

PHP

Resumo

Segundo a [ABNT \(2003, 3.1-3.2\)](#), o resumo deve ressaltar o objetivo, o método, os resultados e as conclusões do documento. A ordem e a extensão destes itens dependem do tipo de resumo (informativo ou indicativo) e do tratamento que cada item recebe no documento original. O resumo deve ser precedido da referência do documento, com exceção do resumo inserido no próprio documento. (...) As palavras-chave devem figurar logo abaixo do resumo, antecidas da expressão Palavras-chave:, separadas entre si por ponto e finalizadas também por ponto.

Palavras-chave: latex. abntex. editoração de texto.

Lista de abreviaturas e siglas

ABNT	Associação Brasileira de Normas Técnicas
abnTeX	ABsurdas Normas para TeX
PHP	Pré-Processador de Hipertexto

Sumário

1	PHP	9
1.1	PHP (Pré-Processador de Hipertexto)	9
1.2	Arquitetura cliente-servidor	9
1.3	Instalação do PHP	10
1.4	Instalação do Apache	11
1.5	Testando o ambiente	12
1.5.1	Função <code>phpinfo</code>	12
1.5.2	Instrução <code>echo</code>	13
1.5.3	Comentários no PHP	14
1.6	Exercícios	14
1.7	Desafio!	14
2	Ambiente de desenvolvimento com Sublime Text	15
2.1	O Sublime Text	15
2.2	Instalando o Sublime Text	15
2.3	Primeiros passos	16
2.4	Comandos do Sublime Text	17
2.5	Adicionando diretórios	17
2.6	Configurações do Usuário	18
2.7	Configurações e Plugins	19
2.7.1	Plugin <code>php-snippets</code>	19
2.8	Desafio!	20
3	Variáveis em PHP	21
3.1	Variáveis	21
3.2	Tipos de Variáveis	22
3.2.1	Tipo Inteiro	22
3.2.2	Tipo Ponto flutuante	23
3.2.3	Tipo <i>Booleano</i>	23
3.2.4	Tipo <i>String</i>	24
3.2.5	Tipo <i>Array</i>	24
3.2.6	Tipo <i>NULL</i>	25
3.3	Constantes	25
3.3.1	Constantes pré-definidas	25
3.3.2	Definindo constantes	25
3.4	Conversão de variáveis	26
3.5	Exercícios	27
3.6	Desafio!	28

4	Operadores em PHP	29
4.1	Operadores	29
4.2	Operador de atribuição	29
4.3	Operadores de <i>strings</i>	29
4.4	Operadores Aritméticos	30
4.5	Operadores Combinados	31
4.6	Operadores de decremento e incremento	32
4.7	Operadores relacionais	33
4.7.1	Operador condicional ternário	34
4.8	Operadores Lógicos	35
4.8.1	Operador NOT	35
4.8.2	Operador AND	36
4.8.3	Operador OR	36
4.9	Precedência de operadores	37
4.10	Exercícios	37
4.11	Desafio!	39
5	Estruturas de controle e repetição	41
5.1	Comando if	41
5.1.1	Comando if..else	42
5.2	Atribuição condicional (ternário)	43
5.3	Estrutura switch	44
5.3.1	Comando switch com break	45
5.3.2	Comando switch completo	46
5.4	Estrutura while	47
5.5	Estrutura do...while	48
5.6	Estrutura for	49
5.7	Estrutura foreach	50
5.8	Comando break	51
5.9	Comando continue	51
5.10	Exercícios	52
5.11	Desafio!	52
6	Manipulação de array's	53
6.1	Criando um array	53
6.2	Arrays associativos	53
6.2.1	Inicializando <i>array's</i>	54
6.2.2	Acessando um <i>array</i>	54
6.3	Percorrendo um <i>array</i>	55
6.4	Acessando um <i>array</i>	56
6.5	Alterando um <i>array</i>	57

6.6	<i>Arrays</i> com duas dimensões	57
6.6.1	Acessando um <i>array</i> de duas dimensões	58
6.6.2	Percorrendo um <i>array</i> de duas dimensões	59
6.7	Visualizando <i>array's</i>	59
6.7.1	Função <code>var_dump</code>	60
6.7.2	Função <code>print_r</code>	60
6.7.3	Funções <code>sort</code> e <code>rsort</code>	61
6.8	Exercícios	61
6.9	Desafio!	61
7	Interações PHP com HTML	63
7.1	Formulários	63
7.1.1	Criando um formulário	63
7.2	Métodos <code>GET</code> e <code>POST</code>	64
7.2.1	Método <code>GET</code>	65
7.2.2	Nosso primeiro formulário com método <code>GET</code>	65
7.2.3	Recebendo dados via método <code>GET</code>	66
7.3	Método <code>POST</code>	67
7.3.1	Nosso primeiro formulário com método <code>POST</code>	68
7.3.2	Recebendo dados via método <code>POST</code>	68
7.4	Exercícios	69
7.5	Desafio!	69
8	Manipulação de funções	71
8.1	Declarando uma função	71
8.2	Passagem de parâmetros	71
8.3	Valor de retorno	72
8.4	Escopo de variáveis em funções	73
8.5	Exercícios	74
8.6	Desafio!	74
9	Manipulação de arquivos e diretórios	75
9.1	Criando e abrindo um arquivo	75
9.2	Gravando em um arquivo	76
9.3	Fechando um arquivo	77
9.4	Lendo de um arquivo	77
9.4.1	Comando <code>file</code>	77
9.5	Trabalhando com arquivos <code>.csv</code>	78
9.5.1	Função <code>str_getcsv</code>	78
9.6	Manipulando diretórios	79
9.6.1	Comando <code>getcwd</code>	79
9.7	Exercícios	80

9.8	Desafio!	80
10	Interações com o navegador	81
10.1	Variável \$_SERVER	81
10.2	<i>Cookies</i>	81
10.2.1	Acessando <i>Cookies</i>	83
10.3	Sessão	84
10.4	Exercícios	85
10.5	Desafio!	85
11	Tratamentos de erros	87
11.1	Tipos de Erro	87
11.2	Exercícios	88
11.3	Desafio!	88
12	Introdução a POO com PHP	89
12.1	O que é programação orientada a objetos?	89
12.2	Classes e Objetos	90
12.3	Atributos	91
12.4	Métodos	92
12.5	Herança	95
12.6	Exercícios	97
12.7	Desafio!	97
13	Conclusão	99
Referências		101
Apêndices		103
APÊNDICE A	Instalação de ambiente de desenvolvimento no Windows	105
APÊNDICE B	Quisque libero justo	107
Anexos		109
ANEXO A	Morbi ultrices rutrum lorem.	111
ANEXO B	Cras non urna sed feugiat cum sociis natoque penatibus et magnis dis parturient montes nascetur ridiculus mus	113

1 PHP

Ao final deste capítulo, o aluno terá as seguintes competências:

1. Entender a arquitetura cliente-servidor;
2. Instalar o servidor web (Apache) e a linguagem PHP; e
3. Testar o ambiente de desenvolvimento.

1.1 PHP (Pré-Processador de Hipertexto)

O PHP (Pré-Processador de Hipertexto), foi criado por *Rasmus Lerdorf* em 1995 e originalmente chamado de “*Personal Home Page Tools*” (Ferramentas para Página Pessoal). Com a aceitação do projeto, muitos programadores passaram a utilizar e propor mudanças, surgindo assim, o PHP que iremos conhecer hoje. O PHP está atualmente na versão 7.0, chamado de PHP7 ou, simplesmente de PHP. A nível de estudo, utilizaremos o PHP 5.3.2, pois é uma versão mais estável e muito utilizada no mercado.

O PHP é uma linguagem de programação que funciona no lado do servidor, ele permite criarmos *sites* dinâmicos, ou seja, o *site* se comporta de acordo com a entrada de dados do usuário. Outros exemplos de linguagem semelhantes são ASP, JSP (Java) e Python.

A linguagem PHP trabalha lado a lado com o HTML (Linguagem de Marcação de Hipertexto), por conta disso vamos precisar saber o básico de HTML, principalmente as *tags* de formulário. Devemos lembrar que o PHP tem pouca relação com o *layout* ou eventos que compõem a aparência de uma página *web*. Portanto, podemos dizer que a maior parte do que o PHP realiza é invisível para o usuário final. O internauta, ao visualizar a página desenvolvida em PHP não será capaz de identificar que a página foi escrita utilizando a tecnologia disponibilizada pelo PHP.

Você arriscaria dizer que o Facebook foi desenvolvido com a linguagem PHP?

1.2 Arquitetura cliente-servidor

Como visto na seção anterior, o PHP funciona do lado do servidor. Para entendermos melhor isso, é necessário entender a estrutura cliente/servidor. Muito utilizada na *internet*. A figura abaixo exemplifica de maneira simples a comunicação entre cliente e servidor.

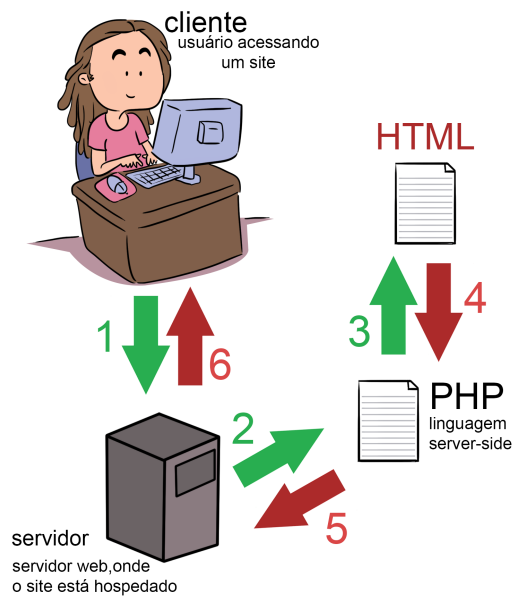


Figura 1.2.1 – Ilustração de uma requisição web.

Dessa figura, podemos tirar algumas palavras chaves. Que são:

Recurso Item disponível na Internet (uma figura, uma página, um arquivo .css);

Cliente Aquele que **requisita** algum recurso (navegador Firefox); e

Servidor Aquele que **provê** algum recurso (servidor Apache).

Portanto, quando o **cliente**, ou seja, o internauta, faz uma **requisição** - digitando na barra de endereços do navegador o *site* <http://projetojovem.seduc.ce.gov.br> e pressionando **ENTER** - o navegador se encarrega de fazer um pedido ao **servidor** que guarda o *site* do projeto e-Jovem.

Passo 1 Usuário requisita página de internet acessando o navegador e digitando o *site*;

Passo 2 O servidor *web* se comunica com a linguagem PHP;

Passos 3 e 4 O PHP e o HTML se juntam em um só arquivo;

Passo 5 O servidor tem agora a página pronta para ser enviada ao usuário; e

Passo 5 O usuário recebe a página completa em seu navegador.

1.3 Instalação do PHP

Para que possamos utilizar o PHP, devemos instalar a linguagem no nosso computador de trabalho. Vamos instalar esses pacotes através do **terminal**. Podemos abrir o **terminal** de várias maneiras. Veja duas delas listadas abaixo:

1. clique com o botão direito na área de trabalho, escolha a opção

Abra o Emulador de Terminal aqui; e

2. acione a combinação de teclas `Alt` + `F2` e digite `xfce4-terminal`.

Em seguida escreva o comando abaixo no `terminal` que acabamos de abrir. Por segurança a senha de usuário será requisitada, e **ela não aparece ao ser digitada**. Não se preocupe, digite a senha e ao final aperte enter.

```
1 $ sudo apt-get install php5 libapache2-mod-php5 php5-gd curl
2     php5-curl php5-xmllrpc php5-cli
```

Se você estiver usando o Linux do Projeto e-Jovem, então esses pacotes já devem ter sido instalados e você visualizou a seguinte tela.

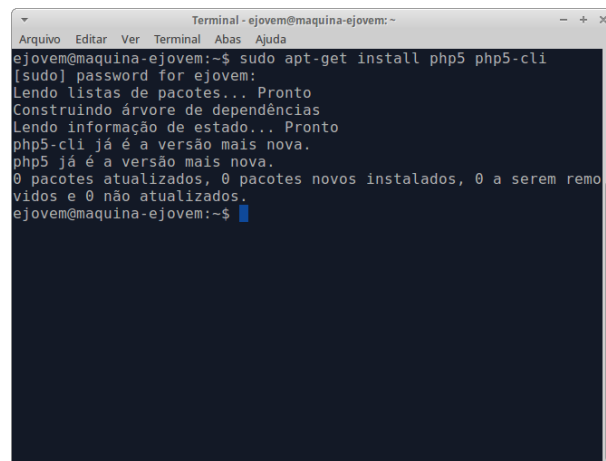


Figura 1.3.1 – Instalação do PHP bem sucedida.

1.4 Instalação do Apache

O servidor Apache é um dos principais aplicativos que fazem a *web* funcionar. Ele é responsável por interpretar os arquivos `.php` e retornar para o cliente, apenas o que ele requisitou. A versão que vamos trabalhar é a 2. O processo de instalação é parecido com o que foi utilizado no PHP. Abra o `terminal` utilizando um dos passos da seção 1.3 e digite a seguinte instrução.

```
1 $ sudo apt-get install apache2
```

Se o sistema utilizado for o Linux do Projeto e-Jovem, então já temos o Apache2 instalado (figura 1.4.1a). Digite no navegador Firefox o endereço de internet `<http://localhost>` (sem os sinais de maior e menor que). A tela será parecida com a mostrada na figura 1.4.1b.

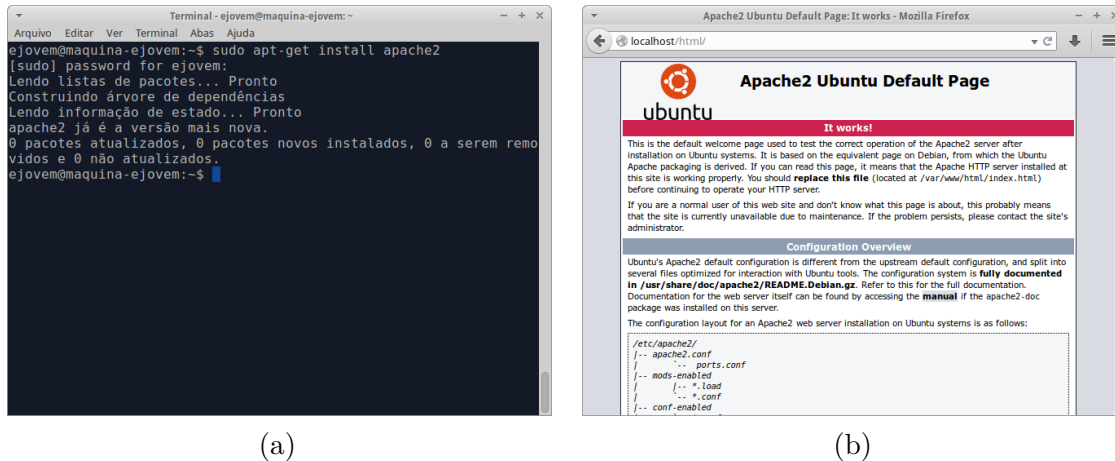


Figura 1.4.1 – (a) Instalação do Apache2 bem sucedida; (b) Verificação do Apache em execução. Digite <http://localhost> no navegador Firefox.

A figura 1.4.1b indica que o Apache está funcionando corretamente. O arquivo apresentado acima pode ser encontrado no diretório `/var/www/`. Será essa a localização dos arquivos que vamos desenvolver. Ou seja, sempre que criarmos um arquivo `.php` ele deverá ser salvo no `/var/www/`.

Para que seja possível o usuário do sistema (no caso você) salve no `/var/www/`, precisamos mudar a permissão de escrita do diretório. Vamos abrir o `terminal` de acordo com o que foi mostrado na seção 1.3. Com o `terminal` aberto, digite o seguinte comando.

```
1 $ sudo chmod -R 777 /var/www
```

O comando acima permite que o usuário comum do sistema grave arquivos no `/var/www/`.

O aplicativo Apache pode ser configurado para funcionar de diversas maneiras. Essa disciplina necessita apenas da configuração básica. Caso queira modificá-la, o aluno poderá ler mais sobre o Apache através do site: <http://httpd.apache.org/docs/2.2>.

Caso você use o sistema operacional Windows na sua casa, veja no apêndice A, lá é explicado como instalar o PHP e o Apache no Windows.

1.5 Testando o ambiente

1.5.1 Função phpinfo

Após a instalação, devemos testar o nosso ambiente de desenvolvimento (composto inicialmente por PHP e Apache). Abra novamente o `terminal`, navegue até o diretório `/var/www/`.

Neste diretório, iremos criar **uma pasta para cada aula** do curso, portanto, hoje criaremos o diretório `aula01` no caminho `/var/www/`. Abra novamente o `terminal` e

1.5.3 Comentários no PHP

O último tópico deste capítulo trata dos comentários que podemos escrever nos nossos arquivos `.php`. Comentários são partes importantes do código desenvolvido. Eles servem para ajudar a entender melhor determinadas partes do código ou ainda para descrever o que o código desenvolvido realiza. Outra funcionalidade importante dos comentários são os de orientar outros programadores, permitindo assim que os desenvolvedores trabalhem em conjunto de maneira mais fácil.

No PHP, os comentários uma linha são indicados pelos caracteres `//` ou `#` e os comentários de múltiplas linhas são representados pelos caracteres `/*` (início) e `*/` (fim).

Veja o nosso arquivo `index.php` após a adição dos comentários explicativos.

```
1 <?php /*
2     linha 1
3     multiplas linhas
4     linha 3
5 */ ?>
6 <?php echo "Aula 01"; ?>
7 <?php phpinfo(); ?>
8 <?php // essa linha nao aparece no navegador ?>
```

1.6 Exercícios

Para realizar as questões abaixo, crie um documento no diretório `/var/www/aula01` de nome `ejovem.php`.

- Q. 01** Como o arquivo `.php` deve ser inicializado? Ou seja, o que deve vir no começo e no final do arquivo?
- Q. 02** Exiba a mensagem “projeto e-jovem” utilizando o comando `echo`.
- Q. 03** Adicione um comentário de uma linha com os dizeres: “Primeira aula de PHP”.
- Q. 04** Adicione um comentário de múltiplas linhas com cada palavra da frase “comentário de múltiplas linhas” em uma linha.
- Q. 05** Exiba a mensagem “Olá Aluno!” utilizando a *tag* `<h1>` e o comando `echo`.

1.7 Desafio!

O desafio deste capítulo é você encontrar na tela do navegador a versão do PHP que vamos trabalhar, a versão do Apache além de nos mostrar qual o diretório padrão em que os arquivos `.php` devem ser salvos.

2 Ambiente de desenvolvimento com Sublime Text

Ao final deste capítulo, o aluno terá as seguintes competências:

1. Instalar o Sublime Text 3 no sistema operacional;
2. Instalar *plugins* do Sublime Text; e
3. Criar uma estrutura de aplicação *web* com PHP.

2.1 O Sublime Text

O Sublime Text é um editor de textos melhorado. Com esse tipo de *software* é possível desenvolver os mais diversos programas, incluindo os *sites*. O Sublime Text hoje dispõe de duas versões que são amplamente utilizadas. A versão 2 - estável porém mais antiga e a versão 3 - caracterizada como versão *beta* porém mais nova. Apesar de no curso sempre trabalharmos com as versões estáveis dos programas, com esse editor usaremos a versão 3.

O editor de textos não vem por padrão nas distribuições Linux. Por isso, é necessário instalá-lo. Vamos abrir o navegador Firefox e digitar a URL <https://www.sublimetext.com/3>. Em seguida, clique no *link* “Download”. Se você estiver usando o Linux do Projeto e-Jovem escolha a opção “Ubuntu 32 bit”. Essa ação vai baixar o arquivo `sublime-text_build-3XXX_i368.deb`. o 3XXX indica a versão (3 no caso) e pequenas mudanças na versão principal respectivamente.

Lembre-se! Saiba onde você salvou o arquivo baixado! Será importante saber a localização dele no momento da instalação!

2.2 Instalando o Sublime Text

Agora que baixamos o arquivo `.deb`, vamos instalá-lo. Abra o **terminal** de acordo com o apresentado na seção 1.3. Utilize os comandos `cd`, `ls` e `pwd` para ir ao diretório que o arquivo `sublime-text_build-3XXX_i368.deb` está salvo. No caso da apostila, o arquivo foi salvo dentro do diretório `/home/ejovem/Downloads`.

Caso você não saiba onde o arquivo se encontra, peça ajuda ao seu instrutor.

Comandos executados no terminal

```
1 $ cd ~/Downloads
2 $ ls
```

Resposta do comando `ls`

```
1 ...
2 sublime-text_build-3114_i368.deb
```

3 . . .

Agora que a gente sabe onde o arquivo se encontra, podemos instalar com o comando `dpkg`. Execute o seguinte código no diretório em que o `sublime-text_build-3XXX_i368.deb` se encontra

```
1 $ sudo dpkg -i sublime-text_build-3114_i368.deb
```

Se a instalação do Sublime Text ocorreu com sucesso. Obtemos a seguinte tela na imagem 2.2.1a. Para abrir o programa, digite a combinação de teclas `[Alt] + [F2]` e digite o comando `subl`. A tela inicial do programa é a exibida na figura 2.2.1b.

