

Table of contents

Introduction

- Resources
- Contact
- Code in this document

Getting started

HTML Basics

- Tags
- Attributes
- A valid HTML document
- The document head
- The document body
- Headers: H*
- Containers
- Inline tags
- Multi element markup
- Forms
- Attributes

PHP Basics

- What makes a PHP-script
- Types and variables
- Conditionals
- Loops
- Exercises

PHP and HTML

- PHP Webserver
- Include / Require
- Forms
- PHP special data arrays
- Exercises

HTML and CSS

- Include style information
- CSS selectors
- Colours
- Background
- Font and text.
- Border
- Margin
- Exercises

[This document in one large webpage...](#)

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@howest.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:

```
<?php
```

[introduction/embed_example](#) | [src](#)

```
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>
```

This is an embed example

output of introduction/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: `Bold Font` (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: `
` (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 and add an appropriate heading.

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

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

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

[html-intro/headers](#) | [src](#)

Header 1

output of html-intro/headers

Header 2

Header 3

Header 4

Header 5

Header 6

Exercise:

- Create a header for each `Hn`-tag

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

Content before...

[html-intro/p-tag](#) | [src](#)

<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...

Content before...

output of html-intro/p-tag

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 a lot to wrap some content and apply styles.

It has no special styles by default

Content before...

[html-intro/div-tag](#) | [src](#)

<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...

Content before...

output of html-intro/div-tag

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.

Content before...

[html-intro/blockquote-tag](#) | [src](#)

<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...

Content before...

output of [html-intro/blockquote-tag](#)

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...

Exercise:

- Create a block of text and wrap it in.
 - No tags
 - div tags
 - p tags
 - blockquote tags And notice the difference

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.

Anchor (links): **a**

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

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

Link to google

[html-intro/a-tag](#) | [src](#)

[Link to google](http://google.com)

output of [html-intro/a-tag](#)

Exercise:

- Create links to
 - google.com
 - howest.be
 - php.net
 - github.com

A newline: `br`

The `br` insert a newline into the document.

This is the first line.

[html-intro/br-tag](#) | [src](#)

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?

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?

Emphasise text: `em`

The `em`-tag allows to emphasise certain text.

This is ``emphasised`` 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.

Multi element markup

Some elements don't make any sense on their own. They should be part of a larger elements-group.

Lists

A HTML-list is composed of `li`-tags enclosed by an `ul` or `ol`-tag.

Unordered lists `ul`

```
<ul>
  <li>list item 1</li>
  <li>list item 2</li>
  <li>list item 3</li>
</ul>
```

[html-intro/list-unordered](#) | [src](#)

- list item 1
- list item 2
- list item 3

output of [html-intro/list-unordered](#)

Ordered lists `ol`

```
<ol>
  <li>list item 1</li>
  <li>list item 2</li>
  <li>list item 3</li>
</ol>
```

[html-intro/list-ordered](#) | [src](#)

1. list item 1
2. list item 2
3. list item 3

output of html-intro/list-ordered

Exercise:

- Make an unordered list with your name, age and gender as items
- Make your name bold, age emphasised and gender quoted.
- Make a top 3 ranked list of your favorite dishes
- Add a fourth dish, but with smaller font .

Tables

A simple table is composed out of:

- a table wrapper: `table`
- rows: `tr`
- header cells `th`
- and normal cells `td`

```
<table>
  <tr>
    <th>Firstname</th>
    <th>Lastname</th>
    <th>Age</th>
  </tr>
  <tr>
    <td>John</td>
    <td>Doe</td>
    <td>21</td>
  </tr>
  <tr>
    <td>Jane</td>
    <td>Doe</td>
    <td>18</td>
  </tr>
</table>
```

[html-intro/simple-table](#) | [src](#)

Firstname Lastname Age

output of `html-intro/simple-table`

John	Doe	21
Jane	Doe	18

Exercise:

- Make a table with two columns: name and score
- Add 3 rows
 - Jan -> 12
 - Piet -> 15
 - Boris -> 7
- Make the names also headers
- Add a column 'passes' and add a V if the number is larger than 10 and an X otherwise.

Forms

Attributes

As already seen with the `a`-tag, attributes can modify the behaviour of an `HTML`-element.

