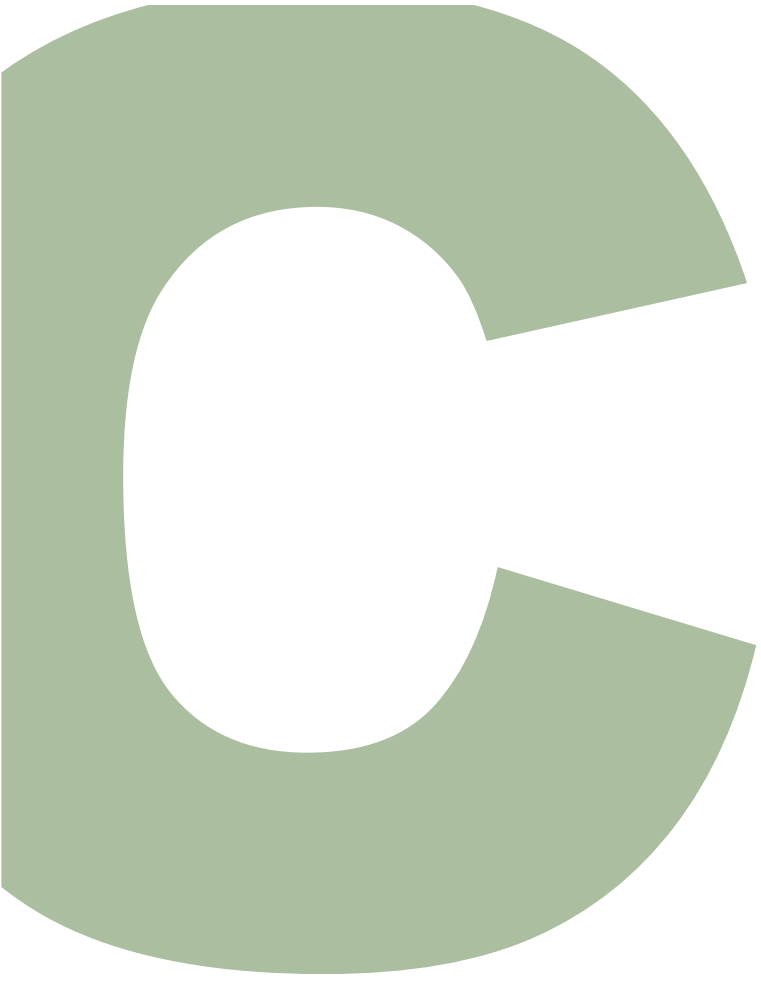


Introdução a Liguagem PHP

Unidade C
Linguagem de Programação Web

UNIDADE C

INTRODUÇÃO A LINGUAGEM PHP

Linguagem PHP

Esta unidade tem por objetivo estudar uma linguagem do lado servidor, que tem seu código interpretado no servidor, sendo que o cliente recebe apenas o resultado do seu processamento. Linguagens do lado servidor são necessárias, especialmente, quando temos a necessidade de acessar um banco de dados (que fica no servidor). A maioria dos sites hoje utiliza esse tipo de linguagem, o que possibilita que as páginas do site sejam dinâmicas, ou seja, são geradas com conteúdo recuperado do banco de dados, o que facilita a manutenção destes sites.

Como já comentado na Unidade A, existem várias linguagens do lado servidor, tais como PHP, JSP, ASP e Python. No entanto, nossa disciplina estudará a linguagem PHP (*Hypertext Preprocessor*) que é atualmente uma das linguagens mais utilizadas, open-source e independente de plataforma.

Agora vamos conhecer mais sobre a linguagem PHP. A seguir abordaremos as ferramentas necessárias para iniciar a programação, a sintaxe básica da linguagem, estruturas condicionais e de repetição, manipulação de arrays, strings e datas, definição de funções e tratar sessões e cookies. Então, mãos à obra...

Características da linguagem

PHP é um acrônimo para "*Hypertext Preprocessor*" (originalmente foi chamado de Personal Home Page, mas quando começou a ganhar adeptos o seu nome foi alterado). O PHP é uma linguagem interpretada, livre, independente de plataforma e utilizada para gerar conteúdo dinâmico. Outras características importantes: pode programar estruturado e/ou orientado a objetos e é de tipagem dinâmica.

A linguagem foi criada por volta de 1994 por Rasmus Lerdorf, com o nome Personal Home Page Tools. Mais tarde, Zeev Suraski desenvolveu o analisador do PHP 3 que contava com o primeiro recurso de orientação a objetos. Pouco depois, Zeev e Andi Gutmans, escreveram o PHP 4, abandonando por completo o PHP 3, dando mais poder à linguagem e maior número de recursos de orientação a objetos. Atualmente a linguagem está na versão 5.

Com o PHP é possível fazer muitas coisas, por exemplo, coletar dados de um formulário, gerar páginas dinamicamente ou enviar e receber cookies. O PHP também tem como uma das características mais importantes o suporte a um grande número de bancos de dados, como dBase, Interbase, mSQL, mySQL, Oracle, Sybase, PostgreSQL e vários outros.

Ferramentas necessárias

Como já mencionado na Unidade A, a programação PHP (linguagem do lado servidor) é interpretada no servidor, isso significa que o PHP fica instalado no servidor e quando uma página PHP é requisitada, o PHP faz o trabalho de interpretar o código e enviar apenas o resultado para o cliente. Então, precisamos saber como vamos trabalhar enquanto estamos na fase de desenvolvimento, pois não vamos ficar

enviando nossas páginas para o servidor para testar. Por isso, vamos trabalhar com um servidor web instalado na nossa máquina e com o PHP configurado neste servidor. A indicação é usarmos a instalação do Xampp. O Xampp é um ambiente destinado a desenvolvedores e faz a instalação das ferramentas necessárias, bem como sua configuração. O ambiente é composto por:

- servidor web Apache;
- interpretadores para linguagens de script: PHP e Perl;
- base de dados MySQL.

O nome desse ambiente significa:

X = para qualquer dos diferentes sistemas operacionais;
A = Apache
M = MySQL
P = PHP
P = Perl

Atualmente XAMPP está disponível para Microsoft Windows, GNU/Linux, Solaris, e MacOS X. A sugestão é utilizar a versão 1.6.8 que é mais estável.

Dica:

O download pode ser feito pelo link:

<http://sourceforge.net/projects/xampp/files/XAMPP%20Windows/>

Parada Obrigatória:

Assista ao vídeo "Instalação e configuração de ferramentas", disponível em nosso Ambiente Virtual de Aprendizagem.

Estrutura do Xampp

Dentro da pasta de instalação do Xampp (provavelmente c:\xampp) está a pasta *htdocs* (documentos de hypertexto). Essa pasta é muito importante, pois todos os arquivos que criaremos ficarão dentro dela. Para iniciar com nossos exemplos, crie uma pasta chamada *curso* dentro de *htdocs* (c:\xampp\htdocs\curso). Em *htdocs* podemos ter várias pastas, por exemplo, quando trabalhamos com vários sites, cada site estará em uma pasta.

Para ter acesso ao nosso servidor local, vamos iniciar o Painel de Controle do Xampp (c\xampp\ xampp-control.exe). A figura C.1 mostra o Painel de Controle. Inicialmente o serviço do Apache não está rodando. Para inicializá-lo clique no botão *Start* ao lado de Apache.

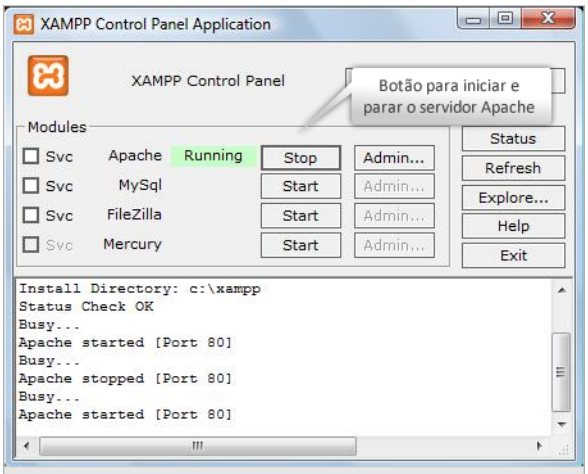


Figura C.1 - Painel de Controle do Xampp
Fonte: do autor

Agora vamos testar se o servidor está rodando. Abra o navegador e digite *http://localhost* na barra de endereço. Se você está visualizando uma página como a da figura C.2, quer dizer que o servidor está rodando. Você também pode acessar o servidor usando o IP local: *http://127.0.0.1*

Figura C.2 - Navegador acessando o servidor local
Fonte: do autor

Estrutura da linguagem

Para começar, a programação de páginas PHP pode ser feita em qualquer editor, por exemplo, o bloco de notas. Mas lembre-se de que agora a extensão dos seus arquivos vão ser *.php*.

Juntamente com uma página (X)HTML vamos escrever código na linguagem PHP. Para isso, precisamos indicar que estamos escrevendo PHP. Isto é feito usando a seguinte sintaxe:

```
<?php

        Comandos

?>
```

Toda codificação PHP fica entre o abre "**<?php**" e fecha "**?>**".

O que precisamos saber antes de escrever nosso primeiro arquivo PHP:

- Para strings pode-se usar tanto aspas simples como duplas.
- O ponto e vírgula (;) deve ser usado como fim de linha de comando.
- Para mostrar algo no navegador pode-se usar **echo** ou **print**. Abaixo são apresentadas formas válidas de usar o *echo* e o *print*:

```
echo(“Bem-vindo ao PHP! “);

echo “Bem-vindo ao PHP! “;

echo ‘Bem-vindo ao PHP! ‘;

print’Bem-vindo ao PHP! ‘;

print(‘Bem-vindo ao PHP!’);
```

- Comentário de linha:

```
//  comentário

# comentário
```

- Comentário para mais de uma linha:

```
/* comentário */
```

Vamos criar nossa primeira página PHP. O código é mostrado na figura C.3. Salve este arquivo com o nome de *curso1.php em c:\xampp\htdocs\curso*.

```
1  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
2  <html xmlns="http://www.w3.org/1999/xhtml">
3  <head>
4  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
5  <title>Primeiro código PHP</title>
6  </head>
7  <body>
8  <?php
9
10     echo "Olá Mundo";
11 ?>
12 </body>
13 </html>
```

Figura C.3 - Primeira página PHP
Fonte: do autor

Para visualizar nossa página no navegador, acesse *http://localhost/curso/curso1.php.* A figura C.4 apresenta o resultado da página *curso1.php* no navegador.