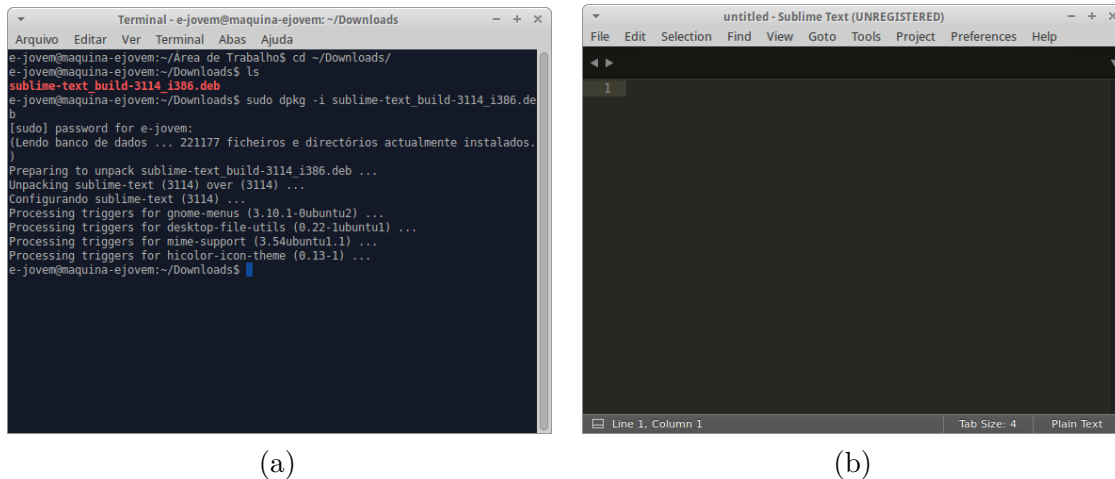


Figura 2.2.1 – (a) Instalação do aplicativo Sublime Text realizada com sucesso; (b) Tela inicial do Sublime Text.

2.3 Primeiros passos

Vamos começar nossos testes com o Sublime Text. Nosso primeiro teste é abrir o arquivo editado anteriormente pelo programa `gedit`. Na tela inicial do Sublime Text, execute a seguinte sequência de menus: `File >> Open File...`. O arquivo deverá estar no diretório `/var/www/aula01`. E tem o nome de `index.php`.

Vamos editar o arquivo. Troque a primeira linha do arquivo para que fique parecido com o apresentado abaixo.

```
1 <?php echo "<h1>Aula 01</h1>"; ?>
2 <?php phpinfo(); ?>
```

Lembre-se! Você pode colocar *tags* HTML dentro das instruções PHP!

O resultado pode ser conferido abaixo.

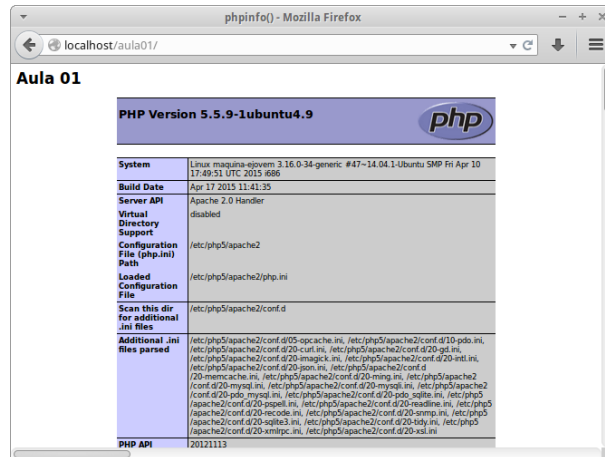


Figura 2.3.1 – Utilizando echo.

2.4 Comandos do Sublime Text

O nosso editor de textos tem um painel de comandos. Nele é possível aplicar comandos relacionados ao Sublime Text. Os comandos que iremos utilizar são em inglês, portanto, a tabela TAL trás a tradução de alguns termos utilizados.

Para acessar o painel digite a combinação de teclas **Ctrl** + **⇧** + **P** enquanto estiver no Sublime Text. O painel deve aparecer na parte superior da janela. Digite o termo *Esse termo ativará o comando de nome View: Toggle Side Bar*. Esse comando ativa ou desativa a barra lateral no Sublime Text (pode ser utilizado a combinação de teclas **Ctrl** + **K**, **Ctrl** + **B**).

Outro comando que podemos utilizar é o de ativar ou desativar o *minimap*. Pressione então as teclas **Ctrl** + **⇧** + **P** e digite o termo *minimap*. O comando completo é *View: Toggle Minimap*.

2.5 Adicionando diretórios

Uma das grandes facilidades do editor de textos Sublime Text e outros editores mais avançados é a apresentação do projeto através da barra lateral (*sidebar*). No nosso caso, um projeto pode ser descrito simplesmente como uma pasta. Então vamos adicionar a pasta **aula01** ao Sublime Text.

Para isso, abra o programa gerenciador de arquivos do sistema. No caso da distribuição do e-Jovem, o programa a ser aberto é o **thunar**. Pressione a combinação de teclas **Alt** + **F2** em seguida digite **thunar**. Navegue até o diretório **/var/www/** clique em cima da pasta **aula01** segure e arraste até o editor de textos Sublime Text.

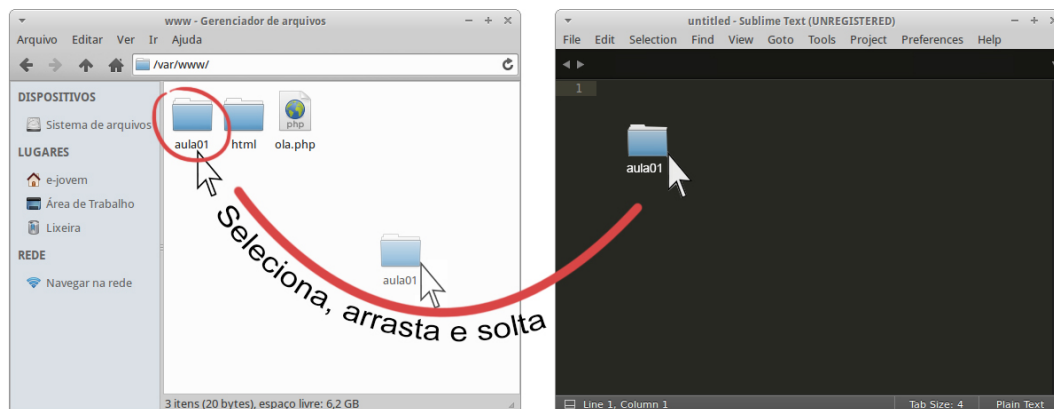


Figura 2.5.1 – Adicione diretório ao Sublime Text de maneira simples, clique em uma pasta e arraste para dentro do editor.

Nesse momento a tela principal do Sublime Text encontra-se com a *sidebar* (barra lateral) ativada e apresentando o diretório que acabamos de adicionar.

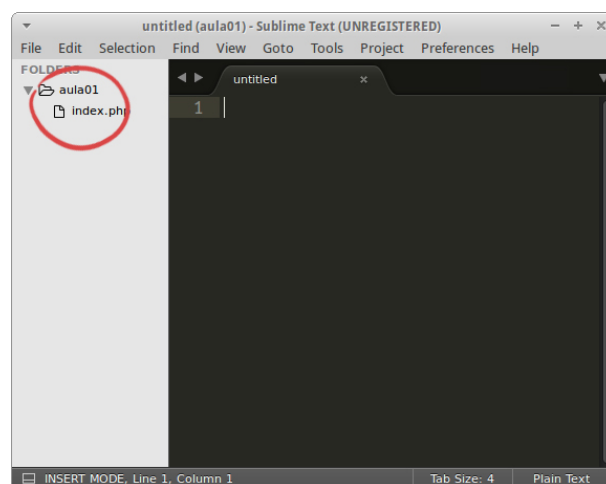


Figura 2.5.2 – Programa Sublime Text com barra lateral visível.

Dica! Outro método que podemos utilizar para realizar a mesma tarefa é utilizar os menus do Sublime Text. O caminho a ser percorrido é: **File** > **Open Folder...** e escolher o diretório `/var/www/aula01`.

2.6 Configurações do Usuário

O Sublime Text permite diversas modificações no seu funcionamento a partir das configurações que o usuário (no caso você) pode realizar. Você deve editar o arquivo que se encontra no menu **Preferences** > **Settings - User**.

Edite o arquivo para que fique parecido com o código abaixo. Você pode procurar na internet por mais comandos que modifiquem o comportamento do Sublime Text.

```

1 {
2   ...
3   "font_size": 16,
4   "highlight_line": true,
5   "font_face": "Ubuntu Mono"
6   ...
7 }

```

2.7 Configurações e Plugins

O Sublime Text se tornou famoso por conta da possibilidade de adicionarmos à ele, diversas outras funcionalidades por meio de *plugins*. Existem *plugins* que auxiliam no desenvolvimento de HTML, CSS e em diversas outras linguagens.

Para tirar o maior proveito dos *plugins*, vamos adicionar um gerenciador de *plugins* ao nosso Sublime Text.

Vamos acessar o site <<http://packagecontrol.io/installation>> através do navegador Firefox. Copie toda o código representado na figura ??.

Agora devemos ir até o Sublime Text e seguir o caminho **View** > **Show console...**. Uma pequena seção irá aparecer na parte debaixo da tela principal do programa. Usando o botão direito do *mouse*, clique no campo como mostrado na figura 2.7.1b escolha a opção *paste* (colar em inglês) e pressione **↵** (Enter). Após a finalização da instalação, percorra o menu **View** > **Hide console...** para esconder o **terminal**.

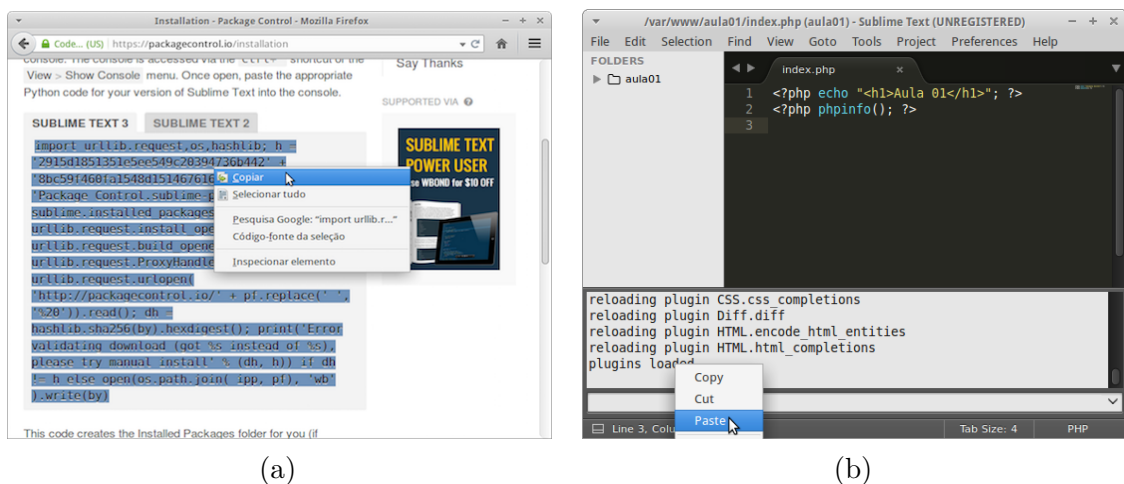


Figura 2.7.1 – (a) Código que deve ser copiado, em seguida, colado no Sublime Text.; (b) Opção a ser escolhida para colar código copiado anteriormente.

2.7.1 Plugin php-snippets

Como teste, instalaremos o pacote **php-snippets**. Esse pacote nada mais é do que um conjunto de códigos pré-prontos. Inicialmente não nos terá muita serventia, mas com

o passar das aulas será bem útil.

Para instalar um *plugin* precisamos executar alguns comandos no painel de comandos. Com o Sublime Text aberto, digite a combinação de teclas **Ctrl** + **⇧** + **P** e digite *package install*. Esse termo ativará o comando *Package Control: Install Package*. Ao pressionar **↵** (Enter) o painel de comandos listará todos os pacotes disponíveis para instalação. Digite o termo *php-snippets*, ativando assim, o pacote de nome *php-snippets*. Veja a figura

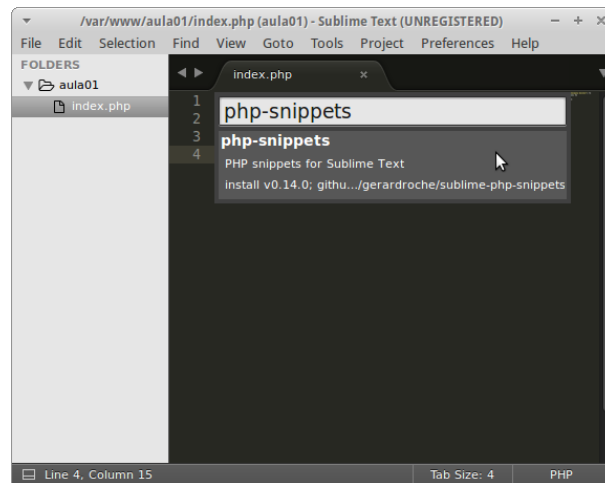


Figura 2.7.2 – Instalação do *plugin* php-snippets.

2.8 Desafio!

O desafio deste capítulo é você configurar o Sublime Text de acordo com o seu gosto. Modifique os seguintes itens:

1. Tamanho e tipo da fonte.
2. Cor do editor (**Preferences** > **Color Scheme** > <opção>)
3. Instalar o *plugin* phpdoc (<<https://packagecontrol.io/packages/PhpDoc>>).

3 Variáveis em PHP

Ao final deste capítulo, o aluno terá as seguintes competências:

1. Entender o que são variáveis e como o PHP trabalha com elas; e
2. Trabalhar com os diversos tipos de variáveis.

3.1 Variáveis

Variáveis são identificadores criados para guardar valores por determinado tempo. No PHP elas são declaradas, inicializadas e armazenadas na memória RAM do servidor *web*. Esse é um dos motivos pelo qual os servidores precisam de grandes quantidades de memória.

Quando desenvolvemos um *site* e o disponibilizamos na *internet*. É nosso desejo que ele seja acessado pela maior quantidade de pessoas possível. Portanto, imagine um servidor com mais de 20 mil acessos simultâneos (mais de 20 mil pessoas visualizando o *site* no mesmo momento). Nesse processo são criadas variáveis diferentes para cada usuário, logo, isso faz com que o processamento que o servidor faz se intensifique. Por conta disso, o servidor deve ser um computador com bastante memória RAM.

Uma variável é inicializada no momento em que é feita a primeira atribuição. O tipo da variável será definido de acordo com o valor atribuído. Esse é um fator importante PHP, pois uma mesma variável pode ser de um tipo e pode assumir no decorrer do código outro valor de tipo diferente.

Para criar uma variável em PHP, precisamos atribuir-lhe um nome de identificação, sempre precedido pelo caractere cifrão (\$). Observe um exemplo:

```
1  <?php
2      // Declarando e atribuindo valores as variaveis
3      $nome = "Joaquim";
4      $sobrenome = "Silva";
5
6      // Imprimindo na tela os valores
7      echo "Nome: " . $nome
8      ...
9  ?>
```

Modifique o código acima para imprimir na tela o valor da variável de nome `$sobrenome`.

Observação! Podem acontecer erros na exibição das mensagens por conta das

codificações de acentuação. Caso isso aconteça, mude a codificação do seu navegador ou utilize as metas de codificação. Para mudar a codificação do Firefox aperte a tecla **Alt** para exibir a barra de menus e em seguida percorra o caminho **Exibir** » **Codificação** » **Unicode**.

Nomes de variáveis devem ser significativas e transmitir a ideia de seu conteúdo dentro do contexto no qual está inserido. Utilize preferencialmente palavras em minúsculo (separadas pelo caractere `_`) ou somente as primeiras letras em maiúsculo quando você tiver duas ou mais palavras. Veja o exemplo abaixo.

```
1 <?php
2 // Exemplos de nomenclaturas
3 $codigo_cliente = "AB123";
4 $codigoCliente = "AB123";
5 ?>
```

Dicas!

- Nunca inicie a nomenclatura de variáveis com números. Ex: `$1nota`;
- Nunca utilize espaço em branco no meio do identificados da variável. Ex: `$nome um`;
- Nunca utilize caracteres especiais (`! @ # & * | [] { } \ ^` entre outros) na nomenclatura das variáveis.
- Evite criar variáveis com nomes grandes demais em virtude da clareza do código-fonte.
- Com exceção de nomes de classes e funções, o PHP é *case sensitive*, ou seja, é sensível a letras maiúsculas e minúsculas. Tome cuidado ao declarar variáveis. Por exemplo: a variável `$codigo` é diferente da variável `$Codigo`.

3.2 Tipos de Variáveis

O PHP tem uma grande flexibilidade na hora de operar com variáveis. Quando definimos uma variável juntamente com seu valor, o PHP atribui um **tipo** à essa variável. Isso permite que o programador não se preocupe muito na definição de tipos de variáveis (pois é feito de forma automática). Porém, devemos ter cuidado com atribuições de valores, evitando erros nos cálculos.

3.2.1 Tipo Inteiro

São os números que pertencem ao conjunto dos números inteiros, abrangendo valores negativos e positivos. No PHP os valores máximos e mínimos do tipo inteiro depende da plataforma, ou seja, do sistema operacional que o PHP está sendo executado.

No geral, esse número pode ter um valor mínimo e máximo por volta de 2 bilhões (positivo e negativo).

Veja exemplos de códigos de atribuição de variáveis do tipo inteiro.

```
1 <?php
2 // Tipo inteiro (negativo e positivo)
3 $nota = 9;
4 $temperatura = -5;
5 ?>
```

3.2.2 Tipo Ponto flutuante

Os números de ponto flutuante (*float* e *doubles*) são números com casas decimais. No Brasil, usamos a vírgula (,) para escrever números decimais. Nas linguagens de programação em geral, utilizamos o ponto (.). Veja exemplos de códigos abaixo.

```
1 <?php
2 // Tipo ponto flutuante (negativo e positivo)
3 $preco = 12.80;
4 $temperatura = -8.5;
5 ?>
```

3.2.3 Tipo *Booleano*

Um *booleano* expressa um valor lógico que pode ser **verdadeiro** ou **falso**. Para especificar um valor *booleano* utilize a palavra chave **true** para verdadeiro e **false** para falso. No exemplo a seguir, declaramos uma variável do tipo *booleano* `$exibir_nome`, cujo conteúdo é **true**.

Esse tipo de variável é bastante útil quando estamos trabalhando com estruturas de decisão ou estruturas de repetição com condição de parada. Esse assunto será abordado posteriormente.

```
1 <?php
2 // Tipo ponto flutuante (negativo e positivo)
3 $exibir_nome = true;
4 // Imprime o valor da variavel
5 // $exibir_nome no navegador
6 print($exibir_nome);
7 ?>
```

Também podemos atribuir outros valores *booleanos* para representação de valores **falso**, veja o exemplo abaixo.

```
1 <?php
2 // Outras maneiras de declarar um valor booleano falso
```

```
3  $valor_falso_exemplo_1 = 0;
4  $valor_falso_exemplo_2 = "";
5  $valor_falso_exemplo_3 = NULL;
6  $valor_falso_exemplo_4 = 0.0;
7  print($valor_falso_exemplo_1);
8  print($valor_falso_exemplo_2);
9  print($valor_falso_exemplo_3);
10 print($valor_falso_exemplo_4);
11 ?>
```

3.2.4 Tipo *String*

Uma *string* é uma cadeia de caracteres alfanuméricos, ou seja, podemos tanto usar números quanto letras. Porém, é necessário declará-las utilizando aspas simples (') ou aspas duplas ("). Veja o exemplo abaixo.

```
1  <?php
2  // Outras maneiras de declarar um valor booleano falso
3  $id = "ab13c";
4  $nome = "Joaquim";
5  $sobrenome = "Silva";
6  print($id);
7  print($nome);
8  print($sobrenome);
9  ?>
```

Caso seja necessário escrever na *string* os caracteres \$, \, ' ou "", devemos utilizar a tabela abaixo.

Tabela 1 – Caracteres especiais em *strings*.

Sintaxe	Significado
\$\$	O símbolo \$
\\	O símbolo \
\'	Aspas simples
\"	Aspas duplas

3.2.5 Tipo *Array*

Array é uma lista de valores armazenados na memória. Os elementos de um *array* podem ser de tipos diferentes (inteiro, decimal, *string* etc). Um *array* pode crescer dinamicamente com a adição de novos itens. O capítulo 6 explica como manipular esse tipo de estrutura.

