



PHP

Hyper-Text Pre-Processor

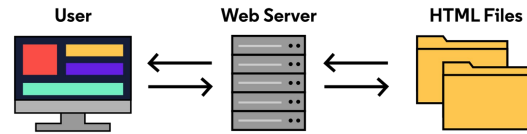
Mohammed Safadi
PHP Expert and Consultant



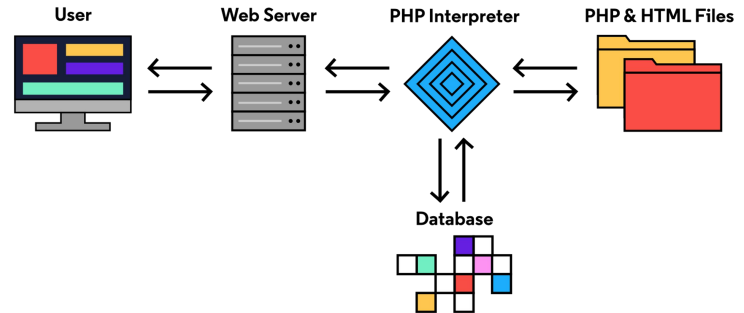
What is PHP

- PHP is an acronym for "Hypertext Preprocessor"
- Widely-used, open source scripting language
- PHP scripts are executed on the server
- Free to download and use
- A server scripting language used for making dynamic web pages.
- A commonly used technology in web development.
- Designed to interact with HTML to generate dynamic websites.

Static

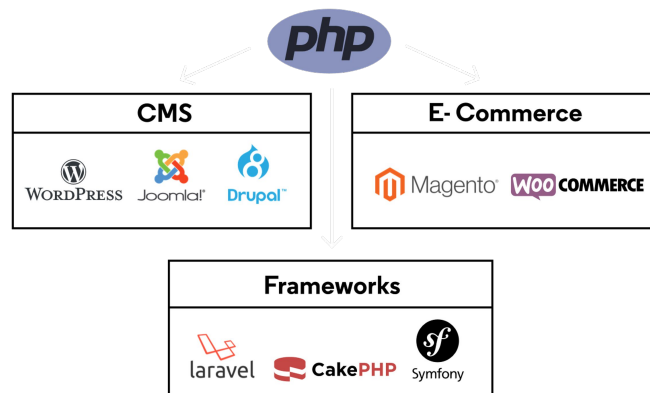


Dynamic



PHP

- PHP is the foundation of:
 - Many CMS (WordPress, Drupal, Joomla)
 - E-Commerce Platforms (WooCommerce, Magento)
 - Web Development Frameworks (Laravel, CakePHP, Symfony)



Install PHP (AMP)



- APACHE Server – Web Server
- MySQL Server – Database Management System
- PHP

- XAMPP (Apache + MariaDB + PHP + Perl)
 - XAMPP is the most popular PHP development environment
 - XAMPP is a completely free, easy to install Apache distribution containing MariaDB, PHP, and Perl.
 - <https://www.apachefriends.org/>

httpd.conf: Apache Configuration



- Listen 80 - Listen: Allows you to bind Apache to specific IP addresses and/or ports
- DocumentRoot "C:/xampp/htdocs" - The directory out of which you will serve your documents

```
<Directory "C:/xampp/htdocs">
    # Possible values for the Options directive are "None", "All",
    # or any combination of:
    #   Indexes Includes FollowSymLinks SymLinksifOwnerMatch ExecCGI MultiViews
    Options Indexes FollowSymLinks Includes ExecCGI

    # AllowOverride controls what directives may be placed in .htaccess files.
    # It can be "All", "None", or any combination of the keywords:
    #   AllowOverride FileInfo AuthConfig Limit
    AllowOverride All

    # Controls who can get stuff from this server.
    Require all granted
</Directory>
```

php.ini: PHP Configuration



- `short_open_tag=Off` - determines whether or not PHP will recognize code between `<?` and `?>` tags
- `max_execution_time=120` - Maximum execution time of each script, in seconds
- `memory_limit=512M` - Maximum amount of memory a script may consume
- `error_reporting=E_ALL` - informs PHP of which errors, warnings and notices you would like it to take action for.
- `display_errors=On` - controls whether or not and where PHP will output errors, notices and warnings too
- `post_max_size=40M` - Maximum size of POST data that PHP will accept.
- `upload_max_filesize=40M` - Maximum allowed size for uploaded files.
- `max_file_uploads=20` - Maximum number of files that can be uploaded via a single request

