
Instituto Federal de Educação, Ciência e Tecnologia de São Paulo

Campus Guarulhos

Apostila de Linguagem de Programação I e II

PHP

Curso: Tecnologia em Análise e Desenvolvimento de Sistemas

Guarulhos, Dezembro de 2014.

Prof^a Gláucya Carreiro Boechat

Sumário

1	Introdução à Linguagem PHP	4
1.1	Características do PHP	4
1.2	Instalação	4
1.2.1	Instalando manualmente	4
1.2.2	Instalando com o XAMPP	5
2	Sintaxe Básica	7
2.1	Sintaxe Básica	7
2.1.1	Separação de instruções	8
2.1.2	Comentários	8
2.2	Variáveis	8
2.2.1	Escopo de variáveis	9
2.3	Constantes	11
2.4	Tipos de dados	11
2.4.1	Booleano	11
2.4.2	Inteiros	11
2.4.3	Strings	12
2.4.4	Arrays	12
2.5	Expressões	13
2.6	Operadores	14
2.6.1	Operadores aritméticos	14
2.6.2	Operadores de atribuição	14
2.6.3	Operadores de string	15
2.6.4	Operadores de incremento e decremento	15
2.6.5	Operadores de comparação	16
2.6.6	Operadores lógicos	16
2.6.7	Operadores lógicos bit a bit	17
2.6.8	Operadores de arrays	18
3	Estruturas de Controle	19
3.1	Estruturas condicionais	19
3.1.1	if	19

3.1.2	if ... else	19
3.1.3	else if	20
3.1.4	Operador ternário ?	21
3.1.5	switch	21
3.2	Comandos de repetição	22
3.2.1	while	22
3.2.2	do ... while	23
3.2.3	for	23
3.2.4	foreach	23
3.3	Comandos de interrupções	24
3.3.1	break	24
3.3.2	continue	24
4	Funções	26
4.0.3	Declaração	26
4.0.4	Argumentos	27
4.0.5	Valor de retorno	28
4.0.6	Funções variáveis	28
5	Tratamento de Formulários	29
5.1	Método Get	29
5.1.1	Variável \$_GET	30
5.2	Método POST	31
5.2.1	Variável \$_POST	31
5.2.2	Variável \$_REQUEST	32
5.2.3	Variáveis de formulários mais complexos	33
6	Sessões e Cookies	35
6.1	Cookies	35
6.1.1	Criação de Cookies	35
6.1.2	Variável \$_COOKIE	36
6.2	Sessões	37
6.2.1	Inicialização de uma sessão	37
6.2.2	Variável \$_SESSION	38
6.2.3	Função de sessões	40
6.3	Cookies x Sessões	40
7	Requisição de arquivos	41
7.1	include	41
7.2	include_once	42
7.3	require	43
7.4	require_once	43

8	Manipulação de Arquivos	45
8.1	fopen	45
8.2	fclose	46
8.3	feof	46
8.4	fgets	46
8.5	fgetc	47
8.6	fwrite	47
9	PHP Orientado à Objetos	49
9.1	Classe	49
9.1.1	Objeto	51
9.1.2	Visibilidade	53
9.1.3	Encapsulamento	54
9.1.4	Construtor e Destrutor	56
9.1.5	Herança	59
9.1.6	Polimorfismo	64
9.1.7	Evitando a herança	66
9.1.8	Classe Abstrata	67
9.1.9	Interfaces	69
10	PHP com Banco de Dados	71
10.1	MySQL	71
10.1.1	phpmyadmin	71
10.2	Comando básicos do MySQL via PHP	72
10.2.1	Conexão com o Banco de Dados	72
10.2.2	Consulta SQL no MySQL	73
10.3	Classes MySQLi e MySQLi_Result	85
10.3.1	Classe MySQLi	85
10.3.2	Classe MySQLi_Result	85
10.3.3	Conexão com o Banco de Dados	86
10.3.4	Selecionar uma base de dados	87
10.3.5	Consulta SQL no MySQL	88

Capítulo 1

Introdução à Linguagem PHP

A abreviação PHP é um acrônimo recursivo para *PHP Hypertext Preprocessor*, originalmente *Personal Home Page* [Achour et al., 2014]. O PHP é uma linguagem de script *open source* projetada para o desenvolvimento web, mas também usada como uma linguagem de programação de propósito geral. Os scripts PHP, diferente do Javascript, são interpretados pelo módulo PHP no lado servidor, gerando uma página web para ser visualizada no lado cliente.

1.1 Características do PHP

Uma das grandes vantagens da linguagem PHP é que ela é gratuita [Niederauer, 2004]. No site oficial do PHP (<http://www.php.net>) é possível obter gratuitamente o arquivo de instalação como também o código-fonte do PHP e sua documentação.

O PHP pode ser utilizado em diferentes plataformas como o Linux, várias variantes do Unix (incluindo HP-UX, Solaris e OpenBSD), Microsoft Windows, Mac OS X, RISC OS, e provavelmente outros. O PHP também é suportado pela maioria dos servidores web atuais, incluindo Apache, Microsoft Internet Information Server, Personal Web Server, Netscape and iPlanet Servers, Oreilly Website Pro Server, Caudium, Xitami, OmniHTTPd, e outros.

Talvez a mais forte e mais significativa característica do PHP é seu suporte a uma ampla variedade de banco de dados. Entre eles estão o MySQL, PostgreSQL, IBM DB2, Oracle (OCI7 and OCI8), mSQL e muitos outros.

1.2 Instalação

Antes de começar a trabalhar com o PHP é preciso instalar e configurar o PHP, um servidor HTTP (web), como o Apache e um banco de dados, como o MySQL. Você pode instalá-los separadamente ou usar um pacote de serviços web pré-configurado com Apache, MySQL e PHP, como o XAMPP.

1.2.1 Instalando manualmente

Primeiramente, instale o servidor HTTP (web) Apache encontrado no seguinte endereço <http://www.apache.org/dyn/closer.cgi> e teste se o mesmo funciona.

Em seguida baixe o Instalador do PHP para o Windows disponível na página de downloads do site oficial do PHP em <http://www.php.net/downloads.php>. Execute o instalador e siga as instruções mostradas pelas telas de instalação. O auxiliar de instalação não irá configurar o Apache, então você precisará configurá-lo manualmente.

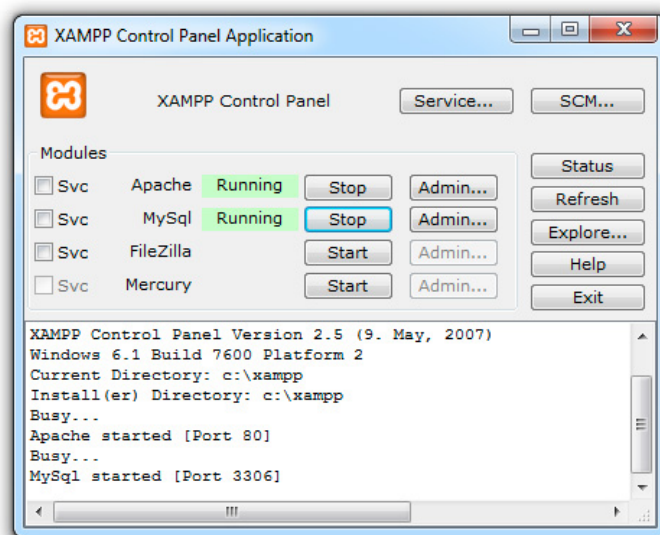
E por último, o MySQL pode ser obtido em <http://www.mysql.com/downloads/>. Para obter maiores informações sobre os arquivos de instalação do Apache, PHP e MySQL consulte suas páginas de downloads citadas acima.

1.2.2 Instalando com o XAMPP

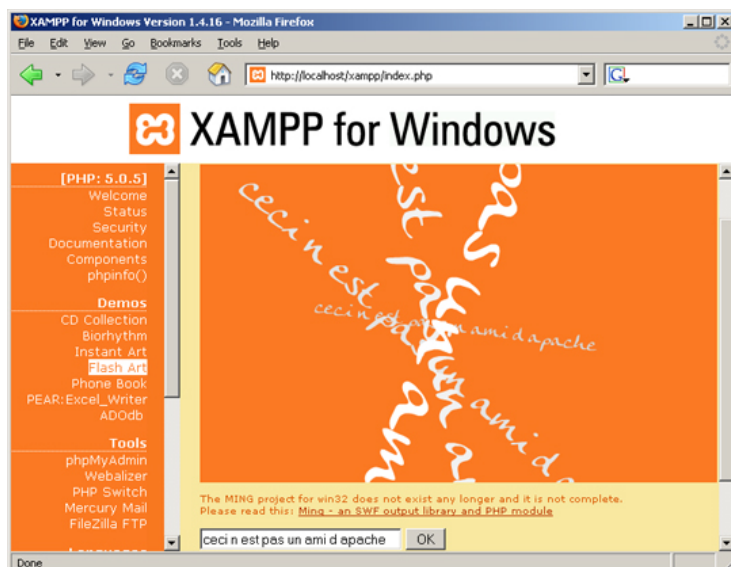
O XAMPP é um pacote de serviços web pré-configurado desenvolvido pelo o *Apache Friends*, projeto sem fins lucrativos para promover o servidor web apache, contendo o servidor HTTP Apache, o MySQL, o PHP e o Perl.

A instalação do XAMPP é feita através do instalador automatico disponível para download no site <http://www.apachefriends.org/download.html>.

Após concluir a instalação, já é possível usar o painel de controle do XAMPP para iniciar ou parar todos os serviços Apache e MySQL. O painel de controle está disponível no menu Iniciar | Programas | XAMPP.



Para testar o funcionamento do XAMPP, abrir a URL <http://localhost/xampp/> ou <http://127.0.0.1/xampp/> no seu navegador de preferência.



Feito isso, seu o PHP estará instalado e pronto para usar. Acesse o diretório onde o XAMPP foi instalado, por exemplo `C:\xampp\htdocs`, e verá várias arquivos e pastas. A pasta *htdocs* é onde deve-se salvar todos os arquivos *.php*. Agora basta abrir seu navegador e testar seus códigos *.php* através do endereço `http://localhost/nome_arquivo.php`

Capítulo 2

Sintaxe Básica

Neste capítulo vamos apresentar as noções básicas de programação em PHP, os tipos de dados, o conceito de variáveis e a manipulação de dados.

2.1 Sintaxe Básica

Todo arquivo em PHP possui a extensão *.php*. O PHP interpreta o código escrito dentro das tags de abertura e fechamento, e o resto do código é ignorado pelo interpretador do PHP. Existem diferentes pares de tags de abertura e fechamento que podem ser usadas no código PHP Achour et al. [2014], são elas:

```
<?php
    echo 'Primeiro par de tags PHP';
?>

<script language="php">
    echo 'Segundo par de tags PHP';
</script>

<?= $expressao = "Terceiro par de tags PHP" ?>
    Atalho <? echo $expressao; ?>

<%
    echo 'Quarto par de tags PHP - estilo ASP';
%>
```

O código em PHP também pode ser embutido no documento HTML, como no exemplo abaixo:

```
<html>
    <body>
        <h1> Exemplo de PHP </h1>
        <?php
            $a = 10;
            echo $a;
        ?>
    </body>
</html>
```


2.1.1 Separação de instruções

Na linguagem PHP, assim como em C ou Perl, todo comando deve terminar com ponto-e-vírgula (;). Em PHP, o último comando antes da tag de fechamento não precisa ter um ponto-e-vírgula para finalizar o comando.

```
<?php
    echo 'Exemplo';
    $a = $b + 3;
?>

<?php
    echo 'Exemplo sem ponto-e-vírgula'
?>
```

2.1.2 Comentários

O PHP suporta comentários no estilo C e Shell Script. O comentário de linha pode começar com // ou # e termina no final da linha ou com a tag de fechamento(? ≤), e o comentário de múltiplas linhas começa com (/*) e termina com (*/).

```
<?php
    echo 'Exemplos de comentários';
    // Estilo de comentário de uma linha

    /* Estilo de comentário para
       múltiplas linhas */

    # Estilo de comentário de uma linha com em Shell Script

    # echo 'Exemplo de comentário';?> A partir não é mais comentário</p>
```

2.2 Variáveis

O nome da variável sempre começa com o caracter '\$' seguida do nome da variável. O primeiro caracter do nome pode ser uma letra ou o caracter '_' e o restante do nome pode conter os caracteres alfanuméricos e o caracter '_' (A-z,0-9,_).

```
$variavel;
$_Variavel123;
$abc;
$ABC;
```

O PHP é *case sensitive*, isto é, letras maiúsculas e minúsculas são diferentes. No exemplo acima, a variável \$abc não é a mesma variável que a variável \$ABC.

Ao contrário das outras linguagens, em PHP não é necessário declarar as variáveis que serão usadas, tampouco definir seu tipo. Por exemplo, ao inicializar a variável \$a com o valor 3, o PHP reconhecerá que a variável \$a é do tipo inteiro.

```
$a = 3;
```

2.2.1 Escopo de variáveis

O escopo de uma variável é um conjunto de regras que determina em quais partes do programa ela pode ser acessada.

Em PHP, uma variável pode ter escopo global, local ou estática. As variáveis globais não podem ter o mesmo nome e o mesmo vale para as variáveis locais de uma mesma função. Entretanto em funções diferentes ou no programa principal não há problema de conflito se as variáveis locais possuírem o mesmo nome.

```
<?php
    $x = 1;          // Escopo global

    function F() {
        $y = 2;      // Escopo local
        echo $y;
    }

    F( );
?>
```

Variável local

Uma variável é chamada local se ela foi criada dentro de uma função e é acessada apenas dentro de uma função. No exemplo a baixo, a variável \$a deixa de existir após o término da função $F()$.

```
<?php

    $a = 1;          // Escopo global

    function Teste() {
        echo $a;      // Referencia uma variável de escopo local não definida
    }

    Teste();
?>
```

Variável global

Uma variável é chamada global se ela foi criada fora de qualquer função. As variáveis globais são visíveis em todas as funções, entretanto para acessá-las é preciso utilizar a palavra chave *global*.

```
<?php
    $a = 1;
    $b = 2;

    function Soma() {
        global $a, $b;    // Referencia as variáveis globais
        $b = $a + $b;
    }

    Soma();
    echo $b;
?>
```

ou através o array *\$GLOBALS*, onde cada chave do array corresponde à uma variável global. Exemplo

```
<?php
    $a = 1;
    $b = 2;

    function Soma() {
        // Alterando o valor da variável global $b
        $GLOBALS['b'] = $GLOBALS['a'] + $GLOBALS['b'];
    }

    Soma();
    echo $b;
?>
```

Variável Estática

Uma variável estática é uma variável que existe somente em escopos locais com o mesmo tempo de vida das variáveis globais. Uma variável estática é inicializada apenas uma vez.

```
<?php
    function F() {
        static $num = 100;
        return $num--;
    }
```

```
for($i=1; $i<100; $i++)
    echo " ".F();
?>
```

Toda vez que a função $F()$ for chamada a variável $\$num$ será decrementada.

2.3 Constantes

Constantes são valores predefinidos no início do programa e que são mantidos fixo ao longo da execução do programa. No PHP, as constantes não podem ter o sinal de cifrão (\$) antes do seu nome e também não podem ser redefinidas ou eliminadas depois de sua criação. Uma constante pode ser do tipo: *boolean*, *integer*, *float* ou *string*.

