1. Introdução

O que é PHP?

PHP é uma linguagem que permite criar sites WEB dinâmicos, possibilitando uma interação com o usuário através de formulários, parâmetros da URL e links. A diferença de PHP com relação a linguagens semelhantes a Javascript é que o código PHP é executado no servidor, sendo enviado para o cliente apenas html puro. Desta maneira é possível interagir com bancos de dados e aplicações existentes no servidor, com a vantagem de não expor o código fonte para o cliente. Isso pode ser útil quando o programa está lidando com senhas ou qualquer tipo de informação confidencial.

O que diferencia PHP de um script CGI escrito em C ou Perl é que o código PHP fica embutido no próprio HTML, enquanto no outro caso é necessário que o script CGI gere todo o código HTML, ou leia de um outro arquivo.

O que pode ser feito com PHP?

Basicamente, qualquer coisa que pode ser feita por algum programa CGI pode ser feita também com PHP, como coletar dados de um formulário, gerar páginas dinamicamente ou enviar e receber *cookies*.

PHP também tem como uma das características mais importantes o suporte a um grande número de bancos de dados, como dBase, Interbase, mSQL, mySQL, Oracle, Sybase, PostgreSQL e vários outros. Construir uma página baseada em um banco de dados torna-se uma tarefa extremamente simples com PHP.

Além disso, PHP tem suporte a outros serviços através de protocolos como IMAP, SNMP, NNTP, POP3 e, logicamente, HTTP. Ainda é possível abrir *sockets* e interagir com outros protocolos.

Como surgiu a linguagem PHP?

A linguagem PHP foi concebida durante o outono de 1994 por **Rasmus Lerdorf**. As primeiras versões não foram disponibilizadas, tendo sido utilizadas em sua *home-page* apenas para que ele pudesse ter informações sobre as visitas que estavam sendo feitas. A primeira versão utilizada por outras pessoas foi disponibilizada em 1995, e ficou conhecida como "**P**ersonal **H**ome **P**age Tools" (ferramentas para página pessoal). Era composta por um sistema bastante simples que interpretava algumas *macros* e alguns utilitários que rodavam "por trás" das *home-pages*: um livro de visitas, um contador e algumas outras coisas.

Em meados de 1995 o interpretador foi reescrito, e ganhou o nome de **PHP/FI**, o "FI" veio de um outro pacote escrito por Rasmus que interpretava dados de formulários HTML (**F**orm **I**nterpreter). Ele combinou os scripts do pacote *Personal Home Page Tools* com o FI e adicionou suporte a mSQL, nascendo assim o PHP/FI, que cresceu bastante, e as pessoas passaram a contribuir com o projeto.

Estima-se que em 1996 PHP/FI estava sendo usado por cerca de 15.000 *sites* pelo mundo, e em meados de 1997 esse número subiu para mais de 50.000. Nessa época houve uma mudança no desenvolvimento do PHP. Ele deixou de ser um projeto de Rasmus com contribuições de outras pessoas para ter uma equipe de desenvolvimento mais organizada. O interpretador foi reescrito por **Zeev Suraski** e **Andi Gutmans**, e esse novo interpretador foi a base para a versão 3.

Quanto à licença, PHP é um software gratuito e de código aberto publicado sob a PHP License, que afirma:

> Produtos derivados deste software não devem ser chamado de PHP, nem pode conter "PHP" em seu nome, sem prévia permissão por escrito da group@php.net. Você pode indicar que o software funciona em conjunto com o PHP, dizendo "Foo para PHP", em vez de chamá-lo "PHP Foo" ou "phpfoo".
>
> Esta restrição no uso do nome *PHP* o torna incompatível com a GNU General Public License (GPL).

Atualmente o PHP encontra-se oficialmente na versão 5, cuja mais atual é a versão 5.6.9. Desde junho/2015 foi lançada a versão PHP 7.0.0. Como assim? Os desenvolvedores que mantém a linguagem, decidiram pular do PHP 6 para o 7, pois muitas alterações presentes na versão 5.6 já representam o PHP 6, tamanha a diferença dela para a versão 5.0 por exemplo, e desta forma atualmente esta disponível apenas a versão 7 Preview da linguagem.

Para nosso curso, utilizaremos a versão compatível com o PHP 7, 5.5, 5.4 e 5.3 embutida no EasyPHP 16.1 (disponível em www.easyphp.org) , que também instala para sua utilização o Apache 2.4.17, MySQL 5.7.9, PhpMyAdmin, dentre outros.