PHP Syntax



- Embedding PHP in HTML is done by placing PHP code between `<?php` and `?>` tags.
- Every statement in PHP must be terminated with a **semicolon ;**.
- PHP files have a **.php** extension
- File always starts with the opening PHP tag `<?php`. The closing tag is implied and left out by convention.
- Whitespace is generally ignored when executing PHP code.
- Keywords are **not case sensitive** in PHP. As a convention, use the standard casing.

echo & print



- **echo** and **print** are more or less the same. They are both used to output data to the screen.
- echo has no return value while print has a return value of 1 so it can be used in expressions.
- echo can take multiple parameters (although such usage is rare) while print can take one argument.
- echo is marginally faster than print.
- The echo statement can be used with or without parentheses: echo or echo().

```
<?php
print "<h2>PHP is Fun!</h2>";
print("Hello world!<br>");
echo "I'm about to learn PHP!<br>";
echo "This ", "string ", "was ", "made ", "with multiple
parameters.";
?>
```


PHP Tags



1. `<?php echo 'if you want to serve PHP code in XHTML or XML documents, use these tags'; ?>`
2. You can use the short echo tag to `<?= 'print this string' ?>`. It's equivalent to `<?php echo 'print this string' ?>`.
3. `<? echo 'this code is within short tags, but will only work ' . 'if short_open_tag is enabled'; ?>`

If a file contains only PHP code, it is preferable to omit the PHP closing tag at the end of the file.

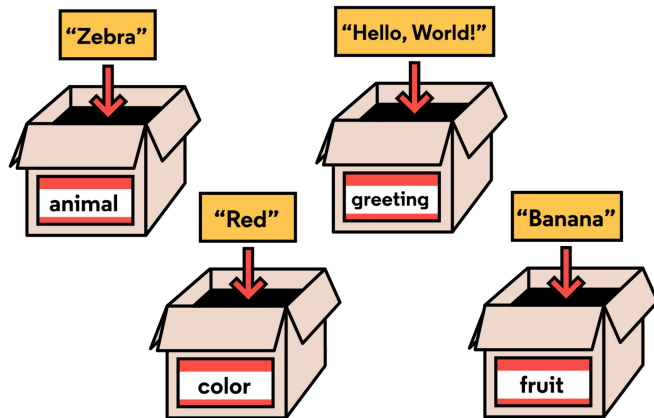
Comments



```
<?php
    echo 'This is a test'; // This is a one-line c++ style comment
    /* This is a multi-line comment
       yet another line of comment */
    echo 'This is yet another test';
    echo 'One Final Test'; # This is a one-line shell-style comment
?>
```

Variables

- Declaring a variable is the process of reserving a word, the variable name, which we'll be able to refer to in our code.
- It's good practice to name the variable in a way that describes the data it holds.
- Assignment is the process of associating that variable name with a specific value so that everytime we use the variable's name the computer will grab that value.

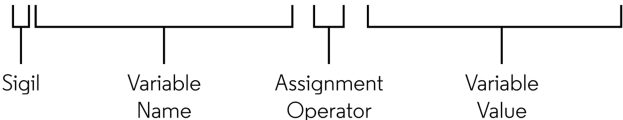


Creating Variables



- To declare a variable we use the dollar sign character (\$) (known as a **sigil**) followed by our chosen variable name.
- The variable name is **case-sensitive**.
- A valid variable name starts with a letter or underscore, followed by any number of letters, numbers, or underscores.
- The type of a variable is determined by the value it stores.

`$my_example = "my value";`



The diagram illustrates the components of the code snippet `$my_example = "my value";`. It uses brackets and vertical lines to identify four parts: the **Sigil** (\$), the **Variable Name** (my_example), the **Assignment Operator** (=), and the **Variable Value** ("my value").