The `a`-tag requires the `href`-attribute to be set. Otherwise the browser has no clue where to take the user on a click.

The attributes are also often used to modify the appearance of an element.

Commonly used attributes:

Class

The class attribute holds a space separated list class-names. The element is member of all the classes specified in the attribute. These classes can be used to style a group of elements the same way.

For example all the elements which are member of the same class (have the same class-name in the class attribute) should have the text colour set to red...

```
<p class="class1" >...</p>
<p class="class1 class-two" >...</p>
```

Id

The `id`-attribute lets you assign a unique identifier to an element.

This identifier should be unique for the whole page and thus occur only once.

```
<p id="unique-identifier" >...</p>
```

If the id is specified in the URL prefixed by a pound symbol (#), the element will be automatically scrolled into view.

```
<!-- http://WWW.example.com/script.php#chapter -->

<p id="chapter1">
Scroll into view...
</p>
```

Style

The style attribute can be used to apply CSS-rules to a single element.

Generally speaking you should not set the styles via this tag. A dedicated style block in the head of page or an external style sheet are better, more scalable, options. It can however come in handy in this introduction to HTML and CSS.

```
<p style="background: red; color: green;">...</p>
```

See [HTML](#) and [CSS](#) for more info about styling an element.

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: `?>`


```
echo "Before php-tags";
```

[php-basics/php-tags](#) | [src](#)

```
<?php
```

```
echo "Within php-tags\n";
```

```
?>
```

```
echo "After php-tags";
```

```
echo "Before php-tags";
```

output of php-basics/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 and 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                                     php-basics/hello-world | src

echo "Hello World";

?>
```

Hello World

output of php-basics/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 an alpha-numeric sequence.

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

<code>\$abc</code>	OK
<code>\$abc123</code>	OK
<code>\$123abc</code>	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	<code>42</code>	Real numbers
Floats	<code>3.1415</code>	Real numbers
Strings	<code>'Hello world'</code>	Strings can be any number or letter in a sequence (enclosed by single <code>'</code> or double <code>"</code> quotes, otherwise php may interpret it as a directive...)
Boolean	<code>true</code> or <code>false</code>	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
```

[php-basics/concatenate](#) | [src](#)

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

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

output of php-basics/concatenate

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

```
<?php
```

[php-basics/variables-in-single-quotes](#) | [src](#)

```
$variable = 'Hello World';
```

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

The variable contains: \$variable!

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

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

```
<?php
```

[php-basics/variables-in-double-quotes](#) | [src](#)

```
$variable = 'Hello World';
```

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

The variable contains: Hello World!

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

Special character sequences

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

Sequence	Result
<code>\n</code>	New line
<code>\r</code>	New line (Windows)
<code>\t</code>	The literal -character
<code>\\$</code>	Literal <code>\$</code> (escaping prevents variable interpretation)
<code>\"</code>	Literal <code>"</code> (escaping prevents string termination).

Example:

`<?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";`

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
<code>-\$a</code>	Negation	Opposite of \$a.
<code>\$a + \$b</code>	Addition	Sum of \$a and \$b.
<code>\$a - \$b</code>	Subtraction	Difference of \$a and \$b.
<code>\$a * \$b</code>	Multiplication	Product of \$a and \$b.
<code>\$a / \$b</code>	Division	Quotient of \$a and \$b.
<code>\$a % \$b</code>	Modulus	Remainder of \$a divided by \$b.
<code>\$a ** \$b</code>	Exponentiation	Result of raising \$a to the \$b 'th power. Introduced in PHP 5.6.

