruturas de repetição, denominadas de loop:

| | |
|---|---|
| **Do Until** <Condição><br>Comandos<br>**Loop** | Executa um bloco de instruções até que a Condição se torne verdadeira |
| **Do**<br>Comandos<br>**Loop Until** <Condição> | Só difere da estrutura anterior pois a condição só é testada no final. Sendo assim, o comando do laço será executado pelo menos uma vez. |
| **Do While** <Condição><br>Comandos<br>**Loop** | Executa um bloco de comandos enquanto a condição for verdadeira |
| **Do**<br>Comandos<br>**Loop While** <Condição> | Só difere da estrutura anterior pois a condição só é testada no final. Sendo assim, o comando do laço será executado pelo menos uma vez. |
| **While** <Condição><br>Comandos<br>**Wend** | Mesma coisa que **Do While ... Loop** |
| **For** Variável=limiteI **to** limiteF **Step** N<br>Comandos<br>**Next** | A Variável será iniciada com o valor limiteI. A cada execução do laço ela é incrementada em N. Quando seu valor ultrapassar de limiteF, o laço se encerra. |
| **For Each** Elemento **In** Coleção<br>Comandos<br>**Next** | Parecido com a estrutura anterior, só que aqui o bloco de instruções é executado para cada elemento existente numa matriz ou numa coleção de objetos. |

Exemplo 15.1 : loop1.asp

```
<% @LANGUAGE=VBSCRIPT%>
        <HTML>
        <HEAD><TITLE>Curso de ASP</TITLE></HEAD>
        <BODY>
        Números pares menores que 100:<BR>*
        <%  Dim i
            For i=0 to 100 step 2   %>
                <%=i%> *
        <%     Next
           i = 1  %>
        <P>Números Impares menos que 100:<BR>*
        <%    Do While i<=100    %>
                <%=i%> *
        <%      i = i + 2
           Loop %>
        </BODY>
</HTML>
```

Exemplo 15.2 : loop2.asp

```
<% @LANGUAGE=VBSCRIPT%>
        <HTML>
        <HEAD><TITLE>Curso de ASP</TITLE></HEAD>
        <BODY>
        <% Dim Nomes(4), Nome
            Nomes(0)=”Lineu”
            Nomes(1)=”Atslands”
            Nomes(2)=”Orlando”
            Nomes(3)=”Yasnaya”
            Nomes(4)=”Luciana”
        For each nome in Nomes  %>
            <%=nome%><BR>
        <%  Next%>
</BODY></HTML>
```

Para encerrar esse capitulo sobre laços, falaremos um pouco do comando EXIT. Ele permite que se saia permanentemente de um loop, de uma função ou de um Sub. A tabela seguinte apresenta o seu formato para cada situação:

| | |
|---|---|
| **Exit Do** | Dentro de laços que começam com **DO** |
| **Exit For** | Dentro de laços que começam com **FOR** |
| **Exit Function** | Para sair de uma Function |
| **Exit Sub** | Para abandonar um Sub |

Exemplo 16 : exit.asp

```
<% @LANGUAGE=VBSCRIPT%>
<HTML>
<HEAD><TITLE>Curso de ASP</TITLE></HEAD>
<BODY>
<% Dim i
     i = 0
     Do
          i=i+1
          if i<=10 then   %>
            <%=CStr(i) & " "%>
<%       else
            Exit Do
     Loop Until False%>
</BODY></HTML>
```

## 6.8 TRATAMENTO DE ERROS

No VBScript existe um objeto interno utilizado para indicar situações de erro em tempo de interpretação: o objeto ERR. É um objeto bastante simples que contém somente duas propriedades:

- ✓ **Description** : Mensagem original do erro
- ✓ **Number** : Número associado ao erro

Quando não fazemos uso desse objeto, os usuários do nosso site podem receber mensagens de erro "indecifráveis". A idéia então é escrever um código capaz de detectar os erros e enviar mensagens amigáveis aos usuários.

Exemplo 17.1 : erro1.asp

```
<% @LANGUAGE=VBSCRIPT%>
<HTML>
<HEAD><TITLE>Curso de ASP</TITLE></HEAD>
<BODY>
<% Dim i
     For i=10 to 0 step –1   %>
        100 Dividido por <%=i%> = <%=(100/i)%><BR>
<% next %>
</BODY></HTML>
```

Para usar nosso objeto devemos adicionar o seguinte comando no início do nosso script: **ON ERROR RESUME NEXT .** Esse comando informa ao interpretador que se ocorrer algum erro na interpretação de algum comando do script, deve-se descartar esse comando, atualizar o objeto ERR e executar a próxima linha. Basta agora escrever código em qualquer lugar do script para verificar a ocorrência do erro para posterior tratamento.

Exemplo 17.2 : erro2.asp

```
<% @LANGUAGE=VBSCRIPT%>
<HTML>
<HEAD><TITLE>Curso de ASP</TITLE></HEAD>
<BODY>
<% On Error Resume Next
      Dim i
      For i=10 to 0 step –1   %>
         100 Dividido por <%=i%> = <%=(100/i)%><BR>
<% next
     if err then  %>
        <B>OCORREU UM ERRO [<%=Err.Number%>] : <%=Err.Description%>
<%end if%>
</BODY></HTML>
```

## 7. OBJETOS INTERNOS

Além da programação com VBScript, ASP disponibiliza cinco objetos "internos" que promovem a interação entre nossos scripts e o ambiente. Trata-se de estruturas especiais que possuem propriedades, métodos, eventos, coleções etc Dentre outras coisas, tais objetos servem para:

- Verificar dados informados pelo clientes Web
- Enviar respostas HTML para tais clientes
- Instanciar objetos ActiveX em seus scripts
- Permitem a comunicação entre clientes conectados ao aplicativo ASP

Objetos internos do ASP são:

| | |
|---|---|
| **Application** | Representa um conjunto de páginas de um mesmo diretório virtual |
| **Session** | Representa uma Sessão aberta com um Cliente via Web Browser |
| **Server** | Representa o Servidor Web em si, permitindo acesso a algumas propriedades do mesmo e a criação de instâncias de Objetos Activex |
| **Response** | Representa as respostas HTML enviadas ao cliente |
| **Request** | Representa os dados enviados por um formulário HTML ao Servidor Web |

## 7.1 APPLICATION

Ao conjunto de páginas ASP de um mesmo diretório virtual damos o nome de Aplicação ASP. Tal aplicação será iniciada na primeira vez que um usuário tentar acessar alguma página desse diretório virtual. Será finalizada quando o servidor web for desligado.

O objeto Application existe para nos possibilitar o armazenamento e recuperação de valores relacionadas a uma aplicação ASP. Com ele podemos criar variáveis de qualquer subtipo cujo valor pode ser acessado ou modificado por qualquer usuário conectado ao diretório virtual.