3.2.6 Tipo *NULL*

Ao atribuímos valor do tipo *NULL* (nulo) a uma variável, estamos determinando que a variável não possui valor número ou alfanumérico e que seu valor é nulo. Veja o exemplo.

```
1 <?php
2     // Tipo NULL
3     $nota = NULL;
4 ?>
```

3.3 Constantes

3.3.1 Constantes pré-definidas

O PHP possui algumas constantes pré-definidas, como por exemplo, constantes que indicam a versão do PHP, o sistema operacional em que o PHP está sendo executado, o arquivo em execução e diversas outras informações. Para ter conhecimento de todas as constantes pré-definidas, podemos utilizar a função `phpinfo()`. Ela exibe uma tabela contendo todas as constantes pré-definidas.

3.3.2 Definindo constantes

O programador também pode definir constantes para serem utilizadas no projeto. Para isso, utilizamos a função `define`. Uma vez que o valor tenha sido definido, não poderá ser alterado. A constante só pode conter valores inteiro, ponto flutuantes ou *string*. Não podemos iniciar uma constante com uma variável do tipo *array* por exemplo.

Um exemplo de utilização da função `define` pode ser visto a seguir.

```
1 <?php
2     // Sintaxe da funcao define()
3     define("VALOR_DA_CONSTANTE", "VALOR");
4 ?>

1 <?php
2     // Sintaxe da funcao define()
3     define("HOST_PRINCIPAL", "192.168.2.100");
4 ?>
```

O nome de uma constante tem as mesmas regras de nomenclatura de qualquer variável. Ele é válido quando começa com uma letra ou *underline* (`_`) seguido por qualquer número de letras, números ou sublinhados.

A seguir listamos algumas diferenças entre constantes e variáveis.

1. Constantes podem ser definidas e acessadas em qualquer lugar do código são indiferentes ao escopo;

2. Constantes só podem conter valores numéricos ou alfanuméricos;
3. Constantes não podem ter um sinal de cifrão (\$) no início;
4. Constantes só podem ser definidas utilizando a função `define` e não por atribuição; e
5. Constantes não podem ser redefinidas ou eliminadas após sua criação.

3.4 Conversão de variáveis

O PHP permite que o programador converta um tipo de variável em outro tipo de variável. As conversões mais comuns são as do tipo inteiro para decimal e do tipo decimal para inteiro. Outra conversão bastante utilizada é de *string* para decimal ou inteiro.

A instrução utilizada para as conversões é denominada de `typecast`. A ação é denominada de *typecasting*, ou seja, o ato de transformar uma variável de um tipo em outro tipo. A tabela a seguir mostra os *typecast* que iremos utilizar.

Tabela 2 – Instruções de transformação de variáveis.

<i>typecasting</i>	Descrição
(int) ou (integer)	Converte em inteiro
(float) ou (double)	Converte em ponto flutuante
(string)	Converte em <i>string</i>

Veja alguns exemplos de códigos de conversões de tipos de variáveis.

```

1 <?php
2 // Conversao do tipo inteiro para ponto flutuante
3 $temperatura = (float)36;
4 echo $temperatura; // resultado: 36
5
6 // Conversao do tipo ponto flutuante para inteiro
7 $parte_inteira = (int)(48.56 + 32);
8 echo $parte_inteira; // resultado: 80
9
10 // Conversao do tipo ponto flutuante para string
11 $preco_total = (string)(123.42);
12 echo $preco_total; //resultado: 123.42
13 ?>
```

No primeiro e último exemplo de código não notamos diferença ao imprimirmos no navegador o valor das variáveis `$temperatura` e `$preco_total` respectivamente. Mas se utilizarmos a função `gettype` teremos o tipo de variável correspondente. Veja o código abaixo.

```

1 <?php
2 ...
```

```

3  echo gettype($temperatura); //resultado: double
4  echo gettype($preco_total); //resultado: string
5  ?>

```

3.5 Exercícios

Q. 01 Qual a principal finalidade de uma variável?

Q. 02 Cite 3 normas ao se nomear uma variável.

Q. 03 Das variáveis listadas abaixo, quais possuem nomenclaturas válidas?

- | | |
|--------------|--------------|
| 1. \$a__b | 8. \$palavra |
| 2. \$a_1_ | 9. \$tele# |
| 3. \$_início | 10. \$123 |
| 4. \$@nome | 11. \$__=__ |
| 5. \$@val_! | 12. \$VALOR |
| 6. \$-nome | 13. \$all |
| 7. \$#valor | |

Q. 04 Crie dez variáveis atribuindo valores diversos (numéricos e *string*), em seguida utilize o comando `echo` para imprimir na tela do navegador.

```

1  <?php
2      $nome = "Joaquim Silva";
3      echo $nome;
4      ...
5  ?>

```

Q. 05 Quais os tipos de variáveis que o PHP trabalha?

Q. 06 Como podemos diferenciar um tipo de variável de outro?

Q. 07 Faça a ligação das variáveis com os respectivos tipos de variável.

- | | |
|-------------------|--------------------------|
| \$var = 10 | () ponto flutuante |
| \$var = "palavra" | () tipo null |
| \$var = 10.22 | () tipo <i>string</i> |
| \$var = true | () tipo numérico |
| \$var = null | () tipo <i>booleano</i> |
| \$var = array() | () tipo <i>array</i> |

Q. 08 Qual a principal finalidade de uma constante e como elas são definidas no PHP?

Q. 09 Crie uma constante e apresente o resultado na tela.

Q. 10 Em que momentos precisamos converter uma variável de um tipo para outro tipo?

Q. 11 Crie conversões com as variáveis `$var1 = 15.20`, `$var2 = 10` e `$var3 = '25'`. Apresente na tela o tipo da variável anterior e atual. Utilize o comando `gettype`.

1. Converta `$var1` em *string*.
2. Converta `$var2` em número real.
3. Converta `$var1` em inteiro.
4. Converta `$var3` em ponto flutuante.

3.6 Desafio!

O desafio deste capítulo é criar um arquivo `.php` declarando variáveis de todos os tipos estudados (inteiro, ponto flutuante e *string*). Apresentá-los no navegador e por fim, converter os valores das variáveis do tipo inteiro e ponto flutuante para *string*. Esse arquivo deve ser criado no diretório `/var/www/aula01`.

4 Operadores em PHP

Ao final deste capítulo, o aluno terá as seguintes competências:

1. Trabalhar com operadores aritméticos;
2. Trabalhar com operadores relacionais; e
3. Trabalhar com operadores de *string*.

4.1 Operadores

Os operadores têm um papel importante dentro de qualquer linguagem de programação. É através deles que podemos realizar diversas operações em um programa. Existem operadores para atribuição, operadores aritméticos, operadores relacionais ou lógicos, e por fim, operadores de *string*.

No PHP, os operadores são utilizados constantemente e nesse capítulo iremos aprender a trabalhar com a maioria deles.

4.2 Operador de atribuição

O operador básico de atribuição é o caractere “=” (igual). Com ele podemos atribuir valores as variáveis como foi visto em exemplos anteriores. Isto quer dizer que o operando da esquerda recebe o valor da expressão da direita, ou seja, a variável da esquerda contém o valor da direita do símbolo igual “=”. Observe o exemplo abaixo:

```
1 <?php
2 // Atribuicao de valores
3 $id = "ab13c";
4 $peso = 66.40;
5 ?>
```

4.3 Operadores de *strings*

Os operadores de *strings* são utilizados para manipular o conteúdo de uma *string*. O PHP disponibiliza, basicamente, dois operadores de *strings*. O primeiro é o operador de concatenação (.) - ele retorna a concatenação das variáveis envolvidas. O segundo operador, é o operador de atribuição e concatenação (.=). Ele acrescenta à variável do lado direito na variável do lado esquerdo do operador. Verifique os exemplos abaixo.

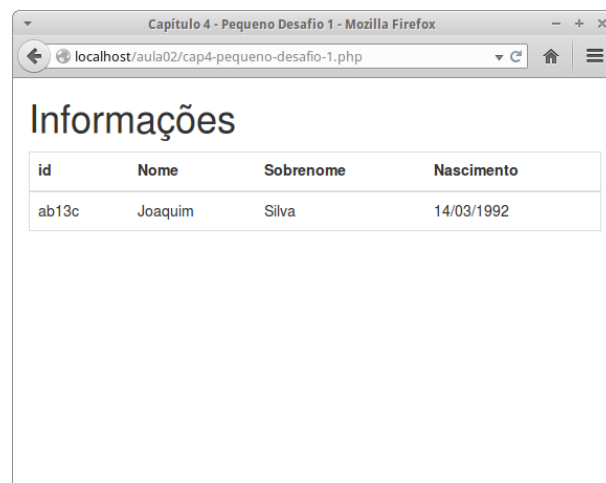
```
1 <?php
2 // Outras maneiras de declarar um valor booleano falso
```

```
3  $id = "ab13c";
4  $nome = "Joaquim ";
5  $sobrenome = "Silva ";
6
7  $nome_sobrenome = $nome . $sobrenome;
8  echo $nome_sobrenome; // resultado: Joaquim Silva
9  echo "<br/>";
10
11 $informacao = $nome_sobrenome .= $id;
12 echo $informacao; // resultado: Joaquim Silva ab13c
13 ?>
```

No exemplo acima, pode-se observar, na atribuição da variável `$informacao` que, temos uma inicialização e atribuição de concatenação em uma mesma linha. Isso é possível no PHP, por mais que seja mais otimizado (mais rápido de ser processado), o código se torna menos legível (mais difícil de ser entendido).

Pequeno desafio!

Utilize o conhecimento adquirido nas aulas passadas para exibir, no navegador, as informações separadas em células de uma tabela. A imagem a seguir deve ser o resultado apresentado.



The screenshot shows a web browser window titled "Capítulo 4 - Pequeno Desafio 1 - Mozilla Firefox". The address bar shows "localhost/aula02/cap4-pequeno-desafio-1.php". The page content displays a table titled "Informações". The table has four columns: "id", "Nome", "Sobrenome", and "Nascimento". The data row shows "ab13c", "Joaquim", "Silva", and "14/03/1992".

id	Nome	Sobrenome	Nascimento
ab13c	Joaquim	Silva	14/03/1992

Figura 4.3.1 – Resultado do primeiro pequeno desafio.

4.4 Operadores Aritméticos

Os operadores aritméticos são utilizados para realizar cálculos matemáticos básicos, tais como: soma, subtração, divisão e multiplicação. Os símbolos mais utilizados são descritos na tabela abaixo.

Tabela 3 – Símbolos matemáticos.

Operação	Operador	Exemplo	Resposta
Adição	+	\$a = 3 + 5	8
Subtração	-	\$a = 6 - 2	4
Multiplicação	*	\$a = 2 * 5	10
Divisão	/	\$a = 15 / 3	5
Módulo (resto da divisão)	%	\$a = 9 % 2	1
Negação	-	\$a = -3	-3

Na tabela acima fizemos operações básicas sem utilizar parênteses. O uso de parênteses segue o mesmo princípio da matemática. Ele serve para dar prioridade a determinado cálculo.

4.5 Operadores Combinados

No PHP é possível combinar os dois operadores visto acima (de atribuição e aritméticos). A partir deles é possível programar de forma mais ágil. Observe o código a seguir.

```

1 <?php
2 // Atribuicao de valores
3 // (operador combinado)
4 $peso = 66.40;
5 $peso += 4;
6 echo $peso;
7 ?>
```

```

1 <?php
2 // Atribuicao de valores
3 // (operador de atribuicao)
4 $peso = 66.40;
5 $peso = $peso + 4;
6 echo $peso;
7 ?>
```

No código acima do lado esquerdo, declaramos a variável `$peso` com um valor inicial, em seguida utilizamos o operador combinado `+=` para incrementar o valor da variável `$peso`. Podemos obter o mesmo resultado com o código acima do lado direito. Escrevemos menos no código do lado esquerdo, por isso, os programadores o utilizam bastante.

Já no código abaixo, utilizamos o operador combinado `.=`. Ele concatena as *strings*.

```

1 <?php
2 // Atribuicao de valores
3 // (operador combinado)
4 $saudacao = "Bom ";
5 $saudacao .= "dia!";
6 echo $saudacao;
7 ?>
```

Nesse exemplo utilizamos o operador de atribuição básico.

```

1 <?php
2 // Atribuicao de valores
3 // (operador de atribuicao)
4 $saudacao = "Bom ";
5 $saudacao = $saudacao + "dia!";
6 echo $saudacao;
7 ?>

```

A tabela abaixo lista os principais operadores de atribuição.

Tabela 4 – Operadores de atribuição combinados.

Operadores	Descrição
=	Atribuição simples
+=	Soma, em seguida, atribui
-=	Subtrai, em seguida, atribui
*=	Multiplica, em seguida, atribui
/=	Divide, em seguida, atribui
%=	Tira o módulo, em seguida, atribui
.=	Concatena, em seguida, atribui

4.6 Operadores de decremento e incremento

Os operadores exemplificados nessa seção são usados para **somar** ou **subtrair** o valor 1 (um) a variável. Esse cálculo pode ser feito antes ou depois da execução de determinada variável. A tabela abaixo mostra tais operadores.

Tabela 5 – Operadores de incremento e decremento.

Operadores	Descrição
\$b = ++\$a	Incrementa o valor de \$a, e atribui à \$b (pré-incremento)
\$b = \$a++	Atribui à \$b, em seguida, incrementa o valor de \$a (pós-incremento)
\$b = --\$a	Decrementa o valor de \$a, e atribui à \$b (pré-decremento)
\$b = \$a--	Atribui à \$b, em seguida, decrementa o valor de \$a (pós-decremento)

O exemplo abaixo mostra a comparação na forma da escrita. Os resultados alcançados serão os mesmos.

```

1 <?php
2 // Operadores de incremento e decremento
3 $preco = 12.40;
4 ++$preco;
5 echo $preco; // resultado: 13.40
6

```



```

7   $preco = 10.20;
8   $preco_sem_mudanca = $preco++;
9   echo "preco sem mudanca: " . $preco_sem_mudanca; // resultado: 10.20
10  echo "<br />";
11  echo "preco: " . $preco; // resultado: 11.20
12  ?>

```

4.7 Operadores relacionais

Os operadores relacionais são utilizados para realizar comparações entre valores (variáveis) ou expressões. Essa comparação sempre resulta em um valor do tipo *booleano*, ou seja, verdadeiro (**true**) ou falso (**false**). Na tabela a seguir, listamos os operadores que iremos trabalhar.

Tabela 6 – Operadores relacionais.

Operadores	Nome	Descrição
==	Igual	Resulta em true se as expressões forem iguais
===	Idêntico	Resulta em true se as expressões forem iguais e do mesmo tipo
!= ou <>	Diferente	Resulta em true se as expressões forem diferentes
<	Menor que	Resulta em true se a primeira expressão for menor que a segunda expressão
>	Maior que	Resulta em true se a primeira expressão for maior que a segunda expressão
<=	Menor ou igual que	Resulta em true se a primeira expressão for menor ou igual a segunda expressão
>=	Maior ou igual que	Resulta em true se a primeira expressão for maior ou igual a segunda expressão

O exemplo de código abaixo descreve o uso de alguns operadores relacionais.

```

1  <?php
2  // Operadores de incremento e decremento
3  $num1 = 5;
4  $num2 = 7;
5
6  $teste = $num1 > $num2; // resultado: false
7  // para imprimir um valor booleano precisamos
8  // transformar em inteiro, exemplo:
9  echo (int)$teste; // resultado: 0
10
11  echo "<br />";
12

```

```
13     $num1 = 3;
14     $num2 = 3;
15     $teste = $num1 == $num2; // resultado: true
16     echo (int)$teste; // resultado: 1
17
18 ?>
```

4.7.1 Operador condicional ternário

É importante explicar o operador condicional ternário (símbolo “?”) para termos um exemplo prático dos operadores relacionais. A sintaxe do operador condicional é exemplificada abaixo:

```
1  <?php
2  // Operador condicional ternario
3  // expressao1 sera um teste
4  // expressao2 sera atribuido a $var caso expressao1 seja verdadeiro
5  // expressao3 sera atribuido a $var caso expressao1 seja falsa
6  $var = expressao1 ? expressao2 : expressao3
7  ?>
```

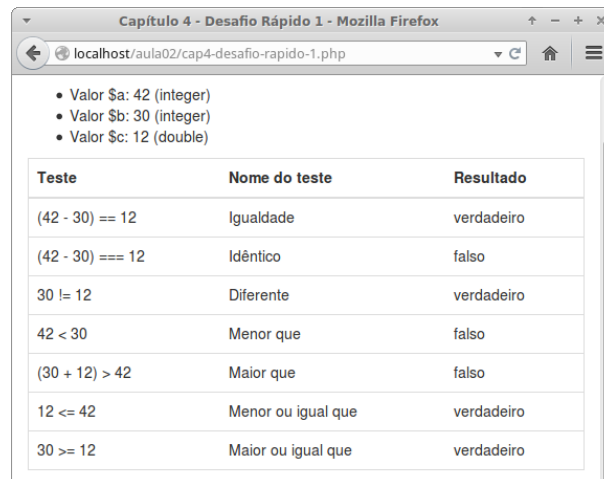
A expressão 1 é sempre um teste a ser realizado com os operadores relacionais. A expressão 2 e expressão 3 são valores que serão atribuídos a variável `$var`. Vamos ver um exemplo:

```
1  <?php
2  // Operador condicional ternario
3  $num1 = 12;
4  $num2 = 12.0;
5  $var = $num1 == $num2 ? "SIM" : "NAO";
6  echo $var; // resultado: SIM
7  ?>
```

Qual será o resultado se mudarmos a expressão 1 (teste) para `$num1 === $num2`?

Desafio rápido!

Faça um pequeno programa que utilize todos os operadores relacionais descritos acima. É necessário criar exemplos que os resultados sejam `true` e `false` para cada um dos casos. Utilize também o operador condicional ternário. A saída no navegador deve ser parecida com a imagem abaixo.



Teste	Nome do teste	Resultado
$(42 - 30) == 12$	Igualdade	verdadeiro
$(42 - 30) === 12$	Idêntico	falso
$30 != 12$	Diferente	verdadeiro
$42 < 30$	Menor que	falso
$(30 + 12) > 42$	Maior que	falso
$12 \leq 42$	Menor ou igual que	verdadeiro
$30 \geq 12$	Maior ou igual que	verdadeiro

Figura 4.7.1 – Resultado do primeiro desafio rápido.

4.8 Operadores Lógicos

Operadores lógicos, também conhecidos como operadores *booleanos*, são utilizados para avaliar expressões lógicas, ou seja, expressões que resultem em valores *booleanos* (verdadeiro ou falso). A tabela a seguir lista cada um desses operadores e sua respectiva função.

Tabela 7 – Operadores lógicos.

Operadores	Função
NÃO (NOT)	Negação
E (AND)	Conjunção
OU (OR)	Disjunção

Com esses operadores, podemos construir a tabela verdade de determinada variável *booleana*. As próximas seções explicam os operadores lógicos NOT, AND e OR.

4.8.1 Operador NOT

Para isso, vamos utilizar os dois valores que a variável pode assumir (Verdadeiro ou Falso) e o operador lógico NOT listado na tabela 7. Veja a tabela abaixo, a variável `$esta_chovendo` pode assumir dois valores, verdadeiro ou falso.

Tabela 8 – Operação de negação.

<code>\$esta_chovendo</code>	NOT <code>\$esta_chovendo</code>
Falso	Verdadeiro
Verdadeiro	Falso

Ou seja, considerando que a variável `$esta_chovendo` pode ser traduzida para a frase: “Hoje está chovendo!”, sua negação seria: “Hoje **não** está chovendo”.

4.8.2 Operador AND

O operador **AND** funciona com base em uma conjunção. Algo só será verdadeiro se os termos envolvidos na expressão sejam verdadeiros. Nosso próximo exemplo leva em consideração a situação de um casamento. Então criamos duas variáveis: `$noiva_presente` e `$noivo_presente`. Se apenas o noivo estiver presente, ou seja, `$noivo_presente = True` e `$noiva_presente = False` o casamento não acontece. Observe na tabela verdade abaixo.

Tabela 9 – Operação de conjunção.

<code>\$noivo_presente</code>	<code>\$noiva_presente</code>	<code>\$noivo_presente AND \$noiva_presente</code>
Falso	Falso	Falso
Falso	Verdadeiro	Falso
Verdadeiro	Falso	Falso
Verdadeiro	Verdadeiro	Verdadeiro

4.8.3 Operador OR

O operador **OR** funciona com base em uma disjunção. Algo será verdadeiro assim que algum termo envolvido na expressão seja verdadeiro. Por exemplo, Joaquim, aluno do programa e-Jovem, vai fazer uma prova de concurso. Para realizá-la, é obrigatório apresentar documento de identidade (variável `$tem_identidade`) **ou** o título de eleitor (variável `$tem_titulo_eleitor`). Joaquim só vai conseguir fazer a prova caso apresente um documento **ou** outro. Dessa maneira, observe a tabela verdade do operador **OR**.

Tabela 10 – Operação de disjunção.