Atenção:

Lembre-se de que o servidor web Apache deve estar rodando. No Painel de Controle do Xampp inicie o Apache.

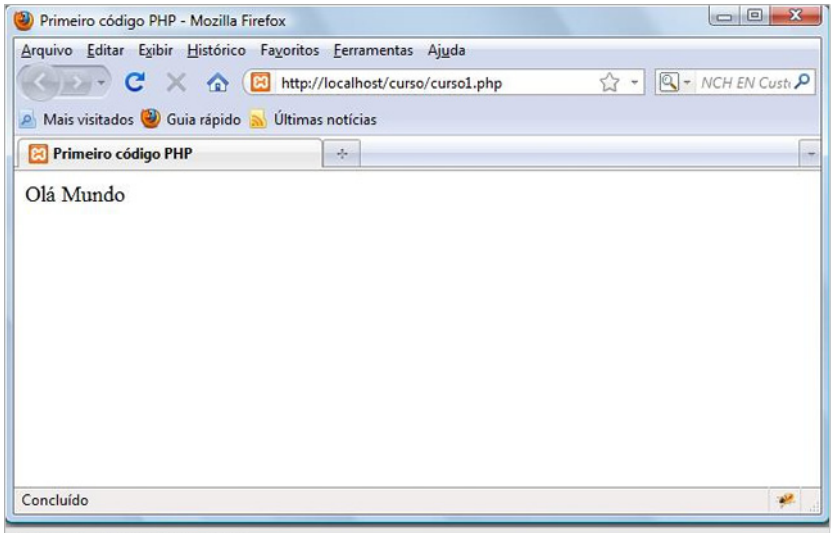


Figura C.4 - Página no navegador
Fonte: do autor

Agora vamos visualizar o código fonte da página. Utilize a tecla de atalho *ctrl + u* para abrir a janela com o código conforme a figura C.5. Observe que o código fonte não mostra nenhuma parte do código PHP. O que aconteceu? O PHP no servidor interpretou a parte de código PHP e retornou apenas o resultado, que neste caso era o texto "Olá Mundo!". É muito importante compreender como funcionam as linguagens do lado servidor. Tudo é processado no servidor, o cliente recebe apenas o resultado.

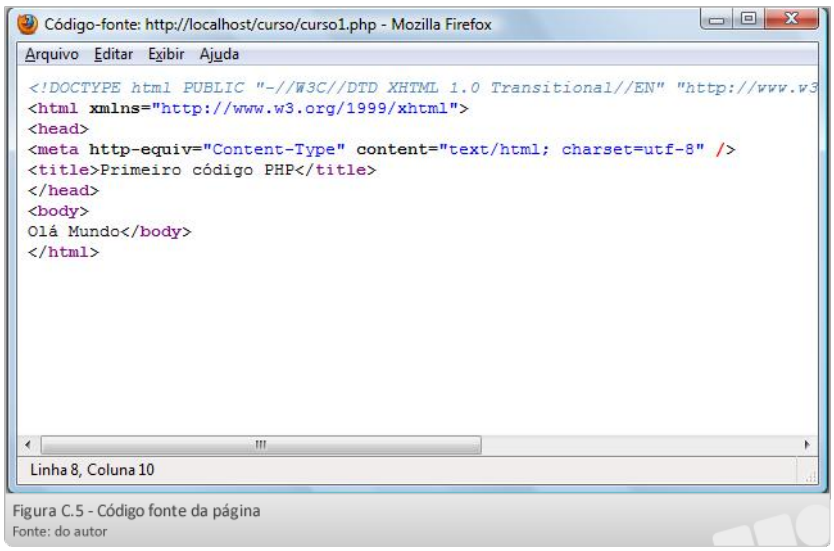


Figura C.5 - Código fonte da página
Fonte: do autor

Variáveis

Assim como na linguagem JavaScript, também no PHP não é necessário fazer uma declaração explícita de variável. Basta atribuir diretamente um valor à variável para ela ser criada e assumir o tipo de acordo com o valor recebido. As variáveis em PHP devem iniciar com o caractere $. Após este caractere, vem o identificador da variável que não pode iniciar com um número (após a primeira letra pode-se usar números). Veja na tabela abaixo alguns exemplos de variáveis válidas e inválidas.

Variáveis válidas	Variáveis inválidas
`$valor`	`$9teste`
`$num1`	`$800`

`$nome_pessoa`	`$1_valor`
`$time_2`	`$6time`

Dica:

O PHP é case sensitive, a variável de nome ***$teste*** é diferente da variável de nome ***$Teste.***

Atenção:

O PHP possui quatro tipos básicos de dados: ***integer***, ***float*** (número de ponto flutuante, ou também *double*), *string* e *boolean*. O tipo é assumido de acordo com o valor atribuído.

Exemplos de atribuição de valores para variáveis:

```
<?php
        $ano    = 2000;
        $nome   = “Maria”;
        $idade = 10;
        $mes    = “Outubro”;
?>
```

Constantes

A definição de constantes em PHP usa a seguinte sintaxe:

```
define(“nome_constante”, “valor_constante”);
```

Exemplo de definição de constantes

```
define(“PI”, 3.14159265);

define(“peso”,80);
```

Exemplo de uso de constante

```
echo “O valor de PI é “. PI;
```

Dica:

nome que para constante não usamos o $. O ponto (.) no exemplo acima foi usado para concatenar a string com o valor da constante.

Atenção:

Lembre-se de que constantes não podem ter seu valor alterado ao longo do programa.

Operadores Lógicos

Operadores lógicos são normalmente utilizados em comandos condicionais, como *if, for* e *while*

Operador lógico	Finalidade
==	Igual
!	Negação (se *true* passa para *false*, se *false* passa para *true*)
!= ou <>	Diferente
>	Maior
<	Menor
>=	Maior ou igual
<=	Menor ou igual
&& ou and	E
|| ou or	Ou

Operadores Matemáticos

Operador Matemático	Finalidade
+	Adição de valores
-	Subtração de valores
*	Multiplicação de valores
/	Divisão de valores

%	Retorna o resto de uma divisão. Exemplo:
	`150 % 13 retornará 7`
	`7 % 3 retornará 1`

Expressões Simples com operadores

Operador	Finalidade
=	Atribuição
+=	Adiciona ao string/valor já existente. Exemplo: `$x += $y` é o mesmo que `$x =$x +$ y`, da mesma forma podem ser utilizados: `-= , *= , /=` ou `%=`

++	Acrescenta 1 no valor
	Exemplo:
	`$X = 3;`
	`$X++; // $X valerá 4`

--	Decrementa 1 no valor
	Exemplo:
	`$X = 3;`
	`$X--; // $X valerá 2`

.	Concatenação
	Exemplo:
	`$X = "linguagem ";`
	`$X .= "PHP ";`
	`echo $X; // linguagem PHP`
	`$X = "linguagem ";`
	`$Y = "PHP";`
	`$Z = $X.$Y;`
	`echo $Z; // linguagem PHP`

Estruturas de Controle

A seguir, veremos as principais instruções condicionais e de repetição. Observaremos que seguem uma sintaxe muito parecida de outras linguagens.

if

A instrução **if** é utilizada quando é necessário tomar decisões. Sua estrutura é:

```
if (<expressão>)
        <Instrução>
```

A expressão é avaliada, caso seja *true* (verdadeira) a instrução é executada; caso contrário, a instrução não é executada.

Se o bloco de instrução possuir mais de uma instrução, deve-se usar início de bloco e fim de bloco, ou seja {}.

```
if (<expressão>){
        <Instrução>
}
```

Caso exista instrução a ser executada quando a expressão retornar false (falso), pode ser usar o else.

```
if (<expressão>){

        <instrução>

}else{

        <instrução>
}
```

switch

O *switch* também pode ser utilizado para testes condicionais e é indicado para situações que exijam vários testes.

Sintaxe:

```
switch (variavel_de_controle) {
  case opção1 :
         comandos ;
         break;
  case opção2 :
         comandos ;
         break;
  default :
         comandos;
}
```

É importante observar que a instrução *break* é utilizada ao final de cada opção. O *break* faz com que a execução pule para o fim do *switch*. Isso é importante, uma vez que quando uma opção verdadeira for encontrada, o switch segue executando todas as instruções até o fim, se não possuir um *break*.

for

Para situações em que precisamos realizar um bloco de instrução diversas vezes, uma opção de solução é o comando de repetição ***for***.

Sintaxe:

```
for (<inicialização>;<condição>;<atualização variável>){

  <instruções>
}
```

while

O *while* é semelhante ao comando *for*, porém, às vezes, o *while* é aplicado quando não sabemos determinar a quantidade de vezes que nosso laço vai executar as instruções. Nesse laço, enquanto a condição do *while* for verdadeira as instruções serão executadas.

Sintaxe:

```
while(<condição>){

  <instruções>
{
```

Dica:

Cuidado para que seu laço não seja infinito. A condição precisa ser falsa em algum momento para sair do laço ou pode usar o comando *exit* para sair do laço.

do .. while

Outra opção é o *do .. while* que é semelhante ao *while*, porém, o teste condicional é feito no fim do laço. Nesse tipo de laço, os comandos são executados pelo menos uma vez.

Sintaxe:

```
do {

  comandos;

}while (<condição>)
```

Processamento de Formulários

Para tornar nossas páginas mais interativas, precisamos manipular dados vindos de formulários. Considere o formulário da figura C.6. Como podemos via PHP recuperar o e-mail informado neste formulário? É o que vamos ver agora.