Para definir uma constante é preciso utilizar a função *define* como a seguir:

```
define("EXEMPLO", "Olá mundo.");
```

Ou utilizar a palavra reservada *const*, esta função está disponível a partir do *PHP 5.3.0*

```
const EXEMPLO2 = 'Exemplo de constante';
```

2.4 Tipos de dados

O PHP suporta os seguintes tipos de variáveis: inteiros, pontos flutuantes, strings, booleanos, array e objetos.

2.4.1 Booleano

O tipo booleano expressa um valor lógico que pode ser verdadeiro(*TRUE*) ou falso(*FALSE*).

Dados	Descrição
<i>TRUE</i>	Verdadeiro
<i>FALSE</i>	Falso

Exemplo de variáveis booleanas.

```
$var = TRUE;
$var2 = FALSE;
```

2.4.2 Inteiros

Uma variável inteira pode ser especificada em notação decimal, octal ou em hexadecimal. Exemplos

Dados	Descrição
$\$numero1 = 210$	Número inteiro na base decimal
$\$numero2 = -5$	Número inteiro negativo
$\$numero3 = 0123$	Número inteiro na base octal (83 na base decimal)
$\$numero4 = 0x1A$	Número inteiro na base hexadecimal (26 na base decimal)

Ponto flutuantes

Um ponto flutuante pode ser especificado utilizando uma das seguintes sintaxes:

<i>Dados</i>	<i>Descrição</i>
$\$numero1 = .15$	Número real com duas casas decimais (0.15)
$\$numero2 = 4e + 6$	Número real grande 4000000
$\$numero3 = 2e - 5$	2/100000

2.4.3 Strings

String é um tipo de dados formado por uma sequência de caracteres alfanuméricos. No PHP é permitido declarar uma string com aspas duplas “ ” ou aspas simples ‘ ’

```
$str = 'IFSP';           //o conteúdo da string é conservado
$str = "IFSP";           //O PHP irá interpretar mais sequências de escape
```

Sequências de escape

<i>Sequência</i>	<i>Descrição</i>
<code>\n</code>	Nova linha
<code>\t</code>	Tabulação horizontal
<code>\"</code>	Aspas duplas
<code>\\$</code>	o caracter \$
<code>[a-zA-Z]*</code>	Expressão regular para qualquer string com letra maiúscula ou minúscula.

2.4.4 Arrays

Um variável do tipo array pode ser composto por chaves dos tipos inteiro e string. Exemplos de inicialização de arrays.

```
$vetor = array("cor" => "azul", 12 => true);
```

A variável *vetor* no exemplo acima possui a chave *cor* com a string *azul* e a chave 12 com o booleano *true*.

```
$vetor = array(1, 2, 3);
```

```
/* vetor(
    [0] => 1
    [1] => 2
    [2] => 3 ) */
```

```
$vetor1 = array(10 => 3, 6, 9, "a" => 100);
$vetor2 = array(10 => 3, 11 => 6, 12 => 9, "a" => 100);
```

Os arrays *vetor1* e *vetor2* são iguais. Na declaração do *vetor1* o valor 6 é adicionado na próxima chave depois da chave 10. e o valor 9 é adicionado na próxima chave depois da chave 11.

```
$vetor3 = array(5 => 1, 12 => 2);  
$vetor3[] = 16;      // $vetor[13] = 16;
```

O valor 16 é adicionado na próxima chave depois da última chave. Como a chave 12 é o maior valor, então o valor será adicionado no elemento $12 + 1$.

```
$vetor3["x"] = "13";
```

A string '13' é adicionada na chave 'x' no *vetor3*.

Para remover um elemento de um array é preciso usar a função *unset*. No exemplo abaixo a função está removendo o elemento 5 do array *vetor3*.

```
unset($vetor3[5]);
```

O próximo comando remove o array *vetor3* do programa.

```
unset($vetor3);
```

Uma variável do array também pode possuir um array dentro do próprio array, exemplo

```
$matriz = array("someVetor" => array(6 => 5, 13 => 9, "a" => 42));
```

2.5 Expressões

Uma expressão é pode ser formada por variáveis, constantes, uma chamada de uma função ou um conjunto de operações aritméticas e lógicas.

```
<?php  
$a = 5;  
$b = $a;      // comporta da mesma forma que $b=5  
  
$a = 3 * 2;  
  
$c = ($a = 15); // Corresponde a escrever $a = 15; $b=15;  
$c = $a = 15;  
  
function F() {  
    return 'abc';  
}  
  
$a = F();      // $a = 'abc'  
$b++;  
$c = ($a == $b);  
?>
```

2.6 Operadores

O PHP oferece uma ampla variedade de operadores, entre binários e unários. A seguir serão apresentados os operadores básicos.

2.6.1 Operadores aritméticos

<i>Operador</i>	<i>Nome</i>	<i>Exemplo</i>
+	Adição	$\$a + \b
-	Subtração	$\$a - \b
-	Negação	$-\$a$
*	Multiplificação	$\$a * \b
/	Divisão	$\$a / \b
%	Módulo	$\$a \% \b

O exemplo abaixo mostra o resultado de diferentes operações matemáticas

```
<?php
$a = 15;
$b = 5;
$c = $a + $b; // $c = 20
$d = $c - $b; // $d = 15
$e = $b * 2; // $e = 10
$f = $c / 10; // $e = 2
$g = $e / 3; // $e = 1
?>
```

2.6.2 Operadores de atribuição

Em PHP o operador de atribuição é usado para escrever o valor de uma variável.

<i>Operador</i>	<i>Nome</i>	<i>Exemplo</i>
=	Atribuição simples	$\$a = \b
+ =	Atribuição com adição	$\$a + = \b
- =	Atribuição com subtração	$\$a - = \b
* =	Atribuição com multiplicação	$\$a * = \b
/ =	Atribuição com divisão	$\$a / = \b
% =	Atribuição com módulo	$\$a \% = \b
. =	Atribuição com concatenação	$\$a . = \b

Exemplos utilizando o operador de atribuição.

```
<?php
$a = 1+1;      // $a = 2
$a += 2;       // $a = $a + 2; ($a = 4)
$a -= 1;       // $a = $a - 1; ($a = 3)
$a *= 2;       // $a = $a * 2; ($a = 6)
$a /= 6;       // $a = $a / 6; ($a = 1)
$a %= 3;       // $a = $a % 3; ($a = 1)
echo $a;
?>
```

2.6.3 Operadores de string

Operador	Nome	Exemplo
=	Concatenação	$\$a = \b
. =	Atribuição com concatenação	$\$a. = \b

Exemplos do operador de concatenação pode ser visto abaixo

```
<?php
$a = "IFSP";
$b = $a . " - Guarulhos";    // $b = IFSP - Guarulhos
$a .= " - São Paulo";       // $a = IFSP - São Paulo
?>
```

2.6.4 Operadores de incremento e decremento

Operador	Nome	Descrição
$++\$a$	Pré-incremento	Incrementa \$a em um, e então retorna \$a
$--\$a$	Pré-decremento	Decrementa \$a em um, e então retorna \$a
$\$a++$	Pós-incremento	Retorna \$a, e então incrementa \$a em um
$\$a--$	Pós-decremento	Retorna \$a, e então decrementa \$a em um

Exemplos dos operadores de incremento e decremento

```
<?php
$a = 10;
echo ++$a;    // Saída: 11
echo $a++;    // Saída: 11
$b = 5;
echo $b--;    // Saída: 5
echo --$b;    // Saída: 3
?>
```


2.6.5 Operadores de comparação

O operador de comparação é usada comparar dois valores.

<i>Operador</i>	<i>Nome</i>	<i>Exemplo</i>
<code>==</code>	Igual	<code>\$a == \$b</code>
<code>!=</code>	Diferente	<code>\$a != \$b</code>
<code><></code>	Diferente	<code>\$a <> \$b</code>
<code>===</code>	Idêntico	<code>\$a === \$b</code>
<code>!==</code>	Não idêntico	<code>\$a !== \$b</code>
<code>></code>	Maior que	<code>\$a > \$b</code>
<code><</code>	Menor que	<code>\$a < \$b</code>
<code>>=</code>	Maior ou igual que	<code>\$a >= \$b</code>
<code><=</code>	Menor ou igual que	<code>\$a <= \$b</code>

Exemplo

```
<?php
    $a=1; $b="001";

    $c = ($a == $b);      // true
    $d = ($a === $b);     // false

    $e = 2;
    $f = ($e <= $a);      // false
?>
```

2.6.6 Operadores lógicos

O operador de comparação é usada comparar dois valores.

<i>Operador</i>	<i>Nome</i>	<i>Exemplo</i>
<code>and</code>	E lógico	<code>(\$a and \$b)</code>
<code>&&</code>	E lógico	<code>(\$a && \$b)</code>
<code>or</code>	OU lógico	<code>(\$a or \$b)</code>
<code> </code>	OU lógico	<code>(\$a \$b)</code>
<code>xor</code>	OU exclusivo	<code>(\$a xor \$b)</code>
<code>!</code>	Não	<code>!\$a</code>

Exemplo ilustrando a utilização dos operadores lógicos

```
<?php
$a=1;
$b="001";
$c=2;

if($a == $b) and ($a < $c)
    $a += $c;

$d = (false || true); // true
?>
```

2.6.7 Operadores lógicos bit a bit

Os operadores lógicos bit a bit permitem fazer operações em cada bit de um número inteiro.

<i>Operador</i>	<i>Nome</i>	<i>Exemplo</i>
&	E lógico	$\$a \& \b
	OU lógico	$\$a \b
^	OU exclusivo	$\$a \wedge \b
~	Não	$\sim \$a$
<<	Deslocamento à esquerda	$\$a << \b
>>	Deslocamento à direita	$\$a >> \b

Exemplo

```
<?php
$a = "120"; // $a = 120 ( binário 1111000)
$b = $a >> 3; // $b = 15 ( 0001111)
$c = $a << 3; // $c = 960 ( 1111000000)
echo "a = $a, b = $b, c = $c"; // a = 120, b = 15, c = 960

$d = "15" & 10; //15 (1111) , 10 (1010)
echo $d; // $d = 10 (1010)
?>
```

2.6.8 Operadores de arrays

O operador de arrays é usado para comparar dois array.

<i>Operador</i>	<i>Nome</i>	<i>Exemplo</i>
+	União	$\$a + \b
==	Igualdade	$\$a == \b
!=	Desigualdade	$\$a != \b
<>	Desigualdade	$\$a <> \b
===	Idêntico	$\$a === \b
!==	Não idêntico	$\$a !== \b

O operador de união (+) acrescenta os elementos do array da direita no array da esquerda, mas as chaves duplicadas não são sobrescritas.

Exemplo

```
<?php
$fruta1 = array("a" => "maça", "b" => "banana");
$fruta2 = array("c" => "laranja", "b" => "melancia",
               "a" => "morango");

$fruta3 = $fruta1 + $fruta2;
// $fruta1 = array("a" => "maça", "b" => "banana", "c" => "laranja");
var_dump($fruta3);

$fruta4 = $fruta2 + $fruta1;

/* $fruta4 = array("c" => "laranja", "b" => "melancia",
                  "a" => "morango"); */
var_dump($fruta4);

if($fruta4 === $fruta2)
    echo "Iguais ";

?>
```

Capítulo 3

Estruturas de Controle

Uma estrutura de controle é aquela que dado o resultado de uma expressão lógica ou relacional é possível decidir se um determinada bloco de comandos deve ou não ser executado.

3.1 Estruturas condicionais

3.1.1 if

O comando *if* é usado para executar o bloco de código se a condição avaliada for verdadeira.

```
if(condição) {  
    //Bloco de comandos;  
}
```

O exemplo abaixo verifica se o valor da variável *\$a* é igual ao valor da variável *\$b*.

```
<?php  
$a=1;  
$b=2;  
  
if($a == $b)  
    echo "\$a = \$b";  
?>
```

3.1.2 if...else

Else é utilizado para indicar um bloco de comandos, caso a condição do *if* não seja satisfeita.

```
if(condição) {  
    //Bloco de comandos;  
}else{  
    //Bloco de comandos;  
}
```

Exemplo:

```
<?php
$a=1;
$b=2;

if($a == $b)
    echo "\$a = \$b";
else
    echo "\$a != \$b";
?>
```

3.1.3 else if

O comando *else if* é usado em situações onde precisa-se verificar mais de uma condição. As condições são avaliadas de cima para baixo. Assim que uma condição verdadeira for encontrada, o bloco de comandos associados à ela é executado.

```
if(condição1) {
    //Bloco de comandos;
}else if(condição2) {
    //Bloco de comandos;
}else{
    //Bloco de comandos;
}
```

Exemplo

```
<?php
$a=1;
$b=2;

if($a < $b)
    echo "\$a < \$b";
else if($a == $b)
    echo "\$a = \$b";
else
    echo "\$a > \$b";
?>
```

3.1.4 Operador ternário ?

O operador ? é uma alternativa dos comandos *if...else* na forma

```
(condição) ? expressão1 : expressão2;
```

O valor de uma expressão ? é determinada pela condição avaliada. Se a condição for verdadeira, então o valor da expressão será expressão1, caso contrário, expressão2 tornará o valor da expressão. Exemplo

```
<?php
    $a = 1;

    $b = ($a > 0) ? $a + 1 : $a - 1;
?>
```

O mesmo código com *if...else* seria

```
<?php
    $a = 1;

    if ($a > 0)
        $b = $a + 1;
    else
        $b = $a - 1;
?>
```

3.1.5 switch

O comando *switch* é utilizado para fazer múltiplos testes da condição. O *switch* procura por uma coincidência entre a *\$variavel* e outros valores associados em cada opção.

```
<?php
    switch ($variavel) {
        case valor1:
            //Bloco de comandos;
            break;
        case valor2:
            //Bloco de comandos;
            break;
        // ...
        default:
            //Bloco de comandos;
            break;
    }
?>
```

O comando *default* é executado quando nenhuma das coincidência for detectada. Exemplo do uso do

switch

```
<?php
    $i = "c";
    switch ($i) {
        case "a":
            echo "i = a";
            break;
        case "b":
            echo "i = b";
            break;
        default:
            echo "i = c";
            break;
    }
?>
```

3.2 Comandos de repetição

Um comando de repetição permite que um conjunto de instruções seja executado até que ocorrência de uma condição.

3.2.1 while

O comando *while* testa a condição e executa um bloco de comandos enquanto a condição for verdadeira.

```
while (condição) {
    //Bloco de comandos;
}
```

Exemplo de uso do comando *while*

```
<?php
    $i=0;
    while ($i < 10) {
        echo "i = ".$i++;
    }
?>
```

3.2.2 do ... while

O comando *do ... while* funciona de maneira semelhante ao comando *while*, com uma pequena diferença que a condição é testada no final da execução do bloco de comandos.

```
do{
    //Bloco de comandos;
}while(condição);
```

Exemplo

```
<?php
    $i=0;
    do{
        echo "i = ".$i++;
    }while ($i < 10);
?>
```

3.2.3 for

O comando *for* possui três expressões: inicialização, condição e incremento. A inicialização serve para inicializar o valor da variável contador. A condição do loop *for* é verificada a cada iteração, caso a condição for verdadeira o loop continua e caso contrário o loop é terminado. A expressão incremento/decremento aumenta ou diminui o valor da variável contador.

```
for(inicialização; condição; incremento/decremento) {
    //Bloco de comandos;
}
```

Exemplo do funcionamento do comando *for*.

```
<?php
    for($i=1; $i<10; $i++){
        echo "\n i = ".$i;
    }
?>
```

3.2.4 foreach

O comando *foreach* percorre cada item do array *\$variavel*. Em cada item, ele armazena a chave e seu respectivo valor na variável *\$chave*.

```
foreach($variável as $chave => $valor) {
    //Bloco de comandos;
}
```


O exemplo abaixo utiliza o comando *foreach* para percorrer o vetor `$cursos`

```
<?php
    $cursos = array('ADS', 'Automação', 'Matemática');

    foreach($cursos as $chave => $nome) {
        echo "<br> curso $chave = $nome";
    }
?>
```

O mesmo código usando *for* pode ser escrito como:

```
<?php
    $cursos = array('ADS', 'Automação', 'Matemática');

    for($i=0; $i<sizeof($cursos); $i++){
        echo "<br> curso $i = $cursos[$i]";
    }
?>
```

3.3 Comandos de interrupções

O PHP fornece algumas formas de interrupção antecipada de um determinado laço ou do comando *switch*.