Após baixar o arquivo, basta executá-lo e seguir as simples instruções na tela.

2. Sintaxe Básica

Delimitando o código PHP

O código PHP fica embutido no próprio HTML. O interpretador identifica quando um código é PHP pelas seguintes tags:

```
<?php
comandos
?>

<script language="php">
comandos
</script>

<?
comandos
?>
```

O tipo de *tags* mais utilizado é o terceiro, que consiste em uma "abreviação" do primeiro. Para utilizá-lo, é necessário habilitar a opção *short-tags* na configuração do PHP.

Separador de instruções

Entre cada instrução em PHP é preciso utilizar o ponto-e-vírgula, assim como em C, Perl e outras linguagens mais conhecidas. Na última instrução do bloco de script não é necessário o uso do ponto-e-vírgula, mas por questões estéticas recomenda-se o uso sempre.

Nomes de variáveis

Toda variável em PHP tem seu nome composto pelo caracter $ e uma string, que deve iniciar por uma letra ou o caracter "_". **PHP é case sensitive**, ou seja, as variáveis $vivas e $VIVAS são diferentes. Por isso é preciso ter muito cuidado ao definir os nomes das variáveis. É bom evitar os nomes em maiúsculas, pois como veremos mais adiante, o PHP já possui alguma variáveis pré-definidas cujos nomes são formados por letras maiúsculas.

Comentários

Há dois tipos de comentários em código PHP:

Comentários de uma linha:

Marca como comentário até o final da linha ou até o final do bloco de código PHP – o que vier antes. Pode ser delimitado pelo caracter "#" ou por duas barras (//).

```
Exemplo:
<?  echo "teste"; #isto é um teste  ?>
<?  echo "teste"; //este teste é similar ao anterior ?>
```

Comentários de mais de uma linha:

Tem como delimitadores os caracteres "/*" para o início do bloco e "*/" para o final do comentário. Se o delimitador de final de código PHP (?>) estiver dentro de um comentário, não será reconhecido pelo interpretador.

Exemplos:

```
<?
  echo “teste”; /* Isto é um comentário com mais
de uma linha, mas não funciona corretamente ?>
*/
```

```
<?
  echo “teste”; /* Isto é um comentário com mais
de uma linha que funciona corretamente
*/
?>
```

3. Criando o primeiro script

Primeiro Exemplo

Neste exemplo, criaremos um script com uma saída simples, que servirá para testar se a instalação foi feita corretamente:

```
<html>
<head><title>Aprendendo PHP</title></head>
<body>

<?php
echo "Primeiro Script";
?>

</body>
</html>
```

Salve o arquivo como “primeiro.php” no diretorio de documentos do Apache. Abra uma janela do navegador e digite o endereço “`http://localhost/primeiro.php`”. Verificando o código fonte da página exibida, temos o seguinte:

```
<html>
<head><title>Aprendendo PHP</title></head>
<body>

Primeiro Script

</body>
</html>
```

Isso mostra como o PHP funciona. O script é executado no servidor, ficando disponível para o usuário apenas o resultado. Agora vamos escrever um script que produza exatamente o mesmo resultado utilizando uma variável:

```
<html>
<head><title>Aprendendo PHP</title></head>
<body>

<?php
$texto = "Primeiro Script";
echo $texto;
?>

</body>
</html>
```

4. Tipos

Tipos Suportados

PHP suporta os seguintes tipos de dados:

- Inteiro
- Ponto flutuante
- String
- Array
- Objeto

PHP utiliza checagem de tipos dinâmica, ou seja, uma variável pode conter valores de diferentes tipos em diferentes momentos da execução do script. Por este motivo não é necessário declarar o tipo de uma variável para usá-la. O interpretador PHP decidirá qual o tipo daquela variável, verificando o conteúdo em tempo de execução.

Ainda assim, é permitido converter os valores de um tipo para outro desejado, utilizando o *typecasting* ou a função `settype` (ver adiante).