Para criar uma variável do nível de aplicação, devemos escrever comandos com seguinte sitaxe:

**Application(“NOME_DA_VARIAVEL”) = VALOR_DA_VARIAVEL**

Uma vez criada, tal variável estará acessível a qualquer usuário da aplicação. Seu valor ficará armazenado até que o servidor web seja desligado.

Como o conteúdo desse tipo de variável pode ser modificado por qualquer usuário conectado à aplicação, poderia haver alguma confusão se vários usuários tentassem alterar esse valor ao mesmo tempo. Para evitar possíveis problemas com a “concorrência", o objeto application disponibiliza dois métodos: LOCK e UNLOCK.

O primeiro bloqueia as variáveis de nível de aplicação para o usuário que invoca tal método. Se qualquer outro “usuário” tentar acessar variáveis desse nível, ficará esperando até a aplicação ser desbloqueada.

A aplicação só será desbloqueada quando o script que a bloqueou termina sua execução, ou quando ocorre o “TimeOut”, ou quando o script invoca o método UNLOCK.

Ainda relacionado a esse objeto existem dois eventos:

| | |
|---|---|
| **Application_OnStart** | Ocorre quando a aplicação é iniciada, ou seja, quando um diretório virtual é acessado pela primeira vez. |
| **Application_OnEnd** | Ocorre quando a aplicação é finalizada, ou seja, quando o web server é desligado. |

Um evento é uma subrotina automaticamente chamada quando o sistema sofre alguma ação especifica. Tais subrotinas não são escritas diretamente nas páginas ASP mas num arquivo a parte nomeado de GLOBAL.ASA.

Sendo assim, quando um diretório virtual for acessado pela primeira vez, o Servidor Web procura em tal diretório a existência desse arquivo. Se encontra, abre o arquivo e procura a subrotina Application_OnStart para executar seus comandos. A mesma coisa acontece quando desligamos o servidor web, só que ele chama a subrotina Application_OnEnd.

No exemplo a seguir criamos uma variável de nível de aplicação chamada DataHoraI para armazenar a Data/Hora em que a aplicação foi iniciada. Outra variável chamada Titulo para armazenar o titulo da aplicação ASP. E uma variável chamada Correio que armazena o e-mail do Web Master:

Exemplo 18.1: Global.asa

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

Sub Application_OnStart()
   Application(“DataHoraI”)=Now
  Application(“Titulo”)=”Curso de ASP”
  Application(“Correio”)=”mailto:lineus@seduc.pi.gov.br”
End Sub

</SCRIPT>
```

Exemplo 18.2 : Application1.asp

```
<% @LANGUAGE=VBSCRIPT %>
<HTML><HEAD><TITLE><%=Application(“Titulo”)%></TITLE></HEAD>
<BODY>
Essa aplicação ASP foi iniciada em <B><%=Application(“DataHoraI”)%></B><BR>
<A Href="<%=Application("Correio")%>">Web Master</a>
</BODY>
</HTML>
```

**Obs** : Só poderá existir um arquivo Global.asa em cada diretório virtual

Observe que o trabalho de manutenção do site pode ficar facilitado. Imagine que todas as páginas asp do seu diretório virtual possuem um padrão de cores, links, cabeçalho, etc. Sendo assim, as páginas teriam muito código em comum. Se desejarmos modificar os padrões do nosso site, teríamos que fazer alterações em todos os arquivos do diretório virtual. Mas se utilizarmos variáveis de nível de aplicação para armazenar essas configurações, não necessitamos mudar todos os arquivos do diretório, mas só o arquivo GLOBAL.ASA.

Exemplo 18.3: Global.asa

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

Sub Application_OnStart()
   Application(“DataHoraI”)=Now
   Application(“Titulo”)=”I Curso de ASP da UFPI”
   Application(“Correio”)=mailto:lineulima@yahoo.com
   Application(“CorFundo”)=”Black”
   Application(“CorTexto”)=”Yellow”
   Application(“TamFonte”)=”4”
End Sub

</SCRIPT>
```

Exemplo 18.4: Application2.asp

```
<% @LANGUAGE=VBSCRIPT %>
<HTML><HEAD><TITLE>
<%=Application("Titulo")%>
</TITLE></HEAD>
<BASEFONT                                    SIZE=<%=Application("TamFonte")%>
COLOR=<%=Application("CorTexto")%>>
<BODY BGCOLOR=<%=Application("CorFundo")%>>
Essa aplicação ASP foi iniciada em <B><%=Application("DataHoraI")%></B><BR>
<A Href="<%=Application("Correio")%>">Web Master</a>
</BODY>
</HTML>
```

## 7.2 SESSION

Toda vez que um usuário Web se conecta a um aplicativo ASP é iniciada uma sessão para o mesmo no servidor Web. Para representar tal sessão, o ASP possui um objeto interno chamado Session.

Na verdade, ele é muito parecido com o objeto Application. A diferença esta em dizer que esse objeto pode armazenar valores ligados apenas a um único visitante do site(o dono da sessão). Com ele podemos criar variáveis de qualquer subtipo cujo valor pode ser acessado ou modificado somente pelo "dono" da sessão.

Para criar uma variável do nível de sessão, devemos escrever comandos com seguinte sitaxe:

**Session("NOME_DA_VARIAVEL") = VALOR_DA_VARIAVEL**

As variáveis de sessão permanecerão na memória(ativas) até a sessão ser encerrada. Isso pode acontecer quando o usuário fechar o web browser, quando ocorre o "TIMEOUT" da sessão, ou quando o script invoca o método ABANDON do objeto Session.

A propriedade TIMEOUT é usada quando o usuário fica parado sem fazer nada no Browser. O default é vinte minutos, mas esse valor pode ser modificado da seguinte forma:

**Session.Timeout = VALOR_MINUTOS**

Ainda relacionado a esse objeto existem dois eventos:

| | |
|---|---|
| **Session_OnStart** | Ocorre quando a sessão é iniciada |
| **Session_OnEnd** | Ocorre quando a sessão é finalizada. |

A exemplo dos eventos do objeto Application, eles também devem ser escritos como subrotinas de um arquivo GLOBAL.ASA.

O exemplo ilustra o conceito de sessão. Temos uma variável a nível de aplicação chamada contador. Ela serve para informar a quantidade de pessoas que acessaram essa aplicação ASP. A idéia é incrementar o valor dessa variável toda vez que uma sessão é iniciada. Também utilizamos uma variável de sessão que informa a hora em que a sessão em questão foi aberta.