3.3.1 break

O comando *break* pode quebrar a execução de um comando *switch* ou interromper a execução de qualquer comando de repetição. Exemplo

```
<?php
    for($i=1; $i<10; $i++){
        if ($x == 5){
            echo "interrompendo o for";
            break;
        }
        echo "\n i = ".$i; // imprime os valores de 1 até 4
    }
?>
```

3.3.2 continue

O comando *continue* funciona de maneira semelhante ao comando *break*, com a diferença que o loop volta para o início dele.

Exemplo

```
<?php
    for($i=1; $i<10; $i++){
        if (($i%2) == 1)
            continue;
        echo "\n i = ".$i;    // imprime os valores pares
    }
?>
```

Capítulo 4

Funções

Em PHP, além das funções embutidas, é possível criar nossas próprias funções. Uma função é um bloco de comando que pode ser usada repetidamente no programa. Quando uma função é criada esta não será executada imediatamente quando a página é carregada, mas sim quando houver uma chamada para a função.

4.0.3 Declaração

Para declarar uma função é preciso começar com a palavra chave *function* seguido do nome da função. O nome da função deve começar com uma letra ou o caracter `'_'` e o restante do nome pode conter os caracteres alfanuméricos e o caracter `'_'`. A forma geral de uma função é

```
function nome_da_função($arg_1, $arg_2, /* ..., */ $arg_n) {  
    //Bloco de comandos  
    echo "Exemplo de função.\n";  
    return $valor_retornado;  
}
```

O comando `return` é opcional, já que não é obrigatório retornar um valor em funções no PHP, outra regra consiste em não permitir que sejam retornados múltiplos valores através deste comando. Para resolver essa necessidade, pode-se retornar listas e arrays.

No PHP as funções não precisam ser criadas antes de serem referenciadas, exceto quando uma função é definida condicionalmente ou quando uma função é definida dentro de outra função. Exemplo

```
<?php  
    $check = true;  
  
/* Neste parte do programa, é possível fazer uma chamada das funções  
g() e m(). Entretanto, as funções f() e n() não podemos ser chamadas  
pois elas ainda não existem.*/  
    g();  
  
// Função definida condicionalmente  
    if ($check) {  
        function f() {  
            echo "f() não existe até que o programa passe por aqui";  
        }  
    }  
}
```

```
// Função definida dentro de outra função
function m() {
    echo "m() existe desde o começo do programa";
    function n() {
        echo "n() não existe até que m() seja chamada";
    }
}
m();
// A partir deste ponto podemos chamar f() e m()
function g() {
    echo "g() existe desde o começo do programa";
}
?>
```

4.0.4 Argumentos

Informações podem ser passadas para funções através de argumentos. O PHP suporta a passagem de argumentos por valor ou por referência, também é permitido valores padrões de argumentos. Por padrão, as informações são passadas por valor para as funções. Assim, os parâmetros que a função recebe é tratado como variável local dentro do contexto da função. Exemplo de função com dois argumentos:

```
<?php
function f($curso, $disciplina) {
    echo "Curso:$curso - disciplina: $disciplina";
}
f("ADS", "Linguagem de Programação");
f("ADS", "Estrutura de Dados");
?>
```

Para que um argumento seja passado por referência em uma função é preciso adicionar o caracter & antes do nome do argumento, na definição da função, exemplo:

```
<?php
function concatenandoStrings(&$string) {
    $string .= 'Dia!';
}
$str = 'Bom ';
concatenandoStrings($str);
echo $str; // saída: 'Bom Dia!'
?>
```

No PHP, uma função também pode definir valores padrão no estilo C++ para argumentos escalares. O valor padrão precisa ser uma expressão constante, e não uma variável ou uma chamada de função ou mesmo um membro de classe. Na declaração das funções, os argumentos que possuírem valores padrões devem vir após os argumentos sem padrões. Caso contrário, a função não funcionará como esperado.

```
<?php
function f($disciplina, $curso = "ADS") {
    echo "Curso:$curso - disciplina:$disciplina";
}
echo f("Banco de Dados");
//Saída: Curso:ADS - disciplina:Banco de Dados
echo f("Linguagem de Programação", null);
//Saída: Curso: - disciplina: Linguagem de Programação
echo f("Estrutura de Dados", "Automação");
//Saída: Curso:Automação - disciplina:Estrutura de Dados
?>
```

NÃO ENTENDI

4.0.5 Valor de retorno

Toda função pode opcionalmente retornar um valor. Esse valor é retornado pelo comando *return*. Exemplo

```
<?php
function quadrado($num) {
    return $num * $num;
}
echo quadrado(3); // Saída: '9'.
?>
```

4.0.6 Funções variáveis

O PHP suporta o conceito de funções variáveis. Isto significa que se você colocar parênteses no final do nome de uma variável, o PHP procurará uma função com o mesmo nome do valor da variável, e tentará executá-la. Exemplo

```
<?php
function mostra($string) {
    echo $string;
}
$func = 'mostra';
// Chamada da função mostra()
$func('Testando a função'); //Saída: Testando a função
?>
```

Capítulo 5

Tratamento de Formulários

Uma das características mais fortes do PHP é a forma como ele trata os formulários HTML. Quando um formulário html é submetido para um script PHP qualquer variável de um formulário será automaticamente disponível para este script. Existem dois métodos de passagem de parâmetros: *Get* e *Post*. O tipo do método é indicado no parâmetro *method* da tag *form*. Exemplo de formulário

```
<html>
  <body>
    <form action="recebe_dados.php" method="POST">
      Nome: <input type="text" name="nome"/>
      Idade: <input type="text" name="idade"/>

      <input type="submit" value="Enviar" name="enviar"/>
      <input type="reset" name="limpar" value="Limpar"/>
    </form>
  </body>
</html>
```

5.1 Método Get

O método Get é o método padrão para o envio de dados, isto significa que se nenhum método for especificado no parâmetro *method* da tag *form* então o método Get será utilizado.

O método Get passa os argumentos de uma página para a próxima, como parte da URL (*Uniform Resource Locator*) [Converse et al., 2004]. O método GET anexa os nomes das variáveis e seus valores na URL designado no atributo *action* do formulário com o separador (?). O símbolo (&) indica o início de uma nova variável e o caracter (=) separa as variáveis dos seus respectivos valores. Exemplo

```
http://www.meusite.com.br/recebe_dados.php?nome=Aluno&idade=19
```

Por motivo de segurança, o método Get não deve ser usado para enviar senhas ou informações confidenciais. O método também não é adequado para variáveis com valores superiores a 2000 caracteres.

Exemplo de formulário usando o método Get

```
<form action="recebe_dados.php" method="get">
  Nome: <input type="text" name="nome" />
  Idade: <input type="text" name="idade"/>

  <input type="submit" value="Enviar" name="enviar">
  <input type="reset" name="limpar" value="Limpar"/>
</form>
```

5.1.1 Variável \$_GET

Ao clicar no botão submit do formulário da página HTML. Os dados do formulário são enviados para o arquivo PHP. O PHP utiliza da variável \$_GET para recolher os dados do formulário enviadas pelo método GET. Exemplo

```
<html>
  <body>
    Olá <?php echo $_GET["nome"]; ?><br>
    você tem<?php echo $_GET["idade"]; ?> anos.
  </body>
</html>
```

Saída:

```
Olá Aluno.
Você tem 19 anos.
```

Exemplo 2

```
<html>
  <body>
    <form method="get" action="cor.php">
      <p>Informe seu nome:</p> <input type="text" name="nome"></p>
      <p>Escolha uma cor:</p>
      <input type="radio" name="cor" value="r" /> Vermelho
      <input type="radio" name="cor" value="g" /> Verde
      <input type="radio" name="cor" value="b" /> Azul      </p>

      <input type="submit" value="Enviar" />
    </form>
  </body>
</html>
```

Arquivo "cor.php"

```
<?php
    switch ($_GET["cor"]) {
        case "r": $background = "rgb(255,0,0)"; break;
        case "g": $background = "rgb(0,255,0)"; break;
        case "b": $background = "rgb(0,0,255)"; break;
        default: $background = "rgb(255,255,255)"; break;
    }
?>

<html>
    <body style="background: <?php echo $background;?>;">
        <? echo "<h1>Olá " . $_GET["nome"] . "</h1>"; ?>
    </body>
</html>
```

5.2 Método POST

Quando o método POST é utilizado no parâmetro *method* da tag *form*, os dados do formulário são encapsulados dentro do corpo da mensagem encaminhada ao servidor. Com o POST não haverá nenhuma mudança visível na URL.

`http://www.meusite.com.br/recebe_dados.php`

O método POST é o método mais seguro e pode ser usado para enviar um número maior de informações. Com esse método é possível enviar imagens ou outros tipos de arquivos.

5.2.1 Variável \$_POST

A variável \$_POST é usada para recolher os dados enviados pelo método POST do formulário. Exemplo de formulário usando o método POST

```
<p>Escolha um curso</p>
<form name="informacoes" method="POST" action="curso.php">

    <select name="curso">
        <option value="ADS" selected>
            Tecnologia em Análise e Desenvolvimento de Sistemas
        </option>
        <option value="Automação">
            Tecnologia em Automação Industrial
        </option>
```



```

<option value="Matemática">
    Licenciatura em Matemática
</option>
<option value="TMSI">
    Técnico de Manutenção e Suporte em Informática
</option>
<option value="TAI">
    Técnico em Automação Industrial
</option>
</select> </p>

<input name="enviar" type="submit" value="Enviar">
</form>

```

Arquivo "curso.php"

```

<html>
<body>
    <h3>Resultado</h3>

    <?php
        echo "<p>Curso selecionado: " . $_POST['curso'] . "</p>\n";
    ?>
</body>
</html>

```

5.2.2 Variável \$_REQUEST

A variável predefinida \$_REQUEST contém as variáveis \$_GET, \$_POST e \$_COOKIE. A variável \$_REQUEST pode ser usada para recolher os dados enviados pelos métodos GET e POST. Exemplo

```

<?php
if(isset($_REQUEST['enviar'])) {
    echo "</p><h2>Dados do formulário</h2> <hr> </p>";
    echo "Nome : {"$_REQUEST['nome']} </br>";
    echo "Comentário : {"$_REQUEST['comentario']}</p><hr>";
} else {
    echo " <h2>Formulário</h2>
        <form name='informacoes' method='POST' action=#>
            Informe seu nome:
            <input name='nome' type='text'><br/><br/>

```

```

        Escreva um comentário: </p>
        <textarea name='comentario' rows='3' cols='25'>
        </textarea>
        <br/><br/>

        <input name='enviar' type='submit' value='Enviar'>
    </form>";
}
?>

```

5.2.3 Variáveis de formulários mais complexos

O PHP entende arrays no contexto de variáveis de formulários. No PHP é possível agrupar variáveis relacionadas, ou usar esse recurso para receber valores de um campo de múltipla seleções. Por exemplo, podemos ter um formulario que manda informações para si mesmo até um comando submetido para mostrar todos os dados [Achour et al., 2014].

O exemplo abaixo agrupa o nome e a idade do usuário no array Dados e os cursos escolhidos no array Curso.

```

<html>
<form action="dados.php" method="post">
    Nome: <input type="text" name="Dados[nome]" /><br />
    Idade: <input type="text" name="Dados[idade]" /><br />
    Curso: <br />
    <select multiple name="Curso[]">
        <option value="ADS">
            Tecnologia em Análise e Desenvolvimento de Sistemas</option>
        <option value="TAI">
            Tecnologia em Automação Industrial</option>
        <option value="LM">
            Licenciatura em Matemática</option>
    </select><br />
    <input type="submit" value="Enviar dados!" />
</form>
</html>

```

Arquivo "dados.php"

```
<html>

<?php
    echo '<pre>';
    echo htmlspecialchars(print_r($_POST, true));
    echo '</pre>';

    echo "<br> Nome : ". $_POST["Dados"]["nome"];
    echo "<br> Idade : ".$_POST["Dados"]["idade"];
    echo "<br> Curso : ".$_POST["Curso"][0];
?>
</html>
```

O exemplo a seguir mostrar as linguagens de programação escolhidas pelo usuário.

```
<?php
    if (isset($_POST['enviar'])) {
        echo "Você gosta das linguagens:<br />";
        foreach($_POST['linguagens'] AS $linguagem)
            echo "$linguagem<br />";
    }
?>

<form action="linguagens.php" method="post">
    Quais linguagens você conhece? </p>
    <input type="checkbox" name="linguagens[]" value="php" />
        PHP<br />
    <input type="checkbox" name="linguagens[]" value="C" />
        C<br />
    <input type="checkbox" name="linguagens[]" value="Java" />
        Java<br />
    <input type="checkbox" name="linguagens[]" value="Python" />
        Python </p>

    <input type="submit" name="enviar" value="Enviar" />
</form>
```

Capítulo 6

Sessões e Cookies

6.1 Cookies

Um cookie é um pequeno pedaço de informação que é mantido na máquina do cliente, seja na memória do navegador ou como um pequeno arquivo gravado no disco rígido do usuário, e posteriormente recuperado pelo servidor. Os cookies podem ser mantidos na máquina do cliente por vários dias.

Exemplos de uso de cookies

- Autenticação de usuário,
- Carrinho de compras,
- Dados de formulários,
- Site de votação,
- Personalização de páginas (cor de fundo)

6.1.1 Criação de Cookies

No PHP, os cookies são definidos pela função `setcookie()` e estes são lidos quase que automaticamente.

Sintaxe

```
setcookie(nome, valor, tempoExpiracao, caminho, domínio);
```

Nome: nome do cookie. Parâmetro obrigatório.

Valor: valor do cookie. Se não fornecido, o servidor tentará excluir o cookie com o nome especificado.

TempoExpiração: tempo de vida do cookie

Caminho: caminho no servidor aonde o cookie estará disponível. Se configurado para '/', o cookie estará disponível para todo o domain.

Domínio: domínio para qual o cookie estará disponível Exemplo: `http://www.ifspguarulhos.edu.br/`

O exemplo abaixo mostra a criação de cookies.

```
<?php
// O cookie usuario1 expira quando fechar o navegador
setcookie("usuario1", "aluno1");
// O cookie usuario2 expira após 1 hora
setcookie("usuario2", "aluno2", time()+3600);
// O cookie usuario3 expira após 1 mês
// 60 seg * 60 min * 24 horas * 30 dias
$tempo = time() + 60*60*24*30;
setcookie("usuario3", "aluno3", $tempo);
?>
```

A função *time()* retorna a hora atual medida no número de segundos desde às 00:00:00 do dia 1 de Janeiro de 1970.