Example:

```
<?php
```

[php-basics/arithmetic](#) | [src](#)

```
$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";
```

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

output of php-basics/arithmetic

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:

```
$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-basics/array-auto-increment](#) | [src](#)

```
$array = [1,2,3];  
  
print_r($array);  
  
$array[] = 'hello';  
  
print_r($array);
```

```

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

```

The keys can however be specified manually:

```
<?php
```

[php-basics/array-custom-keys](#) | [src](#)

```

$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:

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



```
<?php
```

[php-basics/print_r](#) | [src](#)

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

output of php-basics/print_r

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

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 enclosed in square brackets:

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

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

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

Example:

```
<?php
```

[php-basics/array-get-key](#) | [src](#)

```
$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";
```

```

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
```

[php-basics/array-update-value](#) | [src](#)

```

$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
```

[php-basics/array-append](#) | [src](#)

```
$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-basics/array-prepend](#) | [src](#)

```
$array = [1,2,3];
```

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

```
array_unshift( $array, 4 );
```

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

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-basics/array-shift](#) | [src](#)

```
$array = [1,2,3];
```

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

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

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

```
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
```

[php-basics/array-pop](#) | [src](#)

```
$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-basics/array-count](#) | [src](#)

```
$array = [ 1, 2, 3 ];
```

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

```
$array[] = "Add item";
```

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

```
array_shift( $array );
```

```
array_shift( $array );
```

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

```
3
4
2
```

output of [php-basics/array-count](#)

Special arrays

PHP has some special, reserved, arrays. These arrays are created and filled by PHP.

\$argv

This array holds all the arguments passed to a PHP-script from the command line.

```
php print_r_argv.php 'arg1' 'arg2' 123 --options
```

```
Could not open input file: print_r_argv.php
```

Info: Notice that the first argument in this array is always the name of the script!

\$_GET

The `$_GET` -array holds data sent to a webpage via a HTTP-get method.

This corresponds with URL parameters.

```
http://example.com/page.php?arg1=hello&arg2=world&end=!
```

```
Array
(
    [arg1] => hello
    [arg2] => world
    [end] => !
)
```

\$_POST

The `$_POST` -array holds data sent to a webpage via a HTTP-post method.

This is typically done via a form submission...

\$_SESSION

You can store inter-page data in the `$_SESSION` reserved array.

This inter-page data is typically:

- user info
- preferences

\$_FILE

When files are uploaded, PHP stores information about these files in this array.

Example:

```
Array
(
    [file] => Array
        (
            [name] => MyFile.jpg
            [type] => image/jpeg
            [tmp_name] => /tmp/php/php6hst32
            [error] => UPLOAD_ERR_OK
            [size] => 98174
        )
)
```

Conditionals

It can be very handy to execute a piece of code only when certain requirements 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 `{}`).

```
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.

```

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-elseif-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 evaluates 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.

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

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
```

[php-basics/type-juggling](#) | [src](#)

```
$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
<code>\$a and \$b</code>	And	<code>true</code> if both <code>\$a</code> and <code>\$b</code> are <code>true</code> .
<code>\$a or \$b</code>	Or	<code>true</code> if either <code>\$a</code> or <code>\$b</code> is <code>true</code> .
<code>\$a xor \$b</code>	Xor	<code>true</code> if either <code>\$a</code> or <code>\$b</code> is <code>true</code> , but not both.
<code>! \$a</code>	Not	<code>true</code> if <code>\$a</code> is not <code>true</code> .
<code>\$a && \$b</code>	And	<code>true</code> if both <code>\$a</code> and <code>\$b</code> are <code>true</code> .
<code>\$a \$b</code>	Or	<code>true</code> if either <code>\$a</code> or <code>\$b</code> is <code>true</code> .

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

[php-basics/loops-while](#) | [src](#)

```
$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:

```
$var = 1;
```

Add or subtract by 1:

```
$var++; // increment by 1
```

```
$var-- // decrement by 1
```

Add or subtract 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`

[php-basics/loops-for](#) | [src](#)

```
for( $counter = 0; $counter < 10; $counter++ ) {  
  
    echo "Loop iteration: $counter\n";  
}
```

output of php-basics/loops-for

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

$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";
}
```

[php-basics/loops-for-array](#) | [src](#)

```
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
```

output of [php-basics/loops-for-array](#)

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

[php-basics/loops-foreach](#) | [src](#)

```
$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";  
}
```

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`

Exercises

Exercise:

Create a script that:

- receives a number from the command line
- counts from zero to this number
- counts back from this number to zero
- counts from zero to the number in steps of three

php count-to-number.php 9


Count up from 0 to 9:

0
1
2
3
4
5
6
7
8
9

Count down from 9 to 0:

9

Exercise:


Create a script that prints a line of a asterisks  defined by a command line parameter.

```
php print-asterisks.php 9
```

```
*****
```

Exercise:

Create a script that

- prints a square of a asterisks  if one parameter is defined
- Prints a block with width and height if both parameters are defined.

```
php print-square-of-asterisks.php 9
```

```
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****
```

```
php print-square-of-asterisks.php 15 5
```

```
*****  
*****  
*****  
*****  
*****
```

Exercise:

Create a script that prints a left + bottom balanced triangle of asterisks with base defined by parameter.

```
php print-left-bottom-balanced-triangle.php 9
```

```
*  
**  
***  
****  
*****  
*****  
*****  
*****
```


Exercise:

Create a script that prints a right + bottom balanced triangle of asterisks with base defined by parameter.

`php print-right-bottom-balanced-triangle.php 9`

```
*
**
***
****
*****
*****
*****
*****
*****
```

Exercise:

Create a script that prints a center + bottom balanced triangle of asterisks with base defined by parameter.

`php print-center-bottom-balanced-triangle.php 9`

```
  **
 ***
****
*****
*****
```

Exercise:

Create al the triangles again but the base (maximum number of asterisks) should be on top instead of at the bottom...

`php print-left-top-balanced-triangle.php 9`

```
*****
*****
*****
*****
****
***
**
*
```

php print-right-top-balanced-triangle.php 9

```
*****
*****
*****
*****
*****
****
***
**
*
```

php print-center-top-balanced-triangle.php 9

```
*****
*****
*****
***
*
```

Exercise:

Create a script that:

- reads a list of numbers from the command line
- prints the list
- prints the number of numbers (count)
- calculates/prints the min, max and average of the numbers
- prints the list backwards (bonus)
- prints the list sorted (bonus)

php number-statistics.php 9 12 3 5 4 1 8 5

```
The numbers received:
number 0: 9
number 1: 12
number 2: 3
number 3: 5
number 4: 4
number 5: 1
number 6: 8
number 7: 5
```

```
Smallest number: 1
Avg: 5.875
Largest number: 12
```

Exercise:

Create a script that generates the reverse complement of DNA string:

php dna-reverse-complement.php 'ATGCCGATAGGACTATGGACTATCTAGAGATCTATCAGAGAATAT
ATCCGGGATAATCGGATATCGGCGATAC'

```
orig.: ATGCCGATAGGACTATGGACTATCTAGAGATCTATCAGAGAATATATCCGGGATAATCGGATATCGGCGATAC
comp.: TACGGCTATCCTGATACCTGATAGATCTCTAGATAGTCTTATATAGGCCCTATTAGCCTATAGCCGCTATG
```

Bonus:

Print bonds:

php dna-reverse-complement-with-bonds.php 'ATGCCGATAGGACTATGGACTATCTAGAGATCTATCAGAGAATATATCCGGGATAATCGGATATCGGCGATAC'

```
orig.: ATGCCGATAGGACTATGGACTATCTAGAGATCTATCAGAGAATATATCCGGGATAATCGGATATCGGCGATAC
|||||
comp.: TACGGCTATCCTGATACCTGATAGATCTCTAGATAGTCTTATATAGGCCCTATTAGCCTATAGCCGCTATG
```

Info: The PHP functions: `str_split`, `amd`, `strlen` can be of use.

Exercise:

Create a script that generates the reverse complement of DNA string and can cope with:

- with Caps and non caps letters
- white space
- invalid nucleotides (and report these)

php dna-reverse-complement-robust.php 'ATgCXcGAtAgg ACTAtgGaCtA X TCtA g aGaTc TatCAgAgaatAtiXXATCcggaATAATcggAtATCggCGaTaC'

```
orig.: ATgCXcGAtAgg ACTAtgGaCtA X TCtA g aGaTc TatCAgAgaatAtiXXATCcggaATAATcggAtATCggCGaTaC
comp.: TACGGCTATCCTGATACCTGATAGATCTCTAGATAGTCTTATATAGGCCCTATTAGCCTATAGCCGCTATG
```

Invalid NT characters:

X: 4 occurrences

i: 1 occurrences

Exercise:

Create a script that prints the nucleotide frequency of a DNA strand.