Inteiros (integer ou long)

Uma variável pode conter um valor inteiro com atribuições que sigam as seguintes sintaxes:

```
$vivas = 1234; # inteiro positivo na base decimal
$vivas = -234; # inteiro negativo na base decimal
$vivas = 0234; # inteiro na base octal-simbolizado pelo 0
               # equivale a 156 decimal
$vivas = 0x34; # inteiro na base hexadecimal(simbolizado
               # pelo 0x) – equivale a 52 decimal.
```

A diferença entre inteiros simples e `long` está no número de bytes utilizados para armazenar a variável. Como a escolha é feita pelo interpretador PHP de maneira transparente para o usuário, podemos afirmar que os tipos são iguais.

Números em Ponto Flutuante (double ou float)

Uma variável pode ter um valor em ponto flutuante com atribuições que sigam as seguintes sintaxes:

```
$vivas = 1.234;
$vivas = 23e4; # equivale a 230.000
```

Strings

Strings podem ser atribuídas de duas maneiras:

a) utilizando aspas simples (') – Desta maneira, o valor da variável será exatamente o texto contido entre as aspas (com exceção de \\ e \' – ver tabela abaixo)
b) utilizando aspas duplas (") – Desta maneira, qualquer variável ou caracter de escape será expandido antes de ser atribuído.

Exemplo:

```
<?
$teste = "Mauricio";
$vivas = '---$teste--\n';
echo "$vivas";
?>
```

A saída desse script será "---$teste--\n".

```
<?
$teste = "Mauricio";
$vivas = "---$teste---\n";
echo "$vivas";
?>
```

A saída desse script será "---Mauricio--" (com uma quebra de linha no final).

A tabela seguinte lista os caracteres de escape:

Sintaxe	Significado
\n	Nova linha
\r	Retorno de carro (semelhante a \n)
\t	Tabulação horizontal
\\	A própria barra (\ )
\$	O símbolo $
\’	Aspa simples
\”	Aspa dupla

Funções para tratamento de strings

htmlspecialchars

```
string htmlspecialchars(string str);
```

Retorna a string fornecida, substituindo os seguintes caracteres:

- & para '&'
- " para '"'
- < para '<'
- > para ‘>'

htmlentities

```
string htmlentities(string str);
```

Funciona de maneira semelhante ao comando anterior, mas de maneira mais completa, pois converte todos os caracteres da string que possuem uma representação especial em html, como por exemplo:

- º para 'º'
- ª para 'ª'
- á para 'á'
- ç para ‘ç'

nl2br

```
string nl2br(string str);
```

Retorna a string fornecida substituindo todas as quebras de linha (“\n”) por quebras de linhas em html (“
”).

```
Exemplo:
echo nl2br(“Mauricio\nVivas\n”);

Imprime:
Maurício<br>Vivas<br>
```

get_meta_tags

```
array get_meta_tags(string arquivo);
```

Abre um arquivo html e percorre o cabeçalho em busca de “meta” tags, retornando num array todos os valores encontrados.

```
Exemplo:
No arquivo teste.html temos:
...
<head>
<meta name="author" content="jose">
<meta name="tags" content="php3 documentation">
...
</head><!-- busca encerra aqui -->
...

a execução da função:

get_meta_tags(“teste.html”);

retorna o array:

array(“author”=>”jose”,”tags”=>"php3 documentation");
```

strip_tags

```
string strip_tags(string str);
```

Retorna a string fornecida, retirando todas as tags html e/ou PHP encontradas.

```
Exemplo:
strip_tags('<a href="teste1.php3">testando</a><br>');

Retorna a string “testando”
```

urlencode

```
string urlencode(string str);
```

Retorna a string fornecida, convertida para o formato urlencode. Esta função é útil para passar variáveis para uma próxima página.

urldecode

```
string urldecode(string str);
```

Funciona de maneira inversa a urlencode, desta vez decodificando a string fornecida do formato urlencode para texto normal.

Funções relacionadas a arrays

Implode e join

```
string implode(string separador, array partes);
string join(string separador, array partes);
```

As duas funções são idênticas. Retornam uma string contendo todos os elementos do array fornecido separados pela string também fornecida.

```
Exemplo:
$partes = array("a", "casa número", 13, "é azul");
$inteiro = join(" ",$partes);

$inteiro passa a conter a string:
“a casa número 13 é azul”
```

split

```
array split(string padrao, string str, int [limite]);
```

Retorna um array contendo partes da string fornecida separadas pelo padrão fornecido, podendo limitar o número de elementos do array.

```
Exemplo:
$data = “11/14/1975”;
```

```
$data_array = split("/",$data);
```

O código acima faz com que a variável $data_array receba o valor:

```
array(11,14,1975);
```

explode

```
array explode(string padrao, string str);
```

Funciona de maneira bastante semelhante à função `split`, com a diferença que não é possível estabelecer um limite para o número de elementos do array.