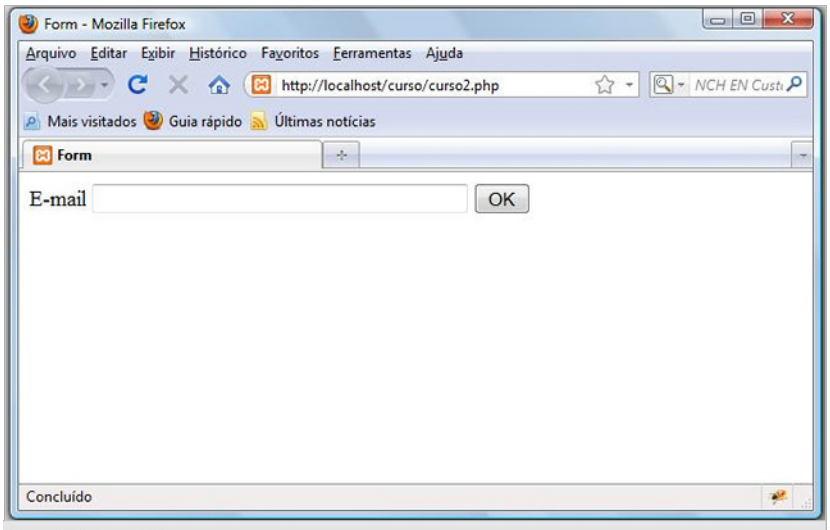


Figura C.6 - Formulário
Fonte: do autor

A figura C.7 apresenta o código (X)HTML correspondente ao formulário acima. Algumas dessas informações importantes estão destacada em vermelho:

- método do formulário: *POST*
- ação do formulário: *tratar_dados.php*
- nome da caixa de texto do e-mail: *email*
- nome do botão: *botao*

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Form</title>
</head>
<body>
<form id="form1" name="form1" method="post" action="tratar_dados.php">
  <label for="email">E-mail</label>
  <input name="email" type="text" id="email" size="40" />
  <input type="submit" name="botao" id="botao" value="OK" />
</form>
</body>
</html>
```

Figura C.7 - Código (X)HTML do formulário
Fonte: do autor

É importante observar que ao clicar no botão "OK" o navegador fará a requisição do arquivo *tratar_dados.php* passando os dados do formulário usando o método POST. A questão agora é como pegar os dados no *tratar_dados.php?*

Para conseguir pegar os valores precisamos conhecer os arrays superglobais do PHP: **$_POST, $_GET e $_REQUEST.**

$_POST: usado quando os dados são enviados pelo método POST.

$_GET: usado quando os dados são enviados pelo método GET.

$_REQUEST: pode ser usado tanto para dados vindos pelo método GET quanto pelo método POST.

No nosso exemplo, os dados estão sendo enviados usando o método POST e o valor que queremos pegar é da caixa de texto cujo *name* é *email*. Então vamos pegar o valor no *tratar_dados.php*:

```
<?php
  $email_recebido = $_POST['email'];
  echo  $email_recebido;
?>
```

O código PHP acima mostra o e-mail informado pelo usuário no formulário. Observe que `'email'` foi usado um índice para o array $_POST. Veja a figura C.8 que mostra os dois arquivos.

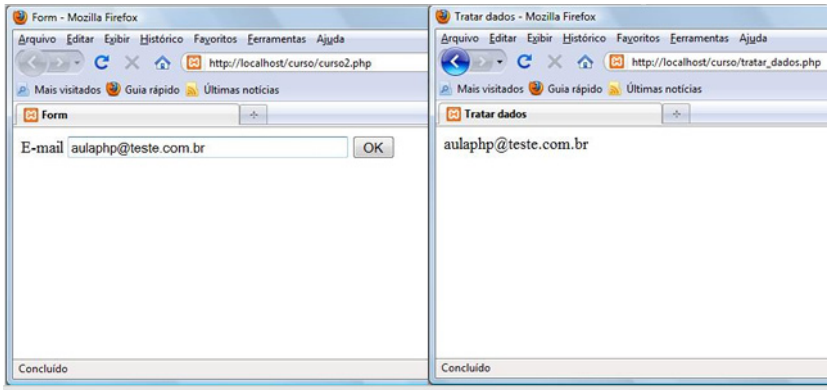


Figura C.8 - Formulário e tratar_dados.php
Fonte: do autor

Outra forma interessante seria verificar se o *tratar_dados.php* foi requisitado pelo formulário. Veja abaixo:

```
<?php
  if ($_POST['botao'] == "OK") {
        $email_recebido = $_POST['email'];
```

```
        echo  $email_recebido;
    }
?>
```

Neste exemplo estamos testando se veio um valor cujo índice é botao, que é o name do botão no formulário. Se ele possuir valor igual a "OK" então significa que veio do form e mostra o valor de email.

Algumas dicas

- Em PHP podemos usar uma variável dentro de aspas duplas. No exemplo abaixo será mostrado o valor correspondente da variável (40).
  Ex.:

```
<?php

        $idade = 40;
        echo  "A idade de João é $idade";

?>
```

- O mesmo não é válido para aspas simples. No exemplo abaixo será mostrado literalmente $idade.
  Ex.:

```
<?php

        $idade = 40;
        echo  'A idade de João é $idade';
?>
```

Dica:

Faça estes dois exemplos apresentados acima para testar.

- O caractere @ é usado para como um operador de controle de erro:
  Ex.:

```
<?php

        $a = @$b / $c; //Suprime mensagem de erro
?>
```

- Teste o código PHP abaixo e veja o que acontece.

```
<?php
        echo ("Olá mundo!");
      echo ("nossa primeira página PHP!");
?>
```

No navegador vai mostrar tudo em uma mesma linha. Mas como fazer para mostrar cada mensagem em uma linha. Lembre-se que está visualizando no navegador, então basta usar o elemento HTML
:

```
<?php
```

```
            echo ("Olá mundo!");
        echo ("<br />nossa primeira página PHP!");
    ?>
```

Síntese

Bem, agora já conhecemos a estrutura básica da linguagem e construímos algumas páginas PHP para testes. De agora em diante, podemos avançar e conhecer outros recursos da linguagem que permitirão criar páginas dinâmicas e mais interessantes.

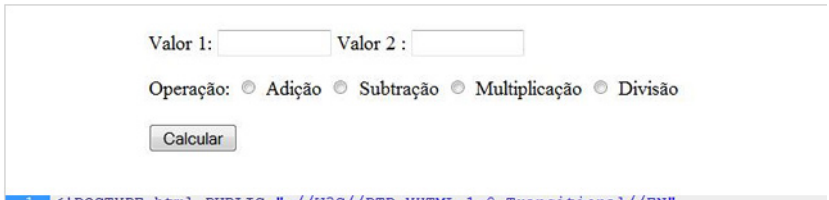
Atividades - Parte 1

1. Faça uma pesquisa sobre as diferentes versões do PHP, apresentando suas principais características e quando (ano) elas foram disponibilizadas.

2. Crie um formulário (X)HTML como o da figura abaixo. O arquivo deve ser nomeado de *form1.html* e na *action* do formulário deve chamar *dados.php*. O *dados.php* deve mostrar todos os dados do *form1.html*.

Figura C.9 - Exercício 2
Fonte: do autor

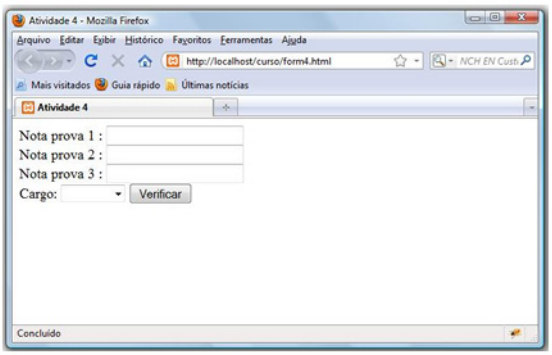
3. Dado o formulário abaixo do arquivo *form3.html*, crie o *calcular.php* que faz o cálculo com os dois valores informados de acordo com a operação selecionada.




```
1  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
2  <html xmlns="http://www.w3.org/1999/xhtml">
3  <head>
4  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
5  <title>Form Atividade 3</title>
6  </head>
7  <body>
8  <form id="form1" name="form1" method="post" action="calcular.php">
9      <label for="v1">Valor 1:</label>
10     <input name="v1" type="text" id="v1" size="10" />
11     <label for="v2">Valor 2:</label>
12     <input name="v2" type="text" id="v2" size="10" />
13     Operação:
14       <input type="radio" name="op" value="A" id="sexo_0" />   Adição
15       <input type="radio" name="op" value="S" id="sexo_1" />   Subtração
16       <input type="radio" name="op" value="M" id="sexo_2" />   Multiplicação
17       <input type="radio" name="op" value="D" id="sexo_3" />   Divisão<br />
18     <input type="submit" name="botao" id="botao" value="Calcular" /> <br />
19 </form>
20 </body>
21 </html>
```

Figura C.10 - Exercício 3
Fonte: do autor

4. Considere o formulário (e código (X)HTML correspondente) abaixo para solucionar o problema a seguir.




```
<form name="form1" method="post" action="calcular4.php">
Nota prova 1 : <input name="p1" type="text" id="p1">  <br>
Nota prova 2 : <input name="p2" type="text" id="p2">  <br>
Nota prova 3 : <input name="p3" type="text" id="p3">  <br>
Cargo: <select name="cargo">
<option></option>
<option value="7">Gerente</option>
<option value="8">Diretor</option>
</select>
<input name="verificar" type="button" id="verificar" value="Verificar">
</form>
```

Figura C.11 - Exercício 4
Fonte: do autor

a. No processo de seleção para cargos de uma empresa, o candidato faz 3 provas, cujos pesos são:

- *Prova 1 = peso 3*
- *Prova 2 = peso 3*
- *Prova 3 = peso 4*

b. Para ser considerado aprovado, o candidato deve ter média acima de 7 para o cargo de gerente e acima de 8 para o cargo de diretor.

c. Crie o arquivo *calcular4.php* que faz o cálculo da nota final do candidato e mostra a situação (APROVADO/REPROVADO).

Linguagem PHP – Parte 2

Continuando a unidade C - Introdução à linguagem PHP, nesta parte 2 do material abordaremos a manipulação de arrays, strings e datas. Ao final desta parte, serão apresentadas a atividades para realização.

Arrays

Array, ou vetor, são estruturas que podem armazenar vários valores, sendo a referência a cada valor através de um índice. O que muda no PHP é que este índice pode ser tanto um número quanto uma string. Vejamos alguns exemplos de criação de arrays em PHP:

- Definição de um array sem valores:

```
$vet = array ();
```

- Criando de array inicializado:

```
$notas = array (5,5,3,6,8);

$nomes = array (“José”,”Maria”,”João”,”Márcia”,”Paulo”);
```

Dica:

Nos exemplos acima, para acessar cada elemento de um vetor, devemos nos referir ao seu índice. Em PHP o primeiro índice de um vetor é o 0 (Zero). Sendo que o último índice é sempre igual ao número de elementos de um vetor - 1. Assim, um vetor de 5 elementos tem índices numéricos inteiros que vão de 0 a 4.

- No exemplo abaixo, quando não é especificado o índice, o valor é atribuído para o próximo índice livre, iniciando em zero:

```
$mes[] = "Janeiro";    // mesmo que $mes[0] = "Janeiro";

$mes[] = "Fevereiro"; // mesmo que $mes[1] = "Fevereiro";
```

Array Associativo

Em um vetor associativo, os índices são strings. Assim, não temos índices numéricos, mas conjuntos de caracteres que representam cada elemento do vetor. Veja abaixo duas formas diferentes de criar um array associativo:

```
$dados = array ("nome" => "Camila",
                    "nascimento" => "25/11/1981",
                     "cargo" => "professor",
                     "rg" => "9990909090");
```

Ou

```
$dados["nome"]              = "Camila";
$dados["nascimento"]        =  "25/11/1981";
$dados["cargo"]             = "professor";
$dados["rg"]                = "9990909090";
```

Laço para mostrar valores de array

Vamos utilizar um tipo especial de laço para mostrar os dados de arrays, é o foreach. O foreach fornece uma forma bastante prática de percorrermos um vetor associativo e tem duas variações, vejamos a sua estrutura:

1. No *foreach* do exemplo abaixo o laço percorre todos os elementos e a cada iteração atribui o valor do elemento para a variável $valor:

```
$nomearray = array(1,2,3,4);
foreach($nomearray as $valor){
       echo "$valor <br />";
}
```

ou

2. Já no exemplo abaixo a cada iteração é atribuído o índice do elemento para a variável *$indice* e o valor para a variável *$valor*:

```
$nomearray = array(1,2,3,4);
foreach($nomearray as $indice => $valor){
        echo "$indice  - $valor <br />";
}
```

Algumas funções para arrays

A tabela abaixo apresenta funções para ordenação de arrays.

sort	ordena de forma crescente (do menor para o maior)
	`$frutas = array ("maça","uva","abacaxi","banana");`
	`sort($frutas); //abacaxi, banana, maça, uva`
	`foreach($frutas as $valor)`
	`  echo "$valor, ";`
	`//saída: abacaxi, banana, maça, uva,`

rsort	ordena de forma decrescente (do maior para o menor)
	`$frutas = array ("maça","uva","abacaxi","banana");`
	`rsort($frutas);`
	`foreach($frutas as $valor)`
	`  echo "$valor, ";`
	`//saída: uva, maça, banana, abacaxi`

asort	ordena um array associativo mantendo a association entre índices e valores
	`$vet = array ("nome" => "Ana", "idade" => "23", "cargo" => "gerente");`
	`asort($vet);`
	`foreach($vet as $valor)`
	`  echo "$valor, ";`
	`//saída: 23, Ana, gerente,`

arsort	ordena um array associativo em ordem decrescente mantendo a associação entre índices e valores
	`$vet = array ("nome" => "Ana", "idade" => "23", "cargo" => "gerente");`
	`arsort($vet);`
	`foreach($vet as $valor)`
	`  echo "$valor, ";`
	`//saída: gerente, Ana, 23,`

ksort	ordena um array por seus índices
	`$vet = array ("nome" => "Ana", "idade" => "23", "cargo" => "gerente");`
	`ksort($vet);`
	`foreach($vet as $ind => $valor)`
	`  echo "$ind: $valor, ";`
	`//saída: cargo: gerente, idade: 23, nome: Ana,`

Outras duas funções interessantes para tratamento de arrays são apresentadas na tabela a seguir: *implode* e *explode*.

implode	Retorna uma string contendo todos os elementos do array, separados pela string passada por parâmetro.
	`$partes = array("a", "casa número", 13, "é azul");`
	`$frase = implode(" ",$partes);`
	`/* $frase passa a conter a string:`
	`"a casa número 13 é azul"`
	`*/`

explode	Retorna um array contendo partes da string fornecida, separadas pelo delimitador fornecido.
	`$data = "11/14/1975";`
	`$vdt = explode("/",$data);`
	`/* O array $vdt vai conter os seguintes valores:`
	`$vdt[0] = 11`
	`$vdt[1] = 14`
	`$vdt[2] = 1975`
	`*/`

Strings

String é uma sequência de letras, dígitos, caracteres de pontuação e outros, que são representados pela linguagem como texto. Strings literais podem ser usadas delimitando por pares de aspas simples ('...') ou aspas duplas ("... "). A seguir, veremos as principais funções do PHP associadas a strings.

Função	Descrição

strlen()	Retorna o número de caracteres de uma string
	`$str = "programação";`
	`strlen($str); // retorna 11`

substr	Retorna uma parte de uma string. Sintaxe:
	`substr(string, posição_inicial, tamanho);`
	`Obs.: lembre-se de que a string começa na posição zero`
	`$str = "lpw@teste.com.br";`
	`substr($str, 3, 6); // Retorna "@teste"`

ucfirst	Converte para maiúsculo o primeiro caractere de uma string.
	`$str = "programação";`
	`ucfirst($str); // Retorna "Programação"`

strtoupper	Converte uma string para maiúsculas.
	`$str = "programação";`
	`strtoupper($str); // Retorna "PROGRAMAÇÃO"`

strtolower	Converte uma string para minúsculas.
	`$str = "Programação";`
	`strtolower($str); // Retorna "programação"`

str_replace()	Substituição de caracteres em uma string. Sintaxe:
	str_replace (string_pesquisar, nova_string, onde_pesquisar)
	`$texto = "10/11/2011";`
	`str_replace("/", "-", $texto);`
	`// retorna "10-11-2011";`

strip_tags()	Retorna uma string, retirando as tags HTML e/ou PHP. `strip_tags('<a href="teste1.php">testando</a> ');` `//Retorna a string "testando"`

htmlspecialchars()	Converte caracteres especiais HTML para string de forma que não sejam interpretados pelo navegador.
	`$texto = "<p>String</p>";`
	`htmlspecialchars($texto);`
	`// Retorna "<p>String<p>", e não "String" em um parágrafo`

urlencode()	Retorna a string, convertida para o formato *urlencode*.
	Esta função é útil para passar valores para uma próxima página através do método GET.
	`$frase = "Título da notícia";`
	`urlencode($frase);`
	`// Retorna "Título+da+notícia";`

nl2br()	Converte a quebra de linha (\n) por quebra de linha em HTML ( ). Obs.: esta função é interessante de ser usada quando há necessidade de mostrar um texto digitado pelo usuário, por exemplo, em um *textarea* `$nome = "Linguagem \nde programação para\n Web";` `nl2br($nome); // Retorna "Linguagem  de programação para  Web"`
strrev()	Retorna a string invertida `echo strrev("Função"); // Retorna "oãçnuF"`
trim()	Retira espaços e linhas em branco do início e do final da string fornecida. `echo trim(" teste \n \n "); // Retorna "teste"`

Datas

Trabalhar com datas e horas também é muitas vezes necessário para incorporar algum recurso nos sites e tratar dados de formulário ou banco de dados. Por exemplo, recuperar a data atual, formatar datas, pegar o dia da semana. Por isso, vamos conhecer as principais funções PHP para manipular datas.

Função	Descrição
mktime()	Retorna o timestamp Unix correspondente para os argumentos dados. Este timestamp é um longo inteiro contendo o número de segundos entre a Era Unix (January 1 1970 00:00:00 GMT) e o tempo especificado. Argumentos podem ser omitidos da direita para esquerda; quaisquer argumentos assim omitidos serão definidos para o valor atual de acordo com a data e a hora local. Sintaxe: `int mktime ([ int $hora [, int $minuto [, int $second [, int $mes [, int $dia [, int $ano [, int $is_dst ]]]]]] ])`
time()	Retorna o timestamp Unix atual. Sintaxe: `time()`
date()	Formata data e hora de acordo com a string format dada Sintaxe: `date("opções formatação", data)`
checkdate()	Valida uma data. Sintaxe: `checkdate ( mes, dia, ano)`

Vejamos na figura C.12 o uso das funções mktime() e time().

Na linha 11: foi criada uma data passando por parâmetro hora, minutos, segundos, mês, dia e ano.

Na linha 15: foi criada a variável $hoje com a data e hora atual.

As linhas 13 e 17: mostram as variáveis criadas. Faça o teste no seu navegador! Você verá que é mostrado um valor para as variáveis e não uma data/hora, como na figura C.13.

```
 9  <?php
10
11      $data_nascimento = mktime(15,20,0,2,20,2000);
12
13      echo "Data de nascimento: $data_nascimento <br/> ";
14
15      $hoje = time();
16
17      echo "Hoje: $hoje";
18  ?>
```

Figura C.12 - Funções *mktime()* e *time()*
Fonte: do autor

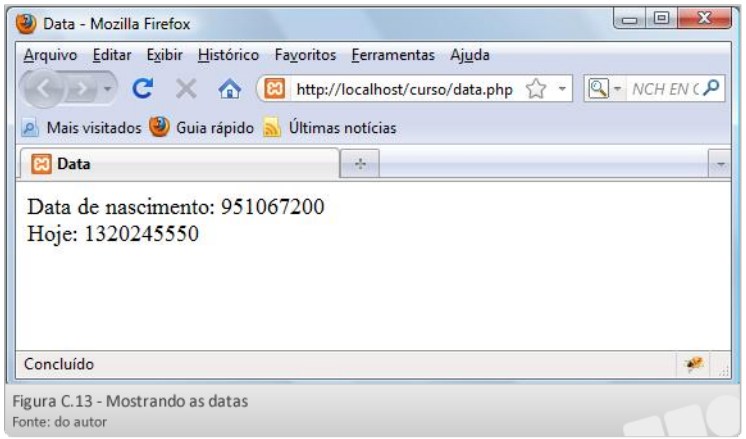


Figura C.13 - Mostrando as datas
Fonte: do autor

Como podemos ver, a data e hora mostradas não estão em um formato conhecido por nós. Precisaremos utilizar o função *date()* para formatar data e hora. A tabela abaixo apresenta as opções de formatação para a função *date()*.

Opção	Descrição
a	Antes/Depois de meio-dia em minúsculo (am ou pm)
A	Antes/Depois de meio-dia em maiúsculo (AM ou PM)
d	Dia do mês, 2 dígitos com preenchimento de zero (01 até 31)
D	Uma representação textual de um dia, três letras (Mon até Sun)
w	Representação numérica do dia da semana 0 (para domingo) até 6 (para sábado)
m	Representação numérica de um mês (01 a 12)
M	Uma representação textual curta de um mês, três letras (Jan a Dec)
t	Número de dias de um dado mês (28 até 31)
L	Se um ano é bissexto (1 se está em ano bissexto, 0 caso contrário)
Y	Uma representação de ano completa com quatro dígitos
y	Uma representação do ano com dois dígitos
g	Formato 12-horas de uma hora sem preenchimento de zero (1 até 12)
G	Formato 24-horas de uma hora sem preenchimento de zero (0 até 23)
h	Formato 12-horas de uma hora com zero preenchendo à esquerda (01 até 12)
H	Formato 24-horas de uma hora com zero preenchendo à esquerda (00 até 23)
i	Minutos com zero preenchendo à esquerda (00 até 59)
s	Segundos, com zero preenchendo à esquerda (00 até 59)

Vamos ver como fica nosso código PHP utilizando agora a função *date()*. Veja na figura C.14.

Nas linhas 12 e 16: foram criadas novas variáveis com as datas formatadas. A figura C.15 mostra o resultado do código no navegador.

