Table of contents

This document in one large webpage...

Introduction

The purpose of this course is to provide an intro to web technologies (HTML, CSS) and dynamic webpages via PHP.

Resources

The main documentation for this course is a site an can be found at https://asoete.github.io/howest-webtechnology. The main advantage of a using a site as a textbook is that the included examples and snippets can be rendered/formatted by the browser on the fly.

A PDF version of this document is also available. But keep in mind that some HTML features can get lost in the conversion to a pdf.

There are no slides available, all key aspects of this course will introduced via examples during the lessons. These examples and the solutions of the exercises made during this course will be published

here.

Keep in mind that this course is a very practical one and that making exercises is the best way to learn, get familiar, with all the aspects featured in this course.

Additional resources

An in depth guide/reference/manual for PHP can be found at http://php.net

For an HTML and CSS reference see http://www.w3schools.com/html and http://www.w3schools.com/css

Contact

If you have questions about this course and/or it's content please ask... You can contact me via arne.soete@howst.be

Code in this document

This course will feature a lot of code. The source-code of all the snippets in this manual can be found here.

The source and the output of each snippets are always displayed whenever a snippet is included.

Example:

introduction/embed_example | src

```
<?php

echo "<h1>This is an embed example</h1>";

if( array_key_exists( 'KEY', $_POST ) ) {

    unset($_POST['KEY']);
}

?>

<p>
  Markup is interpreted by the browser and formatted accordingly..
</p>
```

output of introduction/embed_example

> # This is an embed example
>
> Markup is interpreted by the browser and formatted accordingly..

Getting started

This course requires some software to be installed.

- A web browser: firefox
- A text editor: gedit

- PHP
- GIT

Normally these packages are already installed on the provided VM. If not, they can easily be installed by running:

```
sudo dnf install firefox gedit php git
```

Press `y` when prompted `Is this ok [y/N]:`.

This document and all the exercises/examples are hosted on GitHub. This means a local copy of the source can be obtained easily **and kept in sync with the latest changes and updates**.

If you choose not to use the command line and git. Snapshots of each lessons exercises and examples will be made available for download here.

Init workspace

```
mkdir ~/Documents/webtechnology
cd  ~/Documents/webtechnology
```

- Create your own exercises directory

```
mkdir exercises
```

Get local copy of the *exercises and examples* solutions

- Get an initial copy of the repository:

```
git clone https://github.com/asoete/howest-webtechnology-examples.git examples-solutions
```

  This will create a `examples-solutions` -folder which will hold example solutions for the exercises on a per lesson basis.
- To get the latest version/updates run:

```
# in examples-solutions folder
git pull origin master
```

> **Warning:** If you made local modifications to any of the files in this repository, this update command (`git pull`) will most likely fail. So don't modify the contents in this folder...

> **Info:** When you do encounter errors while pulling, run:
>
> ```
> git fetch --all
> git reset --hard origin/master
> ```
>
> This will reset the repository to be identical to the one on GitHub. **Be warned: local modifications will be lost...**

Get local copy of this site. (Optional)

- Get an initial copy of the repository:

```
git clone https://github.com/asoete/howest-webtechnology.git webtechnology-site
```

- To get the latest version/updates

```
# in webtechnology-site folder
git pull origin master
```

- Start a local instance of the site:

```
# in webtechnology-site folder
make serve
```

  And open http://localhost:8000 in a web browser.

Final result

If you complete all of the steps above, you will end up with a workspace that looks like this:

```
~/Documents/webtechnology
├── examples-solutions
├── exercises
└── site
```

HTML Basics

> **Info:** This course is based on the HTML specification and can differ from older specifications like XHTML and HTML

HTML is an XML subset. This means it is composed out of tags with, optionally, attributes.

Tags

A tag is delimited by `<` and `>`, for example: `<body>`.

There are two types of HTML-tags:

- Non self-enclosing tags
- Self-enclosing tags

Non self-enclosing tags

Non self-enclosing tags exist out of two parts:

1. An opening part: `<tag>`
2. And a closing part: `</tag>`.
   The closing part is identified by the forward slash (`/`) before the tag-name.