6.1.2 Variável \$_COOKIE

O PHP possui uma variável chamada \$_COOKIE é utilizada para recuperar os valores dos cookies. No exemplo a abaixo é mostrado todos os cookies existentes na máquina do usuário.

```
<?php
echo "Olá " . $_COOKIE["usuario"];
// Visualiza todos os cookies
print_r($_COOKIE);
?>
```

O próximo exemplo mostra como verificar se existe algum *cookie*

```
<?php
if(isset($_COOKIE["usuario"])) {
    echo "Olá " . $_COOKIE["usuario"];
}else
    echo "Olá ";
}
?>
```

Podemos setar várias informações num array de *cookies*, exemplo.

```
<?php
//Enviando cookies
setcookie("cookie[um]", "IFSP");
setcookie("cookie[dois]", "Guarulhos");
setcookie("cookie[tres]", "São Paulo");
?>
```

E para recuperar as informações de uma array de *cookies*

```
<?php
// Mostrando os cookies
if (isset($_COOKIE['cookie'])) {
    foreach ($_COOKIE['cookie'] as $nome => $valor) {
        echo "$nome : $valor <br />\n";
    }
}
?>
```

Para excluir um cookie é preciso garantir que o seu tempo de expiração foi expirado, como no exemplo:

```
<?php
setcookie("usuario");

//Tempo de expiração expirado
setcookie("usuario", "", time()-3600);
?>
```

6.2 Sessões

Uma sessão é um período de tempo durante no qual uma determina pessoa navega pelas páginas de um determinado site. Sessões em geral são as formas mais seguras para passar informações sigilosas entre páginas.

O PHP permite configurar uma sessão e variáveis de sessão. Depois que uma sessão é criada, as variáveis de sessão estarão disponíveis para serem usadas em qualquer página do site e enquanto a sessão estiver aberta. E Quando o usuário deixar o site ou fechar o navegador todas as variáveis de sessão serão apagadas.

6.2.1 Inicialização de uma sessão

A inicialização de uma sessão pode ser feita de forma automática ou manual. Para inicializar de forma automática é preciso habilitar a diretiva *session.auto_start* no arquivo *php.ini*. E a inicialização manual é feita a através da função *session_start()*

```
bool session_start(void);
```

A função *session_start()* é usada para criar uma nova sessão, e esta sessão criada recebe um identificador único (UID), chamado de *session_id*. A função também é usada para restaurar os dados de uma sessão com base no UID corrente. Todos os dados de uma sessão são passados via variável GET ou cookie.

Exemplo de criação de uma sessão

```
<?php
    session_start();
?>

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

6.2.2 Variável \$_SESSION

O PHP possui uma a variável chamada de \$_SESSION que é array composto por variáveis de sessão disponíveis para o atual script. E com esta variável é possível recuperar ou armazenar variáveis de uma sessão. Exemplo

```
<?php
    session_start();
    $_SESSION['usuario'] = "aluno";
?>

<html>
    <body>
        <?php
            echo "Olá " . $_SESSION['usuario'];
        ?>
    </body>
</html>
```

O exemplo abaixo mostra como contar o número de visualizações de um site específico.

```
<?php
    session_start();
    if(isset($_SESSION['NumerosVisitas'])) {
        $_SESSION['NumerosVisitas'] =
            $_SESSION['NumerosVisitas']+1;
    }else
        $_SESSION['NumerosVisitas'] = 1;

    echo "Número de visitas : " . $_SESSION['NumerosVisitas'];
?>
```

Para enviar dados para uma sessão através de link.

```
<?php
    session_start();

    $_SESSION['nome'] = 'Aluno';
    $_SESSION['idade'] = 20;
    $_SESSION['data'] = time();

    // Funciona se o cookie de sessão foi aceito
    echo '<br><a href="recebe.php">recebe.php</a>';

    // Ou passando o ID da sessão se necessário
    echo '<br><a href="recebe.php?". SID . "> recebe.php </a>';
?>
```

Arquivo "recebe.php"

```
<?php
    session_start();
    echo "Olá ". $_SESSION['nome'];
    echo "<br> Idade : " . $_SESSION['idade'];
    echo "<br> Data : " . date('Y m d H:i:s', $_SESSION['data']) ;

    echo '<br><a href="envia.php">envia.php</a>';
?>
```

Para destruir todos os dados registrados em uma sessão é preciso usar a função *Session_destroy()* como no exemplo abaixo

```
<?php
    session_start();
    $_SESSION['usuario'] = "aluno";

    if(isset($_SESSION['usuario']))
        session_destroy();
?>
```

E a função *unset()* é usada para liberar variáveis de uma sessão.

```
<?php
    session_start();
    if(isset($_SESSION['usuario']))
        unset($_SESSION['usuario']);
?>
```


6.2.3 Função de sessões

`session_id()` obtém e/ou define o id de sessão atual.

`session_name()` obtém e/ou define o nome da sessão atual.

`session_unset()` libera todas as variáveis de sessão.

`session_encode()` codifica os dados da sessão atual como uma string.

`session_decode()` decifra dado de sessão de uma string.

6.3 Cookies x Sessões

Podemos comparar os cookies e as sessões na tabela abaixo

	<i>Cookies</i>	<i>Session</i>
Informação	No cliente	No servidor
Disponibilidade	As vezes	Sempre
Segurança	Menos seguro	Mais seguro
Tempo	Maior	Menor

Capítulo 7

Requisição de arquivos

Na linguagem PHP, é possível incluir dentro do conteúdo de um arquivo PHP outros arquivos PHP externo com: definições de funções, variáveis, constantes, classes e configurações. Para isso podemos utilizar as funções `include` e `require`.

7.1 `include`

O comando *include* inclui e avalia o arquivo informado [W3Schools, 2014a]. Todo o código do arquivo incluído entrará no escopo do programa na linha onde a inclusão ocorre. Assim, qualquer variável, array ou objeto do arquivo incluído estará disponível no programa. Sintaxe:

```
include 'nome_do_arquivo';
```

Se o arquivo à ser incluído não existir, o comando `include` produzirá uma mensagem de advertência. *erro Fatal (E_WARNING)* e a execução do script continuará.

Exemplo

```
<?php
include "variaveis.php";
echo $a;
echo $b;
?>
```

Arquivo "variaveis.php"

```
<?php
echo "Arquivo contendo as variáveis";
$a = 10;
$b = 20;
?>
```

Caso a inclusão for realizada dentro de uma função, o código do arquivo incluído irá se comportar como se tivesse sido definido dentro da função. Exemplo

```
<?php
function soma($c) {
    include 'variaveis.php';
    return ($a + $b + $c);
}
```

```
$c = 4;
echo "a+b = ".soma($c).", a: '$a.', b : ".$b.", c : ".$c;
?>
```

Para incluir vários arquivos podemos usar um array com os nomes dos arquivos e um comando de repetição, como no exemplo abaixo

```
<?php
$arquivos = array("arq1.php", "arq2.php", "arq3.php");

for($i=0; $i < sizeof($arquivos); $i++){
    include "$arquivos[$i]";
}
?>
```

7.2 include_once

O comando *include_once* possui um comportamento semelhante ao *include*. A única diferença é que se o arquivo já foi incluído, ele não será incluído novamente.

Em caso em que ocorre a inclusão de um mesmo arquivo mais de uma vez, o *include_once* pode ajudar a evitar a redefinições de funções, alteração nos valores das variáveis, etc. Sintaxe:

```
include_once 'nome_do_arquivo';
```

Exemplo da utilização do *include_once*

```
<?php
include_once "arq/funcoes.php";

$c = soma(4.3, 2.5);
$d = subtracao(15, 2);

include_once "arq/funcoes.php"; //o arquivo não será incluído novamente
?>
```

Arquivo "funcoes.php" localizado na pasta "arq".

```
<?php
echo "Funções <br>";

function soma($a, $b) {
    return ($a+$b);
}
```

```
function subtracao($a,$b) {  
    return ($a-$b);  
}  
?>
```

7.3 require

Similar ao comando *include*, o *require* difere na manipulação dos erros. Enquanto o *include* produz uma *warning*, o *require* produzirá uma mensagem de *Erro Fatal (E_COMPILE_ERROR)* caso o arquivo não exista e a execução do programa será parado. Sintaxe:

```
require 'nome_do_arquivo';
```

Exemplo do uso do *require*

```
<html>  
<body>  
    <div class="leftmenu">  
        <?php require 'menu.php'; ?>  
    </div>  
  
    <h1>IFSP - Campus Guarulhos</h1>  
</body>  
</html>
```

Arquivo "menu.php"

```
<?php  
    echo ' <a href="IFSP.php">IFSP</a>  
        <a href="cursos.php">Cursos</a>  
        <a href="estagio.php">Estágio</a>  
        <a href="biblioteca.php">Biblioteca</a>  
        <a href="moodle.php">Moodle</a>  
        ';  
?>
```

7.4 require_once

O comando *require_once* possui um comportamento semelhante ao *require*, exceto que a função irá verificar se o arquivo já foi incluído, em caso positivo, o arquivo não será incluído novamente. Sintaxe:

```
require_once 'nome_do_arquivo';
```

Exemplo

```
<?php
require "arq/funcoes.php";

$c = soma(1,100);
$d = subtracao(9.5,6.3);

require_once "arq/funcoes.php"; //o arquivo não será incluído novamente
?>
```

Capítulo 8

Manipulação de Arquivos

Em aplicações que exigem o armazenamento de poucos dados, podemos trabalhar diretamente com arquivos no formato texto, o que torna mais rápido o processo de armazenamento e recuperação de dados. O PHP oferece uma série de funções exclusivamente para manipulação de arquivos como: abertura, leitura, escrita e fechamento o dos mesmos [Dall'Oglio, 2007].

8.1 fopen

Função para abrir um arquivo. O arquivo pode está na mesma máquina, ou em uma máquina remota. Sintaxe

```
resource fopen(string $nome_arquivo, string $modo[,  
            bool $usar_path[, resource $contexto ]])
```

Se o \$nome_arquivo estiver na forma "protocolo://...", o PHP assumir que seja uma URL. E se o \$nome_arquivo refere-se a um arquivo local, então o PHP abrirá um *stream* para esse arquivo. A função *fopen* retorna FALSE se e a abertura do arquivo falhar.

O arquivo pode ser aberto um dos seguintes modos:

Modo	Descrição
<i>r</i>	Abre um arquivo somente para leitura
<i>r+</i>	Abre um arquivo para leitura e escrita
<i>w</i>	Abre um arquivo somente para escrita
<i>w+</i>	Abre um arquivo somente para escrita e leitura
<i>a</i>	Anexa a um arquivo
<i>a+</i>	Anexa ou cria um arquivo para leitura e escrita
<i>x</i>	Cria um arquivo somente para escrita. Retorna FALSE e um erro se o arquivo já existe
<i>x+</i>	Cria um arquivo para leitura e escrita. Retorna FALSE e um erro se o arquivo já existe

Exemplo

```
<?php  
$file1 = fopen("c:\\pasta\\arquivo.txt", "w");  
$file2 = fopen("/home/usuario/arquivo.gif", "r");  
$file3 = fopen("http://ifspguarulhos.edu.br/arquivo.txt", "r");  
$file4 = fopen("ftp://user:exemplo@ifspguarulhos.edu.br/arquivo.txt",  
            "w");  
?>
```

8.2 fclose

A função *fclose()* é usada para fechar um arquivo aberto apontado pelo identificador de arquivo. A função retorna TRUE em caso de sucesso e FALSE em caso de falha.

```
bool fclose ( resource $identificador )
```

Exemplo

```
<?php
    $file = fopen("arquivo.txt", "r");

    //...

    fclose($file);
?>
```

8.3 feof

A função *feof* testa se o fim do arquivo foi atingido. Se o ponteiro estiver no fim do arquivo (*End Of File*), a função retornará TRUE e se ocorrer um erro (incluindo o limite de tempo de socket), a função retornará FALSE.

```
bool feof( resource $identificador )
```

Exemplo

```
<?php
    $file = fopen("arquivo.txt", "r");

    while (!feof($file)) {
        echo "Ainda não encontrei o fim do arquivo <br>";
    }

    fclose($file);
?>
```

8.4 fgets

Função que lê apenas uma linha do arquivo. A função retorna uma string com até tamanho - 1 bytes lidos do arquivo apontado pelo identificador de arquivo. Se não houver mais dados para ler no arquivo, então a função retorna FALSE.

```
string fgets ( resource $identificador [, int $tamanho ] )
```

O exemplo abaixo lê linha por linha até encontrar o fim do arquivo

```
<?php
$file = fopen("arquivo.txt", "r");

while (!feof($file)) {
    $linha = fgets($file);
    echo "$linha <br>";
}

fclose($file);
?>
```

8.5 fgets

A função *fgets()* é usada para lê somente um caracter do arquivo. A função retorna FALSE quando estiver no fim do arquivo.

```
string fgets( resource $identificador)
```

O exemplo abaixo lê caractere por caractere

```
<?php
$file = fopen("arquivo.txt", "r");

while (!feof($file)) {
    echo fgets($file);
}

fclose($file);
?>
```

8.6 fwrite

A função *fwrite()* escreve o conteúdo de uma string na *stream* apontado pelo identificador de arquivo. A função retorna o número de bytes escritos, ou FALSE em caso de erro.

```
int fwrite( resource $identificador, string $string [, int $tamanho])
```

Se o tamanho do arquivo for fornecido, então a escrita irá parar depois do tamanho em *bytes* ou quando o fim do conteúdo da string for alcançado. Exemplo do funcionamento da função


```
<?php
    $file = fopen("arquivo2.txt","w");

    $conteudo = 'IFSP'

    fwrite($file, $conteudo);
    fwrite($file, ' Guarulhos');

    fclose($file);
?>
```

Capítulo 9

PHP Orientado à Objetos

9.1 Classe

Uma classe é uma abstração ou uma estrutura que define um tipo de dados. Cada classe pode conter atributos(variáveis) e também métodos(funções) para manipular esses atributos. Sintaxe:

```
<?php
class NomeDaClasse{

    var $atributo1;      // Atributos da classe
    var $atributo2;
    //...

    function metodo1(){ // Métodos da classe
    //...
    }
    function metodo1($param1, $param2){
    //...
    }
    //...
}
?>
```

Exemplo

```
<?php
class Pessoa{
    // Atributo da classe
    var $nome = 'Ana';
    // Método da classe
    function mostrar() {
        echo "Nome: ".$this->nome;
    }
}
?>
```

Atributos

Um atributo é qualquer propriedade, característica ou qualidade que possa ser atribuída ao objeto. Exemplos

- A idade de uma pessoa.
- O número de portas de um carro.

Sintaxe:

```
[modificador] nome_do_atributo [=default];
```

ou

```
var nome_do_atributo [=default];
```

Exemplo

```
<?php
class Pessoa{
    // Atributos da classe
    var $nome;
    var $idade = 20;    // atributo iniciado com valor
    // ...

    // Métodos da classe
    // ...
}
?>
```

Métodos

Os métodos definem as funcionalidades ou comportamentos do objeto. Um método equivale a uma função, entretanto ele manipula apenas suas variáveis locais e os atributos que foram definidos na classe. Sintaxe:

```
[modificador] function assinatura(lista de argumentos) {

    // Corpo da método
    ...
}
```

Exemplo

```
<?php
class Pessoa{
    // Atributos da classe
    // ...