```
 9  <?php
10
11      $data_nascimento = mktime(15,20,0,2,20,2000);
12      $fdata_nascimento = date("d/m/Y - H:i:s",$data_nascimento);
13      echo "Data de nascimento: $fdata_nascimento <br/> ";
14
15      $hoje  = time();
16      $fhoje = date("d/m/Y - H:i:s",$hoje);
17      echo "Hoje: $fhoje";
18  ?>
```

Figura C.14 - Data com formatação
Fonte: do autor

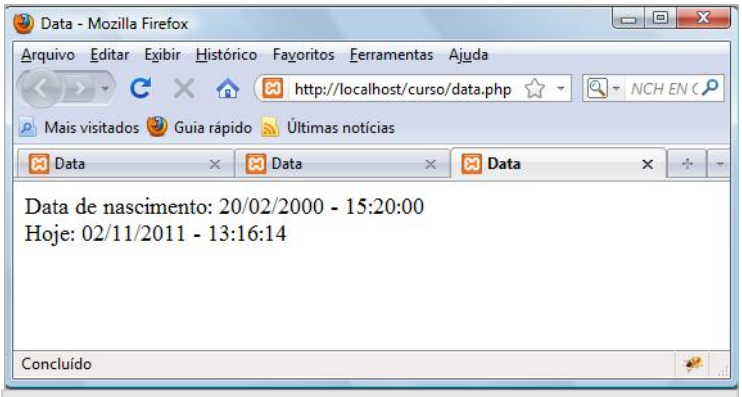


Figura C.15 - Mostrando datas formatadas
Fonte: do autor

Atenção:

A "data e hora" retornadas no exemplo acima são do servidor onde a página está hospedada. No nosso caso, como estamos trabalhando com o servidor local, será a data/hora da máquina.

Saiba Mais:

você pode ver mais opções de formatação acessando:
http://br2.php.net/manual/pt_BR/function.date.php

Outra função interessante para tratamento de datas é a *checkdate*, útil para validar uma data. Imagine que o usuário informa uma data em um formulário, antes de trabalhar com esta data é importante validá-la. Veja o código da figura C.16. No mesmo arquivo, foi colocado o *form* e o código que verifica a data (quando o *form* é submetido chama o próprio arquivo, pois não tem nada em *action*). Algumas observações sobre o código:

Na linha 14: é verificado se o form foi submetido. Neste caso, faz a verificação da data;

Como a função *checkdate* espera três parâmetros, foi necessário quebrar a data em dia, mês e ano (linha 18);

Na linha 24: é usada a função *checkdate*, passando os valores da data que estão no vetor *$vdt*.

```
8  <form id="form1" name="form1" method="post" action="">
9    <label for="data">Data:</label>
10   <input type="text" name="data" id="data" />
11   <input type="submit" name="botao" id="botao" value="OK" />
12 </form>
13   <?php
14       if ($_POST['botao']  == 'OK'){
15
16           $data = $_POST['data'];
17           echo "Data informada: $data <br/>";
18           $vdt  = explode('/',$data);
19           /*
20            $vdt[0] contém o dia
21            $vdt[1] contém o mês
22            $vdt[2] contém o ano
23           */
24           if (checkdate($vdt[1],$vdt[0],$vdt[2])) // verifica se data é válida
25               echo "Data válida";
26           else
27               echo "Data inválida";
28       }
29   ?>
```

Figura C.16 - Código PHP para validação de data
Fonte: do autor

A figura C.17 mostra o resultado do código acima no navegador, sendo uma execução com data válida e outro com data inválida.

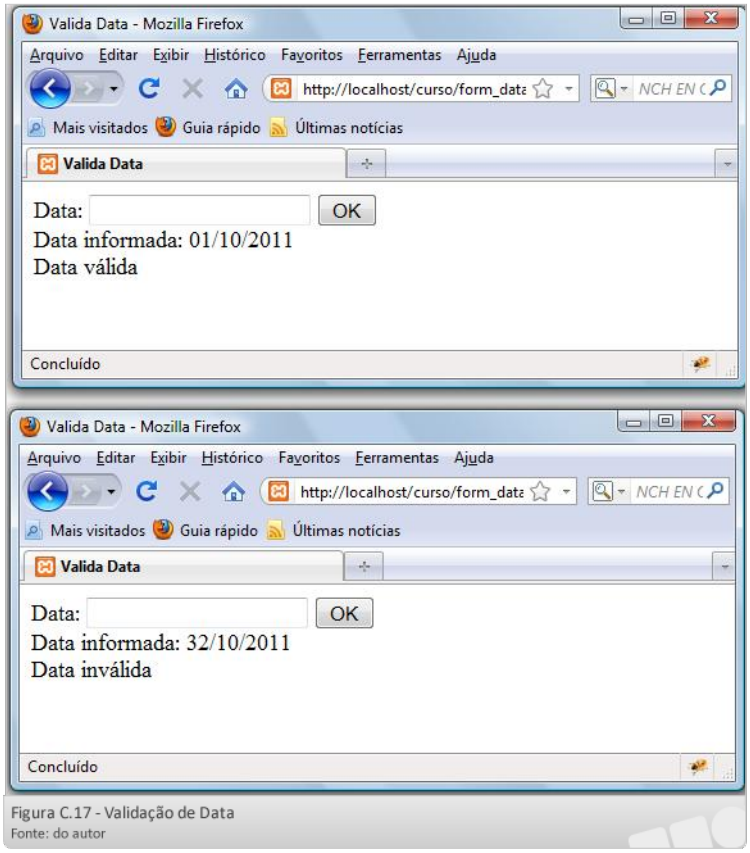


Figura C.17 - Validação de Data
Fonte: do autor

Síntese

Conhecer as funções que foram apresentadas é muito importante para o desenvolvimento de aplicações com PHP, uma vez que é muito comum trabalhar com arrays, strings e datas. Na próxima parte da Unidade C vamos trabalhar com funções definidas pelo programador e voltaremos a utilizar as funções vistas nesta parte.

Atividades - Parte 2

1. Dado o formulário da figura abaixo, crie o arquivo exe1.php que:

a) mostra o texto todo em maiúsculas;
b) mostra o texto todo em minúsculas;
c) mostra o texto com primeira letra em maiúscula;
d) mostra o texto convertendo todos os caracteres de nova linha (\n) para a marcação
;
e) mostra o texto retirando todas as marcações HTML (tags) que tenham sido inseridas;
f) mostra o tamanho do texto (número de caracteres);
g) mostra os 5 primeiros caracteres do texto.

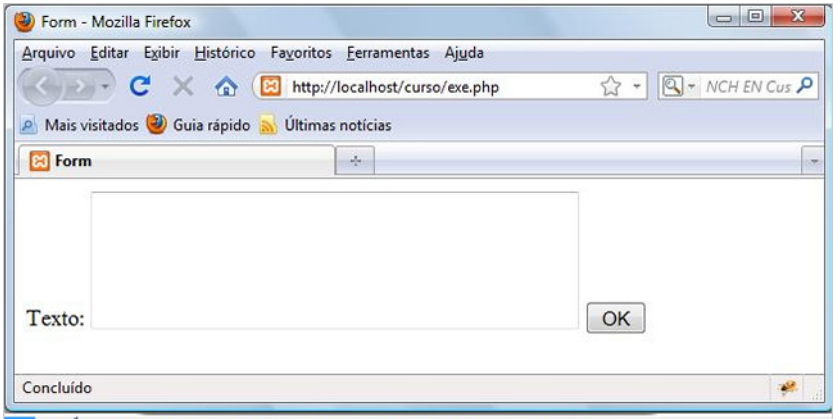


Figura C.18 - Exercício 1
Fonte: do autor

2. Crie um array associativo para armazenar as notas de alunos, onde o índice é o número de matrícula e a nota é o valor correspondente. Faça um código PHP para mostrar as notas de forma ordenada, sendo da maior para a menor nota, juntamente com o número da matrícula do aluno (conforme modelo abaixo).

```
Matrícula      Média

3532           8.9

2314           8.6

2342           8.5

9893           8.4
```

3. Dado o array abaixo, faça um código PHP que mostra o dia da semana para cada uma das datas:

```
$vetor = array("10/04/2011", "14/04/2011 ", "21/04/2011 ", "01/05/2011 ",
"07/09/2011 ", "20/09/2011 ");
```

Por exemplo:

```
10/04/2011 – Domingo

14/04/2011 – Quinta-feira
```

4. Dado o array abaixo, faça um código PHP que conta e mostra quantas datas são do mês de novembro.