These *parts* are used to contain/format certain content.

```
<tag> {{content}} </tag>`
```

Example: `<strong>Bold Font</strong>` (This tags formats its content in a bold font: **Bold Font**)

Self-enclosing tags

A self-enclosing tag has no content to format. So the closing part is left of:

```
<tag>
```

Example: `<br>` (this will insert a newline into your HTML)

Sometimes you may see self-closing tags used like `<tag />`, this trailing tag is optional since HTML5 and can be left of.

Attributes

Attributes modify the behaviour of a tag.

For example the `a`-tag converts a peace of text into a clickable link.

```
<a>My text to click</a>
```

The `href`-attribute defines where the link should take you:

```
<a href="http://go-here-when-clicked.com">My text to click</a>
```

Attributes are also used to modify the appearance of a tag. Later in this course we'll see more detailed examples of this.

A valid HTML document

A valid HTML5 document requires a bit of boilerplate:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <!-- Your webpages metadata -->
</head>
<body>
    <!-- your webpage specific content -->
</body>
</html>
```

This minimal markup tells the browser to treat the document as HTML5.

The document head

The `head`-tag allows the developer to define meta-data about the webpage. It is a wrapper around multiple other tags.

```
<head>
  <!-- meta tags here -->
</head>
```

The `head` -tag may only be defined once in the complete document.

Title

The title tag sets the web-page title. This title is displayed by the browser in the browser-tab.

```
<head>
  <title>My web-page's title...</title>
</head>
```

Style

We will address styling later in this course but for now it is sufficient to know that style information should be included in the head of a web-page.

Raw style

The `style` -tag allows to include raw CSS rules in the documents

```
<head>
  <style type="text/css">
    /* style information here */
  </style>
</head>
```

External style sheets

The `link` -tag allows to external style sheets into the document. (Do not confuse this tag with the `a` -tag...).

```
<head>
  <link href="/link/to/file.css" type="text/css" rel="stylesheet">
</head>
```

We will only ever include CSS-files to style our web-pages. The provided attributes in the example are required to include a CSS-file and avoid browser quirks.

The document body

The `body` -tag should wrap all the content to be displayed.

```
<body>
  <!-- all displayed tags and content go here -->
</body>
```

Exercise:

- Create a valid HTML-document with
  - Title: Hello World
  - Content: `Hello World from the my first web-page`

HTML: Hello World

Create a text file name `hello-world.html`

```
gedit hello-world.html
```

With contents:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <title>Hello World</title>
</head>
<body>
   Hello World from my first web-page.
</body>
</html>
```

Open the local HTML file in the browser.

```
firefox hello-world.html
```

Headers: `H*`

The purpose of a header is to indicate the start of a new block a add an appropriate heading.

The `hn`-tags come in 6 variations. From to highest order header `h1` to the lowest `h6`.

The browser auto-formats these headers accordingly from largest to smallest font-size.

html-intro/headers | *src*

```
<h1>Header 1</h1>
<h2>Header 2</h2>
<h3>Header 3</h3>
<h4>Header 4</h4>
<h5>Header 5</h5>
<h6>Header 6</h1>
```

output of html-intro/headers

Header 1

Header 2

Header 3

Header 4

Header 5

Header 6

Containers

The purpose of these types of tags is to wrap other content. Why the content should be wrapped can vary:

- To indicate semantic meaning (new paragraph, a quote, ...)
- To position and/or style the contents in the container.

They are also referred to as *block*-elements

Paragraphs `p`

The `p`-tag encloses a blob of related text into a paragraph

html-intro/p-tag | *src*

```
Content before...
<p>
  Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy
  eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam
  voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita
  kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.
</p>
Content after...
```

output of html-intro/p-tag

Content before...

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Content after...

Generic container `div`

The `div` defines a *division* in the document. It is used al lot to wrap some content and apply styles.