Variables Examples



```
<?php
$var = 'Bob';
$Var = 'Joe';
echo "$var, $Var";           // outputs "Bob, Joe"

$4site = 'not yet';          // invalid; starts with a number
$_4site = 'not yet';         // valid; starts with an underscore
$täyte = 'mansikka';         // valid; 'ä' is (Extended) ASCII 228.
?>
```

Assign By Reference



- By default, variables are always assigned by value.
- Assign by reference, means that the new variable simply references the original variable. Changes to the new variable affect the original, and vice versa.
- To assign by reference, simply prepend an ampersand (&) to the beginning of the variable which is being assigned

```
<?php
```

```
$foo = 'Bob';  
$bar = &$foo;  
$bar = "My name is $bar";  
echo $bar;  
echo $foo;    // $foo is altered too.
```

```
?>
```

Types



- **Scalar types:**
 - bool
 - int
 - float (floating-point number, aka double)
 - String
- **Compound types:**
 - array
 - object
 - callable
 - Iterable
- **Special types:**
 - resource
 - NULL

```
<?php
$a_bool = TRUE;    // a boolean
$a_str  = "foo";   // a string
$a_str2 = 'foo';   // a string
$a_int  = 12;      // an integer
```

Predefined Variables



- **\$GLOBALS** — References all variables available in global scope
- **\$_SERVER** — Server and execution environment information
- **\$_GET** — HTTP GET variables
- **\$_POST** — HTTP POST variables
- **\$_FILES** — HTTP File Upload variables
- **\$_REQUEST** — HTTP Request variables
- **\$_SESSION** — Session variables
- **\$_ENV** — Environment variables
- **\$_COOKIE** — HTTP Cookies
- **\$php_errormsg** — The previous error message
- **\$http_response_header** — HTTP response headers
- **\$argc** — The number of arguments passed to script
- **\$argv** — Array of arguments passed to script

Variable scope



The scope of a variable is the context within which it is defined.

```
<?php
$a = 1; /* global scope */

function test()
{
    echo $a; /* reference to local scope variable */
}

test();
?>
```

The global keyword



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

function Sum() {
    global $a, $b;

    $b = $a + $b;
}

Sum();
echo $b;
```

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

function Sum() {
    $GLOBALS['b'] = $GLOBALS['a'] + $GLOBALS['b']
];
}

Sum();
echo $b;
```

Note:

Using global keyword outside a function is not an error. It can be used if the file is included from inside a function.

Using static variables



- A static variable exists only in a local function scope.
- It does not lose its value when program execution leaves this scope.

```
<?php
function test() {

    static $a = 0;
    echo $a;
    $a++;

}
?>
```

Variable Variables



```
<?php  
$a = 'hello';  
?>
```

```
<?php  
$$a = 'world';  
?>
```

```
<?php  
echo "$a ${$a}";  
?>
```

```
<?php  
echo "$a $hello";  
?>
```

Constants



- A constant is an identifier (name) for a simple value.
- value cannot change during the execution of the script.
- Constants are case-sensitive.
- Constants are global

```
<?php
```

```
// Valid constant names
define("FOO",      "something");
define("FOO2",     "something else");
define("FOO_BAR",  "something more");
```

```
// Invalid constant names
define("2FOO",     "something");
```

```
// This is valid, but should be avoided
define("__FOO__", "something");
```

```
// Another way to define a constant
const FOO3 = 1;
```

```
?>
```

Magic constants



Constants that change depending on where they are used.

Name	Description
<code>__LINE__</code>	The current line number of the file.
<code>__FILE__</code>	The full path and filename of the file.
<code>__DIR__</code>	The directory of the file.
<code>__FUNCTION__</code>	The function name, or {closure} for anonymous functions.
<code>__CLASS__</code>	The class name.
<code>__TRAIT__</code>	The trait name.
<code>__METHOD__</code>	The class method name.
<code>__NAMESPACE__</code>	The name of the current namespace.
<code>ClassName::class</code>	The fully qualified class name.

PHP Operators

Arithmetic Operators




Example	Name	Result
+\$a	Identity	Conversion of \$a to int or float as appropriate.
-\$a	Negation	Opposite of \$a.
\$a + \$b	Addition	Sum of \$a and \$b.
\$a - \$b	Subtraction	Difference of \$a and \$b.
\$a * \$b	Multiplication	Product of \$a and \$b.
\$a / \$b	Division	Quotient of \$a and \$b.
\$a % \$b	Modulo	Remainder of \$a divided by \$b.
\$a ** \$b	Exponentiation	Result of raising \$a to the \$b'th power.