<code>\$tem_identidade</code>	<code>\$tem_titulo_eleitor</code>	<code>\$tem_identidade OR \$tem_titulo_eleitor</code>
Falso	Falso	Falso
Falso	Verdadeiro	Verdadeiro
Verdadeiro	Falso	Verdadeiro
Verdadeiro	Verdadeiro	Verdadeiro

Verifique no código abaixo o uso dos operadores lógicos no PHP.

```

1 <?php
2 // Operadores logicos
3 // Lembre que 1 significa VERDADEIRO
4 // Lembre que 0 significa FALSO
5 $esta_chovendo = false;
6 echo "esta chovendo? " . (int)!$esta_chovendo; // resultado: 1

```

```
7     echo "<br />";
8
9     $noivo_presente = true;
10    $noiva_presente = true;
11
12    echo "vai ter casamento? " . (int)($noivo_presente and $noiva_presente);
13    echo "<br />";
14
15    $tem_identidade = false;
16    $tem_titulo_eleitor = true;
17
18    echo "vai fazer prova? " . (int)($tem_identidade or $tem_titulo_eleitor);
19    echo "<br />";
20    ?>
```

4.9 Precedência de operadores

Conhecer a precedência dos operadores é importante para obtermos o correto valor da expressão que queremos calcular. A precedência dos operadores indicam quais termos serão calculados primeiro. Imagine que queremos calcular a expressão $5 + 2 + 6 / 3$.

O operador $/$ tem maior precedência em relação ao operador $+$. Por conta disso, devemos primeiro calcular os termos que estão sendo divididos, no nosso exemplo, o termo $6 / 3$, em seguida podemos fazer o somatório. O resultado final será 9.

Caso a intenção seja primeiro realizar a operação com o operador $+$, para só então calcular a expressão com o operador $/$, temos que utilizar parênteses. Veja no exemplo abaixo:

```
1  <?php
2      // Precedencia de operadores
3      $res = 5 + 2 + 6 / 3; // resultado: 9
4      $res = (5 + 2 + 6) / 3; // resultado: 4.33
5  ?>
```

É importante lembrar: o PHP executará todas as operações que estiverem entre parênteses. Se dentro dos parênteses houver diversas operações, a precedência dos operadores será utilizada para definir a ordem de cálculo. Após calcular os termos dentro dos parênteses o PHP volta a calcular os termos que estão fora dos parênteses.

4.10 Exercícios

Q. 01 Qual a finalidade dos operadores de *strings*?

- Q. 02 Quais são os operadores de incremento e decremento? Cite alguns exemplos.
- Q. 03 Qual a finalidade do operador aritmético % (módulo)?
- Q. 04 Cite os operadores relacionais. Mostre 1 exemplo de cada operador.
- Q. 05 Quais são os operadores lógicos/*booleanos*?
- Q. 06 Quais são os operadores de atribuição? Mostre 1 exemplo de cada operador.
- Q. 07 Qual a sintaxe de uso do operador ternário? Cite um exemplo de uso.
- Q. 08 Quais são os operadores utilizados no código abaixo? Qual o resultado final?

```
1 <?php
2     $a = 10;
3     $b = 12.5;
4     $c = $a+$b;
5
6     print($a > $b? "verdadeiro": "falso");
7     print($c >= $b? "verdadeiro" : "falso");
8 ?>
```

- Q. 09 Observe o código abaixo e identifique qual operação é realizada primeiro, depois em segundo e assim por diante.

```
1 <?php
2     $a = 8 * 5 - 3 + 4 / 2 + 19 % 5 / 2 + 1;
3 ?>
```

- Q. 10 Utilize o operador de *string* "." para montar a frase abaixo. Utilize o comando `echo`.

```
1 <?php
2     $s1 = "de";
3     $s2 = "eh um";
4     $s3 = "comunicacao";
5     $s4 = "a";
6     $s5 = "internet";
7     $s6 = "meio";
8 ?>
```

- Q. 11 Observe o código abaixo e diga qual o resultado *booleano* final. Justifique sua resposta explicando o que acontece no código.

```
1 <?php
2     $ex1 = 12.0 < 11.2;
3     $ex2 = 10 * 2 - 3 > 19 % 3 + 10;
4     $ex3 = 10;
5     print( ($ex1 $ex3 = 10 && $ex2) ? "verdadeiro" : "falso");
6 ?>
```

4.11 Desafio!

O seguinte código é uma página inicial de um sistema de notas simplificado. O programador deve inserir no próprio código as 3 notas do aluno. Se a média aritmética das notas for maior que 6, o aluno está aprovado, caso contrário o aluno se encontra de recuperação.

Esse código tem vários problemas e não está executando corretamente, identifique os erros, corrija e apresente ao professor.

```
1  <?php
2      $saudacao = "Bom dia!";
3      $aluno = "Joaquim Silva";
4      $nascimento = 29/06/1992;
5
6      echo "-----" . "<br />";
7      echo $saudacao . "<br />";
8      echo "-----" . "<br />";
9
10     $nota1 = 6.1;
11     $nota2 = "4.0"
12     $nota3 = 9;
13
14     $media = $nota1 + $nota2 + $nota3 / 3;
15     $situacao = $media > 6 ? "APROVADO" : "RECUPERACAO";
16
17     $info = "Aluno: " . $aluno . "<br />";
18     echo $info;
19     echo "Situacao: " . $situacao;
20  ?>
```


5 Estruturas de controle e repetição

Ao final deste capítulo, o aluno terá as seguintes competências:

1. Trabalhar com estruturas de controle; e
2. Trabalhar com estruturas de repetição.

As estruturas que veremos a seguir são comuns para várias linguagens de programação. Entretanto é necessário que a apostila descreva a sintaxe dessas estruturas, resumindo seu funcionamento.

É necessário ainda, entender o conceito de bloco. Um bloco consiste de vários comandos agrupados com o objetivo de relacioná-los com determinado comando ou função. Podemos (e devemos) usar blocos de comandos nas instruções que serão vistas nesse capítulo (`if`, `for`, `while`, `switch`). Os blocos de comandos devem ser utilizados para permitir que um conjunto de instruções façam parte do mesmo contexto desejado.

Blocos em PHP são delimitados pelos caracteres `{` e `}`.

5.1 Comando `if`

Essa estrutura condicional está entre as mais usadas na programação. Sua finalidade é induzir um desvio condicional, ou seja, um desvio na execução natural do programa. Caso a condição (veja sintaxe abaixo) seja verdadeira, então serão executadas a instruções do bloco de comando. Caso a condição não seja satisfeita, o bloco de comando será simplesmente ignorado. Veja a sintaxe e em seguida um exemplo real.

```
1  <?php
2      // Sintaxe estrutura condicional if
3      if (condicao) {
4          # codigo...
5      }
6
7      // Exemplo real
8      $media = 7.5;
9      if ($media >= 7) {
10         echo "Aluno aprovado!";
11     }
12  ?>
```

Se o teste realizado no `if` for falso, ou seja, a expressão `$media >= 7` der um resultado *booleano* falso, o bloco de comandos não será executado.

5.1.1 Comando `if..else`

Um complemento do comando `if` é a adição da palavra chave `else`. Com esse termo é possível tratar também as opções em que o teste é falso. Veja como fica a sintaxe da instrução completa.

```
1 <?php
2 // Sintaxe estrutura condicional if
3 if (condicao) {
4     # codigo...
5 } else {
6     # outro codigo...
7 }
8
9 // Exemplo real
10 $media = 5.5;
11 if ($media >= 7) {
12     echo "Aluno aprovado!";
13 } else {
14     echo "Aluno em recuperacao!";
15 }
16 ?>
```

Como pode ser visto, o teste realizado `$media >= 7` gerou um valor *booleano* falso, resultando na execução do código relacionado ao bloco de comandos `else`, fazendo com que a mensagem exibida na tela do navegador seja “Aluno em recuperação”.

O programador tem a possibilidade de adicionar quantos comandos `if..else` forem necessários. Esses códigos são chamados de “ifs encadeados”. Para facilitar, o PHP criou a palavra chave `elseif` (tudo junto mesmo). Veja a sintaxe de uso.

```
1 <?php
2 // Sintaxe estrutura condicional if encadeado
3 if (condicao) {
4     # codigo...
5 } elseif (condicao 2) {
6     # codigo 2...
7 } else {
8     # outro codigo...
9 }
10
11 // Exemplo real
12 $media = 2.5;
13 if ($media >= 7) {
14     echo "Aluno aprovado!";
15 } elseif ($media < 3) {
16     echo "Aluno reprovado!";
17 } else {
```

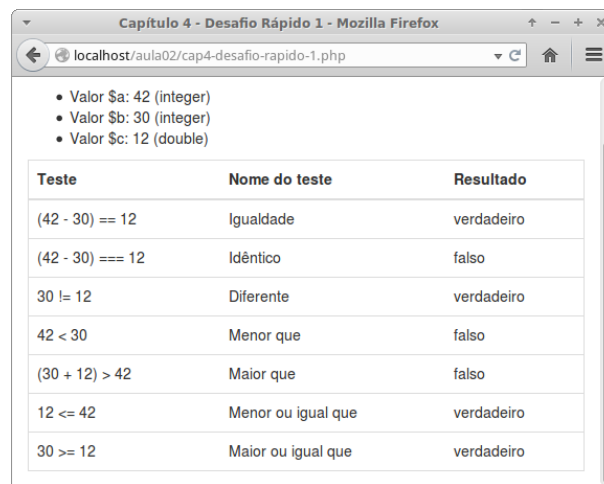
```

18     echo "Aluno em recuperacao!";
19 }
20 ?>

```

Desafio rápido!

Faça um pequeno programa que utilize a idade de um nadador, a partir dela o programa deve indicar em qual categoria o nadador irá concorrer no próximo campeonato. Crianças menores de 5 anos não podem competir. Veja como deve ser a saída do seu programa.



Teste	Nome do teste	Resultado
(42 - 30) == 12	Igualdade	verdadeiro
(42 - 30) === 12	Idêntico	falso
30 != 12	Diferente	verdadeiro
42 < 30	Menor que	falso
(30 + 12) > 42	Maior que	falso
12 <= 42	Menor ou igual que	verdadeiro
30 >= 12	Maior ou igual que	verdadeiro

Figura 5.1.1 – Resultado do primeiro desafio rápido.

5.2 Atribuição condicional (ternário)

Esse tópico foi visto brevemente anteriormente. Vamos relembrar a sintaxe.

```

1 <?php
2     // Operador condicional ternario
3     // expressao1 sera um teste
4     // expressao2 sera atribuido a $var caso expressao1 seja verdadeiro
5     // expressao3 sera atribuido a $var caso expressao1 seja falsa
6     $var = expressao1 ? expressao2 : expressao3
7     ?>

```

Essa instrução se aplica quando queremos uma estrutura resumida, onde podemos ter um resultado mais direto, como por exemplo, atribuir um valor a uma variável dependendo de uma expressão. Observe o exemplo abaixo: a variável `$texto` receberá o valor “menor de 18” ou “maior de 18” de acordo com o teste `$idade > 18`.

```

1 <?php
2     // Operador condicional ternario
3     $idade = 19;

```

```

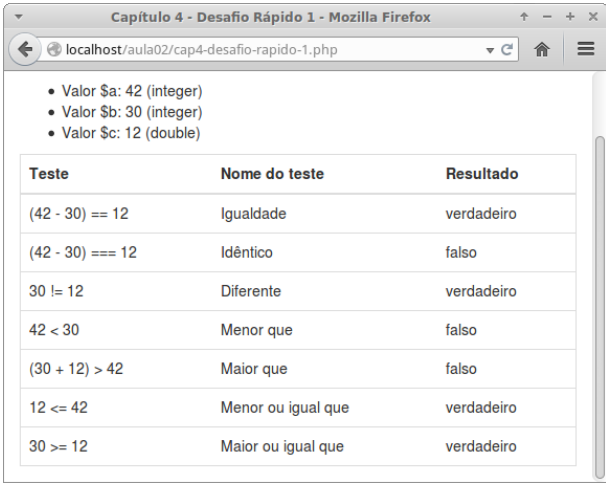
4  $texto = ($idade > 18) ? "maior de 18" : "menor de 18";
5  echo $texto;
6  ?>

```

É uma estrutura semelhante ao comando `if..else`. Cabe ao programador escolher onde cada uma das estruturas é melhor aplicada.

Desafio rápido!

Faça um pequeno programa que verifique se um número é par ou ímpar. A variável `$num` deve ser inicializada pelo programador. O programador deve ainda utilizar a estrutura de condição ternária e em um outro exemplo, utilizar o comando `if..else`. Veja como deve ser a saída do seu programa.



• Valor \$a: 42 (integer)
• Valor \$b: 30 (integer)
• Valor \$c: 12 (double)

Teste	Nome do teste	Resultado
$(42 - 30) == 12$	Igualdade	verdadeiro
$(42 - 30) === 12$	Idêntico	falso
$30 != 12$	Diferente	verdadeiro
$42 < 30$	Menor que	falso
$(30 + 12) > 42$	Maior que	falso
$12 \leq 42$	Menor ou igual que	verdadeiro
$30 \geq 12$	Maior ou igual que	verdadeiro

Figura 5.2.1 – Resultado do primeiro desafio rápido.

5.3 Estrutura switch

Observe que quando temos muitos `if..else` encadeados estamos criando uma estrutura que não é de fácil entendimento, portanto, **não** é considerada uma boa prática de programação. Para resolver esse problema temos uma estrutura semelhante ao `if..else`. O comando `switch` é uma estrutura que simula uma bateria de teste sobre uma variável. Portanto, essa estrutura é utilizada quando é necessário comparar a mesma variável com valores diferentes e executar uma ação específica em cada um desses valores. Veja a sintaxe e um exemplo real.

```

1  <?php
2  // Sintaxe estrutura condicional switch
3  switch ($variavel) {
4      case 'valor 1':
5          # código caso $variavel seja igual a 'valor 1'
6      case 'valor 2':

```

```
7      # codigo caso $variavel seja igual a 'valor 2'
8  }
9
10 // Exemplo real
11 $numero = 3;
12 switch ($numero) {
13     case 1:
14         echo "opcao 1" . "<br />";
15     case 2:
16         echo "opcao 2" . "<br />";
17     case 3:
18         echo "opcao 3" . "<br />";
19     case 4:
20         echo "opcao 4" . "<br />";
21 }
22 ?>
```

Nesse exemplo, o resultado que aparecerá no navegador é: `opcao 3` em uma linha e `opcao 4` em outra linha. A variável `$numero` tem o valor 3. O comando `switch` compara com cada `case` o valor recebido. O bloco executado é o do terceiro `case`, porém, os demais também são executados. Para que tenhamos um resultado satisfatório e não seja apresentado no navegador o restante das opções, é interessante o uso do comando `break` em cada `case`. O comando `break` tem a função de parar o bloco de execução e “sair” do `switch`.

5.3.1 Comando switch com break

O comando `break` é uma instrução (comando) utilizada quando queremos parar o fluxo da execução de um programa dentro do `switch` (e outras estruturas). Observe o mesmo exemplo com o uso do `break`. Temos agora como resultado “opcao 3”. O comando `break` fez com que os demais `case` abaixo do `case 3` não sejam executados.

```
1 <?php
2 // Exemplo real
3 $numero = 3;
4 switch ($numero) {
5     case 1:
6         echo "opcao 1" . "<br />";
7         break;
8     case 2:
9         echo "opcao 2" . "<br />";
10        break;
11    case 3:
12        echo "opcao 3" . "<br />";
13        break;
14    case 4:
```

```
15     echo "opcao 4" . "<br />";
16     break;
17 }
18 ?>
```

5.3.2 Comando switch completo

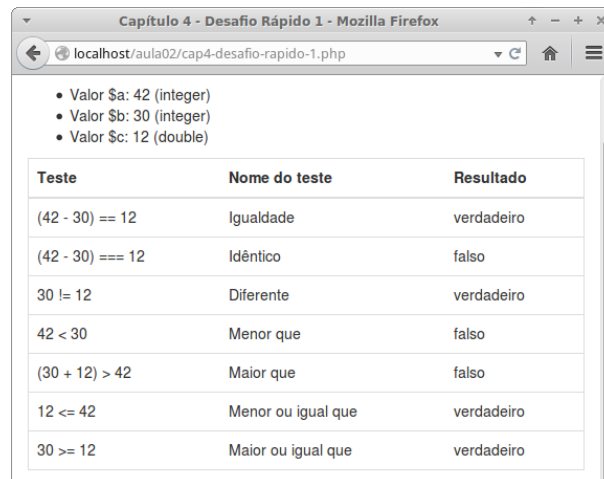
Mas o que acontece se não tivermos um valor que seja satisfatório aos casos existentes no **switch**? A resposta é bem simples, nenhum dos blocos seriam executados. Porém é possível utilizarmos um comando que determina uma opção padrão caso nenhuma das outras venha ter resultado que satisfaça a expressão passada para o comando **switch** chamada **default** (padrão).

```
1  <?php
2  // Exemplo real
3  $numero = 5;
4  switch ($numero) {
5      case 1:
6          echo "opcao 1" . "<br />";
7          break;
8      case 2:
9          echo "opcao 2" . "<br />";
10         break;
11     case 3:
12         echo "opcao 3" . "<br />";
13         break;
14     case 4:
15         echo "opcao 4" . "<br />";
16         break;
17     default:
18         echo "Nenhuma opcao satisfeita";
19 }
20 ?>
```

A instrução passada não condiz com nenhum dos casos existentes. Por esse motivo o bloco pertencente ao comando **default** será executado. O comando **default** pode ser inserido em qualquer lugar dentro do **switch**, porém caso isso aconteça, o uso do comando **break** deve ser adicionado para evitar que os **case** abaixo sejam executados.

Desafio rápido!

Faça um pequeno programa que recebe um símbolo matemático (como *string*) e a partir dele faça as operações matemáticas. Utilize o comando **switch**. Veja como a tela do seu navegador deve ficar.



Teste	Nome do teste	Resultado
(42 - 30) == 12	Igualdade	verdadeiro
(42 - 30) === 12	Idêntico	falso
30 != 12	Diferente	verdadeiro
42 < 30	Menor que	falso
(30 + 12) > 42	Maior que	falso
12 <= 42	Menor ou igual que	verdadeiro
30 >= 12	Maior ou igual que	verdadeiro

Figura 5.3.1 – Resultado do primeiro desafio rápido.

A partir de agora vamos aprender a trabalhar com estruturas de repetição. Elas são muito utilizadas nas linguagens de programação.

5.4 Estrutura *while*

O comando `while` é uma estrutura de controle similar ao comando `if`. Ele possui uma condição para executar um bloco de comandos. A diferença primordial é que o comando `while` estabelece um laço de repetição, ou seja, o bloco de comandos será executado repetidamente enquanto a condição passada for verdadeira. Veja a sintaxe.

```
1 <?php
2 // Sintaxe do comando while
3 while (condicao) {
4     # codigo
5 }
6 ?>
```

Quando estamos usando um laço de repetição, podemos determinar quantas vezes ele deve ou não se repetir. Isso pode ser feito de forma manual - o programador determina, ou automaticamente - quem vai determinar é o fluxo de execução do código. Veja o exemplo a seguir.

```
1 <?php
2 // Exemplo do comando while
3 $num = 1;
4 while ($num < 10) {
5     echo $num . ", ";
6     $num = $num + 1;
7 } // resultado: 1, 2, 3, 4, 5, 6, 7, 8, 9,
8 ?>
```

Nesse exemplo criamos um laço de repetição que tem como **condição de parada** o teste `$num < 10`, além disso, em cada laço executado um incremento na variável `$num` é realizado, fazendo com que o seu valor aumente (se ficou com dúvidas veja seção 4.4). O laço `while` continuará até que a condição não seja mais satisfatória, em outras palavras, até que a variável `$num` não seja menor que 10.

Importante! Tenha cuidado quando estiver trabalhando com estruturas de repetição, pois caso a expressão passada esteja errada, pode ocasionar um *loop* infinito fazendo com que o bloco de código se repita infinitamente. Isso pode ocasionar um travamento do navegador ou até mesmo do próprio servidor WEB.

Vamos ver agora um exemplo em que o laço se repete de forma automática. A condição de parada é uma função do PHP e não um número determinado pelo programador. A função `strlen` recebe uma variável do tipo *string* e retorna a quantidade de caracteres incluindo também os espaços em branco. Ele poderia ser aplicado diretamente na função `echo`, mas no exemplo, ele determina a quantidade de iterações.

```
1  <?php
2      // Exemplo do comando while
3      $num = 0;
4      $texto = "Projeto e-Jovem";
5      while ($num <= strlen($texto)) {
6          $num++;
7      }
8      echo "a frase possui " . $num . " caracteres.";
9      // resultado: a frase possui 15 caracteres.
10  ?>
```

Desafio rápido!

Faça um pequeno programa que imprima no navegador os valores de 3 em 3, formando a sequência 0, 3, 6, 9, 12... Utilize o comando `while`.