It has no special styles by default

html-intro/div-tag | *src*

```
Content before...
<div>
  Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy
  eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam
  voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita
  kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.
</div>
Content after...
```

output of html-intro/div-tag

Content before...

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Content after...

Blockquote `blockquote`

The `blockquote`-tag is used to denote some block of text as a quote from another source.

html-intro/blockquote-tag | *src*

```
Content before...
<blockquote>
  Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy
  eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam
  voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita
  kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.
</blockquote>
Content after...
```

output of html-intro/blockquote-tag

Content before...

> Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Content after...

Inline tags

These tags are inline because they do not start a new block (identified by new lines) as the previous tags.

Their purpose is either to give a specific style and semantic meaning to an element or to extend a certain functionality to the element.

Anchors (links): a

The a is used to link to other web-pages.

In order the function, the href -attribute is required on the a -element.

```
<a href="http://google.com">Link to goole</a>
```

html-intro/a-tag | *src*

> Link to goole
>
> *output of html-intro/a-tag*

> ## Exercise:

A newline: br

The br insert a newline into the document.

```
This is the first line.
This is the second line, but in html all white space is replaced by a single space...<br> The "br"-tag however instructs the browser to continue on a new line...<br><br>Cool right?
```

html-intro/br-tag | *src*

> This is the first line. This is the second line, but in html all white space is replaced by a single space...
> The "br"-tag however instructs the browser to continue on a new line...
>
> Cool right?
>
> *output of html-intro/br-tag*

Emphasise text: em

The em -tag allows to emphasise certain text.

```
This is <em>emphasised</em> inline...
```

html-intro/em-tag | *src*

> This is *emphasised* inline...
>
> *output of html-intro/em-tag*

Small text: small

The small -tag indicates the browser to use a smaller font-size to visualise this content.

```
This is <small>small</small> inline...
```

html-intro/small-tag | *src*

> This is small inline...
>
> *output of html-intro/small-tag*

Inline wrap text: `span`

The `a`

This is ****span**** inline...

html-intro/span-tag | *src*

> This is span inline...
>
> *output of html-intro/span-tag*

Strike text: `strike`

The `a`

This is **<strike>**strike**</strike>** inline...

html-intro/strike-tag | *src*

> This is ~~strike~~ inline...
>
> *output of html-intro/strike-tag*

Bold text: `strong`

The `a`

This is ****strong**** inline...

html-intro/strong-tag | *src*

> This is **strong** inline...
>
> *output of html-intro/strong-tag*

Inline quote text: `q`

The `a`

This is **<q>**quote**</q>** inline...

html-intro/quote-tag | *src*

> This is "quote" inline...
>
> *output of html-intro/quote-tag*

> ### Exercise:
>
> - Make a web-page with links to:
>   - google.com

    - howest.be
    - github.com
- Print the following text so the sentences are broken up as below.

  > HTML is a markup language browser understand to format documents.
  > CSS is a way to style this markup.
  > PHP is a programming language.
  > It is used to dynamically generate HTML-markup.

- Print the following text so `hello world` is emphasised.

  > Let's emphasise hello world in this sentence.

- Print the following text so `hello world` is smaller

  > Let's make hello world smaller in this sentence.

- Print the following text so `hello world` is bold

  > Let's make hello world bold in this sentence.

- Print the following text so `hello world` is crossed of.

  > Let's strike hello world in this sentence.

Attributes

Exercises

title: PHP basics

PHP Basics

> PHP is a widely-used general-purpose scripting language that is especially suited for Web development and can be embedded into HTML.
>
> source: php.net

This means PHP can be used to generate HTML. This allows us to adhere to the DRY ("Don't Repeat Yourself") principle.

What makes a PHP-script

A PHP-script is identified by its `.php` extension and the PHP-tags in the file.

PHP tags

PHP interprets only the code enclosed within the special PHP-tags.