Comparison Operators



Example	Name	Result
<code>\$a == \$b</code>	Equal	true if \$a is equal to \$b after type juggling.
<code>\$a === \$b</code>	Identical	true if \$a is equal to \$b, and they are of the same type.
<code>\$a != \$b</code>	Not equal	true if \$a is not equal to \$b after type juggling.
<code>\$a <> \$b</code>	Not equal	true if \$a is not equal to \$b after type juggling.
<code>\$a !== \$b</code>	Not identical	true if \$a is not equal to \$b, or they are not of the same type.
<code>\$a < \$b</code>	Less than	true if \$a is strictly less than \$b.
<code>\$a > \$b</code>	Greater than	true if \$a is strictly greater than \$b.
<code>\$a <= \$b</code>	Less than or equal to	true if \$a is less than or equal to \$b.
<code>\$a >= \$b</code>	Greater than or equal to	true if \$a is greater than or equal to \$b.
<code>\$a <=> \$b</code>	Spaceship	An int less than, equal to, or greater than zero when \$a is less than, equal to, or greater than \$b, respectively.

Incrementing/Decrementing Operators



Example	Name	Effect
<code>++\$a</code>	Pre-increment	Increments <code>\$a</code> by one, then returns <code>\$a</code> .
<code>\$a++</code>	Post-increment	Returns <code>\$a</code> , then increments <code>\$a</code> by one.
<code>--\$a</code>	Pre-decrement	Decrements <code>\$a</code> by one, then returns <code>\$a</code> .
<code>\$a--</code>	Post-decrement	Returns <code>\$a</code> , then decrements <code>\$a</code> by one.

```
<?php
$a = 5;
echo "Should be 5: " . $a++ . "<br />\n";
echo "Should be 6: " . $a . "<br />\n";
```

```
$a = 5;
echo "Should be 6: " . ++$a . "<br />\n";
echo "Should be 6: " . $a . "<br />\n";
```

Logical Operators



Example	Name	Result
\$a and \$b	And	true if both \$a and \$b are true.
\$a or \$b	Or	true if either \$a or \$b is true.
\$a xor \$b	Xor	true if either \$a or \$b is true, but not both.
! \$a	Not	true if \$a is not true.
\$a && \$b	And	true if both \$a and \$b are true.
\$a \$b	Or	true if either \$a or \$b is true.

Conditional Assignment Operators



Operator	Name	Example	Result
<code>?:</code>	Ternary	<code>\$x = expr1 ? expr2 : expr3</code>	Returns the value of <code>\$x</code> . The value of <code>\$x</code> is <code>expr2</code> if <code>expr1 = TRUE</code> . The value of <code>\$x</code> is <code>expr3</code> if <code>expr1 = FALSE</code>
<code>??</code>	Null coalescing	<code>\$x = expr1 ?? expr2</code>	Returns the value of <code>\$x</code> . The value of <code>\$x</code> is <code>expr1</code> if <code>expr1</code> exists, and is not <code>NULL</code> . If <code>expr1</code> does not exist, or is <code>NULL</code> , the value of <code>\$x</code> is <code>expr2</code> . Introduced in PHP 7

String Operators(Concatenation)



- The concatenation operator ('.'), which returns the concatenation of its right and left arguments.
- The concatenation assignment operator ('.='), which appends the argument on the right side to the argument on the left side.

```
<?php
$a = "Hello ";
$b = $a . "World!"; // now $b contains "Hello World!"

$a = "Hello ";
$a .= "World!"; // now $a contains "Hello World!"
?>
```



Strings

Strings



- A string is series of characters, where a character is the same as a byte.
- A string literal can be specified in four different ways:
 - single quoted
 - double quoted
 - heredoc syntax
 - nowdoc syntax

Single quoted

```
echo 'this is a simple string';
```

```
echo 'You can also have embedded newlines in  
strings this way as it is  
okay to do';
```

```
// Outputs: Arnold once said: "I'll be back"  
echo 'Arnold once said: "I\'ll be back"';
```

```
// Outputs: You deleted C:\*.*?  
echo 'You deleted C:\\*.*?';
```

```
// Outputs: You deleted C:\*.*?  
echo 'You deleted C:\*.*?';
```

```
// Outputs: This will not expand: \n a newline  
echo 'This will not expand: \n a newline';
```

```
// Outputs: Variables do not $expand $either  
echo 'Variables do not $expand $either';
```


Double quoted



PHP strings allow us to place variables directly into double quoted strings.