Exemplo 19.1: Global.asa

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

Sub Application_OnStart()
  Application("DataHoraI")=Now
  Application("Titulo")="I Curso de ASP da UFPI"
  Application("Correio")=mailto:lineulima@yahoo.com
  Application("CorFundo")="Black"
  Application("CorTexto")="Yellow"
  Application("TamFonte")="4"
  Application("Contador")=0
End Sub
```

```
Sub Session_OnStart()
     Application("Contador")=Application("Contador")+1
    Session(“HoraS”)=Time
End Sub

</SCRIPT>
```

A página a seguir encerra a sessão através do método ABANDON:

Exemplo 19.2 sessao1.asp

```
<% @LANGUAGE=VBSCRIPT %>
<HTML><HEAD><TITLE>
</TITLE></HEAD>
Essa sessão foi iniciada às <%=Session(“HoraS”)%><BR>
Você é o visitante de número <%=Session(“Contador”)%><BR>
Desde <%=Application(“DataHoraI”)%>
<HR>
<A HREF=”sessao2.asp”>Encerrar Sessão</A>
</BODY>
</HTML>
```

## 7.3. RESPONSE

Antes de uma página ASP ser enviada ao cliente o Web Server lê seu conteúdo. Se encontra TAGS HTML ou texto, envia diretamente ao cliente. Se encontra scripts, executa-os e repassa o resultado HTML para o Web Browser. Para representar essas respostas HTML, ASP possui o objeto interno RESPONSE.

Com esse objeto podemos enviar simples comandos HTML ou texto, podemos redirecionar o browser para buscar informações em outro URL, podemos bufferizar a resposta ao cliente, bem como interromper o envio da página ASP.

Para tanto, esse objeto possui as seguintes propriedades e métodos:

| | |
|---|---|
| **Write** Texto | Com esse método podemos enviar qualquer texto ao web browser. |
| **End** | Método que termina o envio da página ASP ao cliente, mesmo que ela não tenha chagado ao final. |
| **Buffer =** Valor Booleano | Quando Buffer=True, o servidor web só enviará a página ASP ao cliente quando encerrar toda a leitura da mesma, ou quando utilizarmos nesse script o metódo Flush. Essa propriedade só pode ser utilizada antes de qualquer comando que gere respostas HTML ao cliente. |
| **Redirect** URL | Permite redirecionar o Browser do Cliente para outra página(URL). Se a propriedade Buffer for diferente de True, esse método só poderá ser chamado antes de qualquer comando que gere respostas HTML ao cliente. |

| | |
|---|---|
| **Clear** | Esse método apaga todo o conteúdo do Buffer se o mesmo estiver ativo. |
| **Flush** | Esse método pode ser utilizado para enviar o conteúdo do Buffer para o cliente WEB. |
| **Cookies(“**nome_cookie**”) =** valor | Cria cookies ou altera seu valor. Se a propriedade Buffer for diferente de True, essa propriedade só poderá ser chamado antes de qualquer comando que gere respostas HTML ao cliente. |
| **Expires =** minutos | Essa propriedade informa o tempo(minutos) em que uma página ASP pode permanecer ativa na Cache do Web Browser. |
| **ExpiresAbsolute =** #data hora# | Essa propriedade informa a data e a hora exata em que a página expira da Cache do Web Browser. |
| **AppendToLog** Texto | Esse método escreve um texto no arquivo de LOG do Web Server. |

Com o objeto response podemos escrever uma página ASP 100% script:

Exemplo 20.1 : Response1.asp

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

RESPONSE.WRITE “<HTML><HEAD><TITLE>Curso de ASP</TITLE></HEAD>”
RESPONSE.WRITE “<BODY><CENTER>Testando objeto Response</CENTER>”
RESPONSE.WRITE “</BODY></HTML>”

</SCRIPT>
```

Podemos também escrever uma página ASP que simplesmente Redireciona o usuário para outro site:

Exemplo 20.2 : Response2.asp

```
<% @LANGUAGE=VBSCRIPT %>
<% RESPONSE.REDIRECT “http://www.ufpi.br”%>
```

Quando utilizamos Buffer=True, temos mais controle do que será enviado como resposta ao cliente:

Exemplo 20.3 Response3.asp

```
<% @LANGUAGE=VBSCRIPT %>
<% RESPONSE.BUFFER=TRUE
 DIM Inicio,Fim

 Inicio = "<HTML><HEAD><TITLE>Curso de ASP</TITLE></HEAD><BODY>"
 Fim = "</BODY></HTML>"
RESPONSE.WRITE Inicio
RESPONSE.WRITE "Esse texto não será enviado ao Cliente!"
RESPONSE.WRITE Fim
```

```
RESPONSE.CLEAR
RESPONSE.WRITE Inicio & Chr(13)
RESPONSE.WRITE "Esse texto será enviado ao cliente!" & Chr(13)
RESPONSE.WRITE Fim
%>
```

## 7.4 FORMULÁRIOS HTML

Antes de continuar com os objetos internos do ASP devemos começar a nos preocupar com a construção de páginas interativas: os formulários Web. São interfaces HTML utilizadas para obter dados de um usuário Web para um programa que roda no Web Server.

Num formulário podemos ter os seguintes objetos: caixas de texto, botões, comboboxes, cheked boxes e radio buttons. A idéia é permitir que o usuário digite dados ou escolha opções e com um simples clique num objeto botão, tais dados serão passados para um determinado programa do Web Server.

Para montar um formulário HTML usamos a Tag <FORM> </FORM>. Essa tag possui ainda alguns atributos:

**METHOD**

Indica o método pelo qual os dados informados no formulário serão passados ao programa do Web Server.
Quando METHOD=POST, os dados serão passados junto com a requisição do cliente.
Quando METHOD=GET, os dados serão passados depois da requisição do cliente.

**ACTION**

Indica a URL da aplicação do servidor Web.

**NAME**

Indica o nome do formulário em questão. (Opcional)

Depois dessa tag devemos informar quais são os objetos de interface do formulário.