- Opening tag: `<?php`
- Closing tag: `?>`

php-basics/php-tags | src

```
echo "Before php-tags";

<?php

echo "Within php-tags\n";

?>

echo "After php-tags";
```

output of php-basics/php-tags

```
echo "Before php-tags";

Within php-tags

echo "After php-tags";
```

Notice that the code outside of the PHP-tags is not interpreted and this printed out unchanged.

A valid PHP instruction generally has the form:

```
{{ instruction }};
```

Each statement must be terminated by a semicolon (`;`)!

An exception to this rule are loops and conditionals. These can encapsulate a block of code in curly brackets `{}` and thus end in a `}` ...

Comments

Single line comments

PHP will ignore everything behind a `#` or `//` .

```
echo 'hello world';

// ignore this

# and ignore this

echo 'by world';
```

Multi-line comments

PHP will ignore everything enclosed by `/*` and `*/`.

```
echo 'hello world';

/* ignore this

and ignore this */

echo 'by world';
```

Execute a PHP-script

To get acquainted with php we will start of on the command line and work our way up to PHP as a web server.

PHP at its core is a program which reads a source file, interprets the directives an prints out the result.

Basic invocation:

```
php <script-to-execute>.php
```

The above command will print the output to the `STDOUT`.

Hello World

The obligatory hello world.

Create a file: `hello-world.php` with content:

php-basics/hello-world | src

```
<?php

echo "Hello World";

?>
```

output of php-basics/hello-world

```
Hello World
```

Info: The echo statement prints a string. See echo for more info

Info: If you get an *command not found* error, you probably have to install php. Run: `sudo dnf install php`

Run it via:

```
php hello-world.php
```

on the command line.

Types and variables

Variables are a way to store some information and give the storage space a name. Via this name, the content that was stored can be retrieved.

```
$name = 'value to store';
```

A variable is identified by the leading dollar `$` -symbol followed by a alpha-numeric sequence.

Warning: It is not allowed to start variable name with a number:

$abc	OK
$abc123	OK
~~$123abc~~	Not allowed

PHP knows two main types of variables:

- scalars
- arrays

Scalars

A scalar variable can hold an atomic quantity. For example: one string, or one number, ...

Types

PHP knows four scalar types:

Type	Example	Description
Integers	`42`	Real numbers
Floats	`3.1415`	Real numbers
Strings	`'Hello world'`	Strings can be any number or letter in a sequence (enclosed by single `'` or double `"` quotes, otherwise php may interpret it as a directive...)
Boolean	**`true`** or **`false`**	binary true or false

Declare

Assign a value to a variable:

Generic syntax:

```
$varname = 'value to store';
```

Examples:

```
$int    = 123;
$float  = 4.56;
$string = 'Hello World';
$true   = true;
$false  = false;
```

Printing/Displaying scalars

A scalar can be printed via two methods:

- `echo`
- `print`

Echo

Generic syntax:

```
echo <scalar>;
echo <scalar1>, <scalar2>, ...;
```

Echo outputs the provided scalars.

Multiple scalars can be printed at once, just separate them by a comma `,`.

Example:

```
echo 123;
echo 4.56;
echo 'Hello World';
echo true;
echo false;
```

Print

Generic syntax:

```
print( <scalar> );
```

Print can only output one scalar at the time. (This can be circumvented via concatenation...)

Example:

```
print( 123 );
print( 4.56 );
print( 'Hello World' );
print( true );
print( false );
```

String concatenation and interpolation

Scalars can be combined, concatenated into larger strings.

The concatenation symbol is a dot: `.`.

php-basics/concatenate | *src*

```php
<?php

print( 'This is a string' . ' || ' . 'this is a number: ' . 42 );
```

output of php-basics/concatenate

```
This is a string || this is a number: 42
```

You may have already noticed that printing variables enclosed by single quotes `'` doesn't work. The literal variable name is printed instead.

php-basics/variables-in-single-quotes | *src*

```php
<?php

$variable = 'Hello World';

echo 'The variable contains: $variable!';
```

output of php-basics/variables-in-single-quotes

```
The variable contains: $variable!
```

To instruct PHP to interpret the variables, and other special sequences, the string must be enclosed by double quotes: `"`.

php-basics/variables-in-double-quotes | *src*

```php
<?php

$variable = 'Hello World';

echo "The variable contains: $variable!";
```

output of php-basics/variables-in-double-quotes

```
The variable contains: Hello World!
```

Special character sequences

The following special character sequences are interpreted by PHP and formatted accordingly...