These variables will be parsed which means the computer will read the variables as the value they hold rather than see them as just a sequence of characters.

```
echo "this is a simple string";
```

```
echo "You can also have embedded newlines in  
strings this way as it is okay to do";
```

```
// Outputs: Arnold once said: "I'll be back"  
echo "Arnold once said: \"I'll be back\"";
```

```
// Outputs: You deleted C:\*.*?  
echo "You deleted C:\\*.*?";
```

```
// Outputs: This will expand:  
// a newline  
echo "This will expand: \n a newline";
```

```
$expand = "expand";  
// Outputs: Variables expand either  
echo "Variables $expand either";
```

Escape sequences



Sequence	Meaning
<code>\n</code>	linefeed
<code>\r</code>	carriage return
<code>\t</code>	horizontal tab
<code>\v</code>	vertical tab
<code>\\</code>	backslash
<code>\\$</code>	dollar sign
<code>\"</code>	double-quote

If the string is enclosed in double-quotes ("), PHP will interpret the following escape sequences for special characters:

Heredoc



```
$str = <<<EOD
Example of string
spanning multiple lines
using heredoc syntax.
EOD;
```

```
$foo = new foo();
$name = 'MyName';
```

```
echo <<<EOT
My name is "$name". I am printing some $foo->foo.
Now, I am printing some {$foo->bar[1]}.
This should print a capital 'A': \x41
EOT;
```

Nowdoc



```
<?php
echo <<<'EOD'
Example of string spanning multiple lines
using nowdoc syntax. Backslashes are always treated literally,
e.g. \\ and \'.
EOD;
```

```
$foo = new foo();
$name = 'MyName';
```

```
echo <<<'EOT'
My name is "$name". I am printing some $foo->foo.
Now, I am printing some {$foo->bar[1]}.
This should not print a capital 'A': \x41
EOT;
```

String functions



- **strpos/stripos** — Find the position of the first occurrence of a substring in a string
- **strlen** — Get string length
- **strrpos/strripos** — Find the position of the last occurrence of a substring in a string
- **strtolower** — Make a string lowercase
- **strtoupper** — Make a string uppercase
- **substr** — Return part of a string
- **trim** — Strip whitespace (or other characters) from the beginning and end of a string
- **ucfirst** — Make a string's first character uppercase
- **ucwords** — Uppercase the first character of each word in a string
- **str_pad** — Pad a string to a certain length with another string
- **str_repeat** — Repeat a string
- **str_replace/str_ireplace** — Replace all occurrences of the search string with the replacement string
- **str_ends_with** - Checks if a string ends with a given substring
- **str_starts_with** - Checks if a string starts with a given substring
- **str_contains** - Determine if a string contains a given substring

Conditionals and Logic

if, else, and elseif



```
<?php
```

```
if ($a > $b) {  
    echo "a is bigger than b";  
} elseif ($a == $b) {  
    echo "a is equal to b";  
} else {  
    echo "a is smaller than b";  
}
```

```
?>
```

When converting to bool, the following values are considered false:

- the boolean false itself
- the integer 0 (zero)
- the floats 0.0 and -0.0 (zero)
- the empty string, and the string "0"
- an array with zero elements
- the special type NULL (including unset variables)

switch



```
<?php
switch ($i) {
    case 0:
        echo "i equals 0";
        break;
    case 1:
        echo "i equals 1";
        break;
    case 2:
        echo "i equals 2";
        break;
    default:
        echo "i is not equal to 0, 1 or 2";
}
?>
```


switch: no break



```
<?php
switch ($i) {
    case 0:
    case 1:
    case 2:
        echo "i is less than 3 but not negative";
        break;
    case 3:
        echo "i is 3";
}
?>
```

match



- Introduced in PHP 8 as a more concise and powerful replacement for the "switch" statement.
- Allows developers to compare a single value against multiple possible values and execute different code depending on the match.
- Returns a value based on the matched expression, allowing developers to assign the result to a variable or use it in an expression.
- Unlike switch, the comparison is an identity check (`===`) rather than a weak equality check (`==`).