    // Métodos da classe
    function mostra() {
        echo "Idade: ".$this->idade;
    }
    // ...
}
?>
```

No PHP Orientado à Objeto, todos os atributos de uma classes devem ser manipulados pela variável *\$this*. Sintaxe

\$this->atributo

Exemplo

```
<?php
class Pessoa{
    // Atributos da classe
    var $idade = 20;

    // Métodos da classe
    function mostra() {
        echo "Idade: ".$this->idade;
    }
}
?>
```

9.1.1 Objeto

O objeto pode ser qualquer entidade física, como um computador ou um funcionário, ou um entidade conceitual, como um arquivo ou uma transferência bancária. O objeto é caracterizado pelas suas características(atributos) e pelo seus comportamentos(métodos).

Criação do objeto

No contexto Orientado a Objeto, um objeto é uma instância de uma classe. Cada instância possui uma identidade única e um estado que represanto pelos valores atuais dos atributos do objeto. O objeto é criado através da seguinte forma:

```
$nome_do_objeto = new Nome_da_classe([argumentos]);
```

Exemplo

```
<?php
include_once 'Pessoa.php';

//Criação do objeto do tipo Pessoa
$objeto_pessoa = new Pessoa();
?>
```

Acessando os atributos e métodos do objeto

Para acessar os atributos ou chamar os métodos de um objeto é preciso utilizar a seguinte sintaxe:

```
$nome_do_objeto->nome_do_atributo
```

```
$nome_do_objeto->nome_do_metodo([argumentos])
```

Exemplo

```
<?php
include_once 'Pessoa.php';

//Criação do objeto do tipo Pessoa
$objeto_pessoa = new Pessoa();

$objeto_pessoa->idade = 21; //Alterando a idade do objeto
$objeto_pessoa->mostra(); //Chamando o método mostra()
?>
```

Array de objetos

No PHP Orientado à Objeto também é possível trabalhar com um array de objetos. Exemplo

```
<?php
include_once 'Pessoa.php';
//Criação do array de Pessoas
$peessoas = array();

$peessoas[0] = new Pessoa(); //Objeto do tipo Pessoa
$peessoas[0]->idade = 18; //Atualiza a idade da primeira pessoa($peessoas[0])

$peessoas[1] = new Pessoa(); //Objeto do tipo Pessoa
```

```

    $pessoas[1]->idade = 22;          //Atualiza a idade da segunda pessoa($pessoas[1])

    $pessoas[0]->mostra(); //Chama o método mostra() da primeira pessoa($pessoas[0])
    $pessoas[1]->mostra(); //Chama o método mostra() da segunda pessoa($pessoas[1])
?>

```

9.1.2 Visibilidade

A visibilidade é utilizada para indicar o nível de acessibilidade de um determinado atributo ou método.

Tipos de visibilidade:

Visibilidade Privada (*private*)

Significa que somente os objetos da classe detentora do atributo ou método poderão enxergá-lo ou acessá-lo.

Exemplo

```

<?php
    class ClasseExemplo{
        private $atributo;
        //...
        private function metodo() {
            //...
        }
    }
?>

```

Visibilidade Protegida(*protected*)

Determina que além dos objetos da classe detentora do atributo ou método também os objetos de suas subclasses poderão ter acesso ao mesmo. Exemplo

```

<?php
    class ClasseExemplo{
        protected $atributo;
        //...
        protected function metodo() {
            //...
        }
    }
?>

```

Visibilidade Pública (*public*)

A visibilidade pública determina que o atributo ou método pode ser utilizado por qualquer objeto.

```
<?php
    class ClasseExemplo{
        public $atributo;
        //...
        public function metodo() {
            //...
        }
    }
?>
```

9.1.3 Encapsulamento

O encapsulamento é um mecanismo que provê a proteção de acesso aos membros internos de um objeto Dall'Oglio [2007]. Os atributos da classe nunca devem ser acessados diretamente de fora do escopo de uma classe Para acessar os atributos privados ou protegidos é preciso criar os métodos

- *Get* - Método para obter valores dos atributos
- *Set* - Método para atribuir valores aos atributos

Exemplo 1

ClasseExemplo.php

```
<?php
    class ClasseExemplo{
        private $atributo = 'valor padrão';

        function setAtributo($valor) {
            $this->atributo = $valor;
        }

        function getAtributo() {
            return $this->atributo;
        }
    }
?>
```

exemplo.php

```
<?php
    include_once 'ClasseExemplo.php';
    $objeto = new ClasseExemplo();
    $objeto->setAtributo("Novo valor");
    echo $objeto->getAtributo();
?>
```

Exemplo 2 - Pessoa.php

```
<?php
class Pessoa{
    private $nome;
    private $idade;
    function setNome($nome) {
        $this->nome = $nome;
    }
    function getNome(){
        return $this->nome;
    }
    //...
}
?>
```

Conta.php

```
<?php
class Conta{
    private $agencia;    // Atributos
    private $cc;
    private $titular;
    private $saldo;
    public function setTitular($nome){
        $this->titular = new Pessoa();
        $this->titular->setNome($nome);
    }
    public function getTitular(){
        return $this->titular->getNome();
    }
    //...
}
?>
```


exemplo_conta.php

```
<?php
    include_once 'Pessoa.php';
    include_once 'Conta.php';

    $cc = new Conta();
    $cc->setTitular("Ana");
    echo $cc->getTitular();
?>
```

9.1.4 Construtor e Destrutor

Construtor

O construtor é método especial utilizado para definir o comportamento inicial de um objeto, ou seja, ações executado durante a criação de um objeto. O método construtor é chamado automaticamente no momento em que instanciamos um objeto através do operador *new*. Quando o construtor não for definido, todos os atributos do objeto criado são inicializadas com o valor NULL. Sintaxe

```
void __construct ([lista_de_argumentos]) {
    // corpo do construtor
}
```

Exemplo 1

MinhaClasse.php

```
<?php
class MinhaClasse{
    private $atributo1;
    // ...

    function __construct () {
        echo "Construtor <br>";
        $this->atributo1 = "Minha Classe";
    }
    // ...
}
```

Exemplo.php

```
<?php
    include_once 'MinhaClasse.php';
    $objeto = new MinhaClasse(); // Chama o construtor da MinhaClasse.
?>
```

Exemplo 2

ClasseA.php

```
<?php
class ClasseA{
    private $atributo1;
    // ...

    function __construct($valor) {
        echo "Construtor <br>";
        $this->atributo1 = $valor;
    }

    // ...
}
?>
```

ExemploA.php

```
<?php
include_once 'ClasseA.php';

// Chama o construtor passando um argumento.
$objeto = new MinhaClasse("Exemplo A");
?>
```

Para construir vários construtores para mesma classe é preciso criar o construtor que identifique quantos parâmetros está sendo passado para o construtor e em seguida é chamado o construtor correspondente.

Exemplo

ClasseA.php

```
<?php
class ClasseA{
    function __construct() {
        $a = func_get_args(); //array com os parâmetros enviados
        $i = func_num_args(); //número de parâmetros enviados

        if (method_exists($this, $f='__construct' . $i)) {
            // Chama o construtor correspondente
            call_user_func_array(array($this, $f), $a);
        }
    }
}
```

```

function __construct1($a1) {
    echo('Construtor com 1 param:'. $a1);
}
function __construct2($a1, $a2) {
    echo('Construtor com 2 params:'. $a1.', '. $a2);
}
function __construct3($a1, $a2, $a3) {
    echo('Construtor com 3 params:'. $a1.', '. $a2.', '. $a3);
}
}
?>

```

exemplo.php

```

<?php
include_once 'ClasseA.php';
$obj1 = new ClasseA();
$obj2 = new ClasseA(' IFSP ');
$obj3 = new ClasseA(' IFSP ', 'Guarulhos');
$obj3 = new ClasseA(' IFSP ', 'Guarulhos', 'Sao Paulo');
?>

```

Destrutor

O destrutor ou finalizador é um método especial executado quando a objeto é desalocado da memória. O destrutor é chamado quando:

- Atribuir NULL ao objeto,
- Utilizar a função unset() sobre o objeto,
- O programa é finalizado,
- O método é utilizado para finalizar conexões, apagar arquivos temporários, etc.

Sintaxe do destrutor

```

void __destruct() {
    // corpo do construtor
}

```

Exemplo

MinhaClasse.php

```
<?php
class MinhaClasse{
    private $atributo1;
    function __construct() {
        echo "Construtor <br>";
        $this->atributo1 = "Minha Classe";
    }
    function __destruct() {
        echo "Destruindo " . $this->atributo1 . "<br>";
    }
    //...
}
?>
```

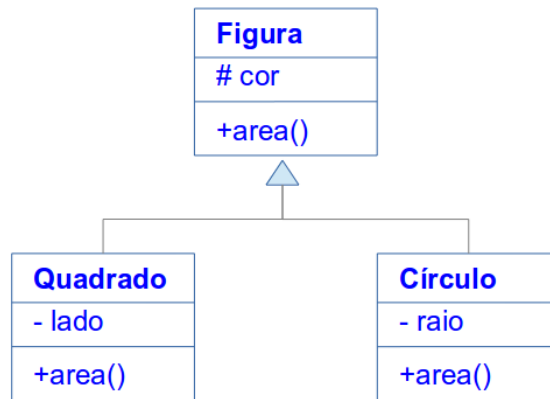
```
<?php
include_once 'MinhaClasse.php';

$objeto = new MinhaClasse();

// $objeto = null;                                // Destruindo o objeto
?>                                                // Destruindo o objeto
```

9.1.5 Herança

A herança é um recurso da programação orientada à objetos que permite o compartilhamento de atributos e métodos entre classes de uma mesma hierarquia (árvore). As classes inferiores da hierarquia (Subclasses) automaticamente herdam todas as propriedades e métodos das classes superiores (Superclasses) Exemplo



Nesse exemplo, a classe Figura é uma Superclasse para classes Quadrado e Círculo.

Hierarquia

- Superclasse (Classe Pai)
 - Possui os atributos e métodos que serão herdados.
 - Desconhece as suas subclasses.
- Subclasse (Classe Filha)
 - Classe considerada como uma especialização da Superclasse.
 - Herda todas as propriedades e métodos das classes superiores.
 - É possível adicionar na Subclasse comportamentos que não são especificados na Superclasse
 - A subclasse pode também ser uma Superclasse de futuras subclasses

Acessibilidade

Todos os atributos e métodos protegidos de uma Superclasse podem ser acessados pelas suas subclasses.

```
protected $atributo;
```

```
protected function metodoX() {}
```

Exemplo de herança.

ClasseA.php

```
<?php
class ClasseA {
    protected $atributo;
    function __construct() {
        echo "<br>Inicializando a ClasseA<br>";
        $this->atributo = "X";
    }
}
```

```

function __destruct() {
    echo "<br><br>Finalizando a ClasseA";
}
public function mostraA(){
    echo "<br>ClasseA - atributo:". $this->atributo;
}
}
?>

```

ClasseB.php

```

<?php
include_once "ClasseA.php";

class ClasseB extends ClasseA{
    private $atributoB;

    function __construct() {
        echo "<br>Inicializando a ClasseB";
        $this->atributoB = "Y";
        $this->atributo = "Z";
    }
    function __destruct() {
        echo "<br>Finalizando a ClasseB";
    }
    public function mostraB(){
        echo "<br>ClasseB - atributo:". $this->atributoB;
        $this->mostraA();
    }
}
?>

```

```

<?php
include_once "ClasseA.php";
include_once "ClasseB.php";

$objA = new ClasseA();
$objA->mostraA();
$objB = new ClasseB();
$objB->mostraB();
?>

```

Em PHP, o construtor da Superclasse não é chamado implicitamente pelo construtor da Subclasse. O construtor da Superclasse pode ser chamado pela Subclasse através da palavra-chave *parent*:

```
parent::__construct([lista_de_argumentos]);
```

Exemplo

ClasseA.php

```
<?php
class ClasseA {

    protected $atributo;

    function __construct() {
        echo "<br>Iniciando a ClasseA<br>";
        $this->atributo = "X";
    }
    function __destruct() {
        echo "<br><br>Finalizando a ClasseA";
    }
    public function mostraA() {
        echo "<br>ClasseA - atributo:". $this->atributo;
    }
}
?>
```

ClasseB.php

```
<?php
include_once "ClasseA.php";

class ClasseB extends ClasseA{

    private $atributoB;

    function __construct() {
        echo "<br>Iniciando a ClasseB";
        $this->atributoB = "Y";
        parent::__construct(); //Chama o construtor da Superclasse
    }
    //...
}
?>
```

exemplo.php

```
<?php

include_once "ClasseA.php";
include_once "ClasseB.php";

$objA = new ClasseA();
$objA->mostraA();

$objB = new ClasseB(); //Chama os construtores da ClasseB e ClasseA
$objB->mostraB();

?>
```

Assim como os construtores, o destrutor da Superclasse também não pode ser herdado pelas Subclasses. O destrutor da Subclasse deve chamar o destrutor da sua Superclasse através da palavra-chave *parent*

```
parent::__destruct();
```

Exemplo - ClasseB modificada para chamar o destrutor da ClasseA.

ClasseB.php

```
<?php

include_once "ClasseA.php";

class ClasseB extends ClasseA{

    private $atributoB;

    function __construct() {
        echo "<br>Inicializando a ClasseB";
        $this->atributoB = "Y";
        parent::__construct();
    }

    function __destruct() {
        echo "<br>Finalizando a ClasseB";
        parent::__destruct(); //Chama o destrutor da Superclasse
    }

    //...
}

?>
```


exemplo.php

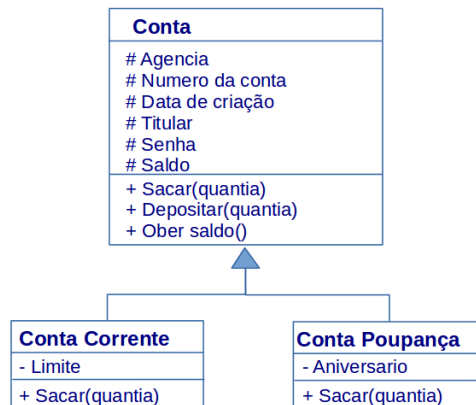
```
<?php
include_once "ClasseB.php";

$objB = new ClasseB();
$objB->mostraB();

?>
```

9.1.6 Polimorfismo

Polimorfismo é uma palavra derivada do grego que significa “muitas formas”. Na programação orientação a objeto é o principio que permite que Subclasses tenham métodos iguais, ou seja métodos que possuem a mesma assinatura da Superclasse, mas com comportamentos diferentes. Os métodos que sofrem o polimorfismo são redefinidos em cada uma das subclasses. Exemplo



No exemplo acima tanto a classe Conta Poupança quanto a Conta Corrente possuem o método sacar(). Na Conta Poupança, é preciso verificar se há saldo na conta. E na Conta Corrente, é preciso verificar se a retirada está dentro do limite do titular, caso for preciso utilizar o limite o imposto IOF (Imposto sobre Operações Financeiras) será cobrado da conta do titular.

Exemplo de polimorfismo em PHP

ClasseA.php

```
<?php
class ClasseA {
    protected $atributoA = "A";

    public function mostra() {
        echo "<br>ClasseA - atributo:". $this->atributoA;
    }
    // ...
}

?>
```

ClasseB.php

```
<?php
    include_once "ClasseA.php";

    class ClasseB extends ClasseA{

        private $atributoB = "B";

        public function mostra() {
            echo "<br>ClasseB - atributo: ".$this->atributoB;
        }

        //...
    }
?>
```

```
<?php
    include_once "ClasseA.php";
    include_once "ClasseB.php";

    $objA = new ClasseA();
    $objA->mostra();           /////Chama o método da ClasseA

    $objB = new ClasseB();
    $objB->mostra();           //Chama o método da ClasseB
?>
```

E para chamar o método original da Superclasse é preciso usar a palavra-chave *parent*

```
parent::metodo([lista_de_argumentos]);
```

Exemplo- Modificando a ClasseB

```
<?php
    include_once "ClasseA.php";
    class ClasseB extends ClasseA{
        private $atributoB;
        public function mostra() {
            echo "<br>ClasseB - atributo: ".$this->atributoB;
            parent::mostra();
        }
    }
?>
```

```
<?php
    include_once "ClasseB.php";

    $objB = new ClasseB();
    $objB->mostra();
?>
```

9.1.7 Evitando a herança

Um método pode ser marcado como um método final utilizando a palavra-chave *final* no início da sua declaração. Um método final impede a sobrescrita do método nas Subclasses. Sintaxe

```
final function metodo([lista_de_argumentos]) {
    //...
}
```

Exemplo

```
<?php
class ClasseA {

    protected $atributoA = "A";

    final function mostra() {
        echo "<br>ClasseA - atributo:". $this->atributoA;
    }

    //...
}
?>
```

Uma classe pode evitar ter Subclasses se ela ser marcado como uma classe final. Como método final, basta adicionar a palavra-chave *final* no início da sua declaração.