Comparações entre strings

similar_tex

```
int similar_text(string str1, string str2, double [porcentagem]);
```

Compara as duas strings fornecidas e retorna o número de caracteres coincidentes. Opcionalmente pode ser fornecida uma variável, passada por referência *(ver tópico sobre funções)*, que receberá o valor percentual de igualdade entre as strings. Esta função é *case sensitive*, ou seja, maiúsculas e minúsculas são tratadas como diferentes.

Exemplo:

```
$num = similar_text("teste", "testando",&$porc);
```

As variáveis passam a ter os seguintes valores:

```
$num == 4; $porc == 61.538461538462
```

strcasecmp

```
int strcasecmp(string str1, string str2);
```

Compara as duas strings e retorna 0 (zero) se forem iguais, um valor maior que zero se `str1 > str2`, e um valor menor que zero se `str1 < str2`. Esta função é *case insensitive*, ou seja, maiúsculas e minúsculas são tratadas como iguais.

strcmp

```
int strcmp(string str1, string str2);
```

Funciona de maneira semelhante à função `strcasecmp`, com a diferença que esta é *case sensitive*, ou seja, maiúsculas e minúsculas são tratadas como diferentes.

strstr

```
string strstr(string str1, string str2);
string strchr(string str1, string str2);
```

As duas funções são idênticas. Procura a primeira ocorrência de `str2` em `str1`. Se não encontrar, retorna uma string vazia, e se encontrar retorna todos os caracteres de `str1` a partir desse ponto.

```
Exemplo:

strstr("Mauricio Vivas", "Viv"); // retorna “Vivas”
```

stristr

```
string strstr(string str1, string str2);
```

Funciona de maneira semelhante à função `strstr`, com a diferença que esta é *case insensitive*, ou seja, maiúsculas e minúsculas são tratadas como iguais.

strpos

```
int strpos(string str1, string str2, int [offset] );
```

Retorna a posição da primeira ocorrência de `str2` em `str1`, ou zero se não houver. O parâmetro opcional `offset` determina a partir de qual caracter de str1 será efetuada a busca. Mesmo utilizando o offset, o valor de retorno é referente ao início de str1.

strrpos

```
int strrpos(string haystack, char needle);
```

Retorna a posição da última ocorrência de `str2` em `str1`, ou zero se não houver.

Funções para edição de strings

chop

```
string chop(string str);
```

Retira espaços e linhas em branco do final da string fornecida.

```
Exemplo:
chop("   Teste      \n  \n  "); // retorna "   Teste"
```

ltrim

```
string ltrim(string str);
```

Retira espaços e linhas em branco do final da string fornecida.

```
Exemplo:
ltrim("   Teste  \n  \n "); // retorna "Teste  \n  \n"
```

trim

```
string trim(string str);
```

Retira espaços e linhas em branco do início e do final da string fornecida.

```
Exemplo:
trim("   Teste  \n  \n "); // retorna "Teste"
```

strrev

```
string strrev(string str);
```

Retorna a string fornecida invertida.

```
Exemplo:

strrev("Teste"); // retorna "etseT"
```

strtolower

```
string strtolower(string str);
```

Retorna a string fornecida com todas as letras minúsculas.

```
Exemplo:

strtolower(“Teste”); // retorna “teste”
```

strtoupper

```
string strtoupper(string str);
```

Retorna a string fornecida com todas as letras maiúsculas.

```
Exemplo:

strtolower(“Teste”); // retorna “TESTE”
```

ucfirst

```
string ucfirst(string str);
```

Retorna a string fornecida com o primeiro caracter convertido para letra maiúscula.

```
Exemplo:

ucfirst(“teste de funcao”); // retorna “Teste de funcao”
```

ucwords

```
string ucwords(string str);
```

Retorna a string fornecida com todas as palavras iniciadas por letras maiúsculas.

```
Exemplo:

ucwords(“teste de funcao”); // retorna “Teste De Funcao”
```

str_replace

```
string str_replace(string str1, string str2, string str3);
```

Altera todas as ocorrências de `str1` em `str3` pela string `str2`.