**CAIXA DE TEXTO SIMPLES**

<INPUT TYPE=Tipo NAME=NomeC VALUE=Valor MAXLENGTH=Tam_Max READONLY>

- Tipo: PASSWORD(só aparecem asteriscos), TEXT(aparecem caracteres normais)
- NomeC : nome da caixa de texto em questão;
- Valor :texto que aparece na caixa (opcional);
- Tam_Max : indica quantos caracteres podem ser digitados(Opcional);
- READONLY : indica que a caixa de texto é somente leitura (Opicional);

Exemplo 21.1 Form1.asp

```
<% @ LANGUAGE= VBSCRIPT %>
<HTML><HEAD><TITLE>Curso de ASP</TITLE></HEAD>
<BODY>
```

```
<FORM ACTION="form2.asp" METHOD=GET>
  Usuário<BR>
 <INPUT TYPE=TEXT NAME="user" VALUE="Lineu" MAXLENGTH=10><BR>
  Senha<BR>
 <INPUT TYPE=PASSWORD NAME="senha" MAXLENGTH=10><BR>
  Data<BR>
 <INPUT TYPE=TEXT NAME="Data" VALUE="<%=Date%>" READONLY>
</FORM>
</BODY>
</HTML>
```

## ÁREA DE TEXTO

```
<TEXTAREA NAME=NomeA COLS=NumCols ROWS=NumLin READONLY>
        Texto Default
</TEXTAREA>
```

- NomeA : Nome da área de texto;
- NumCols: Quantidade de colunas;
- NumLin : Quantidade de linhas;
- READONLY : indica que a área é somente leitura (Opicional);
- Texto Default : texto que aparece na área;

Exemplo 21.2 Form2.asp

```
<% @ LANGUAGE= VBSCRIPT %>
<HTML><HEAD><TITLE>Curso de ASP</TITLE></HEAD>
<BODY>
<FORM ACTION="form2.asp" METHOD=GET>
 <TEXTAREA NAME=”Obs” COLS=50 ROWS=5>
  <% For i=0 to 100
        Response.Write “Linha número “ & i & chr(13)
      Next %>
  </TEXTAREA>
</FORM>
</BODY>
</HTML>
```

## CAIXAS DE COMBINAÇÃO

```
<SELECT SIZE=Tam NAME=NomeCC>
      <OPTION VALUE=valor_passado SELECTED>  Valor Mostrado </OPTION>
<\SELECT>
```

- Tam :Quantidade de linhas (default=1);
- NomeCC : Nome do objeto;
- Valor_Passado : valor correspondente a essa opção;

- SELECTED: Indica que essa será a opção Default;
- Valor_Mostrado : valor mostrado no Web Browser;

Obs : Podem existir várias opções..

Exemplo 21.3 Form3.asp

```
<% @ LANGUAGE= VBSCRIPT %>
<HTML><HEAD><TITLE>Curso de ASP</TITLE></HEAD>
<BODY>
<FORM ACTION="form2.asp" METHOD=GET>
   <SELECT SIZE=1 NAME=”Ano”>
       <OPTION VALUE=1998>1998</OPTION>
       <OPTION VALUE=1999>1999</OPTION>
       <OPTION VALUE=2000 SELECTED>2000</OPTION>
       <OPTION VALUE=2001 >2001</OPTION>
   </SELECT>
</FORM>
</BODY>
</HTML>
```

## BOTÕES DE SELEÇÃO E CHECAGEM

```
<INPUT TYPE=tipo NAME=NomeR VALUE=Valor CHECKED>
```

- Tipo : RADIO(Marca somente um) CHECKBOX(pode marcar mais de um)
- NomeR : nome do objeto;
- Valor : Valor relacionado ao botão de seleção;
- CHECKED: deixa o botão selecionado quando o formulário for carregado.

A idéia é ter vários objetos desses com o mesmo NAME.

Exemplo 21.4 Form4.asp

```
<% @ LANGUAGE= VBSCRIPT %>
<HTML><HEAD><TITLE>Curso de ASP</TITLE></HEAD>
<BODY>
<FORM ACTION="form2.asp" METHOD=GET>
  Sexo<BR>
   Homem <INPUT TYPE=RADIO NAME="Sexo" VALUE="HOMEM"><BR>
   Mulher <INPUT TYPE=RADIO NAME="Sexo" VALUE="MULHER"><BR>
   Indeciso <INPUT TYPE=RADIO NAME="Sexo" VALUE="HOMO"> <P>
  Ocupação <BR>
  Estuda<INPUT TYPE= CHECKBOX NAME="Ocp" VALUE="E"><BR>
  Trabalha<INPUT TYPE= CHECKBOX NAME="Ocp" VALUE="T">
</FORM>
</BODY>
</HTML>
```

**BOTÕES[**

<INPUT TYPE=tipo VALUE=Valor>

Tipo : SUBMIT (Chama página indicada em ACTION) RESET(Limpa formulário)
Valor : Rótulo do Botão

Exemplo 21.5 Form5.asp

```
<% @ LANGUAGE= VBSCRIPT %>
<HTML><HEAD><TITLE>Curso de ASP</TITLE></HEAD>
<BODY>
<FORM ACTION="form5.asp" METHOD=GET>
<INPUT TYPE=TEXT NAME="user" MAXLENGTH=10><BR>
<INPUT TYPE=SUBMIT VALUE=”Enviar”>
<INPUT TYPE=RESET VALUE=”Limpar”>
</FORM>
</BODY>
</HTML>
```

Quando pressionamos o botão Enviar observamos que a página é recarregada. Isso acontece porque o ACTION do nosso formulário é igual a Form5.asp. Se olharmos mais detalhadamente, a caixa de URL do browser vai conter um endereço mais ou menos parecido com: http://nome_servidor/diretório_virtual/Form5.asp?user=Lineu. Isso acontece pois METHOD = GET, user é o nome do nosso objeto caixa de texto do formulário e Lineu foi o valor digitado. Percebeu a importância que tem em nomearmos os objetos dos nossos formulários?

## 7.5 REQUEST

Esse objeto serve para possibilitar a captura em nossas páginas ASP dos dados passados pelos formulários HTML ao Servidor Web.

Existem três formas diferentes de fazermos essa captura, dependendo do METHOD:

| | |
|---|---|
| **GET** | **Request.QueryString(Nome_Objeto)** |
| **POST** | **Request.Form(Nome_Objeto)** |
| **GET OU POST** | **Request(Nome_Objeto)** |

**BOTÕES**

<INPUT TYPE=tipo VALUE=Valor>

Tipo : SUBMIT (Chama página indicada em ACTION) RESET(Limpa formulário)
Valor : Rótulo do Botão

Exemplo 22.1 Request1.asp

```
<% @ LANGUAGE= VBSCRIPT %>
<HTML><HEAD><TITLE>Curso de ASP</TITLE></HEAD>
```

```
<BODY>
<FORM ACTION="Request2.asp" METHOD=GET>
Usuário <INPUT TYPE=TEXT NAME="user" MAXLENGTH=10><BR>
Senha <INPUT TYPE=PASSWORD NAME="senha" MAXLENGTH=10><BR>
<INPUT TYPE=SUBMIT VALUE=”Enviar”>
<INPUT TYPE=RESET VALUE=”Limpar”>
</FORM>
</BODY>
</HTML>
```

Request2.asp

```
<% @ LANGUAGE=VBScript %>
<HTML><HEAD><TITLE>Curso de ASP</TITLE></HEAD>
<BODY>
Usuário = <%=Request.QueryString(“user”)%><BR>
Senha = <%=Request(“senha”)%>
</BODY>
</HTML>
```

Observe que o método GET não é adequando para formulário que exigem alguma informação secreta(senha) pois o que foi digitado no mesmo é mostrado na caixa de endereço do Browser. Sendo assim, procure adaptar o exemplo anterior ao método POST.