5.5 Estrutura do...while

O laço `do...while` funciona de maneira bastante semelhante ao comando `while`. A diferença é que a expressão verificada está no final do bloco de comandos. Essa mudança representa a execução do bloco **pelo menos uma vez**. Diferente do comando `while` onde é possível que o bloco não seja executado nenhuma vez. Veja a sintaxe e um exemplo abaixo.


```
1 <?php
2     // Sintaxe do comando do...while
3     do {
4         # codigo...
5     } while (condicao);
6 ?>
7
8 <?php
9     // Exemplo
10    $num = 0;
11    do {
12        echo $num . ", ";
13        $num = $num + 1;
14    } while ($num < 10);
15    // resultado: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
16 ?>
```

Qual seria o resultado desse código se a variável `$num` fosse inicializada com o valor 11, ou seja, `$num = 11`; na linha 10?

Desafio rápido!

Faça um pequeno programa que imprima na tela somente números pares, iniciando com o 2 e terminando com o valor 20. Utilize o comando `do...while`.

5.6 Estrutura for

A estrutura de repetição `for` tem a finalidade de criar um laço de repetição. Sua inicialização é composta por três partes que estabelecem as regras de funcionamento do `for`. Veja a sintaxe.

```
1 <?php
2     // Sintaxe do comando for
3     for (inicializacao; condicao; atualizacao) {
4         # codigo...
5     }
6 ?>
```

A primeira parte das regras chamada no código acima de “inicialização” é responsável por inicializar a variável que será usada na “condição” e no “atualização”. A segunda parte, de nome “condição” é responsável por fazer o teste, se o teste sobre a condição for verdadeira, o laço do `for` é executado, caso contrário o laço termina. Por fim, a parte “atualização” é responsável por atualizar a variável inicializada anteriormente, geralmente a atualização é um incremento (soma) ou decremento (subtração).

Verifique no exemplo abaixo a utilização do comando `for`.

```
1 <?php
2 // Exemplo
3 for ($k = 0; $k < 10; $k++) {
4     echo $k . ", ";
5 } // resultado: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
6 ?>
```

Desafio rápido!

Faça um pequeno programa que receba duas variáveis `$a` e `$b`. Você deve apresentar na tela do navegador os números correspondentes entre `$a` e `$b`. Por exemplo: `$a = 5` e `$b = 11` deve ser impresso na tela a sequência: 5, 6, 7, 8, 9, 10, 11. Utilize o comando `for`.

5.7 Estrutura foreach

O comando `foreach` é um laço de repetição para interação em *array* (esse conteúdo será abordado no capítulo 6). Trata-se do comando `for` mais simplificado. Sua sintaxe é exemplificada no código abaixo.

```
1 <?php
2 // Sintaxe do comando foreach
3 foreach ($variavel_tipo_array as $elemento) {
4     # codigo...
5 }
6 // Exemplo
7 $alunos = array("Joaquim", "Maria", "Alice");
8 foreach ($alunos as $nome) {
9     echo $nome . ", ";
10 } // resultado: Joaquim, Maria, Alice,
11 ?>
```

Veremos adiante que um *array* é uma variável composta por vários elementos. No caso do exemplo anterior, esses elementos são nomes de pessoas. A finalidade do `foreach` é justamente a cada laço, pegar um desses valores e atribuir a uma variável, nesse caso, `$nome`, até que tenha percorrido todo o *array* (nesse caso, `$alunos`).

Também podemos saber em qual posição o elemento se encontra no *array*, para isso basta adicionar uma nova variável logo após o termo `as` seguido dos caracteres `=>`. Veja a sintaxe completa do comando `foreach` e um exemplo.

```
1 <?php
2 // Sintaxe completa do comando foreach
3 foreach ($variavel_tipo_array as $chave => $valor) {
4     # codigo...
5 }
6 // Exemplo
```

```
7  $alunos = array(0 => "Joaquim", 1 => "Maria", 2 => "Alice");
8  foreach ($alunos as $indice => $nome) {
9      echo $indice . " - " . $nome . "<br />";
10 } // resultado
11 // 0 - Joaquim
12 // 1 - Maria
13 // 2 - Alice
14 ?>
```

O comando **foreach** percorrerá o *array* `$alunos` e irá apresentar na tela o conteúdo das variáveis `$indice` e `$nome`. Lembrando que o conteúdo *array* será abordado mais profundamente no próximo capítulo.

5.8 Comando *break*

O comando **break** já utilizado na instrução **switch** pode ser usado nas estruturas de repetição. Ele terá a mesma função, ou seja, o comando interrompe a execução do laço. Desse modo, o programa sai da estrutura de repetição e continua após o bloco. Veja o exemplo abaixo.

```
1  <?php
2      // Exemplo
3      for ($k = 0; $k < 10; $k++) {
4          echo $k . ", ";
5          if ($k == 5) {
6              break;
7          }
8      } // resultado: 0, 1, 2, 3, 4, 5,
9      ?>
```

No exemplo acima, utilizamos o comando **break** dentro de um laço de repetição com o comando **for**. Utilizamos uma estrutura de decisão (comando **if**) para realizar um teste (`$k == 5`). Caso o teste seja verdadeiro, o programa irá executar a linha que tem a instrução **break**. Dessa maneira, o laço de repetição será interrompido e o restante dos números não serão impressos.

5.9 Comando *continue*

A instrução **continue**, quando executada em uma estrutura de repetição (**for**, **while** etc), ignora as instruções restantes do bloco, voltando ao início. Dessa forma, o programa segue para a próxima verificação da condição de entrada do laço de repetição, funciona de maneira semelhante ao **break**, com a diferença que o fluxo ao invés de ser interrompido é, na realidade, voltado ao início. Veja um exemplo:

```
1 <?php
2 // Exemplo
3 for ($k = 0; $k < 10; $k++) {
4     if ($k == 5) {
5         continue;
6     }
7     echo $k . ", ";
8 } // resultado: 0, 1, 2, 3, 4, 6, 7, 8, 9,
9 ?>
```

Perceba que o número 5 (do nosso teste) não foi apresentado na tela do navegador.

5.10 Exercícios

5.11 Desafio!

O seguinte código é uma página inicial de um sistema de notas simplificado. O programador deve inserir no próprio código as 3 notas do aluno. Se a média aritmética das notas for maior que 6, o aluno está aprovado, caso contrário o aluno se encontra de recuperação.

Esse código tem vários problemas e não está executando corretamente, identifique os erros, corrija e apresente ao professor.

```
1 <?php
2 $saudacao = "Bom dia!";
3 $aluno = "Joaquim Silva";
4 $nascimento = 29/06/1992;
5
6 echo "-----" . "<br />";
7 echo $saudacao . "<br />";
8 echo "-----" . "<br />";
9
10 $nota1 = 6.1;
11 $nota2 = "4.0"
12 $nota3 = 9;
13
14 $media = $nota1 + $nota2 + $nota3 / 3;
15 $situacao = $media > 6 ? "APROVADO" : "RECUPERACAO";
16
17 $info = "Aluno: " . $aluno . "<br />";
18 echo $info;
19 echo "Situacao: " . $situacao;
20 ?>
```

6 Manipulação de array's

Ao final deste capítulo, o aluno terá as seguintes competências:

1. Entender a estrutura do tipo *array*;
2. Trabalhar com variáveis do tipo *array*; e
3. Trabalhar com *array's* de duas dimensões.

O *array* é um tipo de variável que pode guardar diversos valores (chamados de: elementos de um *array*), no PHP, podemos definir como um conjunto de valores (elementos) ordenados. É possível relacionar cada valor com uma chave, no caso mais simples, a chave indica a posição (índice) em que um elemento do *array* se encontra.

6.1 Criando um array

Para criar um *array* pode-se utilizar a função `array`. Veja a sintaxe e um exemplo:

```
1 <?php
2 // Sintaxe de inicializacao de um array
3 $variavel = array(elemento1, elemento2);
4 // Exemplo
5 $alunos = array("Joaquim", "Maria", "Alice");
6 ?>
```

No exemplo acima, foi apresentado a forma mais simples de inicializar uma variável do tipo *array*. A variável `$alunos` pode ser percorrida utilizando o comando `for` por exemplo. Veremos mais a frente.

6.2 Arrays associativos

Nos *array's* associativos determina-se um valor ou nome aos valores do *array*. Observe a sintaxe:

```
1 <?php
2 // Sintaxe de inicializacao de um array
3 $variavel = array('chave 1' => valor 1, 'chave 2' => valor 2);
4 // Exemplo
5 $alunos = array(0 => "Joaquim", 1 => "Maria", 2 => "Alice");
6 ?>
```

No exemplo acima, utilizamos números para representar os índices do *array*. Na seção 6.4 aprenderemos como acessar elementos do *array*.

A seguir, veremos outro exemplo. A figura abaixo representa um *array* que tem como elementos representação de cores. O *array* possui dez posições, cada posição representa uma cor, seu índice (chave) vai de 0 até 9. Veja:

FIGURA!!!

Em código temos:

```
1 <?php
2 $alunos = array(0 => "Verde", 1 => "Azul", 2 => "Violeta",
3     3 => "Cinza", 4 => "Vermelho", 5 => "Amarelo",
4     6 => "Magenta", 7 => "Branco", 8 => "Laranja",
5     9 => "Marrom");
6 ?>
```

No exemplo abaixo, o *array* associativo usa strings como índice, onde cada *string* representa uma chave. Se quisermos representar as informações de uma pessoa, podemos usar a variável do tipo *array*.

```
1 <?php
2 $informacoes = array("nome" => "Joaquim Silva",
3     "endereco" => "rua A, n. 10", "bairro": "Benfica",
4     "cidade" => "Fortaleza");
5 ?>
```

Observe que quando usamos *array's* associativos, a compreensão é mais fácil, dando mais legibilidade ao código, ou seja, é mais fácil entender o que cada valor representa. Veja na representação gráfica abaixo:

FIGURA!!!

6.2.1 Inicializando array's

Outra forma de inicializar *array's* é adicionar valores de acordo com o seu uso. Veja no exemplo abaixo:

```
1 <?php
2 $informacoes["nome"] = "Joaquim Silva";
3 $informacoes["endereco"] = "rua A, n. 10";
4 $informacoes["bairro"] = "Benfica";
5 $informacoes["cidade"] = "Fortaleza";
6 ?>
```

6.2.2 Acessando um array

Para acessarmos o valor do *array* que tem chaves, usamos `$variavel['chave']`. Veja abaixo um exemplo de acesso ao valores armazenados em um *array* dessa natureza.

```
1 <?php
2     $informacoes["nome"] = "Joaquim Silva";
3     $informacoes["endereco"] = "rua A, n. 10";
4     $informacoes["bairro"] = "Benfica";
5     $informacoes["cidade"] = "Fortaleza";
6     // Acessando valores
7     echo "Nome: " . $informacoes["nome"] . "<br />";
8     echo "Cidade: " . $informacoes["cidade"] . "<br />";
9 ?>
```

Dessa forma podemos acessar um *array*. Basta determinar o nome da variável (`$informacoes`) e a chave (pro nosso exemplo, os possíveis valores são: `nome`, `endereco`, `bairro` e `cidade`), onde cada chave tem um valor já determinado. Caso a chave não exista, nada será impresso.

Para acessarmos o valor do *array* que (aparentemente) não tem chaves, usamos os índices que se iniciam do 0 e vai até a quantidade de elementos do *array* menos 1. Veja um exemplo abaixo:

```
1 <?php
2     $alunos = array("Joaquim", "Maria", "Alice");
3     echo $alunos[0] . "<br />";
4     echo $alunos[1] . "<br />";
5     echo $alunos[2] . "<br />";
6     // resultado
7     // Joaquim
8     // Maria
9     // Alice
10 ?>
```

Por mais que não tenha os valores expressamente indicados na variável `$alunos`, o PHP se encarrega de colocar os índices, permitindo ao programador, acessar os valores de maneira fácil.

6.3 Percorrendo um array

O processo de percorrer um *array* pode ser realizado por uma estrutura de repetição (`for`, `while`, `do...while` ou `foreach`). O exemplo a seguir apresenta uma abordagem utilizando o laço `foreach`.

```
1 <?php
2     $informacoes["nome"] = "Joaquim Silva";
3     $informacoes["endereco"] = "rua A, n. 10";
4     $informacoes["bairro"] = "Benfica";
5     $informacoes["cidade"] = "Fortaleza";
6     // Acessando valores
```

```
7  foreach ($informacoes as $chave => $valor) {
8      echo $chave . ": " . $valor . "<br />";
9  } // resultado:
10 // nome: Joaquim Silva
11 // endereco: rua A, n. 10
12 // bairro: Benfica
13 // cidade: Fortaleza
14 ?>
```

Importante! Sempre que se depararem com *array's*, onde haja a necessidade de percorrer seus valores independentemente da chave, procure utilizar mecanismos de programação mais simplificados como o comando `foreach`.

6.4 Acessando um *array*

Quando criamos um *array* temos que ter em mente que estamos criando uma variável que possui vários valores e que os mesmo podem ser acessados a qualquer momento. Cada valor está guardado em uma posição que pode ser acessada através de uma chave. A sintaxe para acesso simplificado de um *array* é a seguinte:

```
1  <?php
2      $informacoes = array("nome" => "Joaquim Silva", "cidade" => "Fortaleza");
3      echo $informacoes["nome"];
4      echo $informacoes["bairro"];
5      // resultado: Joaquim Silva
6  ?>
```

Temos que ter cuidado ao passar uma chave para o *array*, pois ela deve conter o mesmo nome de qualquer umas das chaves existentes no *array*. Caso a chave não exista, o valor não poderá ser resgatado. A sintaxe acima retorna um valor contido na variável `$variavel_tipo_array`. Verifique o exemplo abaixo:

```
1  <?php
2      $informacoes = array("nome" => "Joaquim Silva", "cidade" => "Fortaleza");
3      echo $informacoes["nome"];
4      echo $informacoes["bairro"];
5      // resultado: Joaquim Silva
6  ?>
```

No exemplo acima, apenas o valor “Joaquim Silva” será apresentado na tela, uma vez que a chave `bairro` não está inserida no *array* `$informacoes`.

6.5 Alterando um array

Podemos alterar qualquer valor de um *array*. É muito semelhante ao método de acesso. A diferença está na chamada do *array*. É nesse momento que atribuímos um novo valor. Veja o exemplo abaixo.

```
1 <?php
2   $informacoes = array("nome" => "Joaquim Silva", "cidade" => "Fortaleza");
3   echo $informacoes["nome"] . "<br />";
4   echo $informacoes["cidade"] . "<br />";
5   // modificando valores do array
6   $informacoes["nome"] = "Maria Lima";
7   $informacoes["cidade"] = "Caucaia";
8   // apresentando as novas informacoes
9   echo $informacoes["nome"] . "<br />";
10  echo $informacoes["cidade"] . "<br />";
11 ?>
```

Primeiro os valores iniciais da variável `$informacoes` são apresentados na tela. Em seguida, esses valores são modificados, por fim, os novos valores são impressos. Veja outro exemplo:

```
1 <?php
2   $informacoes = array("nome" => "Joaquim Silva", "cidade" => "Fortaleza");
3   echo $informacoes["nome"] . "<br />";
4   echo $informacoes["cidade"] . "<br />";
5   // modificando valores do array
6   $informacoes["nome"] = "Maria Lima";
7   $informacoes["cidade"] = "Caucaia";
8   // apresentando as novas informacoes
9   echo $informacoes["nome"] . "<br />";
10  echo $informacoes["cidade"] . "<br />";
11 ?>
```

No exemplo acima, modificamos um *array* simples. Para acessá-los é necessário utilizar os índices (que cada elemento tem em *array's* simples). Nesse caso, estamos alterando o preço do produto na instrução `$produto[1] += 3`. Poderíamos ter utilizado a forma `$produto[1] = $produto[1] + 3` para obtermos o mesmo resultado.

6.6 Arrays com duas dimensões

Os *array's* com duas dimensões são estruturas de dados que armazenam os valores em mais de uma dimensão. Os *array's* que vimos até agora armazenam valores em uma dimensão, por isso para acessar às posições utilizamos somente um índice ou chave. Os *array's* de 2 dimensões salvam seus valores na forma de filas e colunas e por isso, é necessário dois índices para acessar cada uma de seus valores.

Outra ideia que temos é que *array's* de duas dimensões são matrizes nos quais algumas de suas posições podem conter outros *array's*. Na figura abaixo temos a representação de um *array* com duas dimensões.

FIGURA!!!

Um *array* de duas dimensões pode ser criado pela função `array`. Veja um exemplo não funcional.

```
1 <?php
2 // Sintaxe
3 $variavel = array(
4     array(elemento_00, elemento_01, elemento_02), // primeira linha
5     array(elemento_10, elemento_11, elemento_12), // segunda linha
6     array(elemento_20, elemento_21, elemento_22) // terceira linha
7 );
8 ?>
```

Observe agora um exemplo real:

```
1 <?php
2 // Exemplo
3 $matriz = array(
4     array("00", "01", "02"), // primeira linha
5     array("10", "11", "12"), // segunda linha
6     array("20", "21", "22") // segunda linha
7 );
8 ?>
```

6.6.1 Acessando um *array* de duas dimensões

Para acessarmos o valor de um *array* de duas dimensões, basta colocar as duas chaves da posição que queremos acessar. É semelhante ao *array* de uma única dimensão. Continuando com o exemplo acima, podemos acrescentar ao final do código as seguintes linhas.

```
1 <?php
2 // continuacao do codigo ...
3 echo "Imprimindo primeira linha" . "<br />";
4 echo "elemento 00: " . $matriz[0][0] . "<br />";
5 echo "elemento 01: " . $matriz[0][1] . "<br />";
6 echo "elemento 02: " . $matriz[0][2] . "<br />";
7 ?>
```

O código acima apresenta no navegador a primeira linha da matriz. Já o código abaixo apresenta no navegador a última coluna da matriz.

```
1 <?php
2 // continuacao do codigo ...
```

```
3 echo "Imprimindo ultima coluna" . "<br />";
4 echo "elemento 02: " . $matriz[0][2] . "<br />";
5 echo "elemento 12: " . $matriz[1][2] . "<br />";
6 echo "elemento 22: " . $matriz[2][2] . "<br />";
7 ?>
```

Perceba que quando queremos imprimir uma linha toda, o primeiro índice não muda, ou seja, no nosso exemplo ele fica zero. Já quando queremos imprimir a coluna, deixamos o segundo índice imutável (sem ser modificado). Desenhar a matriz que você quer representar no papel ajuda no entendimento da lógica de acesso aos elementos da matriz.

6.6.2 Percorrendo um *array* de duas dimensões

Da mesma maneira que percorremos um *array* de uma dimensão, podemos percorrer um *array* de duas dimensões. Será necessário utilizar dois laços de repetição, seja um comando `for`, `while` ou `foreach`. O **primeiro laço de repetição é usado para percorrer as linhas**, o segundo laço utilizado para percorrer os elementos das colunas. Acompanhe no código abaixo.

```
1 <?php
2 // Exemplo
3 $matriz = array(
4     array("00", "01", "02"), // primeira linha
5     array("10", "11", "12"), // segunda linha
6     array("20", "21", "22") // segunda linha
7 );
8 for ($linha = 0; $linha < 3; $linha++) {
9     for ($coluna = 0; $coluna < 3; $coluna++) {
10         echo $matriz[$linha][$coluna] . ", ";
11     }
12     echo "<br />";
13 }
14 ?>
```

6.7 Visualizando array's

Existe um conjunto de funcionalidades internas do PHP prontas para serem utilizadas nas variáveis do tipo *array*. Trata-se de funções pré-definidas, você pode encontrá-las facilmente no site <<http://php.net>>.

Abordaremos agora funções utilizadas exclusivamente para manipulação de *array*, funções de acesso e visualização. Obviamente que não serão abordadas todas as funções,

pois existem várias, mas mostraremos as mais utilizadas, e outras que são definidas como principais.

6.7.1 Função `var_dump`

Essa função é muito usada por programadores que pretendem realizar *debug* (análise mais detalhada para encontrar supostos erros). Observe um exemplo prático:

```
1 <?php
2 // Exemplo
3 $produto = array("id" => "AC12D", "nome" => "detergente", "preco" => 2.40);
4 var_dump($produto);
5 ?>
```

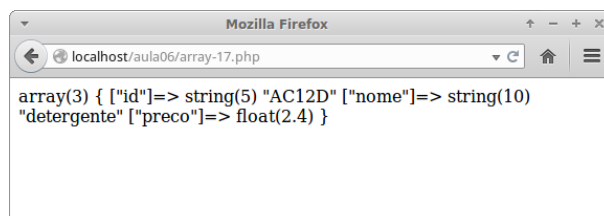


Figura 6.7.1 – Saída da função `var_dump`.

A função `var_dump` é importante pois mostra no resultado o *array* completo incluindo os tipos de variáveis que compõem o *array*. No nosso exemplo, o *array* tem 3 elementos. Mostra o tipo de cada um desses elementos e seus respectivos valores. Os elementos `id` e `nome` são do tipo *string*. Já o elemento `preco` é do tipo *float* (número real).