match Examples



```
<?php
$food = 'cake';

$return_value = match ($food) {
    'apple' => 'This food is an apple',
    'bar' => 'This food is a bar',
    'cake' => 'This food is a cake',
};
```

```
<?php

$age = 23;

$result = match (true) {
    $age >= 65 => 'senior',
    $age >= 25 => 'adult',
    $age >= 18 => 'young adult',
    default => 'kid',
};
```

match Examples



```
<?php
$result = match ($x) {
    // This match arm:
    $a, $b, $c => 5,
    // Is equivalent to these three match arms:
    $a => 5,
    $b => 5,
    $c => 5,
};
```

```
<?php
$result = match ($x) {
    foo() => ...,
    $this->bar() => ..., // $this->bar() isn't called if foo() === $x
    $this->baz => beep(), // beep() isn't called unless $x === $this->baz
    // etc.
};
```

include and require



- The include expression includes and evaluates the specified file.
- When a file is included, the code it contains inherits the variable scope of the line on which the include occurs.
- Any variables available at that line in the calling file will be available within the called file.
- All functions and classes defined in the included file have the global scope.
- require is identical to include except upon failure it will also produce a fatal E_COMPILE_ERROR level error.
- The include_once/require_once expressions are identical to include/require except PHP will check if the file has already been included, and if so, not include (require) it again.

include Example



vars.php
<?php

```
$color = 'green';  
$fruit = 'apple';
```

?>

test.php
<?php

```
echo "A $color $fruit"; // A
```

```
include 'vars.php';
```

```
echo "A $color $fruit"; // A green apple
```

?>

include with return Example

return.php

```
<?php
```

```
$var = 'PHP';
```

```
return $var;
```

```
?>
```

testreturns.php

```
<?php
```

```
$foo = include 'return.php';
```

```
echo $foo; // prints 'PHP'
```

```
$bar = include 'noreturn.php';
```

```
echo $bar; // prints 1
```

```
?>
```

Loops

while



- The first PHP loop that we will cover is the while loop. This type of loop continues to iterate as long as its conditional is true.
- This code outputs the numbers from 1-10, similar to the previous example:

```
<?php

$i = 1;
while ($i <= 10) {
    echo $i++; /* the printed value would be
                $i before the increment
                (post-increment) */
}
```

do while



- A do...while loop is very similar to a while loop.
- The main difference is that the code block will execute once without the conditional being checked.
- After the first iteration, it behaves the same as a while loop.

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

for



A for loop is commonly used to execute a code block a specific number of times.

```
<?php
for ($i = 1; $i <= 10; $i++)
{
    echo $i;
}

for ($i = 1; ; $i++) {
    if ($i > 10) {
        break;
    }
    echo $i;
    $i++;
}

for ($i = 1, $j = 0; $i <= 10; $j += $i, print $i, $i++);
```

foreach



- The foreach loop is used for iterating over an array.
- The code block is executed for every element in the array and the value of that element is available for use in the code block.

```
<?php
$arr = array(1, 2, 3, 4);
foreach ($arr as $value) {
    echo $value;
}

foreach ($arr as $key => $value) {
    echo "{$key} => {$value} ";
}
```

break and continue



- Similar to switch statements, the **break** keyword can be used to terminate any of the loop types early. In our counting example of a while loop, if we add a conditional and a break statement.
- The **continue** keyword is similar to break except it only ends the current iteration early, not the entire loop.

```
<?php
for ($i = 0; $i < 5; ++$i) {
    if ($i == 2) {
        break;
    }
    echo "$i\n";
}
?>
```

```
<?php
for ($i = 0; $i < 5; ++$i) {
    if ($i == 2) {
        continue;
    }
    echo "$i\n";
}
?>
```

Exercises



- Write a PHP program to check whether the first two characters and last two characters of a given string are same.
- Write a PHP program to check if a given string starts with 'Go' or not. (Don't use "str_starts_with" function)
- Write a PHP program to check if a given positive number is a multiple of 3 or a multiple of 7.
- Write a PHP program to check the largest number among three given numbers.
- Write a PHP program to check which number nearest to the value 100 among two given integers. Return 0 if the two numbers are equal.
- Write a PHP program to find the larger value from two positive integer values that is in the range 20..30 inclusive, or return 0 if neither is in that range.
- Write a PHP program to count the number of occurrences of any digit in a string.
- Write a PHP program to return the sum of digits of a an integer number.
- Write a PHP program to reverse any string. (Don't use "strrev" function)