Com o objeto Request somos capazes também de:

| | |
|---|---|
| Ler Cookies | **Request.Cookies(“NomeCookie”)** |
| Saber o endereço IP do usuário | **Request.ServerVariables(“REMOTE_ADDR”)** |
| Saber o METHOD utilizado | **Request.ServerVariables(“REQUEST_METHOD”)** |

Exemplo 22.2 Request3.asp

```
<% @ LANGUAGE=VBScript %>
<HTML><HEAD><TITLE>Curso de ASP</TITLE></HEAD>
<BODY>
Endereço IP do usuário :
<%=Request.servervariables("REMOTE_ADDR")%>
</BODY>
</HTML>
```

**Obs** : Como Request.QueryString, Request.Form, Request.Cookies, Request.ServerVariables possuem coleções de variáveis, as mesmas podem ser lidas com um comando For Each.

Exemplo 22.3 Request4.asp

```
<% @LANGUAGE=VBSCRIPT %>
<HTML><HEAD><TITLE>Curso de ASP</TITLE></HEAD>
<BODY>
```

```
<%FOR EACH ITEM IN Request.ServerVariables
        Response.write ITEM & " – " & Request.ServerVariables(ITEM) & "<BR>"
    NEXT%>
</BODY></HTML>
```

## 7.6 SERVER

Representa o Servidor Web, permitindo acesso a algumas de suas propriedades. Além disso, ele possibilita a instanciação em páginas ASP de componentes ActiveX escritos por outros programadores.

Os objetos ActiveX fazem parte da tecnologia COM para criação de componentes de software reutilizáveis. A idéia é desenvolver objetos com esse padrão para que possam ser reutilizados por "qualquer" linguagem de programação.

Em particular, o ASP pode trabalhar com componentes COM, só que os mesmos não podem possuir interface visual. Sendo assim, em páginas ASP usamos os chamados ActiveX DLL`s, componentes que não fornecem nenhuma interface com o usuário, somente com aplicativos(métodos, propriedades).

Esses componentes estendem bastante nosso poder de programação. Dentre outras coisas, eles nos permitem:

- Acessar Banco de Dados
- Criar/Ler/Alterar arquivos
- Enviar e-mail

| | |
|---|---|
| **ScriptTimeOut** | Propriedade que determina o tempo máximo(segundos) que um script ASP pode ficar executando, antes que o Servidor Web o termine. Isso serve para proibir que scrits ASP fiquem executando "infinitamente". |
| **HTMLEncode** | Método que codifica uma string para o formato HTML. |
| **MapPath** | Método que retorna o PATH real de um determinado diretório virtual do servidor Web. |
| **URLEncode** | Esse método transforma uma string para o formato padrão de URL. |
| **CreateObject** | Cria uma instância de um componente ActiveX na página ASP:<br><br>Set Obj = Server.CreateObject("IDObjeto") |

Para exemplificar o uso do objeto Server trabalharemos com o ActiveX FileSystemObject. Tal componente possui uma serie de propriedades e métodos para manipulação de arquivos e diretórios do servidor Web.

Para instanciar tal objeto numa página ASP escrevemos o seguinte código:

```
Dim Objeto

Set Objeto = Server.CreateObject("Scripting.FileSystemObject")
```

Esse objeto recém instanciado representa o Sistema de Arquivo do servidor Web. Devemos agora criar outro objeto(TextStream), a partir desse, para representar um determinado arquivo. Fazemos isso, utilizando o método OpenTextFile do FileSystemObject:

```
Set Arquivo = Objeto.OpenTextFile(Nome,modo,cria,formato)
```

Onde:

| | |
|---|---|
| **Nome** | Nome do arquivo a ser utilizado pelo script |
| **Modo** | Modo de Abertura do arquivo. 1 para leitura, 2 para gravação por cima, 3 para gravação adicional. |
| **Cria** | Valor Booleano que indica se o arquivo deve ser criado(true) ou não(false) caso não exista. |
| **Formato** | Indica o formato de gravação do arquivo a ser utilizado. –1 Unicode, 0 Ascii |

Para lermos o conteúdo de um ar[quivo, podemos utilizar os seguintes métodos do objeto TextStream:

| | |
|---|---|
| **Read (quantidade)** | Lê um determinado número de caracteres do arquivo |
| **ReadLine** | Lê uma linha inteira do arquivo |
| **ReadAll** | Lê o arquivo inteiro de uma só vez |

Mas se nos interessar gravar informações no arquivo, os métodos disponíveis são:

| | |
|---|---|
| **Write** | Grava uma string no arquivo |
| **WriteLine** | Grava um string no arquivo, incluindo a quebra de linha |
| **WriteBlankLines** | Grava um determinado número de linhas em branco num arquivo |

Esse componente ActiveX ainda possui as seguintes propriedades:

| | |
|---|---|
| **AtEnOfLine** | Indica o fim de uma determinada linha do arquivo |
| **AtEnOfStream** | Indica o Final do Arquivo |
| **Column** | Indica em que coluna do arquivo estamos |
| **Line** | Indica o número da linha atual do arquivo |

O código a seguir mostra como abrimos um arquivo localizado no servidor WEB e exibimos seu conteúdo:

Exemplo 23.1 : Arquivo1.asp

```
<% @Language=vbScript %>
<HTML><HEAD><TITLE>Curso ASP</TITLE></HEAD>
<BODY><CENTER>
<% dim final
  final = "</CENTER></BODY></HTML>"
  On Error Resume Next
  Set Obj = Server.CreateObject("Scripting.FileSystemObject")
  Set arquivo = Obj.OpenTextFile("D:\pessoas.txt",1)
  if Err then
     Response.write "Ocorreu um erro tentando abrir o arquivo!"
     Response.write final
     Response.End
  end if
  Response.Write "Lista de E-mails<BR>"
  Response.write "<HR>"
  do while arquivo.AtEndOfStream=false
    a = arquivo.Readline
    response.write a & "<BR>"
    a = arquivo.Readline
    response.write a & "<HR>"
  loop
  Response.Write "Nova Entrada"
  Response.Write "<FORM ACTION=Arquivo2.asp>"
  Response.Write "NOME : <INPUT TYPE=TEXT NAME=NOME><BR>"
  Response.Write "EMAIL : <INPUT TYPE=TEXT NAME=EMAIL><BR>"
  Response.Write "<INPUT TYPE=SUBMIT VALUE=ENVIAR></FORM>"
  REsponse.Write final
  arquivo.close
%>
```