6.7.2 Função `print_r`

Imprime o conteúdo de uma variável assim como a função `var_dump`, porém apresenta menos detalhes, alguns programadores a preferem por ser mais legível. Exemplo:

```
1 <?php
2 // Exemplo
3 $produto = array("id" => "AC12D", "nome" => "detergente", "preco" => 2.40);
4 print_r($produto);
5 ?>
```

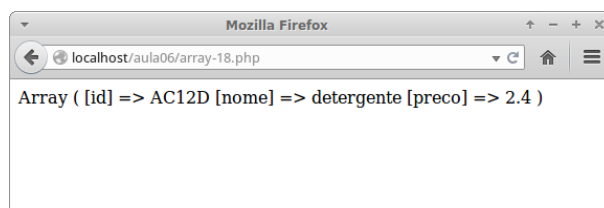


Figura 6.7.2 – Saída da função `print_r`.

6.7.3 Funções `sort` e `rsort`

Muitas vezes é necessário organizar o *array* de acordo com uma ordem. Por exemplo: se temos um conjunto de alunos, podemos deixar seus nomes por ordem alfabética de A a Z. Ou ainda a ordem pode ser de Z a A. Para isso, podemos usar as funções `sort` e `rsort` respectivamente. Veja o exemplo abaixo.

```
1  <?php
2      // Exemplo
3      $alunos = array("Joaquim", "Maria", "Alice", "Breno");
4      echo "Imprimindo variavel sem ordenar" . "<br />";
5      print_r($alunos);
6      // Ordenando variavel alunos
7      sort($alunos);
8      echo "Imprimindo variavel ordenada" . "<br />";
9      print_r($alunos);
10
11  ?>
```

6.8 Exercícios

6.9 Desafio!

7 Interações PHP com HTML

Ao final deste capítulo, o aluno terá as seguintes competências:

1. Entender a estrutura dos métodos `GET` e `POST`; e
2. Enviar dados para o servidor através do navegador.

Neste capítulo abordaremos algumas formas de interações utilizando a linguagem de programação PHP e a linguagem de marcação HTML. Além disso, mostraremos alguns dos componentes mais utilizados para a construção de um sistema ou uma página web, e de que forma o PHP pode receber, processar, e enviar essa informação.

Utilizamos linguagens para exibir os dados da aplicação, seja ela para simples conferência, em relatório ou ainda possibilitando a adição e exclusão de registros. Criaremos a princípio formulários e listagem.

7.1 Formulários

Formulário pode ser definido como um conjunto de campos disponíveis de forma agrupada para serem preenchidos com informações. Na internet, um formulário é composto por vários componentes, além de possuir botões de ação, no qual se define o programa que processará os dados.

7.1.1 Criando um formulário

Para criarmos um formulário, utilizamos a *tag* `<form>`. Dentro dessa *tag* podemos colocar diversos elementos, onde, cada um deles representa uma propriedade em particular. A seguir explicaremos os principais componentes de um formulário.

Todo formulário deve conter no mínimo as seguintes características: ter um nome, um método e uma ação. O nome deve ser único e só pode haver um formulário com esse nome na página PHP. Veremos posteriormente nesse capítulo que podemos usar os métodos `GET` ou `POST`. Já a ação é o caminho do arquivo que vai tratar os dados do formulário. Veja a sintaxe.

```
1 <form name="" method="" action="">
2   <!-- código -->
3 </form>
```

Lembrando que em cada aula iremos criar o diretório da respectiva aula no caminho `/var/www/`. Salve o arquivo abaixo com o nome `index.html` no diretório da aula.

```
1 <!DOCTYPE html>
```

```
2 <html>
3 <head>
4   <meta charset="utf-8">
5   <title>Aula 07</title>
6 </head>
7 <body>
8   <h1>Formulario</h1>
9   <form name="" method="" action="">
10     <!-- codigo -->
11   </form>
12 </body>
13 </html>
```

Ao abrir o código acima no navegador, encontramos apenas uma página em branco com o título “Formulário”. Vamos colocar um campo de texto, permitindo que o usuário digite um texto simples. Para isso, é necessário utilizar a *tag* `<input>` do HTML. Conforme exemplo abaixo. Para simplificar, vamos omitir algumas linhas do código.

```
1 ...
2 <form name="" method="" action="">
3   <input name="email" type="text"></input>
4 </form>
5 ...
```

A *tag* `<input>` pode ser composta por vários atributos. A partir deles, vamos definir o comportamento da *tag*. No nosso exemplo, definimos os atributos nome e tipo. O atributo `name` é utilizado para identificação do elemento na página (ele deve ser único). Já o atributo `type` (tipo em inglês) indica qual campo de formulário a *tag* `<input>` irá representar no navegador. Escreva o código acima e visualize no seu navegador.

7.2 Métodos GET e POST

Agora que relembramos a estrutura de um formulário HTML, vamos aprender sobre os métodos `GET` e `POST`. A função básica desses métodos é a de enviar os dados para o servidor (lembre-se da arquitetura cliente-servidor) para que ele tenha acesso aos dados digitados no formulário. O protocolo HTTP disponibiliza vários métodos para trabalhar com requisições. Os dois métodos básicos de envio de dados ao servidor serão vistos nessa apostila.

Portanto, imagine a seguinte situação: Joaquim, aluno do e-jovem quer pesquisar no Google sobre PHP. Ao digitar o termo de pesquisa na caixa de busca do Google, o navegador realiza uma requisição HTTP de método `GET`. O Google então apresenta uma lista de possíveis sites que o Joaquim pode entrar, ao escolher clicar no primeiro link, uma nova requisição HTTP com o método `GET` foi realizada.

Uma outra situação possível é o Joaquim acessar sua página do Facebook, ele precisa digitar o nome de usuário e a senha, ao acionar o botão “Entrar”, o navegador realiza uma requisição HTTP dessa vez com o método o *POST*.

A figura abaixo ilustra a comunicação que ocorre entre o formulário (cliente) e o arquivo PHP (servidor), seja utilizando o método *GET* ou método *POST*. A diferença básica entre esses métodos será visto nas próximas seções.

ILUSTRAÇÃO DA MILENA

7.2.1 Método *GET*

O método *GET* utiliza a própria URI (normalmente chamada de URL) para enviar dados ao servidor. Quando enviamos um formulário pelo método *GET*, o navegador pega as informações do formulário e coloca junto com a URI de onde o formulário vai ser enviado e envia, separando o endereço da URI dos dados do formulário por um “?” (ponto de interrogação) e “&”.

Esse processo gera uma variável do tipo *array* chamada de `$_GET`. O conteúdo dela é formado pelos dados do formulário enviado.

Quando o usuário faz uma busca no Google, ele faz uma requisição utilizando o método *GET*, você pode verificar na barra de endereço do seu navegador que o endereço ficou com um ponto de interrogação no meio, e depois do ponto de interrogação você pode ler, dentre outros caracteres, o que você pesquisou no Google.

Abaixo temos um exemplo de uma URL do Joaquim pesquisando sobre PHP no Google. Observe:

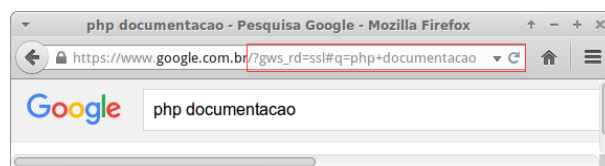


Figura 7.2.1 – Exemplo de URL utilizando o método *GET*.

Podemos notar a passagem de um parâmetro, chamado de `?gws_rd=ssl#q=`. Os últimos termos são os digitados pelo usuário. No nosso exemplo, `php` e `documentacao`.

7.2.2 Nosso primeiro formulário com método *GET*

Para nosso primeiro formulário com método *GET*, vamos simular o envio de um *email*. Imagine que você está cadastrando seu *email* para participar de uma *newsletter* (um boletim informativo). Para isso, o formulário deve ter o campo *email* e um botão de enviar. Utilize seus conhecimentos de HTML para representar a tela abaixo.

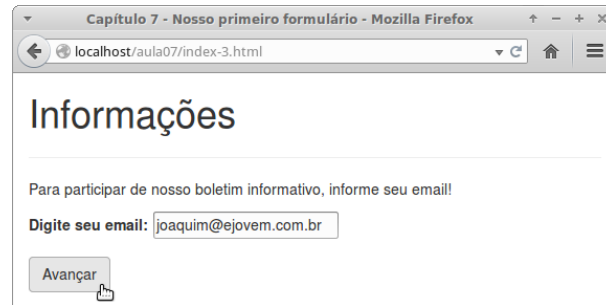


Figura 7.2.2 – Exemplo de um formulário simples.

Ao clicarmos em avançar, será possível, visualizar no navegador as informações inseridas no campo de texto. No nosso exemplo, joaquim@ejovem.com.br. Parte do código que foi utilizado para construir essa página é apresentado abaixo.

```

1  ...
2  <form name="cadastro" method="GET" action="cadastro-newsletter.php">
3    <div class="form-group">
4      <label for="campo-email">Digite seu email: </label>
5      <input name="email" type="text"></input>
6    </div>
7
8    <button type="submit" class="btn btn-default">Avancar</button>
9  </form>
10  ...

```

Perceba os atributos `name`, `method` e `action`. A partir desses atributos será possível trabalhar com os dados inseridos no formulário.

7.2.3 Recebendo dados via método GET

Quando o botão “Avançar” for pressionado, uma requisição será feita ao site <http://localhost> (nosso servidor) enviando todos os dados preenchidos no formulário. O arquivo `.php` responsável por tratar esses dados é o arquivo mencionado no atributo `action`. No nosso primeiro exemplo, o arquivo de nome `cadastro-newsletter.php`.

Caso você ainda não o tenha criado. Crie um novo arquivo dentro de `/var/www/aula07/` chamado `cadastro-newsletter.php`. Inicialmente, coloque o seguinte conteúdo dentro do arquivo.

```

1  <?php
2    print_r($_GET);
3  ?>

```

O código acima exibe no navegador a variável `$_GET` que é um *array*. Perceba que a única chave do *array* é igual ao nome do campo de texto no formulário na página HTML.

Ou seja, o **array** `$_GET` é um **array associativo**, onde as chaves são os atributos `name` dos campos do formulário e os elementos são os valores inseridos no formulário.

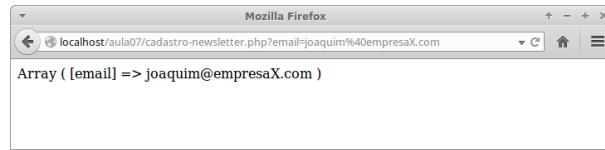


Figura 7.2.3 – Conteúdo enviado através do método GET.

Caso seja necessário pegar apenas a informação de email, devemos lembrar como acessamos elementos em um **array** associativo. Para isso, precisaremos da chave do elemento. Veja no código abaixo.

```
1 <?php
2     print_r($_GET);
3     echo "email: " . $_GET["email"];
4 ?>
```

Desafio rápido!

Altere o arquivo `cadastro-newsletter.php` para receber, além do email do usuário, um nome e data de nascimento. Apresente no navegador o **array** `$_GET` através da função `print_r` e as variáveis separadamente (lembre-se do procedimento de acessar uma variável do tipo **array** associativo).

7.3 Método POST

O método POST é bastante semelhante ao método GET. Ele também é utilizado para transferir os dados do formulário na página HTML para o servidor PHP. A principal diferença está em enviar os dados encapsulados no corpo da mensagem HTTP, ou seja, ao invés das informações inseridas no formulário trafegarem junto à URL. Elas vão “escondidas”.

Sua utilização é mais frequente quando trabalhamos com informações sigilosas, como é o caso de autenticação por meio de usuário e senha, por exemplo: Joaquim deve primeiro se autenticar, inserindo seu *email* e senha, para só depois, ter acesso à caixa de entrada do *email*.

O código do formulário é modificado apenas no atributo `method`. O valor se torna POST. Veja no exemplo abaixo.

```
1 ...
```

```

2 <form name="" method="POST" action="">
3   <!-- código -->
4 </form>
5 ...

```

7.3.1 Nosso primeiro formulário com método POST

Para nosso primeiro formulário com método POST, vamos simular a autenticação em um sistema *web*. Imagine que você está cadastrado nesse sistema e deseja se autenticar. Para isso, o formulário deve ser preenchido com um usuário e a senha do mesmo. Utilize seus conhecimentos de HTML para representar a tela abaixo.

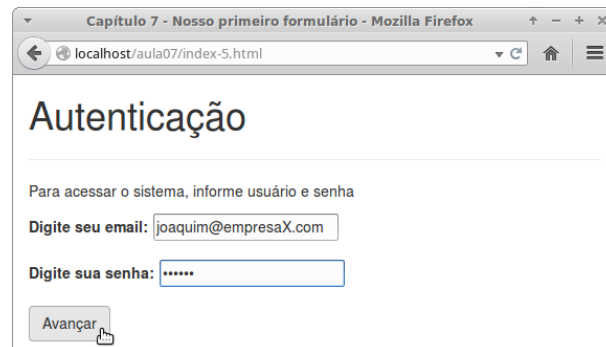


Figura 7.3.1 – Exemplo de um formulário de autenticação simples.

O formulário acima deve ter as seguintes características na *tag* `<form>`:

```

1 ...
2 <form name="entrar" method="POST" action="acesso.php">
3 ...

```

7.3.2 Recebendo dados via método POST

Ao clicarmos em avançar, os dados inseridos no formulário serão tratados no arquivo `acesso.php`. Portanto, vamos criar o arquivo no diretório `/var/www/aula07` com o seguinte conteúdo:

```

1 <?php
2   print_r($_POST);
3 ?>

```

O código acima exibe a variável do tipo *array* com a função `print_r`. Perceba que não estamos mais trabalhando com a variável `$_GET` e sim com a variável `$_POST`, pois o método utilizado no formulário mudou.

Outro detalhe importante a ser percebido é que as informações inseridas no formulário não estão mais visíveis na URL. Para visualizarmos as informações inseridas, devemos programar o arquivo `acesso.php`.

Desafio rápido!

Altere o arquivo `acesso.php` para exibir no navegador o *email* do usuário e sua senha. Utilize o *array* `$_POST`. Lembre-se do procedimento de acessar uma variável do tipo *array* associativo.

7.4 Exercícios

7.5 Desafio!

8 Manipulação de funções

Ao final deste capítulo, o aluno terá as seguintes competências:

1. Entender o propósito da criação de funções; e
2. Criar e manipular funções no PHP.

A maioria das linguagens de programação permitem que o programador crie funções para serem utilizadas no código. As funções são um bloco de código definidos e que tem um objetivo específico. Elas podem ainda receber parâmetros (dados de entrada) e retornar um valor (dados de saída). A maior qualidade das funções é sua propriedade de ser reutilizada, ou seja, o programador declara uma vez e a utiliza em diversas partes do seu código.

8.1 Declarando uma função

Declarar uma função no PHP é simples. Essa seção irá lhe guiar pelo processo. O código abaixo apresenta a sintaxe de uma declaração de função. Na linha 3 utilizamos a instrução `function` e em seguida o nome da função, os termos entre parenteses são os parâmetros daquela função. O bloco da função é definido pelos caracteres `{` e `}`, ou seja, toda vez que executarmos a função definida, o código entre as chaves é que será executado. Na linha 7 a instrução `return` é responsável por retornar o valor calculado. Nem sempre uma função recebe algum parâmetro, por isso, eles são opcionais, assim como a instrução `return`, que também é opcional.

```
1 <?php
2 // Sintaxe da criacao de uma funcao
3 function nome_da_funcao($argumento1, $argumento2,...)
4 {
5     // comando 1;
6     // comando 2;
7     return $valor;
8 }
9 ?>
```

8.2 Passagem de parâmetros

Como vimos anteriormente na sintaxe da função, podemos passar ou não parâmetros pra uma função. porém existem alguns tipos de passagem de parâmetros: Por valor (*by value*), por referência (*by reference*) e por argumentos variáveis - esses últimos dois não serão abordados nessa apostila.

O tipo de passagem de parâmetro por valor tem a seguinte característica: se o conteúdo da variável for alterado, essa alteração não afeta a variável original. Exemplo:

```
1 <?php
2 // criacao da funcao
3 function imprime_h1($texto) {
4     $novo_texto = "<h1>". $texto . "</h1>";
5     echo $novo_texto;
6 }
7 ?>
8 <?php
9 // usando a funcao criada anteriormente
10 imprime_h1("Seja bem vindo!");
11 ?>
```

No exemplo acima, a mensagem “Seja bem vindo!” não é alterada.

8.3 Valor de retorno

Toda função pode opcionalmente retornar um valor, ou simplesmente executar os comandos e não retornar valor algum. No PHP não é possível que uma função retorne mais de um valor, mas é permitido fazer com que uma função retorne um valor composto, como o *array*.

Observe o exemplo abaixo. Nele criamos uma função para calcular o índice de massa corporal de uma pessoa (IMC). Essa função recebe como parâmetro dois argumentos (por valor). A altura representada pela variável `$altura` e o peso representado pela variável `$peso`.

```
1 <?php
2 // criacao da funcao
3 function calcula_imc($peso, $altura) {
4     $imc = $peso / ($altura * $altura);
5     return $imc;
6 }
7 ?>
8 <?php
9 // usando a funcao criada anteriormente
10 $peso = 62;
11 $altura = 1.75;
12 $imc = calcula_imc($peso);
13 echo "indice de massa corporal: " . $imc;
14 // resultado de $imc = 20.244897959
15 ?>
```

A função utiliza as duas variáveis para fazer o cálculo `$peso / ($altura * $altura)`. No exemplo acima é apresentada a declaração da função e logo após a chamada

da função criada. Passamos como parâmetros para essa função o peso = 62 e a altura = 1.75.

Desafio rápido!

Crie um arquivo no diretório `/var/www/aula08/` com o nome `funcoes.php` e desenvolva 4 funções. Todas as funções devem receber como parâmetro, dois valores numéricos (inteiros ou reais).

A função de nome `somar` retorna a soma dos parâmetros. A função de nome `subtrair` retorna a diferença do primeiro parâmetro pelo segundo. Já a função `dividir` retorna o quociente da divisão do primeiro parâmetro pelo segundo. Por fim, a última função de nome `multiplicar`, multiplica os dois valores e retorna o resultado.

8.4 Escopo de variáveis em funções

Um conceito importante em programação são os tipos de declarações de variáveis. A visibilidade das variáveis dependerá de onde ela é declarada. Os acessos a essas variáveis podem ser definidos da seguinte forma:

Variáveis locais (código: 8.1): São aquelas declaradas dentro de uma função e não tem visibilidade fora dela;

Código 8.1 – Exemplo: Variáveis Locais

```
1 <?php
2     // variaveis locais
3     function teste() {
4         $a = "variavel local";
5     }
6     teste();
7     echo $a; // resultado: nada sera impresso
8 ?>
```

Nada foi impresso pois a variável `$a` só existe dentro da função `teste()`.

Variáveis globais (código: 8.2): São variáveis declaradas fora do escopo de uma função, porém tem visibilidade (pode ser acessada) de dentro de uma função sem ser passada como parâmetro. Para isso declaramos a variável e fazemos a sua chamada logo após com o uso do termo `global`; e

Código 8.2 – Exemplo: Variáveis globais

```
1 <?php
2     // variaveis locais
3     function teste() {
```

```
4     global $a;
5     $a = "variavel global";
6 }
7 teste();
8 echo $a; // resultado: sera impresso: variavel global
9 ?>
```

Variáveis estáticas (código: 8.3): Podemos armazenar variáveis de forma estática dentro de uma função. Significa que ao fazermos isso, temos o valor preservado independente da última execução. Usamos o operador `static` para declaramos a variável.

Código 8.3 – Exemplo: Variáveis estáticas

```
1 <?php
2 // variaveis estaticas
3 function teste() {
4     static $a;
5     $a = $a + 10;
6     echo $a . "<br />";
7 }
8 teste(); // imprime 10 na tela
9 teste(); // imprime 20 na tela
10 teste(); // imprime 30 na tela
11 ?>
```

8.5 Exercícios

8.6 Desafio!

9 Manipulação de arquivos e diretórios

Ao final deste capítulo, o aluno terá as seguintes competências:

1. Abrir e fechar arquivos; e
2. Editar conteúdo de um arquivo texto;

Assim como em outras linguagens de programação é muito importante trabalharmos com manipulações de arquivos e diretórios em PHP. Através dessa funcionalidade podemos manipular um arquivo ou diretório dentro do servidor *web*, criando assim arquivos responsáveis por guardar informações referentes à página ou referente aos usuários do sistema.

Trabalhar com arquivos, significa que devemos realizar no mínimo duas operações: abrir e fechar um arquivo.

9.1 Criando e abrindo um arquivo

No PHP vamos utilizar a função `fopen`. Sua sintaxe é simples. Precisamos passar o caminho (*path*) e o nome do arquivo a ser criado além do modo de abertura do arquivo. Veja a sintaxe:

```
1 <?php
2 // Sintaxe da funcao fopen
3 $identificador = fopen(<caminho/nome_do_arquivo>, modo);
4 ?>
```

Os modos de abertura de um arquivo que o desenvolvedor pode utilizar são listados abaixo. É importante que o programador entenda sobre cada um dos modos, para a correta utilização dos arquivos. Será apresentado apenas 3 modos (os mais utilizados).