Sequence	Result
`\n`	New line
`\r`	New line (Windows)
`\t`	The literal -character
`\$`	Literal `$` (escaping prevents variable interpreation)

Sequence	Result
`\"`	Literal `"` (escaping prevents string termination).

Example:

php-basics/escape-sequences | *src*

```php
<?php

$variable = "hello world";

echo "1. The value of the variable is: $variable.";
echo "2. The value of the variable is: $variable.\n";
echo "\t3. The value of the variable is: $variable.\n";
echo "4. The value of the variable is: \$variable.\n";
echo "5. The value of the variable is: \"$variable\".\n";
```

output of php-basics/escape-sequences

```
1. The value of the variable is: hello world.2. The value of the variable is: hello world.
   3. The value of the variable is: hello world.
4. The value of the variable is: $variable.
5. The value of the variable is: "hello world".
```

Basic arithmetic

Floats an Integers can be used in arithmetic.

Example	Name	Result
`-$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`	Modulus	Remainder of $a divided by $b.
`$a ** $b`	Exponentiation	Result of raising $a to the $b 'th power. Introduced in PHP 5.6.

Example:

```php
<?php

$a = 42;
$b = 3.1415;
$c = 5;

echo $a + $b  . "\n";
echo $a - $b  . "\n";
echo $a * $b  . "\n";
echo $a / $b  . "\n";
echo $a % $c  . "\n";
echo $a ** $c . "\n";
```

php-basics/arithmetic | *src*

output of php-basics/arithmetic

```
45.1415
38.8585
131.943
13.369409517746
2
130691232
```

Arrays

Arrays are able to hold more than one item.

An item is stored in the array at a named location. If no name/key/index is explicitly specified, an numeric index from `0` to `n` (where n is the number of items in the array minus one) is used as the keys.

Declare

An array can be declared in two ways:

```php
$array = array( /* list of array items */ );

$array = [ /* list of array items */ ];
```

The `[]`-method can only be used from PHP version 5.4 and higher.

A normal typical array is a **list of values**. The keys of those values are automatically assigned, starting with zero `0` and auto incrementing for each element added.

See below for how to print and add values to arrays

```php
<?php

$array = [1,2,3];

print_r($array);

$array[] = 'hello';

print_r($array);
```

php-basics/array-auto-increment | *src*

output of php-basics/array-auto-increment

```
Array
(
    [0] => 1
    [1] => 2
    [2] => 3
)
Array
(
    [0] => 1
    [1] => 2
    [2] => 3
    [3] => hello
)
```

The keys can however be specified manually:

php-basics/array-custom-keys | *src*

```php
<?php

$array = [
    'key1' => 'value1',
    'two'  => 2,
    3      => 'hello world',
];

print_r($array);
```

output of php-basics/array-custom-keys

```
Array
(
    [key1] => value1
    [two] => 2
    [3] => hello world
)
```

Print/Display arrays

The function `print_r` can be used to print an array.

Generic syntax:

```php
print_r( $array );
```

```php
<?php

$array = array(
   'one' => 1,
   'two' => 'three',
   4     => 'four',
   'hello' => 'world'
);

print_r( $array );
```

php-basics/print_r | src

```
Array
(
    [one] => 1
    [two] => three
    [4] => four
    [hello] => world
)
```

output of php-basics/print_r

Get a value from an array

A value can be retrieved by specifying the array variable name followed by the index you which to retrieve encloded in square brackets:

```php
$array[<key>];
```

If the key is a string, the appropriate quoting must be used.

```php
$array['<key>'];
```