```
final class Nome_da_classe {

    // Atributos

    // Métodos
}
```

Exemplo - Classe Final

```
<?php
final class ClasseFinal {

    protected $atributoA = "A";

    function mostra() {
        echo "<br>Classe Final";
    }

    //...
}
?>
```

9.1.8 Classe Abstrata

Uma classe abstrata é uma classe que serve de base para outras classe. Uma classe abastrata nunca pode ser instanciada na forma de objeto, somente suas Subclasses poderam ser instaciadas. A linguagem PHP irá automaticamente impedir que se instanciem objetos de classes abastratas. Sintaxe

```
abstract class Nome_da_classe {

    // Atributos

    // Métodos
}
```

Exemplo ClasseA.php

```
<?php
abstract class ClasseA {

    protected $atributoA = "A";

    public function mostra() {
        echo "<br>ClasseA - atributo:". $this->atributoA;
    }

    //...
}
?>
```

ClasseB.php

```
<?php
    include_once "ClasseA.php";

    class ClasseB extends ClasseA{

        //...
    }
?>
```

```
<?php
    include_once "ClasseA.php";
    include_once "ClasseB.php";

    //Fatal error: Cannot instantiate abstract class ClasseA
    $objA = new ClasseA();

    $objB = new ClasseB();
    $objB->mostra();
?>
```

Método Abstrato

Um método abstrato é um método que fornece assinatura para um método, mas sem nenhuma implementação. E qualquer classe que possuir métodos abstratos também deve ser abstrata. A implementação deverá ser feita nas subclasses. O principal uso de classes e métodos abstratos é garantir que cada subclasse sobrescreva os métodos abstratos da superclasse. Sintaxe

```
abstract function nome_do_metodo([lista_de_argumentos]);
```

Exemplo - ClasseA.php

```
<?php

abstract class ClasseA {

    abstract function mostra();

    //...
}
?>
```

ClasseB.php

```
<?php

include_once "ClasseA.php";

class ClasseB extends ClasseA{

    private $atributoB;

    public function mostra() {

        echo "<br>ClasseB - atributo:". $this->atributoB;

    }

    // ...

}

?>
```

```
<?php

include_once "ClasseB.php";

$objB = new ClasseB();
$objB->mostra();

?>
```

9.1.9 Interfaces

Interfaces são um conjunto de métodos que determinadas classes deverão implementar. Na declaração desses métodos devem ser informados a visibilidade, os nomes dos métodos e os respectivos parâmetros, sem qualquer implementação. A classe que implementar uma interface tem a obrigação de implementar todos os métodos definidos pela interface. Exemplo

IPessoa.php

```
<?php

interface IPessoa{

    function setNome($nome);

    function getNome();

}

?>
```

Pessoa.php

```
<?php
    include_once 'IPessoa.php';

    class Pessoa implements IPessoa{
        private $nome;
        private $idade;
        //...

        function setNome($nome) {
            $this->nome = $nome;
        }

        function getNome() {
            return $this->nome;
        }
        //...
    }
?>
```

Exemplo

```
<?php
    include_once 'Pessoa.php';

    //Criação do objeto do tipo Pessoa
    $objeto_pessoa = new Pessoa();

    $objeto_pessoa->setNome("Ana");
    echo $objeto_pessoa->getNome();
?>
```

Capítulo 10

PHP com Banco de Dados

10.1 MySQL

O MySQL é o banco de dados mais popular usado com PHP. O MySQL é compatível com as linguagens PHP, C/C++, C#, Java entre outras. Ele é ideal para pequenas e grandes aplicações e ainda suporta o padrão SQL (*Structured Query Language*).

A combinação PHP/MySQL é multiplataforma, por exemplo podemos desenvolver na plataforma Windows e usar na plataforma UNIX. O download do MySQL pode ser feito em <http://www.mysql.com>.

10.1.1 phpmyadmin

O *phpMyAdmin* é uma ferramenta de software livre escrito em PHP, destinado a lidar com o gerenciamento do MySQL pela web. O *phpMyAdmin* suporta uma ampla variedade de operações em MySQL, MariaDB e Drizzle.



Operações como criar, alterar e remover tabelas e bases de dados, inserir, editar e remover dados de tabelas podem ser realizadas através de uma interface com o usuário. Além de possuir a capacidade de executar diretamente qualquer consulta SQL. O *phpMyAdmin* está disponível para download em <http://www.phpmyadmin.net>

10.2 Comando básicos do MySQL via PHP

10.2.1 Conexão com o Banco de Dados

Abrir conexão

No PHP, utilizamos a função `mysqli_connect()` para criar uma conexão com a base de dados no servidor *MySQL*. Sintaxe:

```
mysqli_connect(host, login, senha);
```

ou

```
mysqli_connect(host, login, senha, bd_nome);
```

Onde `host` é o hostname ou ip do servidor, `login` é o usuário do banco de dados MySQL, `senha` é a senha do usuário do banco de dados e `bd_nome` é nome da base de dados. Exemplo

```
<?php
// Criando uma conexão com o bando de dados
$con = mysqli_connect("localhost", "root", "");

// Checando a conexão
if (mysqli_connect_errno($con)) {
    echo "Erro ao conectar com a base de dados: ".
        mysqli_connect_error();
} else {
    echo "Conexão Aberta";
}
?>
```

Exemplo2

```
<?php
// Criando uma conexão com a base de dados bd_exemplo
$con = mysqli_connect("localhost", "root", "", "bd_exemplo");

// Checando a conexão
if (mysqli_connect_errno($con)) {
    echo "Erro ao conectar com a base de dados: ".
        mysqli_connect_error();
}
?>
```

Fechar conexão

A função `mysqli_close()` fecha a conexão com banco de dados estabelecida com a função `mysqli_connect`.

```
mysqli_close($con);
```

Caso a variável `$con` não for fornecida então a última conexão estabelecida será fechada. Exemplo

```
<?php
// Criando conexão com o bando de dados
$con = mysqli_connect("localhost", "root", "", "bd_exemplo");

// Checar a conexão
if (mysqli_connect_errno($con)) {
    echo "Erro ao conectar com a base de dados: ".
        mysqli_connect_error();

} else {
    echo "Conexão Aberta";
    mysqli_close($con);
}
?>
```

10.2.2 Consulta SQL no MySQL

A função `mysqli_query()` é utilizada para realizar uma query(consulta) SQL no MySQL. A função recebe a conexão estabelecida juntamente com uma string de query(consulta) SQL.

```
mysqli_query($con, $sql);
```

A função retornará um objeto `mysqli_result` em caso de sucesso com os comandos SELECT, SHOW, DESCRIBE e EXPLAIN ou retorna TRUE nos outros casos. Em caso de falha a função retorna FALSE.

Criação de uma nova base de dados

Para criar uma base de dados é preciso utilizar a seguinte comando SQL

```
$sql = "CREATE DATABASE bd_nome";
```

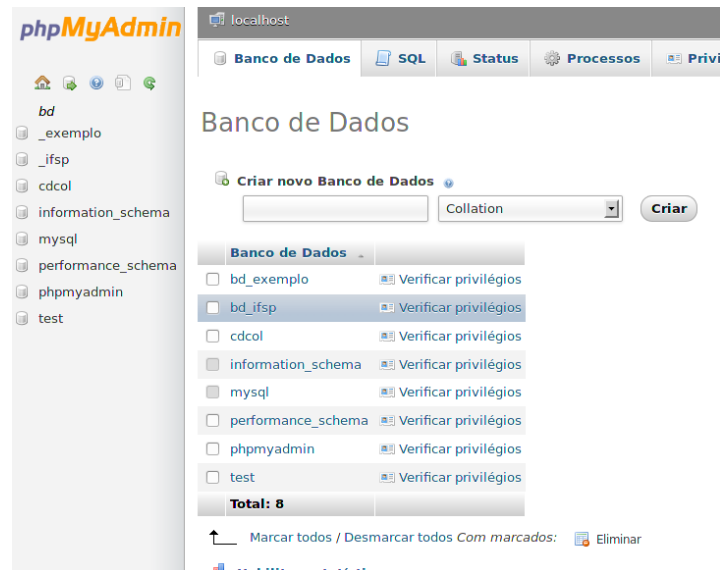
```
mysqli_query($con, $sql);
```

Exemplo de criação de base de dados

```
<?php
$con = mysqli_connect (localhost,root,);

if (mysqli_connect_errno($con)) {
    echo "Erro ao conectar com a base de dados: ".
    mysqli_connect_error();
}else{
    // Criando base de dados
    $sql="CREATE DATABASE db_ifsp";
    if (mysqli_query ($con,$sql)) {
        echo "Base de Dados criada com sucesso!!";
    }else{
        echo "Erro: " .mysqli_error($con);
    }
    mysqli_close ($con);
}
?>
```

Podemos verificar pelo *phpmyadmin* que a nova base de dados *db_ifsp* foi criada com sucesso.



Remoção de uma base de dados

Uma base de dados pode ser removida através da query.

```
$sql = "DROP DATABASE bd_nome";
mysqli_query ($con, $sql);
```

Exemplo

```
<?php

$con = mysqli_connect (localhost,root,);

$sql="DROP DATABASE db_ifsp";

if (mysqli_query ($con,$sql)) {
    echo "Base de Dados removida com sucesso!!";
}else{
    echo "Erro: " .mysqli_error ($con);
}
mysqli_close ($con);
?>
```

Criação de tabelas na base de dados

A instrução *CREATE TABLE* é usada para criar uma tabela na base de dados. As tabelas são organizadas em linhas e colunas e cada tabela deve possuir um nome [W3Schools, 2014b]. Sintaxe

```
$sql = "CREATE TABLE nome_tabela (coluna1 tipo_de_dado (tamanho), coluna2
      tipo_de_dado (tamanho), ....)";

mysqli_query ($con,$sql);
```

O seguinte exemplo cria a tabela chamada Pessoa com três colunas: "Nome", "Sobrenome" e "Idade".

```
<?php
// Criando conexão com a base de dados bd_ifsp
$con = mysqli_connect ("localhost","root","","bd_ifsp");

$sql="CREATE TABLE Pessoa (Nome CHAR(30), Sobrenome CHAR(30),
      Idade INT) ";

if (mysqli_query ($con,$sql)) {
    echo "Tabela criada com sucesso!!";
}else{
    echo "Erro: " .mysqli_error ($con);
}
mysqli_close ($con);
?>
```

Inserção de dados em uma tabela

A instrução *INSERT INTO* é utilizado para adicionar novos registros a uma tabela de banco de dados. É possível inserir dados de duas formas:

```
$sql = "INSERT INTO nome_tabela VALUES (valor1, valor2, valor3, ...)";
```

```
mysqli_query($con, $sql);
```

ou

```
$sql = "INSERT INTO nome_tabela ( coluna1, coluna2, coluna3, ...)
VALUES (valor1, valor2, valor3, ...)";
```

```
mysqli_query($con, $sql);
```

O exemplo abaixo mostra a inserção de dois registros na tabela Pessoa.

```
<?php
// Criando conexão com a base de dados bd_ifsp
$con = mysqli_connect("localhost", "root", "", "bd_ifsp");

// Check connection
if (mysqli_connect_errno()) {
    echo "Falha ao conectar com o MySQL: " . mysqli_connect_error();
}

$sql1 = "INSERT INTO Pessoa
VALUES ('Ana', 'Souza', 20)";

mysqli_query($con, $sql1);

$sql2 = "INSERT INTO Pessoa (Nome, Sobrenome, Idade)
VALUES ('Carlos', 'Costa', 21)";

mysqli_query($con, $sql2);

mysqli_close($con);
?>
```

Podemos verificar pelo *phpmyadmin* que as duas pessoas foram inseridas na tabela Pessoa.

Exemplo de formulário para inserir dados em uma tabela

```
<html>
<body>
  <form action="inserir.php" method="post">
    Nome: <input type="text" name="nome"> </p>
    Sobrenome: <input type="text" name="sobrenome"></p>
    Idade: <input type="text" name="idade"> </p>

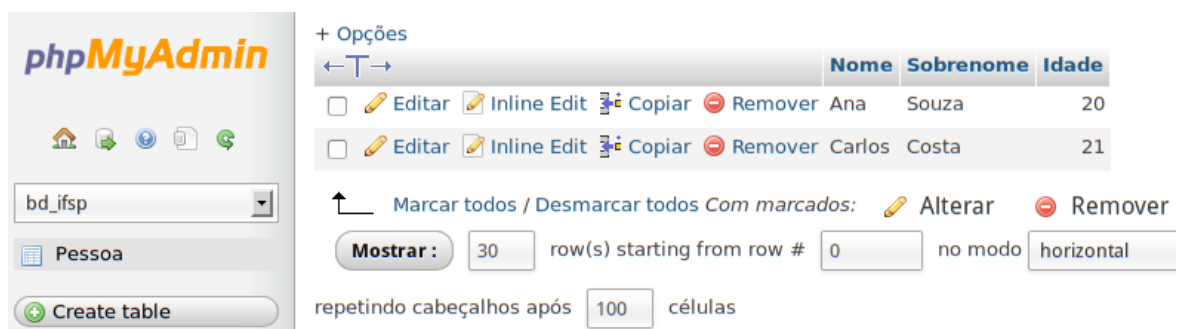
    <input type="submit" value="Enviar">
  </form>
</body>
</html>
```

Arquivo "inserir.php"

```
<?php
$con = mysqli_connect("localhost","root","","bd_ifsp");

if (mysqli_connect_errno($con)) {
    echo "Erro: " . mysqli_connect_error();
}else{
    $sql = "INSERT INTO Pessoa
            VALUES ('$_POST[nome]', '$_POST[sobrenome]', $_POST[idade]) ";

    if(mysqli_query($con,$sql)){
        echo "Pessoa inserida com sucesso!!!";
    }else{
        echo "Erro: " .mysqli_error($con);
    }
    mysqli_close($con);
}
?>
```



phpMyAdmin

bd_ifsp

Pessoa

Create table

+ Opções

	Nome	Sobrenome	Idade
<input type="checkbox"/> Editar <input type="checkbox"/> Inline Edit <input type="checkbox"/> Copiar <input type="checkbox"/> Remover	Ana	Souza	20
<input type="checkbox"/> Editar <input type="checkbox"/> Inline Edit <input type="checkbox"/> Copiar <input type="checkbox"/> Remover	Carlos	Costa	21

↑ Marcar todos / Desmarcar todos Com marcados: ☐ Alterar ☐ Remover

Mostrar: 30 row(s) starting from row # 0 no modo horizontal

repetindo cabeçalhos após 100 células

Consultar dados

A instrução *SELECT* é realizar uma consulta na base de dados. O resultado da consulta é armazenado no objeto *\$resultado* do tipo *mysqli_result*

```
$sql = "SELECT coluna1, coluna2, ...  
      FROM nome_tabela";  
  
$resultado = mysqli_query($con, $sql);
```

O seguinte exemplo seleciona os nomes e sobrenomes dos dados armazenados na tabela Pessoa.