A próxima página ASP mostra como escrever num arquivo localizado no servidor WEB. Lembrando que essa página deve ser acessada por um formulário WEB com um objeto de NAME=Nome e outro NAME=Email:

Exemplo 23.2 Arquivo2.asp

```
<% @Language=vbScript %>
<HTML><HEAD><TITLE>Curso ASP</TITLE></HEAD>
<BODY><CENTER>
<% dim final
  final = "</CENTER></BODY></HTML>"
  On Error Resume Next
  Set Obj = Server.CreateObject("Scripting.FileSystemObject")
  Application.Lock
  Set arquivo = Obj.OpenTextFile("D:\pessoas.txt",8)
  if Err then
     Response.write "Ocorreu um erro tentando abrir arquivo!"
     Response.write final
```

```
      Response.End
  end if
   Arquivo.WriteLine(Request("Nome"))
   Arquivo.WriteLine(Request("Email"))
   arquivo.close
   if Err then
      Response.write "Ocorreu um erro tentando gravar no arquivo!"
      Response.write final
      Response.End
   else
     Response.Write "Dados inseridos com sucesso!" & "<BR>"
     Response.Write "Nome:" & Request("Nome") & "<BR>"
     Response.Write "Email :" & Request("Email")
   end if
   Response.Write final
%>
```

## 7.7 ACESSANDO BANCO DE DADOS

É possível criar páginas ASP que tenham a habilidade de recuperar ou alterar informações em um banco de dados. Para tal, devemos utilizar o componente Microsoft Data Access que já vem instalado com o IIS.

Para implementar esse acesso a um banco de dados relacional o Data Access utiliza componentes inseridos no pacote ADO(Activex Data Object), uma tecnologia baseada no ODBC(Open DataBase Connectivity).

### 7.7.1 ODBC

O ODBC é o meio mais conhecido para acessar um banco de dados em ambiente Windows. Ele facilita a vida do desenvolvedor mascarando as particularidades de implementação dos Banco de Dados com que os mesmos irão trabalhar.

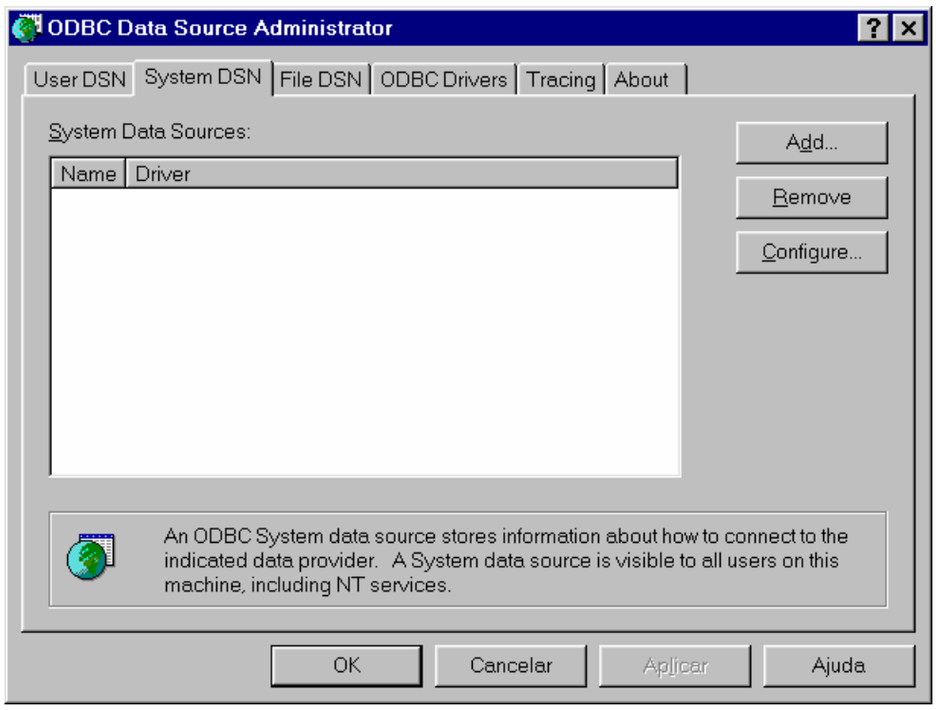
A idéia é que cada fabricante de banco de dados crie um drive ODBC. Sendo assim, quando um programa tenta acessar determinado banco de dados via ODBC o windows procura um drive apropriado para este, verifica como deve proceder e então efetua as operações requisitadas.

Antes de colocar a mão na massa e desenvolver páginas ASP com acesso a banco de dados devemos:

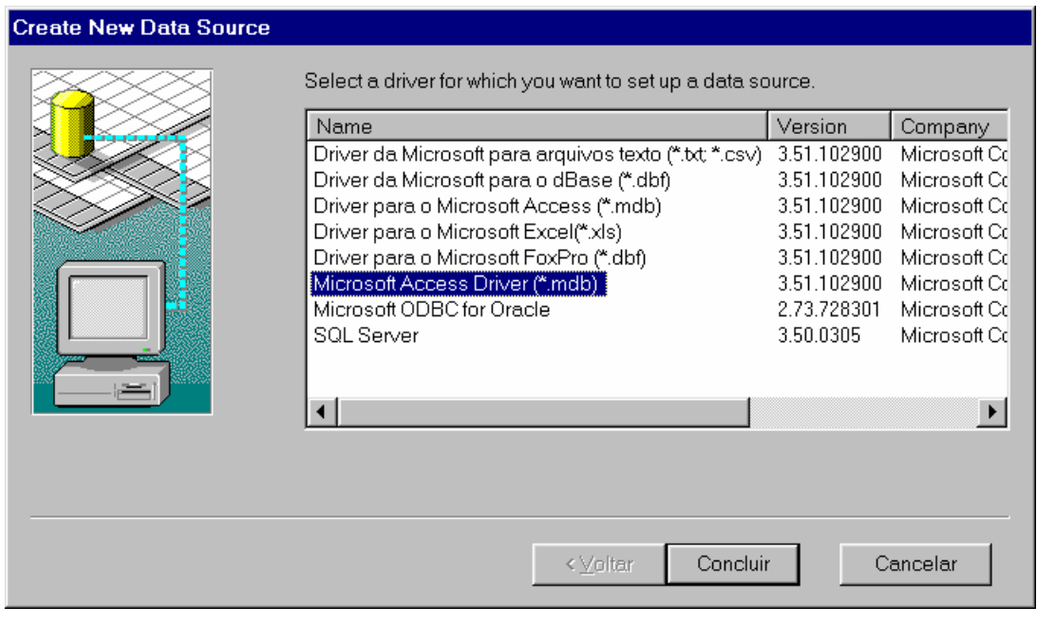
- ✓ Criar um Banco compatível com ODBC
- ✓ Registrar o Banco como uma origem de dados ODBC

Esse último passo permite que aplicações que acessem Banco de Dados via ODBC precisem apenas saber o nome dessa origem de dados, dispensando-se as preocupações com a localização exata e o tipo de banco de dados acessado.