Example:

```php
<?php

$array = [1,2,3];

echo $array[0] . "\n";
echo $array[1] . "\n";
echo $array[2] . "\n";

$array_assoc = [
   'key1' => "value1",
   'key2' => "value2",
   'key3' => "value3",
];

echo $array_assoc['key1'] . "\n";
echo $array_assoc['key2'] . "\n";
echo $array_assoc['key3'] . "\n";
```

php-basics/array-get-key | src

output of php-basics/array-get-key

```
1
2
3
value1
value2
value3
```

Update a value in an array

An array value can be targeted by its key. This key can also be used to update the value:

```
$array[<key>] = <new value>;
```

Example:

php-basics/array-update-value | *src*

```php
<?php

$array = [
    "value1",
    "value2",
    "value3",
    100 => "hundred",
    'key' => "value",
];

print_r($array);

$array['key'] = "new value for key";

$array[1] = 'index 1 now points here';

print_r($array);
```

output of php-basics/array-update-value

```
Array
(
    [0] => value1
    [1] => value2
    [2] => value3
    [100] => hundred
    [key] => value
)
Array
(
    [0] => value1
    [1] => index 1 now points here
    [2] => value3
    [100] => hundred
    [key] => new value for key
)
```

Manipulating arrays

Add an item to the end of an array:

Adding an element in front of an array can be accomplished by the function `array_push`.

php-basics/array-append | src

```php
<?php

$array = [1,2,3];

print_r( $array );

array_push( $array, 4);

print_r( $array );

// or

$array[] = 5;

print_r( $array );
```

output of php-basics/array-append

```
Array
(
    [0] => 1
    [1] => 2
    [2] => 3
)
Array
(
    [0] => 1
    [1] => 2
    [2] => 3
    [3] => 4
)
Array
(
    [0] => 1
    [1] => 2
    [2] => 3
    [3] => 4
    [4] => 5
)
```

Add an item in front of an array:

Adding an element at the end of an array can be accomplished by the function `array_unshift`.

```php
<?php

$array = [1,2,3];

print_r( $array );

array_unshift( $array, 4 );

print_r( $array );
```

php-basics/array-prepend | src

output of php-basics/array-prepend

```
Array
(
    [0] => 1
    [1] => 2
    [2] => 3
)
Array
(
    [0] => 4
    [1] => 1
    [2] => 2
    [3] => 3
)
```

Extract the first element from an array

Extracting the first element from an array can be accomplished by the function `array_shift`.

```php
<?php

$array = [1,2,3];

print_r( $array );

echo array_shift( $array ) . "\n";

print_r( $array );
```

php-basics/array-shift | src

output of php-basics/array-shift

```
Array
(
    [0] => 1
    [1] => 2
    [2] => 3
)
1
Array
(
    [0] => 2
    [1] => 3
)
```

Extract the last element from an array

Extracting the last element from an array can be accomplished by the function `array_pop`.

php-basics/array-pop | *src*

```php
<?php

$array = [1,2,3];

print_r( $array );

echo array_pop( $array ) . "\n";

print_r( $array );
```

output of php-basics/array-pop

```
Array
(
    [0] => 1
    [1] => 2
    [2] => 3
)
3
Array
(
    [0] => 1
    [1] => 2
)
```

Count the elements in an array

Counting the elements in an array can be accomplished by the function `count`.

```php
<?php

$array = [ 1, 2, 3 ];

echo count($array) . "\n";

$array[] = "Add item";

echo count($array) . "\n";

array_unshift( $array );
array_unshift( $array );

echo count($array) . "\n";
```

php-basics/array-count | *src*

```
3
4
4
```

output of php-basics/array-count

Conditionals

It can be very handy to execute a piece of code only when certain requirement are met. This kind of behaviour can be accomplished via conditionals

The `if` language structure defines the conditions to fulfil and the accompanying block of code to run if the conditions evaluate to `true` (enclosed in curly brackets `{}`).

```php
if( /* <condition> */ ) {

    /* execute this code here */
}
```

Additionally an `else`-block can be defined. The code in this block will be executed when the *`if`-condition* evaluated to false.

```php
if( /* condition */ ) {

    /* execute when condition is true */
}
else {

    /* execute when condition is false */
}
```