```
<?php  
$con = mysqli_connect("localhost", "root", "", "bd_ifsp");  
  
if (mysqli_connect_errno($con)) {  
    echo "Erro: " . mysqli_connect_error();  
} else {  
    $sql = "SELECT Nome, Sobrenome  
          FROM Pessoa ";  
  
    $resultado = mysqli_query($con, $sql);  
  
    echo "<h2>Pessoas</h2>";  
  
    while($pessoa = mysqli_fetch_array($resultado)) {  
        echo $pessoa['Nome'] . " " . $pessoa['Sobrenome'] . "<br>";  
    }  
    mysqli_close($con);  
}  
?>
```

O exemplo acima usamos a função *mysqli_fetch_array()* para retornar a primeira linha do conjunto de registros como um *array*. Cada chamada para *mysqli_fetch_array()* retorna a próxima linha no conjunto de registros. O loop *while* percorre todos os registros no conjunto de registros. Para imprimir o valor de cada linha, usamos o PHP variável *\$pessoa* (*\$pessoa['Nome']* e *\$pessoa['sobrenome']*).

Resultado da consulta:

```
Pessoas  
  
Ana Souza  
Paulo Lima  
Carlos Costa
```

O Próximo exemplo seleciona todos os dados armazenados na tabela Pessoa. O caractere * é usado para selecionar todos os dados em uma tabela.

```
<?php
$con = mysqli_connect("localhost","root","","bd_ifsp");

if (mysqli_connect_errno($con)) {
    echo "Erro: " . mysqli_connect_error();
}else{
    $sql = " SELECT *
            FROM Pessoa ";
    $resultado = mysqli_query($con,$sql);

    echo "<h2>Pessoas</h2>";

    while($pessoa = mysqli_fetch_array($resultado)) {
        echo $pessoa['Nome'] . " " . $pessoa['Sobrenome'] . ", " . " .
            $pessoa['Idade'] . "anos<br>";
    }
    mysqli_close($con);
}
?>
```

Saída:

Pessoas

Ana Souza, 20 anos

Paulo Lima, 19 anos

Carlos Costa, 21 anos

Cláusula where

A cláusula *where* é usada para extrair apenas os registros que satisfazem um critério especificado.

```
$sql = "SELECT coluna1, coluna2,...
        FROM nome_tabela
        WHERE coluna_i operador valor";

$resultado = mysqli_query($con,$sql);
```


O exemplo abaixo seleciona o nome e sobrenome da tabela Pessoa que possui idade maior que 19.

```
<?php
$con = mysqli_connect("localhost","root","","bd_ifsp");

if (mysqli_connect_errno($con)) {
    echo "Erro: " . mysqli_connect_error();
}else{
    $sql = "SELECT Nome, Sobrenome
            FROM Pessoa
            WHERE (Idade > 19) ";
    $resultado = mysqli_query($con,$sql);

    echo "<h2>Pessoas</h2>";

    while($pessoa = mysqli_fetch_array($resultado)) {
        echo $pessoa['Nome'] . " " .
            $pessoa['Sobrenome'] . "<br>";
    }
    mysqli_close($con);
}
?>
```

Saída:

Pessoas

Ana Souza

Carlos Costa

Ordenar uma consulta

A palavra-chave *ORDER BY* palavra-chave é usada para classificar os dados de um conjunto de registros. Os registros são classificados em ordem crescente (*ASC by default*) ou decrescente (*DESC*).

```
$sql = "SELECT coluna1, coluna2,...
        FROM nome_tabela
        ORDER BY colunas ASC|DESC;";

$resultado = mysqli_query($con,$sql);
```

O próximo exemplo seleciona todos os dados armazenados na tabela Pessoas, e classifica o resultado pelo Sobrenome em ordem decrescente.

```

<?php
    $con = mysqli_connect("localhost","root","","bd_ifsp");

    if (mysqli_connect_errno($con)){
        echo "Erro: " . mysqli_connect_error();
    }else{
        $sql = "SELECT *
                FROM Pessoa
                ORDER BY Sobrenome DESC ";

        $resultado = mysqli_query($con,$sql);

        echo "<h2>Pessoas</h2>";
        while($pessoa = mysqli_fetch_array($resultado)){
            echo $pessoa['Nome'] . " " . $pessoa['Sobrenome'] . " " .
                $pessoa['Idade'] . " anos<br>";
        }
        mysqli_close($con);
    }
?>

```

Saída

Pessoas

Ana Souza, 20 anos

Paulo Lima, 19 anos

Carlos Costa, 21 anos

O próximo exemplo seleciona todos os dados armazenados na tabela Pessoas, e depois classifica o resultado pelo Sobrenome seguido do Nome

```

<?php
    $con = mysqli_connect("localhost","root","","bd_ifsp");

    if (mysqli_connect_errno($con)){
        echo "Erro: " . mysqli_connect_error();
    }else{
        $sql = "SELECT *
                FROM Pessoa
                ORDER BY Sobrenome, Nome ";

        $resultado = mysqli_query($con,$sql);
    }

```

```

echo "<h2>Pessoas</h2>";
while($pessoa = mysqli_fetch_array($resultado)){
    echo $pessoa['Nome'] . " " . $pessoa['Sobrenome'] . " " .
        $pessoa['Idade'] . " anos<br>";
}
mysqli_close($con);
}
?>

```

Saída:

Pessoas

Carlos Costa, 21 anos

Paulo Lima, 19 anos

Ana Souza, 20 anos

Paginação dos resultados

A palavra-chave *LIMIT* é usada para limitar o número de linhas do resultado da consulta e a palavra-chave *OFFSET* indica o início da consulta.

```

$sql = "SELECT coluna1, coluna2,...
        FROM nome_tabela
        LIMIT limite
        OFFSET inicio";

$resultado = mysqli_query($con,$sql);

```

O exemplo abaixo seleciona todos os dados armazenados na tabela Pessoas, iniciando a consulta a partir do segundo registro e depois retorna apenas os cinco primeiros registros.

```

<?php
$con = mysqli_connect("localhost","root","","bd_ifsp");

if (mysqli_connect_errno($con)) {
    echo "Erro: " . mysqli_connect_error();
}else{
    $sql = "SELECT *
            FROM Pessoa
            LIMIT 5
            OFFSET 2";

    $resultado = mysqli_query($con,$sql);

```

```

echo "<h2>Pessoas</h2>";
while($pessoa = mysqli_fetch_array($resultado)){
    echo $pessoa['Nome'] . " " . $pessoa['Sobrenome'] . ", " .

        $pessoa['Idade'] . " anos<br>";
}
mysqli_close($con);
}
?>

```

Saída:

Pessoas

Paulo Lima, 19 anos

Carlos Costa, 21 anos

Atualização dos dados de uma tabela

A instrução *UPDATE* é usada para atualizar os registros existentes em uma tabela.

```

$sql = "UPDATE nome_tabela
        SET coluna1=valor, coluna2=valor, ...
        WHERE colunas = valor";

mysqli_query($con,$sql);

```

O exemplo a seguir altera todos os registros da tabela Pessoa com *Idade* = 19 para *Idade* = 23.

```

<?php
$con = mysqli_connect("localhost","root","","bd_ifsp");

if (mysqli_connect_errno($con)) {
    echo "Erro: " . mysqli_connect_error();
}else{
    $sql1 = "UPDATE Pessoa
            SET Idade = 23
            WHERE Idade = 19 ";
    mysqli_query($con,$sql1);

    $sql2 = "SELECT * FROM Pessoa WHERE Idade = 23";
    $resultado = mysqli_query($con,$sql);
}

```

```

echo "<h2>Pessoa</h2>";
$pessoa = mysqli_fetch_array($resultado)
echo $pessoa['Nome'] . " " . $pessoa['Sobrenome'] . ", " .
    $pessoa['Idade'] . " anos<br>";
mysqli_close($con);
}
?>

```

Saída:

Pessoa

Paulo Lima, 23 anos

Remoção de dados de uma tabela

A instrução *DELETE FROM* é usada para remover registros de uma tabela de banco de dados.

```

$sql = "DELETE FROM nome_tabela
        WHERE coluna = valor";

```

```

mysqli_query($con, $sql);

```

O seguinte exemplo remove os registros da tabela Pessoa com o nome = 'Carlos'

```

<?php
// Criando conexão com a base de dados bd_ifsp
$con = mysqli_connect("localhost","root","","bd_ifsp");

// Check connection
if (mysqli_connect_errno()) {
    echo "Falha ao conectar com o MySQL: " . mysqli_connect_error();
}

$sql = "DELETE FROM Pessoa
        WHERE Nome ='Carlos' ";

mysqli_query($con, $sql);

mysqli_close($con);
?>

```

10.3 Classes MSQli e MySQLi_Result

10.3.1 Classe MSQli

Classe que representa uma conexão entre o PHP e um banco de dados MySQL Welling and Thomson [2005]. Exemplos de atributos e métodos:

- Atributos
 - *affected_rows* - Número de linhas afetadas pela operação MySQL anterior.
 - *error* - String que descreve o ultimo erro.
 - *server_info* - Versão do servidor MySQL.
 - *server_version* - Versão do servidor MySQL como um integer.
 - ...
- Métodos
 - *__construct()* - Abre uma conexão com servidor MySQL.
 - *close()* - Fecha uma conexão aberta anteriormente.
 - *select_db()* - Seleciona uma base de dados(database).
 - *query()* - Realiza uma consulta na base de dados.
 - ...

10.3.2 Classe MySQLi_Result

Representa o conjunto de resultados da consulta feita à base da dados(database)

- Atributos
 - *num_rows* - Número de linhas retornado da consulta
 - *field_count* - Número de colunas retornada da consulta
 - *lengths* - Tamanho das colunas da atual linha retornado da consulta
 - *current_field* - Linha atual retornado do método *fetch_field()*
 - ...
- Métodos
 - *mysqli_fetch_assoc()* - Obtem uma linha do conjunto de resultados como uma matriz associativa.
 - *mysqli_fetch_row()* - Obtém uma linha do resultado como uma matriz numerada *fetch_object()*.
O método retorna a linha atual do conjunto de resultados como um objeto
 - *free()* - Libera memória do conjunto de resultados chamado
 - ...

10.3.3 Conexão com o Banco de Dados

A conexão com o banco de dados é realizada através da chamada do construtor da classe MySQLi.

```
__construct(string $host, string $username, string $passwd,  
            string $dbname, string $port, string $socket);
```

Exemplo 1

```
<?php  
$host = "localhost"; $username = "root"; $passwd = "";  
  
// Criacao do objeto $db e conexão com o banco de dados  
$db = new mysqli($host,$username,$passwd);  
  
if ($db->connect_error){// Verifica a conexão  
    die('Erro('.$db->connect_errno.')'.$db->connect_error);  
}else{  
    echo "Conexão aberta";  
}  
?>
```

Exemplo 2

```
<?php  
$host = "localhost"; $username = "root"; $passwd = "";  
$dbname = "ifsp";  
  
// Criacao do objeto $db e conexão com a base de dados "ifsp"  
$db = new mysqli($host,$username,$passwd,$dbname);  
  
if ($db->connect_error){// Verifica a conexão  
    die('Erro('.$db->connect_errno.')'.$db->connect_error);  
}else{  
    echo "Conexão aberta";  
}  
?>
```

Fechar conexão

O método *close()* fecha uma conexão com banco de dados aberta anteriormente.

```
bool close ( void )
```

Exemplo do uso do método.

```
<?php
    $host = "localhost";    $username = "root";    $passwd = "";
    $dbname = "ifsp";

    $db = new mysqli($host,$username,$passwd,$dbname);
    if ($db->connect_error){// Verifica a conexão
        die('Erro('.$db->connect_errno.')'.$db->connect_error);
    }else{
        echo "Conexão aberta";
        //...

        $db->close();
    }
?>
```

10.3.4 Selecionar uma base de dados

O método *select_db()* seleciona uma nova base de dados *default* para as consultas à base de dados selecionada.

```
bool select_db(string $dbname)
```

Exemplo

```
<?php
    $host = "localhost";    $username = "root";    $passwd = "";
    $dbname = "ifsp";

    $db = new mysqli($host,$username,$passwd);
    if ($db->connect_error){// Verifica a conexão
        die('Erro('.$db->connect_errno.')'.$db->connect_error);
    }else{
        echo "Conexão aberta";

        $db->select_db("books");
        //...

        $db->close();
    }
?>
```


10.3.5 Consulta SQL no MySQL

O método *query()* é utilizada para realizar uma *query*(consulta) SQL no MySQL. O método recebe a string de *query*(consulta) SQL.

```
mixed query(string $query)
```

O método retornará um objeto *mysqli_result* em caso de sucesso com os comandos SELECT, SHOW, DESCRIBE e EXPLAIN ou retorna TRUE nos outros casos. Em caso de falha a função retorna FALSE.

Criação de base de dados

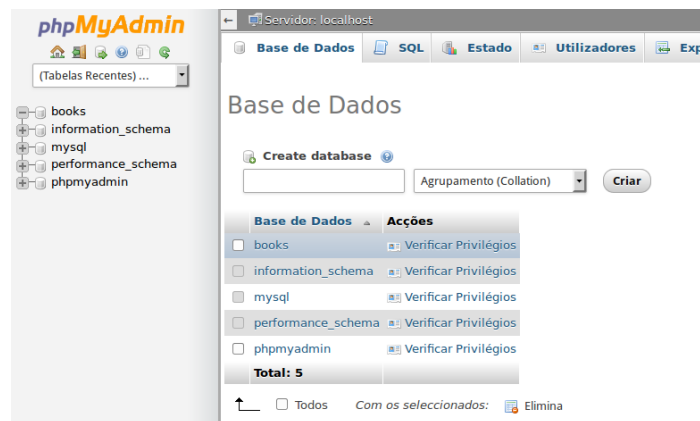
Para criar uma base de dados é preciso utilizar a seguinte comando SQL

```
$sql = "CREATE DATABASE bd_nome";  
$db->query($sql);
```

Exemplo de criação de uma base de dados

```
<?php  
$host = "localhost";    $username = "root";    $passwd = "";  
$dbname = "books";  
  
$db = new mysqli($host,$username,$passwd);  
  
if ($db->connect_error){// Verifica a conexão  
    die('Erro('.$db->connect_errno.')'.$db->connect_error);  
}else{  
  
    $sql = "CREATE DATABASE ".$dbname;  
    if($db->query($sql)){  
        echo "Base de dados criada com sucesso";  
    }  
  
    //...  
  