php dna-frequency.php 'ATGCCGATAGGACTATGGACTATCTAGAGATCTATCAGAGAATATATCCGGGATAATCGGATATCGGCGATAC'

```
input: ATGCCGATAGGACTATGGACTATCTAGAGATCTATCAGAGAATATATCCGGGATAATCGGATATCGGCGATAC
```

STATS:

A: 24 nts -> 32.876712328767 %

T: 18 nts -> 24.657534246575 %

G: 18 nts -> 24.657534246575 %

C: 13 nts -> 17.808219178082 %

Exercise:

Create a script that prints the frequency of the characters in a string.

- sort by frequency: low to high + high to low
- sort by character (and reverse)
- case-insensitive (bonus)

php character-frequency.php 'Hello world, this is a random 123#\$ string.'

input: Hello world, this is a random 123#\$ string.

STATS:

'H': 1 occurrences -> 2.3255813953488 %
'e': 1 occurrences -> 2.3255813953488 %
'l': 3 occurrences -> 6.9767441860465 %
'o': 3 occurrences -> 6.9767441860465 %
' ': 7 occurrences -> 16.279069767442 %
'w': 1 occurrences -> 2.3255813953488 %
'r': 3 occurrences -> 6.9767441860465 %
'd': 2 occurrences -> 4.6511627906977 %
'i': 1 occurrences -> 2.3255813953488 %
't': 2 occurrences -> 4.6511627906977 %

Info: See: `sort`, `asort`, `ksort`, ... for different sort functions

PHP and HTML

PHP Webserver

PHP has a built in web-server. This means that no external server like Apache or Nginx is required to start a web-site and interlink the pages on this site.

Starting a server

The server is started with one command on the command line:

```
php -S localhost:<port> [-t /path/to/folder]
```

Example:

```
php -S localhost:8080
```

This previous command will start a web-server in the current working directory and will be accessible at the URL: `http://localhost:8080`.

You can pick any port, as long as it is between 1024 and 65535. By convention `8000` or `8080` are picked because of the resemblance with the official HTTP-port: `80`.

As mentioned before, by default the server will start in the current working directory. If you wish the root of the site to be another directory, specify it via the `-t` option.

Info: More info about this command can be found by executing the `man php` command on the command line.

By default the web-server will search and execute serve the `index.html` or `index.php` file in the servers root directory. (root directory = the directory where the server was started)

Making a simple page

```
mkdir my-website
cd my-website
echo "Hello world" > index.html
php -S localhost:8080
firefox localhost:8080
```

You should be greeted with `Hello world` ...

Because the web-pages are served via a PHP server, all PHP files (ending in `.php`) will be interpreted by the webserver. This allows us to generate the HTML content dynamically.

Exercises

Exercise:

Create a PHP page that prints `hello world` when served by a web-server

Exercise:

Create a PHP page that resembles your CV.

Exercise:

Create a web-page that prints

- an ordered list of three your three favourite dishes (dynamically)
- a list of (three) hobby's

Exercise:

Create three web-pages that interlink to one another.

- Home
 - Title

- Name
- Age
- Hobby's
 - Title
 - list
- Favourite dishes
 - Title
 - list

Include / Require

PHP allows us to include one file into another. This is done via the `include` or `require`;

The difference between the two is that `require` will fail if the specified file can't be included where `include` will merely warn about the failed inclusion.

```
include('path/to/file.php');
```

```
require('path/to/another-file.php');
```

Exercise:

Create a web-page who includes another file.

Forms

Forms can be used to send data from the web-page to the server. This data can be read and processed via PHP.

A form is composed out of a form-tag and data tags.

Form tag

```
<form action="<action>" method="<method>">
  <!-- content -->
</form>
```

The form tag has two required attributes:

- action
- method

Action

This attribute specifies the page the data should be sent to.

To send the data back to the same page, specify: `#` or the URL of the current page.

```
<form action="#"></form>
<form action="http://server.com/script-handle-data.php"></form>
```

Method

The method defines how the data should be send to the server.

There are two main methods:

- GET:
- POST

GET

GET mains appending the data as URL parameters.