On top of this, multiple conditions can be chained into an `if-elsif-else` construct.

```
if( /* condition 1 */ ) {

    /* execute when condition 1 is true */
}
elseif( /* condition 2 */ ) {

    /* execute when condition 2 is true */
}
elseif( /* condition 3 */ ) {

    /* execute when condition 3 is true */
}
else {

    /* execute when conditions 1, 2 and 3 are false */
}
```

Conditionals can also be nested:

```
if( /* condition 1 */ ) {

    if( /* condition 2 */ ) {

        /* execute when condition 1 and 2 evaluate to true */
    }
    else {

        /* execute when conditions 1 evalutes to true and condition 2 to false */
    }
}
else {

    /* execute when condition 1 evaluates to false*/
}
```

Comparison operators

Example	Name	Result
`$a == $b`	Equal	`true` if `$a` is equal to `$b` after type juggling.
`$a === $b`	Identical	`true` if `$a` is equal to `$b`, and they are of the same type.
`$a != $b`	Not equal	`true` if `$a` is not equal to `$b` after type juggling.
`$a <> $b`	Not equal	`true` if `$a` is not equal to `$b` after type juggling.
`$a !== $b`	Not identical	`true` if `$a` is not equal to `$b`, or they are not of the same type.
`$a < $b`	Less than	`true` if `$a` is strictly less than `$b`.

Example	Name	Result
`$a > $b`	Greater than	`true` if `$a` is strictly greater than `$b`.
`$a <= $b`	Less than or equal to	`true` if `$a` is less than or equal to `$b`.
`$a >= $b`	Greater than or equal to	`true` if `$a` is greater than or equal to `$b`.

PHP is a dynamically type language. This means the type of a variable is not set in stone but PHP will try its best to guess the types of variables and convert them (juggle them from one type to the other) where its deemed necessary.

For example:

php-basics/type-juggling | src

```php
<?php

$string = '1 as a string';

var_dump($string);

# $string to int = 1 the `+` triggers the type juggling
var_dump( $string + 0);

# ------------------------------------------------------------------------

var_dump( '1' == 1, 1 == true, 'abc' == true );
var_dump( '1' === 1, 1 === true, 'abc' === true );
```

output of php-basics/type-juggling

```
string(13) "1 as a string"
int(1)
bool(true)
bool(true)
bool(true)
bool(false)
bool(false)
bool(false)
```

Info: `var_dump` prints a variable with type information

Logical operators

Multiple comparisons can be bundled together into one condition. They are combined via the logical operators:

Example	Name	Result
`$a and $b`	And	`true` if both `$a` and `$b` are `true`.

Example	Name	Result
`$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`.

Example:

Ma_embed_php(php-basics/logical-operators)

These logical operators can be combined at will. Brackets `()` can be used to enforce precedence.

Ma_embed_php(php-basics/logical-precedence)

Loops

Loops enable you to repeat a block of code until a condition is met.

While

This construct will repeat until the defined condition evaluates to false:

```
while( /* <condition> */ ) {

    /* execute this block */
}
```

Warning: Incorrectly formatted code can result in an endlessly running script. If this happens, use `Ctrl` + `c` on the command line to abort the running script.

Danger: The never ending loop:

```
# This will run until interrupted by the user.

while(1) {

    echo "Use `Ctrl`+`c` to abort this loop\n";
}
```

php-basics/loops-while | *src*

```php
<?php

$iterations = 10;

while( $iterations > 0 ) {

    echo "Countdown finished in $iterations iterations\n";
    $iterations = $iterations - 1;
}
```

output of php-basics/loops-while

```
Countdown finished in 10 iterations
Countdown finished in 9 iterations
Countdown finished in 8 iterations
Countdown finished in 7 iterations
Countdown finished in 6 iterations
Countdown finished in 5 iterations
Countdown finished in 4 iterations
Countdown finished in 3 iterations
Countdown finished in 2 iterations
Countdown finished in 1 iterations
```