```
$feriados = array("07/09/2010", "20/09/2010", "12/10/2010", "02/11/2010",
"15/11/2010", "08/12/2010");
```

Considere a entrada de uma informação de data de nascimento no formato dd/mm/aaaa. Faça um código PHP que mostra o signo da pessoa para aquela data. Faça um formulário para informar a data e testar o código. Para auxiliar na definição são fornecidos dois vetores:

a) o vetor $ultimo_dia contém o último dia do mês válido para o signo
b) o vetor $signos contém os signos

```
$ultimo_dia = array(20,19,20,20,20,20,21,22,22,22,21,21);

$signos = array('Capricórnio ', 'Aquário ', 'Peixes ', 'Áries ', 'Touro ',
```

```
'Gêmeos ', 'Câncer ', 'Leão ', 'Virgem ', 'Libra ', 'Escorpião ', Sagitário
');
```

Mês	Último dia	Signo
01	20	Capricórnio
02	19	Aquário
03	20	Peixes
04	20	Áries
05	20	Touro
06	20	Gêmeos
07	21	Câncer
08	22	Leão
09	22	Virgem
10	22	Libra
11	21	Escorpião
12	21	Sagitário

Exemplos:

```
Para a data 02/08/1975 -> observar o mês 8 -> o dia 02 é menor ou igual ao
último dia, então é leão

Para a data 30/11/1991 -> observar o mês 11 -> o dia 30 é maior que o último
dia, então pega o mês seguinte -> neste caso o signo é Sagitário
```

5. Análise os trechos de código PHP abaixo e marque o bloco que possui erros assinalando-os:

a)

```
$doc = array();
$doc['rg'] = "00.000.000-X";
$doc['cpf'] = "000.000.000-00";
$doc['cartao'] = 12345;
asort($doc);
foreach($doc as $i => $dado)
    echo "<br /> $i = $dado ";
```

b)

```
$doc = array("rg" => "00.000.00-
X",
"cpf" => "000.000.000-00",
   "cartao" => 12345);
arsort($doc);
foreach($doc as $i => $dado)
    echo "<br /> $i = $dado ";
```

c)

```
$doc = array();
$doc['rg'] = "00.000.000-X";
$doc['cpf'] = "000.000.000-00";
$doc['cartao'] = 12345;
echo "<br /> RG     =  $doc[rg]
";
echo "<br /> CPF    =  $doc[cpf]
";
echo "<br /> Cartão =
$doc[cartao] ";
```

d)

```
$doc[rg]     = "00.000.000-X";
$doc[cpf]    = "000.000.000-00";
$doc[cartao] = 12345;
sort($doc);
foreach($dado as $doc)
    echo "<br /> $dado ";
```

Linguagem PHP – Parte 3

Nesta parte 3 da Unidade C vamos trabalhar funções. O uso de funções deixa o código mais organizado e modular, possibilitando a reutilização de código toda vez que precisamos realizar uma mesma tarefa. Ao longo deste material vamos apresentar situações problema em que se torne interessante o uso de funções. Ao final desta parte serão apresentadas as atividades para realização.

Funções

Uma função é um conjunto de instruções destinado a uma tarefa bem específica e que podemos utilizar várias vezes. Por exemplo, podemos precisar de funções para verificar se um CPF é válido, tratar valores recebidos de formulários, como datas e strings. A sintaxe para a criação de uma função em PHP é:

```
function nome_da_funcao(parâmetros) {


        // instruções

}
```

Atenção:

Lembre-se de que PHP é case sensitive para variáveis/funções definidas e que o nome de uma função deve ser único, ou seja, não podemos ter duas funções com o mesmo nome. Além disso, o nome da função não pode iniciar por número e não se podem usar caracteres como ponto, vírgula, espaço.

Os parâmetros da função são opcionais, mas os parênteses devem sempre aparecer. Uma função só é executada quando chamamos a função, ou seja, apenas a sua definição não executa as instruções pertencentes a ela. A chamada ou invocação de uma função se faz pelo nome da função com parênteses, por exemplo:

```
nome_da_funcao();
```

Uma função pode retornar um valor, neste caso, usa-se a palavra reservada ***return***. Vamos a um exemplo. A figura C.19 apresenta a definição de uma função chamada somar que recebe três valores por parâmetro. A função soma os três valores e retorna o valor. A definição da função começa na linha 3 e termina na linha 8. A seguir, na linha 11 é feita a chamada da função, passando os três valores para cálculo. Como a função retorna o valor total, será mostrado no navegador o resultado da soma.

```
1  <?php
2
3      function somar($v1, $v2, $v3){
4
5          $soma = $v1 + $v2 + $v3;
6
7          return $soma;
8      }
9
10
11     echo somar(4,7,9);
12 ?>
```

Figura C.19 - Definição de função
Fonte: do autor

Juntamente com funções, é importante compreender o escopo de variáveis em PHP:

- Para se utilizar uma variável que seja vista tanto dentro como fora da função deve-se declará-la como sendo GLOBAL, ou seja, fora da função.
- Uma variável declarada dentro de uma função será válida apenas dentro da própria função. Por exemplo, o código abaixo, não imprimirá nada em virtude de *$texto* possuir escopo local (função).

```
4       function mostrar(){
5
6           $texto = "teste de variável local";
7
8       }
9
10      echo $texto;
```

Figura C.20 - Exemplo de variável local
Fonte: do autor

A seguir vamos ver exemplos de várias funções.

Exemplo de função que recebe parâmetro e não retorna valor

A função apresentada na figura C.21, chamada tabuada, recebe um número por parâmetro e mostra a tabuada do número recebido. Veja que a função não retorna valor, já mostra diretamente dentro da função os valores. Na função também foi tratado o valor recebido por parâmetro. A função *is_numeric* retorna *true* se o valor passado por parâmetro é numérico e *false* em caso negativo. Se o valor for numérico é feito um teste se ele é maior ou igual a 1 e menor igual a 9, neste caso é feito um laço para mostrar a tabuada. Se o valor não estiver no intervalo então mostra uma mensagem "Informe um número entre 1 e 9".

```
1   <?php
2
3   //exemplo de função que recebe parâmetro e não retorna valor
4
5       function tabuada($num){
6
7           if (!is_numeric($num))
8               echo "Informe um número";
9           else
10          if ((int)$num >= 1 && (int)$num <=9){
11              echo "Tabuada do $num <br />";
12              for($i=1;$i<=10;$i++){
13
14                  $x = $num * $i;
15                  echo "$num * $i = $x <br />";
16              }
17
18          }else
19              echo "Informe um número entre 1 e 9";
20      }
21
22      tabuada(9);
```

Figura C.21 - Definição da função tabuada
Fonte: do autor

A figura C.22 mostra a chamada da função no navegador.

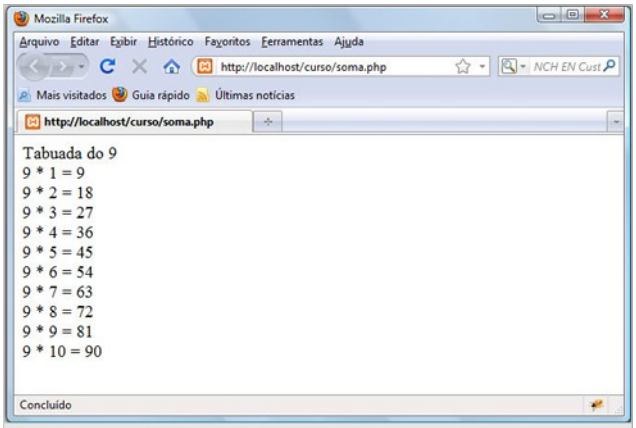


Figura C.22 - Resultado da chamada função tabuada
Fonte: do autor

Exemplo de função que recebe um vetor por parâmetro e retorna valor

A função apresentada na figura C.22, chamada somar_vetor, recebe por parâmetro um vetor e calcula a soma dos seus valores. Alguns apontamentos:

Na linha 7: Foi criada a variável $soma, sendo inicializada com zero.

Na linha 9: É usado um laço foreach para percorrer o vetor.

Na linha 10: É acumulado o valor do elemento do vetor na variável $soma.

Na linha 12: Após sair do laço, a função retorna o valor da variável $soma.

A linha 13: Fecha a função.

A linha 15: Define um array chamado $vetor com alguns valores.

Na linha 17: Mostra uma mensagem com o valor, retornado da chamada da função somar_vetor, passando o $vetor por parâmetro.

```
1  <?php
2
3  //exemplo de função que recebe por parâmetro um vetor e retorna valor
4
5      function somar_vetor($v){
6
7          $soma = 0;
8
9          foreach($v as $valor )
10             $soma += $valor;
11
12         return $soma;
13     }
14
15     $vetor = array(44,88,33,1,8,22,1,85,21,57);
16
17     echo "Soma dos valores do vetor: ". somar_vetor($vetor);
18
19 ?>
```

Figura C.23 - Definição da função somar_vetor
Fonte: do autor

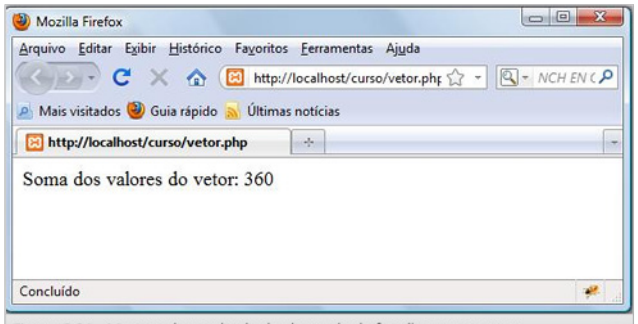


Figura C.24 - Mostrando resultado da chamada da função somar_vetor
Fonte: do autor

Exemplo de função que recebe vários parâmetros e retorna valor

A função definida no código da figura C.25, recebe por parâmetro dois números e a operação (+,-, * ou /) que deve ser efetuada para os dois valores. A função deve retornar o resultado do cálculo. Alguns apontamentos:

- a função é definida entre as linhas 2 e 22.
- está sendo usada na função a estrutura *switch* para verificar o que foi recebido em *$op*.
- quando a operação for de divisão está sendo feito um teste para verificar se o denominador não é zero para não ocorrer erro na divisão.

Na linha 21: É retornado o valor calculado.

- para testar a função foram criadas três variáveis (linhas 23 a 25). Neste caso, os valores são 6 e 5 e a operação é

multiplicação.

Na linha 27: É feita a chamada da função passando as variáveis por parâmetro.

- A figura C.26 mostra o resultado da chamada da função no navegador.

```php
<?php
function calculadora($v1, $v2, $op){
    $res=0;
    switch ($op) {
        case "+" :
            $res = $v1 + $v2;
            break;
        case "-":
            $res = $v1 - $v2;
            break;
        case "*":
            $res = $v1 * $v2;
            break;
        case "/":
            if ($v2 != 0)
                $res = $v1 / $v2;
            break;
        default :
            $res = "Operação inválida!";
    }
    return $res;
}
$num1     = 6;
$num2     = 5;
$operacao = "*";
echo "Resultado de $num1 $operacao $num2 = ".
     calculadora($num1, $num2, $operacao);
?>
```

Figura C.25 - Definição da função calculadora
Fonte: do autor

Atenção:

Note que as variáveis que são passadas por parâmetro para uma função não precisam ter o mesmo nome dos argumentos definidos na função.

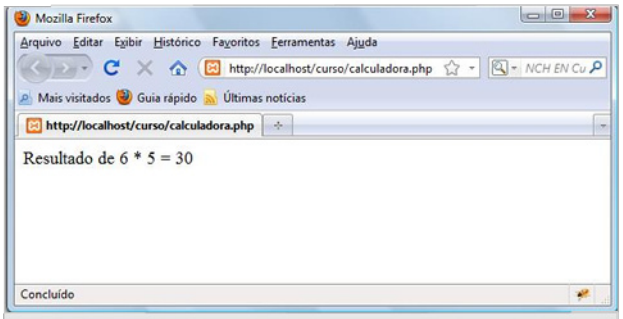


Figura C.26 - Mostrando resultado da chamada da função calculadora
Fonte: do autor

Funções require e include

As funções ***require*** e ***include*** incluem um arquivo de forma completa no script. A principal diferença entre os dois é que, se usarmos *require()* para incluir um arquivo que não pode ser incluído (talvez o arquivo não exista), um erro fatal será gerado e a execução de código na página será imediatamente suspenso. Se usarmos *include()* e o arquivo de inclusão não puder ser localizado, teremos uma advertência, mas a execução do código na página não será interrompida.