Para criar uma origem de dados devemos executar o programa ODBC do Painel de Controle do Windows.



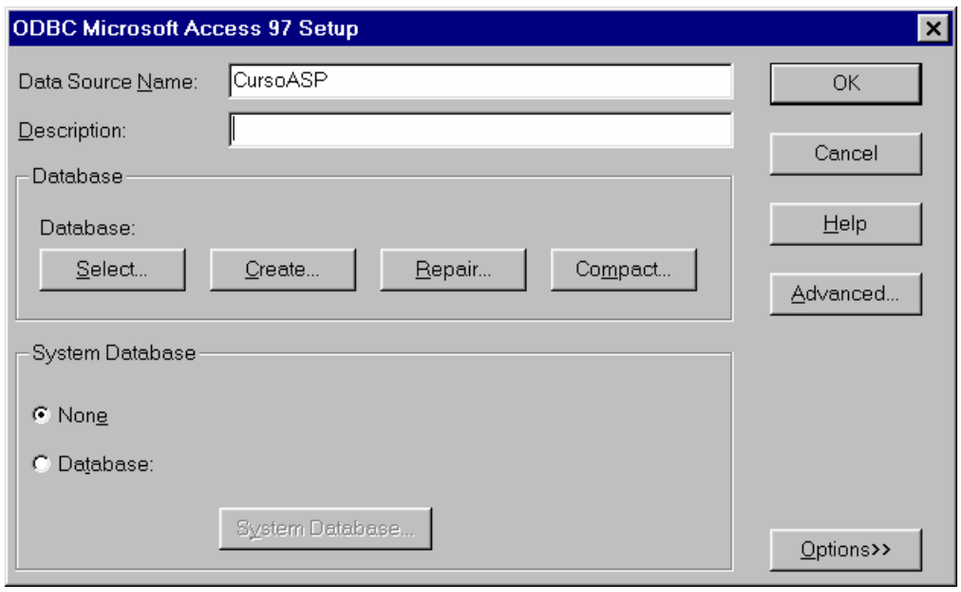

Geralmente para uma aplicação Web é criada uma Origem de dados na opção System DSN. Clicando no botão adicionar temos:




Aqui devemos escolher o driver ODBC a ser utilizado para acessar o Banco de Dados. Para o nosso exemplo usaremos o Microsoft Access. O nome do nosso Banco será Curso.mdb e terá uma tabela chamada Pessoal com os campos:

| | |
|---|---|
| **Codigo** | AutoIncremental (Chave Primária) |
| **Nome** | Texto(50) (Não Nulo) |
| **Email** | Texto(50) |
| **DataN** | Data/Hora |

O nome da nossa Fonte de Dados será CursoAsp.




Nessa janela especificaremos o nome da nossa fonte e o seu PATH(Select).

Feito isso, nossa fonte de dados foi criada. Quando nossas aplicações necessitarem abrir tal banco de dados, basta informar agora o nome dessa fonte(CursoASP).

## 7.7.2 ADO

Para utilizar banco de dados em nossas páginas ASP devemos instanciar os objetos do ADO:

- ✓ **Connection** : representa uma conexão ativa com um banco de dados ODBC. Esse é o componente mais importante do ADO, sendo que os demais só poderão ser criados a partir dele.
- ✓ **Command** : representa um comando a ser executado pela fonte de dados ODBC. Com ele podemos criar comandos SQL compilados no servidor(preparados).
- ✓ **Recordset**: representa um conjunto de registros resultantes do processamen to de um comando SQL em um objeto Connection. É esse objeto que permite a navegação numa tabela ou consulta do banco de dados.
- ✓ **Fileds** : Representa os campos de um RecordSet.

O primeiro objeto que deve ser instanciado é o Connection:

**Set Conexao = Server.CreateObject(“ADODB.Connection”)**

Após sua definição, precisamos ativar a conexão com a fonte de dados ODBC. Para isso, usamos o método Open do Connection. Sintaxe:

```
ObjConexao.Open FonteODBC , Usuário, senha

Onde :
FonteODBC : é o nome da fonte OBDC ou uma string de conexão com o Banco
Usuário : é o nome do usuário da Banco de Dados(opcional)
Senha : é a senha desse usuário(opcional)
```

Exemplo:

**Conexao.Open "CursoASP"**

O objeto Connection possui vários outros métodos, sendo que os principais são:

| | |
|---|---|
| **Close** | Fecha a conexão e libera recursos no Servidor de Banco de Dados. |
| **Execute (ComandoSQL)** | Executa um comando SQL na Conexão aberta. Quando o comando SQL for de seleção, o método retorna um objeto do tipo RecordSet com o resultado da consulta. Quando o comando SQL for de execução esse método não retorna nada. |
| **BeginTrans** | Inicia uma transação no Banco de Dados |
| **CommitTrans** | Finaliza com sucesso uma transação iniciada |
| **RollBackTrans** | Desfaz todos os comandos de uma transação aberta. |

Para instanciarmos um RecordSet procedemos da seguinte forma:

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

Em seguida devemos indicar a Fonte de Dados ODBC para o Recordset:

**Tabela.ActiveConnection = "CursoASP"**

Por fim, abrimos o recordset utilizando o método Open e informando o comando SQL:

**Tabela.Open "Select * From Pessoa"**

Mas a forma mais fácil de se instanciar um Recorset é pela utilização do método Execute do Connection:

**Set Tabela = Conexao.Execute("Select * from pessoa")**

Os principais métodos e propriedades do objeto RecordSet são:

| | |
|---|---|
| **BOF** | True indica início do RecordSet |
| **EOF** | True indica o final do RecordSet |
| **Close** | Fecha o recordset aberto |
| **MoveFirst** | Move o cursor para o primeiro registro de um RecordSet |
| **MoveLast** | Move o cursor para o último registro de um RecordSet |
| **MoveNext** | Move o cursor para o próximo registro de um RecordSet |
| **MovePrevious** | Move o cursor para registro anterior de um RecordSet |
| **Update** | Salva as alterações feitas no RecordSet no Banco de Dados. Também serve para alterar o registro corrente do RecordSet<br>Exemplo: |

| | |
|---|---|
| | Tabela.Update “email”,”lsantos@ufpi.br ” |
| **AddNew** | Adiciona um registro ao RecordSet<br>Ex:<br>Tabela.AddNew<br>Tabela(“Nome”)=”Kelly”<br>Tabela(“DataN”)=#26/05/1980#<br>Tabela.Update |
| **Delete** | Elimina um registro de um Recordset consequentemente, de uma tabela do Banco de Dados. |