Info: The pattern `$variable = $variable + 1` is used a lot in programming. Therefore shorthand versions if this,and similar operations, are available:

```php
$var = 1;

# Add or substract by 1:
$var++; // increment by 1
$var--  // decrement by 1

# Add or substract by n:
# $var += n;
# $var -= n;

$var += 3;
$var += 100;
$var -= 42;
$var -= 4;
```

Exercise:

- Make a script that counts from 0 to 10
- Modify the script to count from 50 to 60
- Modify the script to count from 0 to 10 and back to 0
- Modify the script to count from 0 to 30 in steps of 3.

Only while loops are allowed.

For

For is similar to while in functionality. It also loops until a certain condition evaluates to `true`. The main difference is the boilerplate required to construct the loop.

The `for`-construct forces you to define the counter variable and the increments right in the construct.

```
for( <init counter>; <condition>; <increment counter> ) {

    /* execute this block */
}
```

Notice the semi-colons `;` between each of the `for`-parts!

php-basics/loops-for | *src*

```php
<?php

for( $counter = 0; $counter < 10; $counter++ ) {

    echo "Loop interation: $counter\n";
}
```

output of php-basics/loops-for

```
Loop interation: 0
Loop interation: 1
Loop interation: 2
Loop interation: 3
Loop interation: 4
Loop interation: 5
Loop interation: 6
Loop interation: 7
Loop interation: 8
Loop interation: 9
```

Exercise:

- Make a script that counts from 1 to 10
- Modify the script to count from 0 to 10 and back to 0
- Modify the script to count from 0 to 30 in steps of 3.

Only for loops are allowed.

The for construct can also be used to loop over all elements in an array:

php-basics/loops-for-array | src

```php
<?php

$array = [
    1,
    2,
    'three',
    'value'
];

print_r($array);

for( $i = 0; $i < count($array); $i++ ){

    echo "\$array has value: '". $array[$i] . "' at index $i\n";
}
```

output of php-basics/loops-for-array

```
Array
(
    [0] => 1
    [1] => 2
    [2] => three
    [3] => value
)
$array has value: '1' at index 0
$array has value: '2' at index 1
$array has value: 'three' at index 2
$array has value: 'value' at index 3
```

Exercise:

- Fill an array with: [one, two, three, four, five];
- Print each word on a single line.
- Modify the script to also print the index before the word: `$index: $value`

Foreach

The for and the while construct have their limitations regarding arrays. What if we have an array with custom keys (not a sequential list of integers...)?

We can solve this problem with the `foreach` construct. This construct is specifically designed to iterate over array items.

```
foreach( <array> as [<key-placeholder> =>] <value-placeholder>) {

    /* use key and value here*/
}
```

Info: The `key-placeholder =>` part is placed into square brackets to indicate that this part of the construct is optional. The part can be omitted when we have no need of the key in the accompanying block but are only interested in the values...

php-basics/loops-foreach | src

```php
<?php

$array = [ 1, 2, 'three', 'value' ];

print_r($array);

foreach( $array as $value ) {

   echo "The obtained value is: `$value`\n";
}

# ------------------------------------------------------------------------ #

$array = [
   1 , 2, 3,
   'key1' => 'value1',
   100 => 'hello'
];

print_r($array);

foreach( $array as $key => $value ) {

   echo "Key: `$key` has value: `$value`\n";
}
```

output of php-basics/loops-foreach

```
Array
(
    [0] => 1
    [1] => 2
    [2] => three
    [3] => value
)
The obtained value is: `1`
The obtained value is: `2`
The obtained value is: `three`
The obtained value is: `value`
Array
(
    [0] => 1
    [1] => 2
    [2] => 3
    [key1] => value1
    [100] => hello
)
Key: `0` has value: `1`
Key: `1` has value: `2`
Key: `2` has value: `3`
Key: `key1` has value: `value1`
Key: `100` has value: `hello`
```

HTML extended: beyond the basics

External stylesheets

Forms