- **r** (*read*): este modo abre o arquivo somente para leitura, ou seja, o programador não vai conseguir escrever nenhum dado no arquivo enquanto ele estiver aberto nesse modo;
- **w** (*write*): abre o arquivo somente para escrita, caso o arquivo não exista, tenta criá-lo (é necessário que o diretório tenha permissão de escrita); e
- **a+** (*append*): abre o arquivo para leitura e escrita, caso o arquivo não exista, o PHP tentará criá-lo.

Importante!

Para trabalharmos com arquivos é sempre importante verificarmos se a pasta ou o arquivo tem permissões dentro do sistema operacional, caso isso não aconteça, o arquivo não será criado, lido ou até mesmo gravado. Portanto, vamos mudar as permissões da pasta `aula08`. Execute os comandos a seguir a partir do terminal do Linux:

```
1 $ mkdir -p /var/www/aula08
2 $ chmod 777 /var/www/aula08
```

Vamos iniciar criando um arquivo de texto na pasta da nossa aula, no caso, `/var/www/aula08`. Em seguida escreva no arquivo `index.php` o seguinte conteúdo:

```
1 <?php
2     echo "<h1>Trabalhando com arquivos</h1>" . "<br />";
3     $fid = fopen("/var/www/aula08/disciplinas.txt", "w");
4     ?>
```

Esse código irá criar o arquivo `disciplinas.txt` caso ele não exista e vai retornar o identificador para a variável `$fid` (que significa *file identifier* em inglês ou identificador de arquivo em português). Ao acessar o endereço `<http://localhost/aula08/>` o seu navegador vai apresentar apenas a mensagem “Trabalhando com arquivos”, porém, se verificarmos o diretório `/var/www/aula08`, vamos visualizar o arquivo “disciplinas.txt”.

9.2 Gravando em um arquivo

Após o uso do comando `fopen`, temos o identificador apontando para o arquivo. Com ele podemos fazer alterações ou manipulações. Nessa seção vamos gravar dados dentro do arquivo com o uso do comando `fwrite`. Veja a sintaxe abaixo:

```
1 <?php
2     // Sintaxe da funcao fwrite
3     fwrite(identificador, "texto");
4     ?>
```

Os parâmetros da função `fwrite` são descritos abaixo:

- **identificador** (*handle*): variável identificadora do arquivo. Essa variável é retornada pela função `fopen`; e
- **texto** (*string*): texto a ser gravado no arquivo.

Vamos modificar o arquivo `index.php` para conter o seguinte conteúdo:

```
1 <?php
2     echo "<h1>Trabalhando com arquivos</h1>" . "<br />";
```

```
3 $fid = fopen("/var/www/aula08/disciplinas.txt", "w");
4 fwrite($fid, "programacao web\r\n");
5 ?>
```

O segundo parâmetro - o texto a ser gravado - recebeu ao final da *string* os caracteres `\r\n`. Esses caracteres fazem com que o PHP “aperte um Enter” ao salvar o arquivo.

9.3 Fechando um arquivo

Todo arquivo aberto deve ser fechado após sua utilização. Caso contrário, ele não poderá ser reutilizado em algum outro local do código. Para fechar um arquivo você deve passar o identificador para a função `fclose`.

```
1 <?php
2 // Sintaxe da funcao fclose
3 fclose(identificador);
4 ?>
```

9.4 Lendo de um arquivo

Após abrirmos um arquivo, outra operação que podemos efetuar é a leitura do conteúdo existente no arquivo. Essa operação é feita linha por linha. Podemos ler os valores existentes nos arquivos usando diversas funções. Nessa apostila veremos essa funcionalidade com a função `file`.

9.4.1 Comando `file`

O comando `file` ler um arquivo e retorna um *array* com todo o seu conteúdo. Cada posição do *array* representa uma linha do arquivo começando pelo índice 0. Veja a sintaxe a seguir:

```
1 <?php
2 // Sintaxe da funcao file
3 $variavel = file(<caminho/nome_do_arquivo>);
4 ?>
```

Seguindo o exemplo do arquivo `index.php`, adicione, usando o Sublime Text, outras disciplinas. O código responsável por realizar a leitura do arquivo é apresentado em seguida.

```
1 programacao web
2 logica de programacao
3 banco de dados
4 software livre
```

```
1 <?php
2     $linhas = file("/var/www/aula08/disciplinas.txt");
3     foreach ($linhas as $linha) {
4         echo $linha . "<br />";
5     }
6 ?>
```

9.5 Trabalhando com arquivos .csv

Os arquivos .csv são arquivos de formato regulamentado, ou seja, eles tem uma estrutura certa a ser seguida. CSV significa *Comma-Separated Values* que traduzindo para o português é: valores separados por vírgula. Um exemplo de arquivo .csv pode ser visualizado abaixo:

```
1 id,nome,endereco,cidade
```

A primeira linha do arquivo diz respeito ao cabeçalho, é nessa linha que informamos quais campos serão gravados. No nosso exemplo, os campos que vamos guardar são: **id**, **nome**, **endereco** e **cidade**. Nas linhas seguintes nós vamos guardar os **registros**, ou seja, as informações. Veja o arquivo `informacoes.csv` com a adição de informações dos usuários.

```
1 id,nome,endereco,cidade
2 1,Joaquim Silva,"rua A, n. 10",Fortaleza
3 2,Maria Lima,"rua B, n. 13",Fortaleza
```

Crie esse arquivo com o aplicativo Sublime Text e salve com o nome `informacoes.csv` no diretório `/var/www/aula08`. Iremos trabalhar com ele nos próximos exemplos.

O código a seguir abre o arquivo `informacoes.csv` no modo `r`. Isso indica que o programa será capaz de trabalhar com o arquivo de uma maneira: apenas lendo o arquivo.

9.5.1 Função `str_getcsv`

O código abaixo realiza a leitura do arquivo .csv e guarda na variável `$dados` os valores de cada linha.

```
1 <?php
2     $linhas = file("/var/www/aula08/informacoes.csv");
3     $dados = array();
4
5     foreach ($linhas as $linha) {
6         $dados = str_getcsv($linha);
7         print_r($dados);
8         echo "<br />";
9     }
10 ?>
```

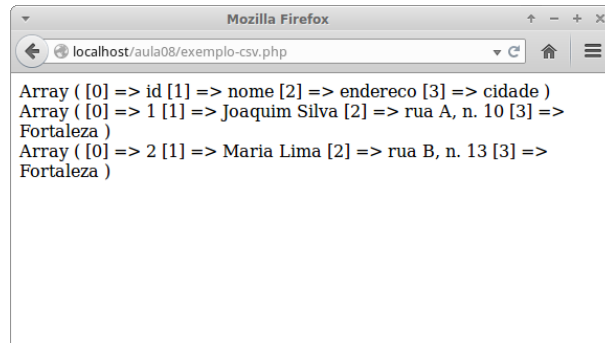


Figura 9.5.1 – Resultado do código acima.

Desafio rápido!

Altere o código exemplificado na seção 9.5.1 para exibir os dados em uma tabela. O resultado final esperado é o apresentado na figura abaixo:

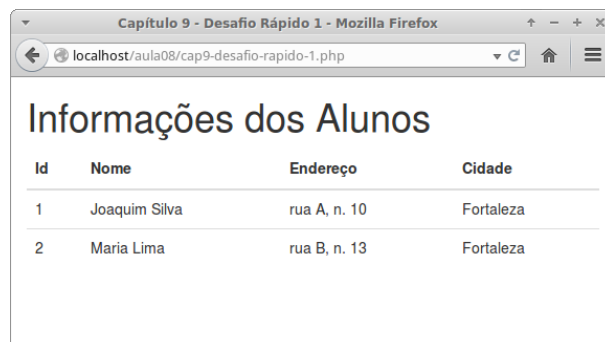


Figura 9.5.2 – Resultado esperado do desafio rápido.

9.6 Manipulando diretórios

Existem alguns comandos básicos para manipulação de diretórios. Mostraremos na apostila, apenas como obter o diretório atual, ou seja, o diretório em que o arquivo `.php` está sendo executado. Outras operações como criar e apagar um diretório são explicadas na documentação oficial do PHP.

9.6.1 Comando `getcwd`

Essa função retorna o diretório atual do arquivo PHP que está sendo executado. Por exemplo, se existe o arquivo `/var/www/aula08/index.php` e nesse arquivo `index.php` estiver escrito `$diretorio = getcwd()`, então o valor da variável `$diretorio` será `/var/www/aula08`. Reescreva o código abaixo.

```
1 <?php
2     $diretorio = getcwd();
3     echo $diretorio;
4 ?>
```

Desafio rápido!

Crie um diretório dentro de `/var/www/aula08` com seu nome. Por exemplo:
📁 `/var/www/aula08/joaquim`. Salve o código acima com o nome `index.php` dentro desse novo diretório. Qual o resultado?

9.7 Exercícios

9.8 Desafio!

10 Interações com o navegador

Ao final deste capítulo, o aluno terá as seguintes competências:

1. Acessar informações do navegador;
2. Criar e acessar dados em cookies;
3. Criar e acessar dados em sessões;

10.1 Variável `$_SERVER`

A linguagem de programação PHP também permite acessar informações do navegador (*browser*) automaticamente. Isso pode ser muito útil quando queremos coletar informações sobre o cliente, como por exemplo, o tipo de navegador, qual o sistema operacional ou outras informações. O código a seguir mostra informações sobre o navegador do usuário:

```
1 <html>
2     <head>
3         <title>Apostila de PHP</title>
4     </head>
5     <body>
6         <?php echo $_SERVER['HTTP_USER_AGENT'] ?>
7     </body>
8 </html>
```

`$_SERVER` é uma variável global do PHP, do tipo *array*, que armazena uma série de informações sobre o servidor e o cliente, como endereço do servidor, IP do servidor, nome do arquivo que está em execução etc. No caso, a chave `HTTP_USER_AGENT` exibe as informações sobre o navegador. Abaixo um exemplo desse retorno:

10.2 *Cookies*

cookies são mecanismos para armazenar e consultar informações. Eles são armazenados na máquina do cliente, ou seja, o *cookie* está armazenado na máquina que acessou o site. Os *cookies* possuem várias atribuições que são definidas pelo programador.

Por exemplo, Joaquim, aluno do e-jovem, vai ajudar sua mãe a comprar alguns eletrodomésticos, por isso acessou uma loja virtual, escolheu uma geladeira e um fogão, ou seja, adicionou eles ao carrinho de compras virtual da loja. Por algum motivo, Joaquim teve que sair e desligou o computador. Mais tarde ao voltar pra casa, volta a entrar na loja virtual e percebe que os itens adicionados ainda estão no carrinho de compras.

Esse é um exemplo básico de uso de *cookies* no desenvolvimento de sites. O site guardou a informação que os itens geladeira e fogão tinham sido adicionados ao carrinho. Quando Joaquim acessou o site pela segunda vez, essas informações foram enviadas ao servidor do site, processadas e por fim, apresentadas ao usuário.

Os *cookies* são armazenados em uma pasta específica do navegador, por isso não é possível encontrá-lo na pasta “Meus Documentos” ou no diretório `/home/aluno` por exemplo. Mesmo assim, pessoas mal intencionadas podem acessá-los. Por conta disso, o uso de *cookies* **não** é recomendado quando se trata de informações sigilosas. O usuário tem ainda a opção “habilitar *cookies*” que pode ser desativada a qualquer momento pelo visitante, impedindo assim, a criação de *cookies* no sistema. Mas em cada navegador essa opção pode mudar de nome.

A linguagem PHP atribui *cookies* utilizando a função `setcookie`. Essa função deve ser incluída antes da tag `<html>` num arquivo `.php`.

A sintaxe do comando `setcookie` possui muitos parâmetros. Abaixo está representada a função com alguns desses parâmetros.

```
1 <?php
2 // Sintaxe da funcao setcookie
3 setcookie("nome_do_cookie", "valor_do_cookie", "tempo_de_vida");
4 ?>
```

A tabela abaixo relaciona cada um dos parâmetros com suas respectivas funções.

INCLUIR TABELA

Observe no exemplo abaixo a criação de um *cookie* através da função `setcookie`:

```
1 <?php
2 $valor = "valores a serem gravados";
3 setcookie("Teste_Cookie", $valor);
4 ?>
```

Criamos uma variável na linha `??`, em seguida a função `setcookie` recebe como parâmetro somente o seu nome e o valor a ser gravado.

Usaremos o navegador Mozilla Firefox para visualizar o *cookie* criado. Acesse o site `<http://localhost>` na barra de endereços do navegador. Em seguida, você pode pressionar as teclas `[CTRL]+[I]`, selecionar a aba “Segurança” em seguida selecionar o botão “Exibir cookies”. Lembre-se de criar o código acima primeiro e depois acessar a página pelo navegador de sua máquina. A imagem abaixo deverá ser exibida no seu monitor.

INCLUIR PRINTSCREEN!!

Veja que outras informações como caminho, enviar, e validade não foram especificados, porém aparecem na tela mesmo assim. No exemplo abaixo o programador atribui o tempo de vida do *cookie*. para isso será necessário capturar o dia e a hora com a função

`time` somar o tempo adicional de vida do *cookie*. Esse tempo deve ser informado em segundos. Isso faz com que o *cookie* exista na máquina do cliente de acordo com a quantidade de tempo determinado pelo programador. Verifique o exemplo abaixo.

```
1 <?php
2     $valor = "valores a serem gravados";
3     $tempo = time() + 3600;
4     setcookie("Teste_Cookie", $valor, $tempo);
5
6     $valor = "valores a serem gravados";
7     $tempo = time() + 3600;
8     setcookie("Teste_Cookie", $valor, $tempo);
9 ?>
```

Esse *cookie* tem a validade de 3600 segundos, ou seja 1 hora, sabendo disso e supondo que o usuário fez o acesso às 14:47:36, o *cookie* tem validade até às 15:47:36. Isso é muito importante para a programação dos *cookies*. Se quisermos que ele exista por um determinado tempo, temos que calcular tudo em segundos. Veja um outro exemplo a seguir.

```
1 <?php
2     $valor = "valores a serem gravados";
3     $tempo = time() + (3600*24*7);
4     setcookie("Teste_Cookie", $valor, $tempo);
5 ?>
```

Esse *cookie* tem seu tempo de vida de 7 dias, pois 3600 segundos = 1 hora, 24 horas = 1 dia e 7 * horas_de_um_dia resulta em 7 dias.

10.2.1 Acessando Cookies

É possível acessar os valores dos *cookies* criados anteriormente. Para isso, basta utilizar a variável `$_COOKIE['nome_do_cookie']`, veja um exemplo:

```
1 <?php
2     $valor = "valores a serem gravados";
3     $tempo = time() + (3600*24*7); // tempo de vida
4     setcookie("Teste_Cookie", $valor, $tempo); // grava o valor
5     echo $_COOKIE['Teste_Cookie']; // imprime
6 ?>
```

Observe agora um exemplo de um código utilizado para contar as visitas de um site usando *cookie*:

```
1 <?php
2     $valor = 1;
3     $tempo = time() + (3600*24*7); // tempo de vida
4     $valor = $_COOKIE['contador']+1; // inicia a contagem
```

```
5  setcookie("contador", $valor, $tempo); // grava o valor
6  echo $_COOKIE['contador']." visitas no site!"; // imprime
7  ?>
```

O resultado na tela é de acordo com a quantidade de vezes que o cliente entrou no site ou atualizou o mesmo.

10.3 Sessão

Sessões são mecanismos muito parecidos com os tradicionais *cookies*. A principal diferença é que as sessões são armazenadas no próprio servidor.

As sessões são métodos de manter (ou preservar) determinados dados no servidor, ao invés do computador do usuário. A sessão fica ativa (mantendo os dados armazenados) enquanto o navegador estiver aberto, ou enquanto a sessão não expirar (por inatividade, ou por conta da lógica do programa). Para criarmos uma sessão utilizaremos a função abaixo:

```
1  <?php
2  session_start();
3  ?>
```

Dessa forma estamos iniciando um conjunto de regras. Essa função deve sempre está no início do seu programa, com exceção de algumas regras. Agora trabalharemos com essa sessão, primeiro podemos determinar o tempo de vida da sessão com o seguinte comando:

```
1  <?php
2  session_cache_expire(5);
3  session_start();
4  ?>
```

Neste caso, a função `session_cache_expire` vem antes da função `session_start`. O código faz o seguinte: primeiro ele avisa que a sessão, quando iniciada, deve expirar em 5 minutos, e depois a inicia. Com o uso da variável global `$_SESSION` podemos gravar valores na sessão, veja um exemplo:

```
1  <?php
2  session_start();
3  $var = "Sessao Criada!";
4  $_SESSION['minha_sessao'] = $var;
5  ?>
```

Criamos na sessão uma chave com o nome `minha_sessao` e atribuímos a ela o valor gravado na variável do tipo *string* chamada `$var`. Essas informações ficam gravadas no servidor, em seguida é possível resgatar o valor da seguinte forma:

```
1 <?php
2     session_start();
3     echo $_SESSION['minha_sessao'];
4 ?>
```

Observe que `$_SESSION` tem seu tratamento igual a uma variável do tipo *array*. Para resgatar o valor gravado na sessão, basta fazer a chamada a essa variável `$_SESSION` passando como chave, o nome especificado, no exemplo: `minha_sessao`.

Abaixo temos um exemplo com o uso da função `isset`, que verifica se uma variável existe ou não, retornando um valor *booleano* (`true` ou `false`):

```
1 <?php
2     session_start();
3     if (isset($_SESSION['minha_sessao'])) {
4         echo "Sessao Ativa";
5     } else {
6         echo "Sessao nao existe!";
7     }
8 ?>
```

Para desativarmos uma sessão podemos utilizar dois tipos de mecanismos: um deles é o uso da função `session_destroy` que tem como finalidade destruir a sessão por completo, e todas as informações nela contidas. Outra forma é apagarmos apenas uma informação (variável) gravada na sessão, com o uso da função `unset`. Veja abaixo exemplos de uso das funções `session_destroy` e `unset`.

```
1 <?php
2     session_start();
3     session_destroy();
4 ?>
```

Usamos `unset` quando queremos desativar determinada sessão, O exemplo abaixo destrói a sessão especificada:

```
1 <?php
2     session_start();
3     unset($_SESSION['minha_sessao']);
4 ?>
```

Dessa forma destruímos a informação (variável) `minha_sessao`, armazenada na sessão ativa, mantendo as demais informações intactas.

10.4 Exercícios

10.5 Desafio!

11 Tratamentos de erros

Ao final deste capítulo, o aluno terá as seguintes competências:

1. Entender os tipos de erro e suas possíveis causas;
2. Tratar os erros;

11.1 Tipos de Erro

Durante o processo de programação é comum surgirem erros que interrompem, causam um mal funcionamento ao programa ou apresentam um resultado inesperado. Normalmente esses erros são provocados pelo próprio programador, e podem ser:

- **Erro de sintaxe:** quando o programador escreve uma instrução da forma errada, por exemplo, esquece de colocar um “;” (ponto-e-vírgula) no final da linha;
- **Erro de semântica:** quando o programador escreveu um código correto mas não previu alguma outra situação ou dependência para que a instrução programada funcione. Exemplo: ao programar a abertura de um arquivo, não verificou antes se o arquivo realmente existe;
- **Erros de lógica:** quando o programador não analisa corretamente o fluxo lógico do que se espera que o programa execute.

Existem também os erros provocados por fatores externos, alheios ao programador, em relação a própria linguagem, ao computador, servidores, sistemas operacionais etc. As linguagens de programação possuem mecanismos para interromper o programa em caso de erro, e o PHP oferece um conjunto de tipos de erro para alertar ao programador e, assim, auxiliá-lo na identificação da causa.

Geralmente o PHP apresenta os seguintes tipos de mensagens de erro (mais comuns):

- **NOTICE:** quando alguma regra da linguagem não é seguida corretamente, que pode ser um erro mas o programa segue sua execução normalmente, serve apenas como alerta ao programador;
- **WARNING:** quando há realmente um erro, mas não-fatal, ou seja, o programa continua em execução;
- **ERROR:** erro fatal, o programa é interrompido.

O programador PHP deve estar sempre atento a essas mensagens de erro, ler, interpretar e solucionar o problema indicado na mensagem. Porém, quando o programa estiver “no ar”, ou seja, em produção, com usuários de verdade fazendo uso do programa,

é importante que essas mensagens de erro sejam suprimidas, até porque essas mensagens de erro só interessam ao programador.

Para exibir a mensagem direta no navegador procure ativar a opção `display_errors`, as funções serão ativadas para *On* (ligada) ou *Off* (desligada). O arquivo `php.ini` no sistema operacional Linux geralmente fica no diretório: `/etc/php5/apache2/php.ini`. Por padrão a exibição das mensagens está desligada, porém, durante o desenvolvimento é interessante permitir que o PHP mostre as mensagens.