Para exibir os valores dos campos de todos os registros de um Recorset usamos a coleção de objetos Fields:

```
Do While not Tabela.EOF
     Response.Write Tabela.Fields(0).Value & “<BR>”
    Response.Write Tabela.Fileds(“Nome”) & “<BR>”
    Response.Write Tabela.Fields(“email”) & “<BR>”
    Response.Write Tabela.Fields(“DataN”) & “<HR>”
     Tabela.MoveNext
Loop
```

Exemplo 24.1 banco1.asp

```
<% @LANGUAGE=VBSCRIPT %>
<% OPTION EXPLICIT %>
<%  DIM CONEXAO,TABELA
  SET CONEXAO = SERVER.CREATEOBJECT(“ADODB.CONNECTION”)
 CONEXAO.OPEN “CursoASP”
 SET TABELA = CONEXAO.EXECUTE (“SELECT * FROM PESSOA”)    %>
<HTML><HEAD><TITLE>Curso ASP</TITLE></HEAD><BODY><CENTER>
<H1> RELAÇÃO DE PESSOAS </H1>
<%
        Do While not Tabela.EOF
             Response.Write Tabela.Fields(0).Value & “<BR>”
            Response.Write Tabela.Fileds(“Nome ) & “<BR>”
            Response.Write Tabela.Fields(“email”) & “<BR>”
            Response.Write Tabela.Fields(“DataN”) & “<HR>”
             Tabela.MoveNext
        Loop
%>
</CENTER></BODY></HTML>
```

Para efetuar operações de Inclusão, Deleção ou Modificação num Banco de Dados utilizamos comandos SQL do tipo INSERT, DELETE e UPDATE, respectivamente, no método Execute do Connection:

Exemplo 24.2 banco2.asp?Codigo=1 APAGAR

```
<% @LANGUAGE=VBSCRIPT %>
```

```
<% OPTION EXPLICIT %>
<%  DIM CONEXAO,SQL
   SET CONEXAO = SERVER.CREATEOBJECT(“ADODB.CONNECTION”)
  CONEXAO.OPEN “CursoASP”
  SQL = “DELETE FROM PESSOA WHERE CODIGO=” & REQUEST(“CODIGO”)
  CONEXAO.EXECUTE (SQL) %>
<HTML><HEAD><TITLE>Curso ASP</TITLE></HEAD><BODY><CENTER>
  Pessoa de Código igual a <%=REQUEST(“CODIGO”)%> foi excluida do Banco!
 </CENTER></BODY></HTML>
```

Exemplo 24.3 banco3.asp?codigo=1&email=lineulima@yahoo.com ALTERAR

```
<% @LANGUAGE=VBSCRIPT %>
<% OPTION EXPLICIT %>
<%  DIM CONEXAO,SQL
   SET CONEXAO = SERVER.CREATEOBJECT("ADODB.CONNECTION")
  CONEXAO.OPEN "CursoASP"
  SQL = "UPDATE PESSOA SET "
  SQL = SQL & "EMAIL = '" & REQUEST("EMAIL") & "' "
  SQL = SQL & "WHERE CODIGO=" & REQUEST("CODIGO")
  CONEXAO.EXECUTE (SQL) %>
<HTML><HEAD><TITLE>Curso ASP</TITLE></HEAD><BODY><CENTER>
  Novo email da pessoa <%=REQUEST("CODIGO")%> é <%=REQUEST("email")%>
 </CENTER></BODY></HTML>
```

Exemplo 24.4 banco4.asp?Nome=Lineu&Email=lsantos@ufpi.br&Datan=02/10/1978 INSERE

```
<% @LANGUAGE=VBSCRIPT %>
<% OPTION EXPLICIT %>
<%  DIM CONEXAO,SQL
   SET CONEXAO = SERVER.CREATEOBJECT("ADODB.CONNECTION")
  CONEXAO.OPEN "CursoASP"
  SQL = "INSERT INTO PESSOA(NOME,EMAIL,DATAN)VALUES("
  SQL = SQL & "'" & REQUEST("NOME") & "',"
  SQL = SQL & "'" & REQUEST("EMAIL") & "',"
  SQL = SQL & "#" & REQUEST("DATAN") & "#)"
  CONEXAO.EXECUTE (SQL) %>
<HTML><HEAD><TITLE>Curso ASP</TITLE></HEAD><BODY><CENTER>
  <%=REQUEST("NOME")%> foi inserido(a) no Banco!
 </CENTER></BODY></HTML>
```

A seguir, código da página FORMBANCO.asp que contém formulários para acessar as páginas anteriores.

FORMBANCO.asp

```
<% @ LANGUAGE= VBSCRIPT %>
<HTML><HEAD><TITLE>Curso de ASP</TITLE></HEAD>
```

```
<BODY><CENTER>
<H1>INSERIR USUÁRIO</H1>
<FORM ACTION="Banco4.asp" METHOD=GET>
  Nome do usuário <BR>
  <INPUT TYPE=TEXT NAME="Nome" MAXLENGTH=50><BR>
  Email do usuário <BR>
  <INPUT TYPE=TEXT NAME="email" MAXLENGTH=50><BR>
  Data de Nascimento <BR>
  <INPUT TYPE=TEXT NAME="datan" MAXLENGTH=10
VALUE="<%=DATE%>"><BR>
  <INPUT TYPE=SUBMIT VALUE="INSERIR"><BR>
</FORM>
<HR>
<H1>APAGAR USUÁRIO</H1>
<FORM ACTION="Banco2.asp" METHOD=GET>
  Código do usuário <BR>
  <INPUT TYPE=TEXT NAME="Codigo" MAXLENGTH=5><BR>
  <INPUT TYPE=SUBMIT VALUE="APAGAR"><BR>
</FORM>
<HR>
<H1>ALTERAR EMAIL</H1>
<FORM ACTION="Banco3.asp" METHOD=GET>
  Código do usuário <BR>
  <INPUT TYPE=TEXT NAME="Codigo" MAXLENGTH=5><BR>
  E-mail do usuário <BR>
  <INPUT TYPE=TEXT NAME="email" MAXLENGTH=50><BR>
  <INPUT TYPE=SUBMIT VALUE="MODIFICAR"><BR>
</FORM>
<HR>
</CENTER>
</BODY>
</HTML>
```

lineulima@yahoo.com