Arrays

Arrays



- An array in PHP is actually an ordered map.
- A map is a type that associates values to keys.
- An array can be created using the array() language construct or [].
- It takes any number of comma-separated key => value pairs as arguments.
- The comma after the last array element is optional and can be omitted.
- The key can either be an int or a string. The value can be of any type.
- Additionally the following key casts will occur:
 - Strings containing valid decimal ints, unless the number is preceded by a + sign, will be cast to the int type.
 - Floats are also cast to ints.
 - Booleans are cast to ints, too, i.e. the key true will actually be stored under 1 and the key false under 0.
 - Null will be cast to the empty string.
 - Arrays and objects can not be used as keys.
- If multiple elements in the array declaration use the same key, only the last one will be used as all others are overwritten.

Creating Arrays



```
<?php
// Indexed array – numeric index
$array = array("bar", "foo", 24, 2.4);

// Associative array – named keys
$array = array(
    "foo" => "bar",
    "bar" => "foo",
);

// Using the short array syntax
$array = [
    "foo" => "bar",
    "bar" => "foo",
];
?>
```

Array Example



```
<?php
$array = array("foo", "bar", "hello", "world");
var_dump($array);
```

```
<?php
$array = array(
    "a",
    "b",
    6 => "c",
    "d",
);
var_dump($array);
```

Array Keys Example



```
$array = array(
    1      => 'a',
    '1'    => 'b', // the value "a" will be overwritten by "b"
    1.5    => 'c', // the value "b" will be overwritten by "c"
    -1     => 'd',
    '01'   => 'e', // as this is not an integer string it will NOT override the key for 1
    '1.5'  => 'f', // as this is not an integer string it will NOT override the key for 1
    true   => 'g', // the value "c" will be overwritten by "g"
    false  => 'h',
    ''     => 'i',
    null   => 'j', // the value "i" will be overwritten by "j"
    'k',   // value "k" is assigned the key 2.
    2      => 'l', // the value "k" will be overwritten by "l"
);

var_dump($array);
```

Array Example



```
$array = array(
    "foo" => "bar",
    42    => 24,
    "multi" => array(
        "dimensional" => array(
            "array" => "foo"
        )
    )
);

var_dump($array["foo"]);
var_dump($array[42]);
var_dump($array["multi"]["dimensional"]["array"]);
```

Array Modification



```
<?php
$arr = array(5 => 1, 12 => 2);

$arr[] = 56;    // This is the same as $arr[13] = 56;
                // at this point of the script

$arr["x"] = 42; // This adds a new element to
                // the array with key "x"

unset($arr[5]); // This removes the element from the array

unset($arr);    // This deletes the whole array
?>
```

Array destructuring



```
<?php
$source_array = ['foo', 'bar', 'baz'];

[$foo, $bar, $baz] = $source_array;

echo $foo;      // prints "foo"
echo $bar;      // prints "bar"
echo $baz;      // prints "baz"
?>
```

Array destructuring



```
<?php
$source_array = [
    [1, 'John'],
    [2, 'Jane'],
];

foreach ($source_array as [$id, $name]) {
    // logic here with $id and $name
}
?>
```

Array functions

- `array_pop` — Pop the element off the end of array
- `array_push` — Push one or more elements onto the end of array
- `array_shift` — Shift an element off the beginning of array
- `array_unique` — Removes duplicate values from an array
- `array_unshift` — Prepend one or more elements to the beginning of an array
- `array_values` — Return all the values of an array
- `array_keys` — Return all the keys or a subset of the keys of an array
- `array_map` — Applies the callback to the elements of the given arrays
- `sort` — Sort an array in ascending order
- `asort` — Sort an array in ascending order and maintain index association
- `count` — Counts all elements in an array or in a Countable object
- `ksort` — Sort an array by key in descending order
- `krsort` — Sort an array by key in ascending order
- `array_search` — Searches the array for a given value and returns the first corresponding key
- `in_array` — Checks if a value exists in an array
- `array_key_exists` — Checks if the given key or index exists in the array

Functions

User-defined functions



- Any valid PHP code may appear inside a function, even other functions and class definitions.
- A valid function name starts with a letter or underscore, followed by any number of letters, numbers, or underscores.
- Functions need not be defined before they are referenced.
- Values are returned by using the optional **return** statement.