Funções diversas

chr

```
string chr(int ascii);
```

Retorna o caracter correspondente ao código ASCII fornecido.

ord

```
int ord(string string);
```

Retorna o código ASCII correspontente ao caracter fornecido.

echo

```
echo(string arg1, string [argn]... );
```

Imprime os argumentos fornecidos.

print

```
print(string arg);
```

Imprime o argumento fornecido.

strlen

```
int strlen(string str);
```

Retorna o tamanho da string fornecida.

Arrays

Arrays em PHP podem ser observados como mapeamentos ou como vetores indexados. Mais precisamente, um valor do tipo array é um dicionário onde os índices são as chaves de acesso. Vale ressaltar que os índices podem ser valores de qualquer tipo e não somente inteiros. Inclusive, se os índices forem todos inteiros, estes não precisam formar um intervalo contínuo

Como a checagem de tipos em PHP é dinâmica, valores de tipos diferentes podem ser usados como índices de array, assim como os valores mapeados também podem ser de diversos tipos.

```
Exemplo:
<?
$cor[1] = “vermelho”;
$cor[2] = “verde”;
$cor[3] = “azul”;
$cor[“teste”] = 1;
?>

Equivalentemente, pode-se escrever:

<?
$cor = array(1 => “vermelho, 2 => “verde, 3 => “azul”, “teste => 1);
?>
```

Listas

As listas são utilizadas em PHP para realizar atribuições múltiplas. Através de listas é possível atribuir valores que estão num array para variáveis. Vejamos o exemplo:

```
Exemplo:

 list($a, $b, $c) = array(“a”, “b”, “c”);
```

O comando acima atribui valores às três variáveis simultaneamente. É bom notar que só são atribuídos às variáveis da lista os elementos do array que possuem índices inteiros e não negativos. No exemplo acima as três atribuições foram bem sucedidas porque ao inicializar um array sem especificar os índices eles passam a ser inteiros, a partir do zero. Um fator importante é que cada variável da lista possui um

índice inteiro e ordinal, iniciando com zero, que serve para determinar qual valor será atribuído. No exemplo anterior temos $a com índice 0, $b com índice 1 e $c com índice 2. Vejamos um outro exemplo:

```
$arr = array(1=>”um”,3=>”tres”,”a”=>”letraA”,2=>”dois);
list($a,$b,$c,$d) = $arr;
```

Após a execução do código acima temos os seguintes valores:

```
$a == null
$b == “um”
$c == “dois”
$d == “tres”
```

Devemos observar que à variável $a não foi atribuído valor, pois no array não existe elemento com índice 0 (zero). Outro detalhe importante é que o valor “tres” foi atribuído à variável $d, e não a $b, pois seu índice é 3, o mesmo que $d na lista. Por fim, vemos que o valor “letraA” não foi atribuído a elemento algum da lista pois seu índice não é inteiro.

Os índices da lista servem apenas como referência ao interpretador PHP para realizar as atribuições, não podendo ser acessados de maneira alguma pelo programador. De maneira diferente do array, uma lista não pode ser atribuída a uma variável, servindo apenas para fazer múltiplas atribuições através de um array.

Funções para Array

Array

```
array array(...);
```

É a função que cria um array a partir dos parâmetros forncidos. É possível fornecer o índice de cada elemento. Esse índice pode ser um valor de qualquer tipo, e não apenas de inteiro. Se o índice não for fornecido o PHP atribui um valor inteiro sequencial, a partir do 0 ou do último índice inteiro explicitado. Vejamos alguns exemplos:

```
Exemplo 1
$teste = array("um", "dois","tr"=>"tres",5=>"quatro","cinco");

Temos o seguinte mapeamento:
0 => “um” (0 é o primeiro índice, se não houver um explicito)
1 => “dois” (o inteiro seguinte)
“tr” => “tres”
```

```
5 => “quatro” (valor explicitado)
6 => “cinco” (o inteiro seguinte ao último atribuído, e não o
próximo valor, que seria 2)

Exemplo 2
$teste                          =                          array("um",
6=>"dois","tr"=>"tres",5=>"quatro","cinco");

Temos o seguinte mapeamento:
0 => “um”
6 => “dois”
“tr” => tres
5 => “quatro” (seria 7, se não fosse explicitado)
7 => “cinco” (seria 6, se não estivesse ocupado)
```