```php
<?php

    include("arquivo.php");

?>
```

Ou

```
<?php

    require(“arquivo.php”);

?>
```

Porque estamos vendo estas duas funções? Porque é muito comum e interessante que o desenvolvedor crie seus arquivos de funções e faça uso deles sempre que precisar com *include* ou *require*. Imagine que você pode criar um arquivo chamado *funcoes.php* com diversas funções comuns que você costuma usar, como na figura C.27. Sempre que necessitar usar essas funções, pode fazer a inclusão no arquivo que precisar. A figura C.28 mostra o código do arquivo *exemplo.php* que na linha 10 faz a inclusão do funcoes.php. A partir da inclusão, qualquer uma das funções pode ser usada. A linha 14 faz uso da função *valida_data* do *funcoes.php.* As funções *include* e *require* são bastante utilizadas pelos desenvolvedores e é recomendado que você organize seus arquivos para usá-las.

```php
<?php

function valida_cpf($cpf){
    //código da função
}

function valida_data($data){
    //código da função
}

function formata_data($data){
    //código da função
}

function formata_numero($num){
    //código da função
}

?>
```

Figura C.27 - Arquivo funcoes.php
Fonte: do autor

```php
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Exemplo</title>
</head>
<body>
<?php

    include('funcoes.php');

    $dt = "30/11/2000";

    if (valida_data($dt)){

        // código....

    }

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

Figura C.28 - Arquivo exemplo.php incluindo o funcoes.php
Fonte: do autor

Síntese

Trabalhar com funções é fundamental para melhorar a organização do código e possibilitar a sua reutilização. Agora você precisa praticar, por isso, faça as atividades propostas, retomando todo o material da Unidade C.

Atividades - Parte 3

1. Faça uma função PHP que recebe três parâmetros: um vetor de strings e duas strings ($string1 e $string2). Para cada elemento do vetor, a função deve substituir todas as ocorrências da string1 pela string2. A função deve retornar o vetor ordenado de forma crescente.

2. Considerando o vetor abaixo:

```
$vetor = array("10/04/2011", "14/04/2011", "21/04/2011", "01/05/2011",
"07/09/2011", "20/09/2011");
```

Faça uma função PHP que recebe por parâmetro:

a) um vetor (conforme estrutura acima) e
b) um número correspondente a um mês.

A função deve mostrar todas as datas do mês recebido, sendo no formato abaixo:

```
07 de setembro – Quarta-feira

20 de setembro – Sexta-feira
```

3. Faça uma função PHP que recebe por parâmetro um vetor de notas e gera e retorna um vetor com apenas as notas abaixo de 6.0, ordenando pela chave (índice) do vetor. Considere um vetor associativo.

4. Faça uma função que recebe por parâmetro o *$ano_de_nascimento* e retorna a idade da pessoa. Faça um formulário para testar a função.

5. Faça uma função que recebe por parâmetro $litros e $km e retorna quantos Km por litro um carro faz. Faça um formulário para testar a função.

6. Faça uma função PHP que recebe por parâmetro um vetor de datas de feriados.

A função deve verificar se a data de hoje é um feriado. Caso a data seja um feriado, a função deve retornar "SIM" em caso negativo retornar "NÃO". Considere o vetor abaixo como exemplo e mostre a chamada da função.

```
$feriados = array("07/09/2010", "20/09/2010", "12/10/2010", "02/11/2010",
"15/11/2010", "08/12/2010");
```

Linguagem PHP – Parte 4

Nesta parte 4 da Unidade C. vamos introduzir o conceito de sessões e cookies e mostrar como eles podem ser usados em PHP. Estes dois recursos são muito importantes no desenvolvimento de aplicações para o ambiente Web e é imprescindível que o desenvolvedor tenha conhecimento para manipulá-los. Ao final desta parte são apresentadas a atividades para realização.

Sessões

É muito comum em sites a necessidade de manter informações do usuário enquanto este estiver navegando (manter informações entre as diversas páginas dentro do site). O exemplo mais comum é quando você informa login e senha em algum site para acessar uma área de dados restrita ao seu usuário, por exemplo, para e-mails. Ou ainda, quando você compra pela internet e coloca os produtos escolhidos no carrinho de compras, os dados do carrinho de compras são exclusivamente seus.

Mas então, como estas informações são mantidas entre as páginas? Bem, a forma mais usual é a SESSÃO.

Primeiro precisamos entender o que é a sessão, para depois vermos como trabalhar com ela em PHP.

A Sessão, de modo geral, é uma área do servidor onde podemos guardar valores para recuperar depois. Cada usuário que se conecta ao site recebe uma sessão (um identificador). Os dados colocados na sessão só são removidos quando a sessão é encerrada (navegador é fechado) ou quando a sessão expira por um período de inatividade configurado pelo programador.

Agora vamos começar a trabalhar na prática para que você entenda melhor.

A sessão precisa ser iniciada em cada página que você for usar ou definir um valor. Para abrir a sessão é só usar a função abaixo:

```
session_start(); // Inicia a sessão
```

Atenção:

É necessário iniciar a sessão antes de qualquer echo ou de inserir qualquer HTML fora de blocos php. Geralmente o início da sessão está no começo da página.

Depois de iniciada a sessão você pode definir valores usando o array superglobal do PHP:

```
$_SESSION[“login”] = “ maria”;
```

Vamos visualizar isto em uma sessão. Crie o arquivo *sessao1.php* conforme código da figura C29.

```
1  <?php
2
3      session_start();
4
5      $_SESSION["login"] = "maria";
6
7      echo "Criando variável na sessão...";
8
9  ?>
```

Figura C.29 - Código do arquivo sessao1.php
Fonte: do autor

Ao visualizar o arquivo *sessao1.php* no navegador, apenas a mensagem será mostrada. Mas o que nos interessa é verificar onde a variável ficou armazenada no servidor. Como estamos trabalhando de forma

local, podemos acessar estes dados. Abra a pasta c:\xampp\tmp. Você verá que um arquivo foi gerado para a sessão criada a partir do código do arquivo *sessao1.php*. A figura C.30 mostra a pasta com o nome do arquivo gerado. O nome do arquivo inicia com *sess_ e* depois vem o identificador da sessão.

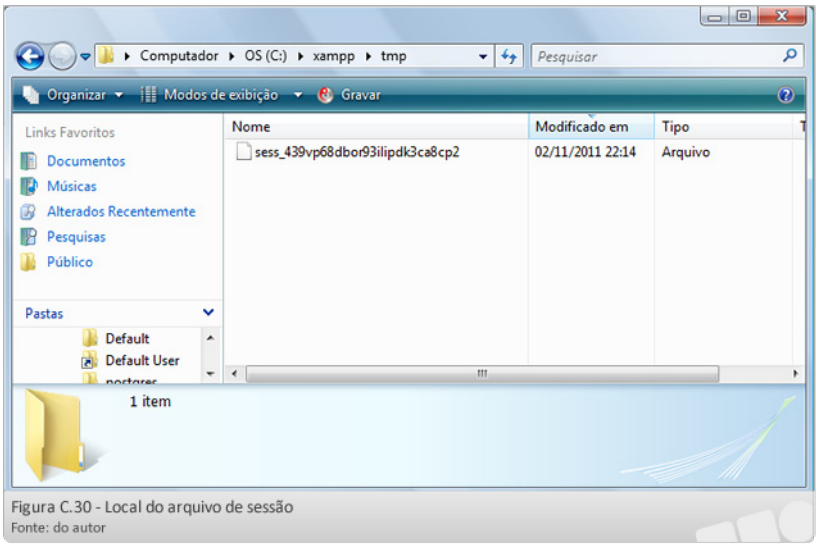


Figura C.30 - Local do arquivo de sessão
Fonte: do autor

Ao abrir o arquivo, é possível verificar os dados da sessão. A figura C.31 mostra o arquivo aberto no bloco de notas. Primeiro vem o nome da variável colocada na sessão, depois do pipeline vem a letra s indicando que a variável é do tipo string, e depois dos dois pontos vem o valor correspondente à variável (maria).

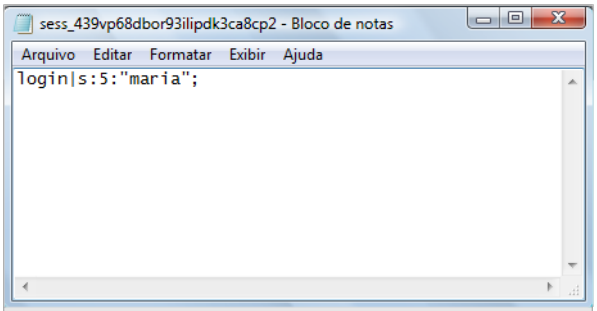


Figura C.31 - Dados da sessão
Fonte: do autor

Contudo, uma sessão pode conter muitas variáveis. Vamos alterar um pouco nosso código para acrescentar mais variáveis. Veja como ficou na figura C.32. Também usamos agora a função *session_id()* que retorna o identificador da sessão, aquele que também é usado no nome do arquivo. Assim, você poderá ver no navegador o identificador da sessão. A figura C. 33 mostra o arquivo agora com as novas variáveis registradas na sessão.

```php
<?php

    session_start();

    $_SESSION["login"] = "maria";
    $_SESSION["data"]  = "02/11/2011";
    $_SESSION["hora"]  = "22:20";

    $id_sessao = session_id(); //identificador da sessão

    echo "Criando variáveis na sessão $id_sessao ";

?>
```