Pseudo code to demonstrate function uses



```
<?php
function foo($arg_1, $arg_2, /* ..., */ $arg_n)
{
    echo "Example function.\n";
    return $retval;
}
?>
```

Functions within functions



```
<?php
function foo()
{
    function bar()
    {
        echo "I don't exist until foo() is called.\n";
    }
}
```

```
/* We can't call bar() yet since it doesn't exist. */
foo();
```

```
/* Now we can call bar(), foo()'s processing has made it accessible. */
bar();
```

Passing function parameters by reference



```
<?php
function add_some_extra(&$string)
{
    $string .= 'and something extra.';
}
$str = 'This is a string, ';

add_some_extra($str);
echo $str; // outputs 'This is a string, and something extra.'
?>
```

Default argument values



```
<?php
function makecoffee($type = "cappuccino")
{
    return "Making a cup of $type.\n";
}
echo makecoffee();
echo makecoffee(null);
echo makecoffee("espresso");
?>
```

Incorrect usage of default function arguments



```
<?php
function makeyogurt($container = "bowl", $flavour)
{
    return "Making a $container of $flavour yogurt.\n";
}

echo makeyogurt("raspberry");
// "raspberry" is $container, not $flavour
?>
```

Named Arguments



```
<?php
function makeyogurt($container = "bowl", $flavour = "raspberry",
    $style = "Greek")
{
    return "Making a $container of $flavour $style yogurt.\n";
}

echo makeyogurt(style: "natural");
?>
```

As of PHP 8.0.0, [named arguments](#) can be used to skip over multiple optional parameters.

Variable-length argument lists



- PHP has support for variable-length argument lists in user-defined functions by using the ... token.
- Argument lists may include the ... token to denote that the function accepts a variable number of arguments.
- The arguments will be passed into the given variable as an array.

```
<?php
function sum(...$numbers) {
    $acc = 0;
    foreach ($numbers as $n) {
        $acc += $n;
    }
    return $acc;
}

echo sum(1, 2, 3, 4);

echo sum(...[1, 2, 3, 4]);
?>
```

Older versions of PHP



```
<?php
function sum() {
    $acc = 0;
    foreach (func_get_args() as $n) {
        $acc += $n;
    }
    return $acc;
}

echo sum(1, 2, 3, 4);
?>
```

Type declared arguments/return



```
<?php
function add(int $a, int $b) : int {
    return $a + $b;
}

echo add(1, 2);
echo add(1, 'world'); // Error! Argument 2 is string not integer
?>
```

Variable functions

```
<?php
function foo() {
    echo "In foo()<br />\n";
}

function bar($arg = '') {
    echo "In bar(); argument was '$arg'.<br />\n";
}

$func = 'foo';
$func();           // This calls foo()

$func = 'bar';
$func('test');    // This calls bar()
?>
```

Anonymous functions



```
<?php
echo preg_replace_callback('~-([a-z])~', function ($match) {
    return strtoupper($match[1]);
}, 'hello-world'); // outputs helloWorld
```

```
$greet = function($name)
{
    printf("Hello %s\r\n", $name);
};
```

```
$greet('World');
$greet('PHP');
?>
```

Anonymous functions, also known as **closures**, allow the creation of functions which have no specified name.

Inheriting variables from the parent scope



```
<?php
$message = 'hello';

// No "use"
$example = function () {
    var_dump($message);
};
$example();

// Inherit $message when the function is defined.
$example = function () use ($message) {
    var_dump($message);
};
$example();
```

Arrow Functions



- Arrow functions were introduced in **PHP 7.4**
- Arrow functions have the basic form **fn (argument_list) => expr.**

```
<?php
```

```
$y = 1;
```

```
$fn1 = fn($x) => $x + $y;
```

```
// equivalent to using $y by value:
```

```
$fn2 = function ($x) use ($y) {  
    return $x + $y;  
};
```

```
?>
```

Exercises



- Write a PHP function that accepts an array of integers and return a new array after removing odd numbers.
- Write a PHP function that accepts an array of strings and return the longest string found in the array, the function should have a 2nd argument that will hold the index of the item that have the longest string in the input array.
- Write a function that accepts 2 arrays and return a new array that holds the value of multiplying each item in the first array by the corresponding item in the second array.
- Write a function to calculate the factorial of a number (a non-negative integer). The function accepts the number as an argument.
- Write a function to check whether a number is prime or not.