Em geral, não é recomendável utilizar arrays com vários tipos de índices, já que isso pode confundir o programador. No caso de realmente haver a necessidade de utilizar esse recurso, deve-se ter bastante atenção ao manipular os índices do array.

range

```
array range(int minimo, int maximo);
```

A função range cria um array cujos elementos são os inteiros pertencentes ao intervalo fornecido, inclusive. Se o valor do primeiro parâmetro for maior do que o do segundo, a função retorna `false` (valor vazio).

shuffle

```
void shuffle(array &arr);
```

Esta função “embaralha” o array, ou seja, troca as posições dos elementos aleatoriamente e não retorna valor algum.

sizeof

```
int sizeof(array arr);
```

Retorna um valor inteiro contendo o número de elementos de um array. Se for utilizada com uma variável cujo valor não é do tipo array, retorna 1. Se a variável não estiver setada ou for um array vazio, retorna 0.

Funções de "navegação"

Toda variável do tipo array possui um ponteiro interno indicando o próximo elemento a ser acessado no caso de não ser especificado um índice. As funções seguintes servem para modificar esse ponteiro, permitindo assim percorrer um array para verificar seu conteúdo (chaves e elementos).

reset

```
mixed reset(array arr);
```

Seta o ponteiro interno para o primeiro elemento do array, e retorna o conteúdo desse elemento.

end

```
mixed end(array arr);
```

Seta o ponteiro interno para o último elemento do array, e retorna o conteúdo desse elemento.

next

```
mixed next(array arr);
```

Seta o ponteiro interno para o próximo elemento do array, e retorna o conteúdo desse elemento.

Obs.: esta não é uma boa função para determinar se um elemento é o último do array, pois pode retornar `false` tanto no final do array como no caso de haver um elemento vazio.

prev

```
mixed prev(array arr);
```

Seta o ponteiro interno para o elemento anterior do array, e retorna o conteúdo desse elemento. Funciona de maneira inversa a `next`.

pos

```
mixed pos(array arr);
```

Retorna o conteúdo do elemento atual do array, indicado pelo ponteiro interno.

key

```
mixed key(array arr);
```

Funciona de maneira bastante semelhante a `pos`, mas ao invés de retornar o elemento atual indicado pelo ponteiro interno do array, retorna seu índice.

each

```
array each(array arr);
```

Retorna um array contendo o índice e o elemento atual indicao pelo ponteiro interno do array. o valor de retorno é um array de quatro elementos, cujos índices são 0, 1, “key” e “value”. Os elementos de índices 0 e “key” armazenam o índice do valor atual, e os elementos de índices 1 e “value” contém o valor do elemento atual indicado pelo ponteiro.

Esta função pode ser utilizada para percorrer todos os elementos de um array e determinar se já foi encontrado o último elemento, pois no caso de haver um elemento vazio, a função não retornará o valor `false`. A função `each` só retorna `false` depois q o último elemento do array foi encontrado.

```
Exemplo:

/*função que percorre todos os elementos de um array e
imprime seus índices e valores */
function imprime_array($arr) {
  reset($arr);
  while (list($chave,$valor) = each($arr))
    echo “Chave: $chave. Valor: $valor”;
}
```

Funções de ordenação

São funções que servem para arrumar os elementos de um array de acordo com determinados critérios. Estes critérios são: manutenção ou não da associação entre índices e elementos; ordenação por elementos ou por índices; função de comparação entre dois elementos.

sort

```
void sort(array &arr);
```

A função mais simples de ordenação de arrays. Ordena os elementos de um array em ordem crescente, sem manter os relacionamentos com os índices.

rsort

```
void rsort(array &arr);
```

Funciona de maneir ainversa à função `sort`. Ordena os elementos de um array em ordem decrescente, sem manter os relacionamentos com os índices.

asort

```
void asort(array &arr);
```

Tem o funcionamento bastante semelhante à função `sort`. Ordena os elementos de um array em ordem crescente, porém mantém os relacionamentos com os índices.

arsort

```
void arsort(array &arr);
```

Funciona de maneira inversa à função `asort`. Ordena os elementos de um array em ordem decrescente e mantém os relacionamentos dos elementos com os índices.

ksort

```
void ksort(array &arr);
```

Função de ordenação baseada nos índices. Ordena os elementos de um array de acordo com seus índices, em ordem crescente, mantendo os relacionamentos.