Figura C.32 - Código do arquivo sessao1.php com mais variáveis
Fonte: do autor

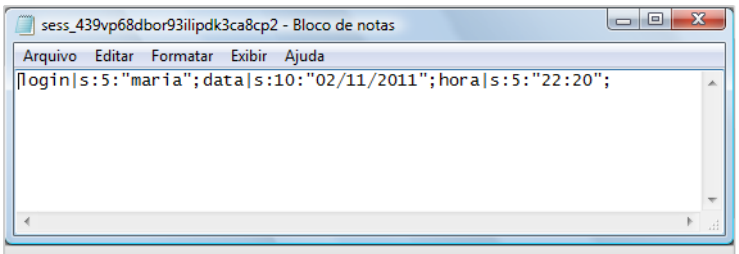


Figura C.33 - Dados da sessão
Fonte: do autor

Como recuperar estes dados da sessão? O código da figura C.34 mostra como verificar se existe um dado na sessão e como podemos recuperá-lo. Usamos a função *isset* para verifica se a variável está definida (retorna *true* se estiver e *false* se não existir).

```php
<?php
    session_start();

    if (isset($_SESSION['login'])) {

        echo $_SESSION['login'];

    }else {

        echo "usuário não logado";
    }

?>
```

Figura C.34 - recuperar valor da sessão
Fonte: do autor

Outras funções importantes para trabalhar com sessões são apresentadas na tabela abaixo:

Função	Descrição
`unset()`	Exclui a uma variável específica da sessão. Ex.: `unset($_SESSION[“login”]);`
`session_destroy()`	Destrói toda a sessão, eliminando todas as variáveis salvas nela. Antes de chamar a função *session_destroy()*, deve-se primeiro abrir a sessão com *session_start()*.Essa função é normalmente utilizada quando um usuário requisita sua saída da aplicação (*logout*).
`session_cache_expire()`	Retorna o tempo de expiração da sessão ou define novo tempo de expiração em minutos. Para redefinir o tempo de expiração, deve ser executada antes de *session_start()*. O tempo padrão de duração da sessão é de 180 minutos. Para alterar o tempo padrão, deve-se chamar *session_cache_expire()* a cada página que faça uso da sessão.

Cookies

Outra forma de manter dados para posterior acesso em outras páginas do site é através do uso de cookies. Só que diferentemente da sessão, o **cookie** é um arquivo texto armazenado no computador do usuário e pode ser recuperado posteriormente pelo servidor.

O uso de cookies, por exemplo, permite que o usuário após ter se autenticado em um site, desligue o computador, acesse o site um tempo depois e não precise se autenticar novamente. Ex: GMail, Hotmail, Yahoo, etc. Para utilizar esse recurso, geralmente o usuário precisa aceitar algum tipo de opção como

"salvar as minhas informações neste computador". Além disso, é preciso que o navegador do usuário esteja com a opção de armazenar cookies habilitada.

Vamos ver como isso pode ser configurado no Firefox:

- clique no menu Ferramentas ou *Tools* e depois em Opções ou *Options*;
- clique na guia Privacidade ou *Privacy* e, na seção Cookies, ative a opção "Sites podem definir Cookies" ou "*Allow sites to set cookies*". Veja na figura C.35.

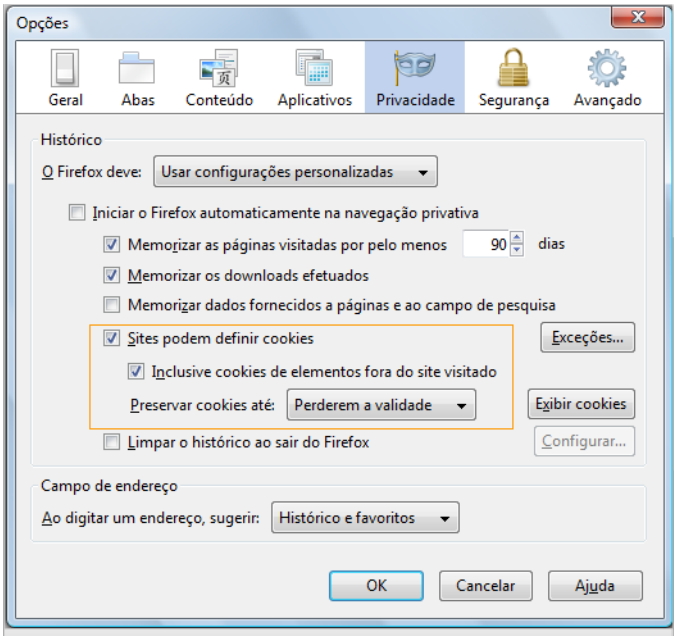


Figura C.35 - Opções do Firefox
Fonte: do autor

Vamos ver agora como definir um cookie em PHP:

```
<?php

            setcookie('nome_cookie', 'valor do cookie');

?>
```

Na definição de um cookie os dois primeiros parâmetros são obrigatórios: *nome do cookie e valor do cookie*. No entanto, outros parâmetros podem ser definidos. Veja abaixo:

```
setcookie(nome, valor, expira, caminho, domínio, seguro, http)
```

Nome	nome do cookie.
Valor	valor do cookie.
Expira	tempo para o cookie expirar (número de segundos desde 01/01/1970).
Caminho	caminho no servidor aonde o cookie estará disponível.
Domínio	domínio para qual o domínio estará disponível.
Seguro	(*true/false*) indica que o cookie só poderá ser transmitido sob uma conexão segura HTTPS do cliente.
Http	(*true/false*) quando for TRUE, o cookie será acessível somente sob o protocolo HTTP. Isso significa que o cookie não será acessível por linguagens de script, como JavaScript.

Vamos a um exemplo. Veja o código da figura C.36 e faça o teste na sua máquina.

```php
<?php

    setcookie( 'cookie_lpw_tics' , 'Linguagem de Programação para Web' , time()+ 3600 );

    echo "Definindo um cookie...";
?>
```

Figura C.36 - Código
Fonte: do autor

Você deve ter observado que na função *setcookie* foram passados três parâmetros. O terceiro parâmetro é referente ao tempo de validade do cookie. Nesse caso, foi definido que o cookie tem validade de uma hora (tempo atual + 3600 segundos). Vamos ver onde este cookie foi criado na sua máquina. Se o seu navegador for Firefox, acesse o menu Ferramentas e depois Opções. Clique em "Exibir Cookies". A figura C.37 mostra a visualização da janela que é apresentada. Na figura é possível observar as informações relativas ao cookie.

Dica:

Se você possuir muitos cookies, uma opção é excluir todos eles e depois executar o arquivo para geração do cookie. Assim, você vai encontrar facilmente o cookie gerado.

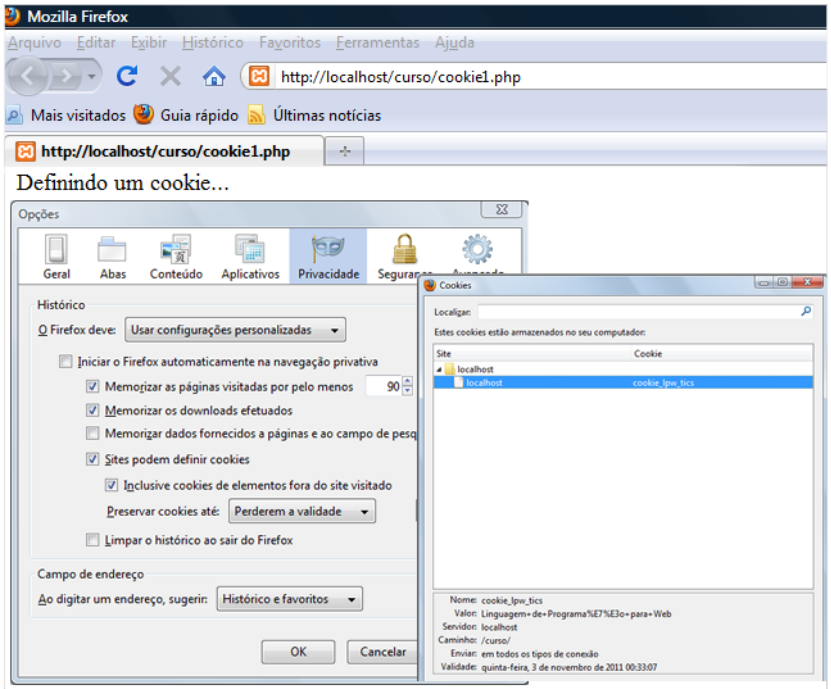


Figura C.37 - Informações relativas ao cookie
Fonte: do autor

Para recuperar os dados do cookie usamos o array superglobal do PHP $_COOKIE[], passando o nome do cookie como parâmetro. Veja na figura C.38 um código de exemplo.

```php
<?php

    $dados  = $_COOKIE['cookie_lpw_tics'];

    echo "Dados recuperados do cookie: $dados";

?>
```

Figura C.38 - Recuperar dados do cookie
Fonte: do autor

Se for necessário remover um cookie, basta usar a mesma função setcookiet com o nome do cookie e com a string vazia (um exemplo pode ser visto na figura C.39).

```php
<?php

    setcookie('cookie_lpw_tics', '');

    echo "excluindo cookie...";
?>
```

Figura C.39 - Excluindo o cookie
Fonte: do autor

Síntese

O uso de sessões e cookies é muito comum em web sites, por exemplo, sessões é sempre usado para autenticação de usuários e cookies para armazenar dados do perfil do usuário para uma posterior visita, podendo mostrar produtos que são de seu interesse. Bem, agora é com você, faça as atividades propostas para ganhar mais experiência com essas novas funções.

Leitura:

Para saber mais e ver outros exemplos sobre sessões e cookies, leia o capítulo 13 da seguinte bibliografia: NIEDERAUER, J. Desenvolvendo websites com PHP: Aprenda a criar websites dinâmicos e interativos. São Paulo: Novatec, 2004.

Atividades - Parte 4

1. Crie uma página PHP que coloca na sessão um contador iniciando em 1. Cada vez que a página for executada dentro da sessão (existindo já a variável) o contador é incrementado em 1. Mostre sempre o valor do contador na página e o identificador da sessão.

2. Dado o código PHP da figura abaixo, faço o arquivo *dados_sessao.php* para mostrar todos os dados da sessão.

```php
<?php

    session_start();

    $_SESSION["data"]  = date("d/m/Y");
    $_SESSION["hora"]  = date("H:i:s");
    $_SESSION["dia"]   = date("w");

    $id_sessao = session_id(); //identificador da sessão

    echo "Criando variáveis na sessão $id_sessao ";
?>
<br /><br />
<a href="dados_sessao.php"> Ver dados da sessão</a>
```

Figura C.40 - Exercício 2
Fonte: do autor

3. Utilizando cookies, crie uma página com um mecanismo que conte o número de acessos de um mesmo usuário a essa página. Mantenha esta contagem válida por 1 minuto (desde o último acesso).

4. Crie um formulário de login para autenticação e mantenha os dados de login do usuário por meia hora, mesmo depois de o usuário ter fechado o navegador.