    $db->close();  
}  
?>
```

Podemos verificar pelo *phpmyadmin* que a nova base de dados *books* foi criada com sucesso.



Remoção de uma base de dados

Uma base de dados pode ser removida através da *query*.

```
$sql = "DROP DATABASE bd_nome";  
$db->query($sql);
```

Exemplo de remoção de base de dados

```
<?php  
$host = "localhost";   $username = "root";   $passwd = "";  
  
$db = new mysqli($host,$username,$passwd);  
if ($db->connect_error){// Verifica a conexão  
    die('Erro('.$db->connect_errno.')' . $db->connect_error);  
}else{  
    echo "Conexão aberta";  
  
    $dbname = "books";  
  
    $sql = "DROP DATABASE ".$dbname;  
  
    if($db->query($sql)) {  
        echo "Base de dados deletada com sucesso";  
    }  
    // ...  
  
    $db->close();  
}  
?>
```

Criação de tabelas na base de dados

A instrução *CREATE TABLE* é responsável pela criação de uma tabela na base de dados. Sintaxe

```
$sql = "CREATE TABLE nome_tabela(coluna1 tipo_de_dado(tamanho),
                                   coluna2 tipo_de_dado(tamanho), ...);";
$db->query($sql);
```

O seguinte exemplo cria a tabela chamada Books com quatro colunas: "isbn", "author", "title" e "price".

```
<?php
    $host = "localhost";  $username = "root";  $passwd = "";
    $dbname = "books";

    $db = new mysqli($host,$username,$passwd,$dbname);

    if ($db->connect_error){// Verifica a conexão
        die('Erro('.$db->connect_errno.')' . $db->connect_error);
    }else{

        $sql = "CREATE TABLE books(isbn char(13) NOT NULL,
                                   author char(30), title char(60), price float(4,2),
                                   PRIMARY KEY(isbn))";

        if($db->query($sql)){
            echo "Tabela criada com sucesso.";
        }

        $db->close();
    }
?>
```

Inserção de dados em uma tabela

A instrução *INSERT INTO* é utilizado para adicionar novos registros em uma tabela na base de dados. É possível inserir dados de duas formas:

```
$sql = "INSERT INTO nome_tabela VALUES (valor1, valor2, valor3, ...)";
$db->query($sql);
```

ou

```
$sql = "INSERT INTO nome_tabela ( coluna1, coluna2, coluna3, ...)
      VALUES (valor1, valor2, valor3, ...)";
$db->query($sql);
```

O exemplo abaixo mostra a inserção de dois livros na tabela books.

```
<?php
    $host = "localhost";    $username = "root";    $passwd = "";
    $dbname = "books";

    $db = new mysqli($host,$username,$passwd,$dbname);

    if ($db->connect_error){// Verifica a conexão
        die('Erro('.$db->connect_errno.')'.$db->connect_error);
    }else{

        $sql1 = "INSERT INTO books
                VALUES('8535911693', 'Jorge Amado', 'Capitães da Areia',
                30.80)";

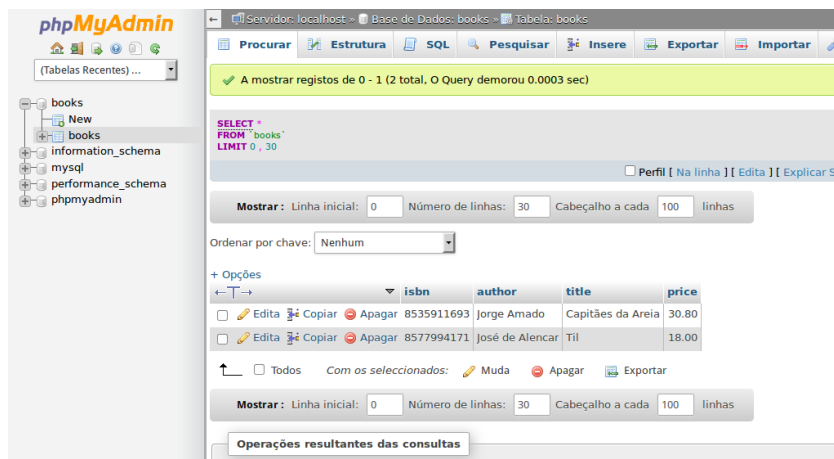
        if($db->query($sql1)){
            echo "Livro inserido com sucesso.";
        }

        $sql2 = "INSERT INTO books(isbn,author, title,price)
                VALUES('8577994171','José de Alencar', 'Til', 18.00)";

        if($db->query($sql2)){
            echo "Livro inserido com sucesso.";
        }

        $db->close();
    }
?>
```

Podemos verificar pelo *phpmyadmin* que os dois livros foram inseridos na tabela books.



Exemplo de formulário para inserir dados em uma tabela

```
<html>
<body>
  <h1>Dados do livro</h1>
  <form action="insert_book.php" method="post">
    ISBN:<input type="text" name="isbn" maxlength="13" size="13">
    Autor:<input type="text" name="author" maxlength="30" size="30">
    Título:<input type="text" name="title" maxlength="60" size="30">
    Preço R$:<input type="text" name="price" maxlength="7" size="7">
    <input type="submit" value="Enviar">
  </form>
</body>
</html>
```

Arquivo "insert_book.php"

```
<?php
echo 'Livraria';

$isbn    = $_POST['isbn'];
$author  = $_POST['author'];
$title   = $_POST['title'];
$price   = $_POST['price'];

if (!$isbn || !$author || !$title || !$price) {
    echo 'Você não enviou todos os dados necessários.';
    exit;
}
```

```

$price = doubleval($price);

$db = new mysqli("localhost","root","", 'books');

if ($db->connect_error) {
    die('Erro('.$db->connect_errno.')'.$db->connect_error);
}

$sql = "insert into books values(
    '". $isbn. "', '". $author. "', '". $title. "', '". $price. "')";

if ($db->query($sql)) {
    echo 'Livro inserido com sucesso.';
}

$db->close();
?>

```

Consultar dados

A instrução *SELECT* é realizar uma consulta na base de dados. O resultado da consulta é armazenado no objeto `$result` do tipo *mysqli_result*

```

$sql = "SELECT coluna1, coluna2, ...
      FROM nome_tabela";
$result = $db->query($sql);

```

O seguinte exemplo seleciona o título e o autor dos dados armazenados na tabela books.

```

<?php
$host = "localhost";
$username = "root";
$password = "";
$dbname = "books";

$db = new mysqli($host, $username, $password, $dbname);

if ($db->connect_error) {
    die('Erro('.$db->connect_errno.')'.$db->connect_error);
} else {

```

```

$sql = "SELECT title, author FROM books";

if ($result = $db->query($sql)) {
    while ($book = $result->fetch_object()) {
        echo '<p>Título: ' . $book->title ;
        echo ', Autor: ' . $book->author . '</p>';
    }
    $result->free();
}
$db->close();
}
?>

```

O exemplo acima é chamado o método *fetch_object()* do objeto *\$result* para retornar a linha atual do conjunto de resultados como um objeto. Os atributos desse objeto representam os nomes dos campos encontrados no conjunto de resultados. Caso não existam mais linhas no conjunto de resultados, então é retornado *NULL*. No exemplo acima, o objeto *\$book* é composto pelos atributos 'title' e 'author'. Resultado da consulta:

Resultado da busca

Título: Vidas Secas, Autor: Graciliano Ramos
Título: Capitães da Areia, Autor: Jorge Amado
Título: Til, Autor: Jose de Alencar

O Próximo exemplo seleciona todos os dados armazenados na tabela books. O caractere * é usado para selecionar todos os dados da tabela.

```

<?php
    $host = "localhost";    $username = "root";    $passwd = "";
    $dbname = "books";

    $db = new mysqli($host,$username,$passwd,$dbname);

    if ($db->connect_error) {
        die('Erro(' . $db->connect_errno . ') ' . $db->connect_error);
    }else{

        $sql = "SELECT * FROM books";

        if ($result = $db->query($sql)) {

```

```

while ($book = $result->fetch_object()) {
    echo '<p>Título: ' . $book->title ;
    echo '<br/> Autor: ' . $book->author;
    echo '<br />ISBN: ' . $book->isbn;
    echo '<br />Preço: ' . $book->price.'</p>';
}
$result->free();
}

$db->close();
}
?>

```

Saída:

Resultado da busca

Título: Vidas Secas
 Autor: Graciliano Ramos
 ISBN: 8501067342
 Preço:24.50

Título: Capitaes da Areia
 Autor: Jorge Amado
 ISBN: 8535911693
 Preço:30.80

Título: Til
 Autor: Jose de Alencar
 ISBN: 8577994171
 Preço:18.00

Cláusula where

A cláusula *where* é usada para extrair apenas os registros que satisfazem um critério especificado.

```

$sql = "SELECT coluna1, coluna2,...
        FROM nome_tabela
        WHERE coluna_i operador valor";

```

```

$result = $db->query($sql);

```

O exemplo abaixo seleciona o título e o autor da tabela books que possui preço maior que 15, 50.

```

<?php
    $host = "localhost";  $username = "root";  $passwd = "";
    $dbname = "books";

    $db = new mysqli($host,$username,$passwd,$dbname);

    if ($db->connect_error){
        die('Erro('.$db->connect_errno.')'.$db->connect_error);
    }

```



```

}else{

    $sql = "SELECT title, author

    if ($result = $db->query($sql)) {

        while ($book = $result->fetch_object()) {
            echo '<p>Título: ' . $book->title ;
            echo '<br/>Autor: ' . $book->author . '</p>';
        }
        $result->free();
    }

    $db->close();
}
?>

```

Saída:

Resultado da busca

Título: Vidas Secas
 Autor: Graciliano Ramos
 Título: Captaes da Areia
 Autor: Jorge Amado
 Título: Til
 Autor: Jose de Alencar

Exemplo de busca em uma tabela

```

<html>
<body>
<h1>Pesquisar livros no acervo</h1>
<form action="buscaLivro.php" method="post">
    <select name="searchtype">
        <option value="author">Autor</option>
        <option value="title">Título</option>
        <option value="isbn">ISBN</option>
    </select>
    <input name="searchterm" type="text">

    <input type="submit" value="Buscar">
</form>
</body>
</html>

```

Arquivo "insert_book.php"

```
<?php

echo '<h1>Resultado da busca</h1>';

$searchtype = $_POST['searchtype'];
$searchterm = $_POST['searchterm'];

if (!$searchtype || !$searchterm) {
    echo 'Retorne, e informe novamente os dados.';
    exit;
}

$db = new mysqli("localhost", "root", "", 'books');
if ($db->connect_error) {
    die('Erro(' . $db->connect_errno . ') ' . $db->connect_error);
}

$sql = "SELECT * FROM books "
        . "WHERE ".$searchtype." LIKE '%" . $searchterm . "%'";

if ($result = $db->query($sql)) {

    echo 'Número de livros encontrados: ' . $result->num_rows;

    while ($book = $result->fetch_object()) {
        echo '<p>Título: ' . $book->title ;
        echo '<br/> Autor: ' . $book->author;
        echo '<br />ISBN: ' . $book->isbn;
        echo '<br />Preço: ' . $book->price;
        echo '</p>';
    }
    $result->free();
}

$db->close();

?>
```

Ordenar uma consulta

A palavra-chave *ORDER BY* é usada para classificar os dados de um conjunto de registros. Os registros são classificados em ordem crescente (*ASC by default*) ou decrescente (*DESC*).

```
$sql = "SELECT coluna1, coluna2, ...  
      FROM nome_tabela  
      ORDER BY colunas ASC|DESC";  
$result = $db->query($sql);
```

O exemplo a seguir seleciona o autor e o título da tabela books, e classifica o resultado pelo título em ordem decrescente

```
<?php  
$host = "localhost"; $username = "root"; $passwd = "";  
$dbname = "books";  
  
$db = new mysqli($host,$username,$passwd,$dbname);  
  
if ($db->connect_error) {  
    die('Erro('.$db->connect_errno.')'.$db->connect_error);  
}else{  
    $sql = "SELECT author, title  
          FROM books  
          ORDER BY title DESC";  
  
    if ($result = $db->query($sql)) {  
        while($book = $result->fetch_object()) {  
            print "<p>Título: ". $book->title;  
            print ", Autor: ".$book->author."</p>";  
        }  
        $result->free();  
    }  
  
    $db->close();  
}  
?>
```

Saída

Resultado da busca

Título: Vidas Secas, Autor: Graciliano Ramos

Título: Til, Autor: Jose de Alencar

Título: Sentimento do Mundo, Autor: Carlos Drummond de Andrade

Título: Memórias Postumas de Bras Cubas, Autor: Machado de Assis

Atualização dos dados de uma tabela

A instrução *UPDATE* é usada para atualizar os registros existentes em uma tabela.

```
$sql = "UPDATE nome_tabela
      SET coluna1=valor, coluna2=valor, ...
      WHERE colunas = valor";
$db->query($sql);
```

O exemplo a seguir altera todos os registros da tabela books com *price* = 18.00 para *price* = 15.99.

```
<?php
    $host = "localhost";  $username = "root";  $passwd = "";
    $dbname = "books";

    $db = new mysqli($host,$username,$passwd,$dbname);

    if ($db->connect_error) {
        die('Erro('.$db->connect_errno.')'.$db->connect_error);
    }else{

        $sql = "UPDATE books
              SET price = 15.99
              WHERE price = 18.00";

        if($db->query($sql)) {
            echo "Livro alterado com sucesso.";
        }

        $db->close();
    }
?>
```

Remoção de dados de uma tabela

A instrução *DELETE* é usada para remover registros de uma tabela de banco de dados.

```
$sql = "DELETE FROM nome_tabela
      WHERE coluna = valor";
$db->query($sql);
```

O seguinte exemplo remove os registros da tabela books com o título = 'Til'

```
<?php
    $host = "localhost";  $username = "root";  $passwd = "";
    $dbname = "books";

    $db = new mysqli($host,$username,$passwd,$dbname);

    if ($db->connect_error) {
        die('Erro('.$db->connect_errno.')' . $db->connect_error);
    }else{

        $sql = "DELETE FROM books
                WHERE title = 'Til'";

        if($db->query($sql)){
            echo "Livro removido com sucesso.";
        }
        $db->close();
    }
?>
```

Referências Bibliográficas

Mehdi Achour et al. Manual do php, 2014. URL http://us2.php.net/manual/pt_BR/. [Acessado em 14/02/2014].

Tim Converse, Joyce Park, and Clark Morgan. *PHP5 and MySQL Bible*. Bible. Wiley, 2004. ISBN 9780764571824.

Pablo Dall'Oglio. *PHP – Programando com Orientação a Objetos*. Novatec Editora Ltda, São Paulo, Brasil, 2007.

Juliano Niederauer. *Desenvolvendo Websites com PHP*. Novatec, 2004.

W3Schools. PHP 5 Tutorial, 2014a. URL <http://www.w3schools.com/php/default.asp>. [Acessado em 14/02/2014].

W3Schools. SQL Tutorial, 2014b. URL <http://www.w3schools.com/sql/default.asp>. [Acessado em 14/02/2014].

Luke Welling and Laura Thomson. *PHP e MySQL: Desenvolvimento Web*, volume 3. Elsevier-Campus, 2005.