usort

```
void usort(array &arr, function compara);
```

Esta é uma função que utiliza outra função como parâmetro. Ordena os elementos de um array sem manter os relacionamentos com os índices, e utiliza para efeito de comparação uma função definida pelo usuário, que deve comparar dois elementos do array e retornar 0, 1 ou –1, de acordo com qualquer critério estabelecido pelo usuário.

uasort

```
void uasort(array &arr, function compara);
```

Esta função também utiliza outra função como parâmetro. Ordena os elementos de um array e mantém os relacionamentos com os índices, utilizando para efeito de comparação uma função definida pelo usuário, que deve comparar dois elementos do array e retornar 0, 1 ou –1, de acordo com qualquer critério estabelecido pelo usuário.

uksort

```
void uksort(array &arr, function compara);
```

Esta função ordena o array através dos índices, mantendo os relacionamentos com os elementos., e utiliza para efeito de comparação uma função definida pelo usuário, que deve comparar dois índices do array e retornar 0, 1 ou –1, de acordo com qualquer critério estabelecido pelo usuário.

Objetos

Um objeto pode ser inicializado utilizando o comando *new* para instanciar uma classe para uma variável.

```
Exemplo:
class teste {
          function nada() {
                echo "nada";
          }
}

$vivas = new teste;
$vivas -> nada();
```

A utilização de objetos será mais detalhada mais à frente.

Booleanos

PHP não possui um tipo booleano, mas é capaz de avaliar expressões e retornar *true* ou *false*, através do tipo `integer`: é usado o valor 0 (zero) para representar o estado *false*, e qualquer valor diferente de zero (geralmente 1) para representar o estado *true*.

Transformação de tipos

A transformação de tipos em PHP pode ser feita das seguintes maneiras:

Coerções

Quando ocorrem determinadas operações ("+", por exemplo) entre dois valores de tipos diferentes, o PHP converte o valor de um deles automaticamente (coerção). É interessante notar que se o operando for uma variável, seu valor não será alterado.

O tipo para o qual os valores dos operandos serão convertidos é determinado da seguinte forma: Se um dos operandos for `float`, o outro será convertido para `float`, senão, se um deles for `integer`, o outro será convertido para `integer`.

```
Exemplo:
$vivas = "1";        // $vivas é a string "1"
$vivas = $vivas + 1; // $vivas é o integer 2
$vivas = $vivas + 3.7;// $vivas é o double 5.7
$vivas = 1 + 1.5          // $vivas é o double 2.5
```

Como podemos notar, o PHP converte `string` para `integer` ou `double` mantendo o valor. O sistema utilizado pelo PHP para converter de *strings* para números é o seguinte:

- É analisado o início da `string`. Se contiver um número, ele será avaliado. Senão, o valor será 0 (zero);
- O número pode conter um sinal no início ("+" ou "-");

- Se a `string` contiver um ponto em sua parte numérica a ser analisada, ele será considerado, e o valor obtido será `double`;
- Se a `string` contiver um "e" ou "E" em sua parte numérica a ser analisada, o valor seguinte será considerado como expoente da base 10, e o valor obtido será `double`;

```
Exemplos:
$vivas = 1 + "10.5";       // $vivas == 11.5
$vivas = 1 + "-1.3e3";     // $vivas == -1299
$vivas = 1 + "teste10.5"; // $vivas == 1
$vivas = 1 + "10testes";  // $vivas == 11
$vivas = 1 + "  10testes";      // $vivas == 11
$vivas = 1 + "+  10testes";     // $vivas == 1
```

Transformação explícita de tipos

A sintaxe do *typecast* de PHP é semelhante ao C: basta escrever o tipo entre parenteses antes do valor

```
Exemplo:
$vivas = 15;              // $vivas é integer (15)
$vivas = (double) $vivas  // $vivas é double (15.0)
$vivas = 3.9              // $vivas é double (3.9)
$vivas = (int) $vivas     // $vivas é integer (3)
                          // o valor decimal é truncado
```

Os tipos de *cast* permitidos são:

(int), (integer) ⇒ muda para integer;

(real), (double), (float) ⇒ muda para float;

(string) ⇒ muda para string;

(array) ⇒ muda para array;

(object) ⇒ muda para objeto.

Com a função settype

A função `settype` converte uma variável para o tipo especificado, que pode ser "integer", "double", "string", "array" ou "object".

```
Exemplo:
$vivas = 15;              // $vivas é integer
settype($vivas,double)    // $vivas é Double
```