Veja o Exemplo:

```
1 <?php
2     strtolower();
3 ?>
```

A função `strtolower` recebe um texto e retorna o mesmo texto com letras minúsculas. Logo, se a função precisa receber um parâmetro e, no exemplo, esse parâmetro não foi fornecido, certamente vai gerar um erro:

Warning: strtolower() expects exactly 1 parameter, 0 given in /var/www/aula10/index.php on line 2.

Essa mensagem será exibida se a opção `display_errors` estiver setada para *On* no arquivo `php.ini`.

Caso não queira ou não possa modificar o arquivo `php.ini`, é possível, fazer com que o erro seja exibido com a função `ini_set`, da seguinte maneira: `ini_set('display_errors', 1)`, seguida da função `error_reporting`, da seguinte maneira: `error_reporting(E_ALL)`. Essas duas linhas indicam que tipo de erro o PHP deve exibir. O argumento `E_ALL`, neste caso, indica que qualquer erro deve ser exibido. Por exemplo a função `error_reporting(E_WARNING)` vai exibir apenas os erros do tipo *warning*.

```
1 <?php
2     ini_set('display_errors', 1);
3     error_reporting(E_WARNING);
4     strtolower();
5 ?>
```

11.2 Exercícios

11.3 Desafio!

12 Introdução a POO com PHP

Ao final deste capítulo, o aluno terá as seguintes competências:

1. Entender os conceitos da Programação Orientada a Objetos;
2. Criar classes com atributos e métodos;
3. Criar objetos;
4. Herdar classes;

12.1 O que é programação orientada a objetos?

A programação Orientada a Objetos (POO) é um estilo de programação, ou uma forma de programar, que ajuda a agrupar tarefas a respeito de um mesmo tema, organizar o código e entender como estruturar melhor a solução de um problema de modo que fique mais fácil também, posteriormente, fazer mudanças e dar manutenção ao programa.

Usar esse estilo de programação orientado a objeto requer entender alguns conceitos. Como o próprio nome já diz, orientado a objetos indica que vamos usar “objetos”, e esse é o primeiro conceito que vamos ver. **Objeto é a representação de algo que podemos atribuir características e ações.** Por exemplo, um carro é um “objeto”, podemos dar a ele características (cor, modelo, fabricante etc.) e ações (buzinar, acelerar, estacionar etc.). Uma pessoa também pode ser representada num sistema como um objeto porque tem características (cor do cabelo, altura, peso, etc.) e executa ações (falar, andar, correr, dormir, etc.).

Da mesma forma em um *site*, podemos dizer que o usuário do *site* é um objeto, um cliente, um produto ou serviço, uma foto, um vídeo etc. Em um comércio eletrônico, cada produto a venda é um “objeto”, assim como o cliente que está comprando o produto e o carrinho de compras criado pelo cliente, são todos “objetos”. Em uma rede social, cada foto, amigo, postagem, são objetos.

Objeto, portanto, é um conceito e também um modo de pensar a programação, organizando e programando cada componente de uma forma separada (pensando em objetos), mas fazendo com que suas ações possam interagir com outros objetos, formando um sistema. Por exemplo, um objeto “pessoa”, através de uma ação de “dirigir” pode interagir com um objeto “carro”.

Um aspecto interessante da Programação Orientada a Objetos é o reuso. Reuso significa reutilizar, aproveitar o que já foi programado em um objeto para ser usado por outro objeto. Por exemplo, um carro e uma moto, apesar de serem objetos diferentes, compartilham de algumas características e ações. O carro e a moto tem peso, tem um

fabricante, tem pneus, precisam ser abastecidos, são conduzidos por uma pessoa etc. Logo, se é programado a função de “abastecer”, essa única função serve para os dois (carro e moto), não é preciso repetir o mesmo código para os dois tipos de objetos.

12.2 Classes e Objetos

Na Programação Orientada a Objeto o programador precisa desenvolver um modelo que vai ser usado como base para a criação de um objeto, com todas as suas características e comportamento, antes de usar esse objeto no sistema. Esse modelo do objeto chamamos de “Classe”, e contém a definição de um determinado objeto, ou como esse objeto vai funcionar.

Uma “classe” é, portanto, um modelo de um objeto, com a programação de suas características (atributos) e ações (métodos).

Em PHP, para criar uma classe usamos o comando `class`. Exemplo:

```
1 <?php
2     class Carro {
3         // código da classe
4     }
5 ?>
```

No código anterior, usamos o comando `class` para criar uma nova classe chamada “Carro”. Essa classe vai ser um modelo para todos os objetos do tipo “carro”.

Normalmente, como um meio de organizar melhor o sistema, colocamos cada classe em um arquivo `.php` diferente, mas não é necessário fazer isso por enquanto nesta aula, veremos no próximo capítulo como funciona essa organização dos arquivos.

Após programarmos uma classe, podemos criar um ou mais objetos com base nessa classe. O processo de criar um objeto com base nessa classe é chamada de “instanciação”. Então, criar um objeto significa instanciar uma classe. Podemos fazer isso com o comando `new`, da seguinte forma:

```
1 <?php
2     $meucarro = new Carro();
3 ?>
```

Nesse código anterior criamos uma variável `$meucarro`, e essa variável é um objeto do tipo “Carro”, ou seja, um objeto criado com base na classe “Carro”. Ainda com base na mesma classe podemos criar vários objetos, todos com as mesmas características e ações programadas na classe:

```
1 <?php
2     $palio = new Carro();
3     $fiesta = new Carro();
```

```
4 ?>
```

12.3 Atributos

No exemplo anterior criamos uma classe “Carro” mas não programamos mais nada. Vamos agora ver como podemos incluir e usar os atributos da classe. Atributo é o nome dado a uma característica ou propriedade da classe, que vai estar disponível a todos os objetos instanciados a partir dessa classe. Seguindo o mesmo exemplo, podemos ter uma classe Carro com os atributos de cor e marca.

```
1 <?php
2     class Carro {
3         public $cor;
4         public $marca;
5     }
6 ?>
```

Usamos o comando `public` seguido do nome de uma variável (`$cor` e `$marca`). Essas variáveis, dentro da classe, se tornam atributos da classe. Neste exemplo não foi inserido nenhum valor para essas variáveis, ou seja, são atributos sem valor padrão, os valores vão ser colocados depois do objeto criado, onde cada objeto pode ter uma cor e uma marca diferente.

Vamos agora instanciar essa classe, criando um objeto e atribuindo os valores:

```
1 <?php
2     $fiesta = new Carro(); // cria um novo objeto do tipo Carro
3     $fiesta->cor = "vermelho"; // define o valor "vermelho" para o atributo cor
4     $fiesta->marca = "Ford"; // define o valor "Ford" para o atributo marca
5
6     $palio = new Carro(); // cria um segundo objeto tambem do tipo Carro
7     $palio->cor = "amarelo";
8     $palio->marca = "Fiat";
9 ?>
```

Depois da classe Carro programada com dois atributos (cor e marca), podemos criar objetos com base nessa classe e usar em cada um deles os atributos disponíveis. No exemplo, `$palio` e `$fiesta` são dois objetos com as características de cor e marca, conforme descrito na classe, e cada um deles é um objeto distinto. Quando definimos “amarelo” ao atributo “cor” do objeto `$palio`, podemos por exemplo exibir com o comando `echo`:

```
1 <?php
2     $palio = new Carro();
3     $palio->cor = "amarelo";
4     $palio->marca = "Fiat";
```

```
5
6     echo $palio->cor; // exibe o resultado "amarelo"
7     ?>
```

Observe que para os atributos de um objeto usamos o sinal de (->), e o nome do atributo não tem o sinal de \$, ficando apenas no nome do objeto.

Veja outro exemplo, dessa vez os atributos da classe Carro com alguns valores pré definidos:

```
1  <?php
2      class Carro {
3          public $cor = "branco";
4          public $marca;
5          public $airbag = true;
6      }
7
8      $fusca = new Carro();
9      echo $fusca->cor; // exibe o resultado "branco"
10     ?>
```

12.4 Métodos

Como visto anteriormente, uma classe tem atributos (suas características) e métodos (suas funções). Já vimos como trabalhar com os atributos, agora vamos conhecer os métodos, ou o que um objeto pode fazer.

Continuando com o exemplo da classe Carro, vamos imaginar que além dos atributos cor e marca, tem também uma função para dar a partida. Esse método da classe é construído da mesma forma que uma função do PHP:

```
1  <?php
2      class Carro {
3          public $cor;
4          public $marca;
5
6          public function ligar() {
7              echo "Carro ligado";
8          }
9      }
10     ?>
```

Agora sim, temos uma visão completa de uma classe, com seus atributos e métodos. Vamos criar um objeto Carro:

```
1  <?php
2      $meucarro = new Carro();
```

```

3         $meucarro->cor = "azul";
4         $meucarro->marca = "Chevrolet";
5         $meucarro->ligar();                                     // exibe "Carro
6     ?>

```

Observe que para executar um método do objeto usamos novamente o sinal de (->) seguido do nome do método e depois os parênteses, já que como foi visto anteriormente no curso, se trata de uma função. Essa função `ligar()` do exemplo apenas exibe uma mensagem, mas podemos programa-la para fazer outras coisas, como mudar os atributos do próprio objeto.

Outro exemplo, dessa vez criando um método que exibe os atributos do objeto:

```

1  <?php
2      class Carro
3      {
4          public $cor;
5          public $marca;
6          public $dono;
7          public $placa;
8
9          public function info()
10         {
11             echo $this->dono." tem um carro ".$this->cor." da ".$this->placa;
12         }
13     }
14
15     $meucarro = new Carro();                                     //instanciando a classe Carro
16     $meucarro->cor = "amarelo";                                   // setando os atributos
17     $meucarro->marca = "Fiat";
18     $meucarro->dono = "Joaquim";
19     $meucarro->info();                                           // exibe "Joaquim tem u
20 ?>

```

Vamos ver um exemplo onde ao invés de exibir uma mensagem, o método `ligar()` muda um atributo.

```

1  <?php
2      class Carro {
3          public $cor;
4          public $marca;
5          public $ligado = "nao"; // atributo com valor padrao = "nao"
6
7          public function ligar() {
8              $this->ligado = "sim"; // muda o valor do atributo para "sim"
9          }
10         public function desligar() {
11             $this->ligado = "nao"; // muda o valor do atributo para "nao"

```

```

12         }
13     }
14     ?>

```

Agora temos o termo `$this`, que é usado somente dentro de uma classe e faz referência a algum método ou atributo da própria classe. No caso, o método `ligar()`, quando executado, vai mudar o valor do atributo “ligado” de “não” para “sim”. Vamos ver nosso carro funcionando:

```

1  <?php
2      ...
3      $meucarro = new Carro();
4      $meucarro->cor = "verde";
5      $meucarro->marca = "Fiat";
6      echo $meucarro->ligado;           // exibe "nao", valor padrao da classe
7      $meucarro->ligar();               // executamos o metodo ligar()
8      echo $meucarro->ligado;           // agora exibe "sim", valor alterado pelo m
9      $meucarro->desligar();            // executamos o metodo desligar()
10     echo $meucarro->ligado;           // agora exibe "nao", valor alterado pelo m
11     ?>

```

Claro que podemos melhorar essa classe, por exemplo, alterando o tipo de atributo “ligado” para booleano (`true` ou `false`) e inserindo uma lógica para que o carro, depois de ligado, não possa ser ligado novamente, o mesmo para o método de desligar.

```

1  <?php
2      class Carro {
3          public $cor;
4          public $marca;
5          public $ligado = false;           // atributo
6
7          public function ligar() {
8              if ($this->ligado) {           // verifica
9                  echo "O carro ja esta ligado"; // se o atr
10             }
11             else {
12                 $this->ligado = true;       // caso con
13             }
14         }
15         public function desligar() {
16             // tente fazer este metodo desligar com a mesma logica
17         }
18     }
19     ?>

```

12.5 Herança

Um outro conceito dentro da programação orientada a objetos é o de Herança. Herança significa que é possível criar uma classe que herda atributos e métodos de outra classe. O sentido disso é organizar mais ainda, de forma que um código possa ser reutilizado, facilitando o desenvolvimento.

Vamos imaginar, pegando o exemplo anterior do Carro, que existem vários tipos de carro diferentes, e cada categoria tem suas próprias características. Por exemplo, carros de corrida, ônibus, jipe, buggy etc. Então, seria complicado uma única classe representando todos os carros, porque existem categorias com características próprias, mas ao mesmo tempo eles compartilham de características semelhantes. Então, vamos deixar essas características semelhantes em um único lugar, e aproveitar essas informações para serem usadas em outras classes.

Vamos analisar essa seguinte classe:

```
1  <?php
2      class Carro {
3          public $cor;
4
5          public function ligar() {
6              echo "Carro ligado";
7          }
8          public function desligar() {
9              echo "Carro desligado";
10         }
11     }
12  ?>
```

Esta classe é uma classe genérica de um Carro, podemos dizer que todo carro tem uma cor, pode ser ligado e desligado. Podemos criar objetos a partir dessa classe para outros carros. Porém, um carro de Fórmula 1, por exemplo, pode ter outros atributos e métodos que um carro convencional não tem. Para aproveitarmos o que já é comum a todos os carros, podemos criar uma nova classe para a categoria de Fórmula 1 usando (herdando) a classe genérica de Carro e incluindo o que for específico da categoria. Vamos ver como isso funciona:

```
1  <?php
2      class Formula1 extends Carro {
3          # codigo
4      }
5  ?>
```

Observe a palavra **extends**, que é um comando para criar a herança. Este código cria uma nova classe Formula1 como uma extensão da classe Carro. A classe Formula1 é

como se fosse a classe “filha” da classe `Carro`. Vamos criar um objeto para `Formula1`:

```
1 <?php
2     $ferrari = new Formula1();
3     $ferrari->ligar();                      // exibe o resultado "Carro ligado"
4 ?>
```

Mesmo sem programar nada na classe `Formula1`, por ela ser “filha” de `Carro` ela herda seus atributos e métodos, por isso podemos usar a função `ligar()` em um objeto do tipo `Formula1`. Agora vamos incrementar essa classe `Formula1`:

```
1 <?php
2     class Formula1 extends Carro {
3         public $piloto;
4         public $equipe;
5         public $velocidade = 0;
6     }
7 ?>
```

No exemplo anterior, incluímos 3 novos atributos a classe `Formula1`, piloto, equipe e velocidade, sendo a velocidade iniciada com o valor 0. Agora a classe `Formula1` tem seus próprios atributos, além daqueles que vem herdados da classe `Carro`, ou seja, a cor. Vamos incluir um método para aumentar a velocidade do `Formula1`:

```
1 <?php
2         class Formula1 extends Carro {
3             public $piloto;
4             public $equipe;
5             public $velocidade = 0;
6
7             public function acelerar() {
8                 $this->velocidade = $this->velocidade + 100;
9             }
10        }
11
12        $ferrari = new Formula1();
13        $ferrari->piloto = "Schumacher";
14        $ferrari->equipe = "Ferrari";
15        $ferrari->cor = "vermelho";
16        $ferrari->ligar();                      // exibe o resultado "Carro ligado"
17        $ferrari->acelerar();                  // executa acelerar, aumentando em 100 a ve
18        echo $ferrari->velocidade;            // exibe "100";
19 ?>
```

Dessa forma, usando a Herança, se for necessário mudar alguma coisa no método de ligar o carro, essa mudança vai automaticamente refletir em todas as classes que herdaram da classe `Carro` esse método. Esse processo torna a programação mais intuitiva e fácil de manter.

12.6 Exercícios

12.7 Desafio!

13 Conclusão

Sed consequat tellus et tortor. Ut tempor laoreet quam. Nullam id wisi a libero tristique semper. Nullam nisl massa, rutrum ut, egestas semper, mollis id, leo. Nulla ac massa eu risus blandit mattis. Mauris ut nunc. In hac habitasse platea dictumst. Aliquam eget tortor. Quisque dapibus pede in erat. Nunc enim. In dui nulla, commodo at, consectetur nec, malesuada nec, elit. Aliquam ornare tellus eu urna. Sed nec metus. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

Phasellus id magna. Duis malesuada interdum arcu. Integer metus. Morbi pulvinar pellentesque mi. Suspendisse sed est eu magna molestie egestas. Quisque mi lorem, pulvinar eget, egestas quis, luctus at, ante. Proin auctor vehicula purus. Fusce ac nisl aliquam ante hendrerit pellentesque. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Morbi wisi. Etiam arcu mauris, facilisis sed, eleifend non, nonummy ut, pede. Cras ut lacus tempor metus mollis placerat. Vivamus eu tortor vel metus interdum malesuada.

Sed eleifend, eros sit amet faucibus elementum, urna sapien consectetur mauris, quis egestas leo justo non risus. Morbi non felis ac libero vulputate fringilla. Mauris libero eros, lacinia non, sodales quis, dapibus porttitor, pede. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Morbi dapibus mauris condimentum nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Etiam sit amet erat. Nulla varius. Etiam tincidunt dui vitae turpis. Donec leo. Morbi vulputate convallis est. Integer aliquet. Pellentesque aliquet sodales urna.

Referências

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. *NBR 6028*: Resumo - apresentação. Rio de Janeiro, 2003. 2 p. Citado na página [2](#).

Apêndices

APÊNDICE A – Instalação de ambiente de desenvolvimento no Windows

Nunc velit. Nullam elit sapien, eleifend eu, commodo nec, semper sit amet, elit. Nulla lectus risus, condimentum ut, laoreet eget, viverra nec, odio. Proin lobortis. Curabitur dictum arcu vel wisi. Cras id nulla venenatis tortor congue ultrices. Pellentesque eget pede. Sed eleifend sagittis elit. Nam sed tellus sit amet lectus ullamcorper tristique. Mauris enim sem, tristique eu, accumsan at, scelerisque vulputate, neque. Quisque lacus. Donec et ipsum sit amet elit nonummy aliquet. Sed viverra nisl at sem. Nam diam. Mauris ut dolor. Curabitur ornare tortor cursus velit.

Morbi tincidunt posuere arcu. Cras venenatis est vitae dolor. Vivamus scelerisque semper mi. Donec ipsum arcu, consequat scelerisque, viverra id, dictum at, metus. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut pede sem, tempus ut, porttitor bibendum, molestie eu, elit. Suspendisse potenti. Sed id lectus sit amet purus faucibus vehicula. Praesent sed sem non dui pharetra interdum. Nam viverra ultrices magna.

Aenean laoreet aliquam orci. Nunc interdum elementum urna. Quisque erat. Nullam tempor neque. Maecenas velit nibh, scelerisque a, consequat ut, viverra in, enim. Duis magna. Donec odio neque, tristique et, tincidunt eu, rhoncus ac, nunc. Mauris malesuada malesuada elit. Etiam lacus mauris, pretium vel, blandit in, ultricies id, libero. Phasellus bibendum erat ut diam. In congue imperdiet lectus.

APÊNDICE B – Quisque libero justo

Quisque facilisis auctor sapien. Pellentesque gravida hendrerit lectus. Mauris rutrum sodales sapien. Fusce hendrerit sem vel lorem. Integer pellentesque massa vel augue. Integer elit tortor, feugiat quis, sagittis et, ornare non, lacus. Vestibulum posuere pellentesque eros. Quisque venenatis ipsum dictum nulla. Aliquam quis quam non metus eleifend interdum. Nam eget sapien ac mauris malesuada adipiscing. Etiam eleifend neque sed quam. Nulla facilisi. Proin a ligula. Sed id dui eu nibh egestas tincidunt. Suspendisse arcu.

Anexos

ANEXO A – Morbi ultrices rutrum lorem.

Sed mattis, erat sit amet gravida malesuada, elit augue egestas diam, tempus scelerisque nunc nisl vitae libero. Sed consequat feugiat massa. Nunc porta, eros in eleifend varius, erat leo rutrum dui, non convallis lectus orci ut nibh. Sed lorem massa, nonummy quis, egestas id, condimentum at, nisl. Maecenas at nibh. Aliquam et augue at nunc pellentesque ullamcorper. Duis nisl nibh, laoreet suscipit, convallis ut, rutrum id, enim. Phasellus odio. Nulla nulla elit, molestie non, scelerisque at, vestibulum eu, nulla. Ut odio nisl, facilisis id, mollis et, scelerisque nec, enim. Aenean sem leo, pellentesque sit amet, scelerisque sit amet, vehicula pellentesque, sapien.

ANEXO B – Cras non urna sed feugiat cum sociis natoque penatibus et magnis dis parturient montes nascetur ridiculus mus

Sed consequat tellus et tortor. Ut tempor laoreet quam. Nullam id wisi a libero tristique semper. Nullam nisl massa, rutrum ut, egestas semper, mollis id, leo. Nulla ac massa eu risus blandit mattis. Mauris ut nunc. In hac habitasse platea dictumst. Aliquam eget tortor. Quisque dapibus pede in erat. Nunc enim. In dui nulla, commodo at, consectetur nec, malesuada nec, elit. Aliquam ornare tellus eu urna. Sed nec metus. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.