Say we want to send the username and the age of a user back to the server, the URL could look like this:

```
http://server.com/script-to-handle-data.php?username=johnd&age=21
```

This method has two gotchas:

1. The data is sent visible to the server. **Never use this method to send sensitive data** back to the server.
2. The number of characters allowed in an URL is limited. So large amounts of data can not be sent this way...

An advantage of this method is that the URL with the data attached can be bookmarked or shared.

POST

POST The post method comes is where **GET** falls short.

The data is sent in the body of the HTTP request and is this invisible and not limited by size.

This method is most often used to send data from forms back to the server

```
<form action="#" method="get"></form>
<form action="#" method="post"></form>
```

Data tags

Other special tags are used to present or request data from the user.

In order to send the data, contained in the elements, back to the server, the name attribute must be set on the elements. This name can than be used on the backend to retrieve the values entered by the user.

Input

```
<input type="text" value="" name="">
```

The input tag encompasses a lot of "data types". The **type** -attribute can be used to modify the behaviour of this tag.

Types:

- checkbox

- email
- file
- number
- password
- radio
- text

The value attributes holds the default value of the element. If not defined, the element will be empty.

There the `radio` and `checkbox` type don't take a value (only) a state, the `checked` can be replaces the value attribute.

```
<ul>
  <li><input type="checkbox" name="input-type-checkbox"></li>
  <li><input type="checkbox" checked name="input-type-checkbox"></li>
  <li><input type="file" value="Hello World" name="input-type-file"></li>
  <li><input type="email" value="hello.world@mail.com" name="input-type-email"></li>
  <li><input type="number" value="42" name="input-type-number"></li>
  <li><input type="password" value="hello world" name="input-type-password"></li>
  <li><input type="radio" name="input-type-radio"></li>
  <li><input type="radio" checked name="input-type-radio"></li>
  <li><input type="text" value="hello world" name="input-type-text"></li>
  <li><input type="submit" value="hello world" name="input-type-submit"></li>
</ul>
```

[php-and-html/input](#) | [src](#)

- ☐
- ☒
- Choose File No file selected
- hello.world@mail.com
- 42
-
- ☐
- ☒
- hello world
- hello world

output of php-and-html/input

Info: In order to send files to the server, the form attribute: `enctype` must be set to `multipart/form-data`

```
<form action="#" method="post" enctype="multipart/form-data"></form>
```

See later for more details on how to upload files...

Select

The select tag allows the user to choose options out of a predefined set:

```
<select name="name-sent-backend">
  <option value="1">Option 1</option>
  <option value="2">Option 2</option>
  <option value="3">Option 3</option>
</select>
```

As can be seen in the example, a select is composed out of multiple options.

The default behaviour is that only one option can be selected at once.

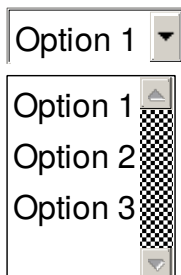
This can be altered via the `multiple`-attribute.

```
<select name="name-sent-backend">
  <option value="1">Option 1</option>
  <option value="2">Option 2</option>
  <option value="3">Option 3</option>
</select>

<br>

<select multiple name="name-sent-backend">
  <option value="1">Option 1</option>
  <option value="2">Option 2</option>
  <option value="3">Option 3</option>
</select>
```

[php-and-html/select](#) | [src](#)



output of [php-and-html/select](#)

Sub-group can be created via `optgroup`

```

<select name="name-sent-backend">
  <optgroup label="numbers">
    <option value="1">Option 1</option>
    <option value="2">Option 2</option>
    <option value="3">Option 3</option>
  </optgroup>
  <optgroup label="letters">
    <option value="a">Option a</option>
    <option value="b">Option b</option>
    <option value="c">Option c</option>
  </optgroup>
</select>

```

[php-and-html/select-groups](#) | [src](#)

```

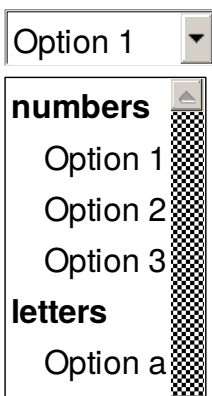
<br>

```

```

<select multiple name="name-sent-backend">
  <optgroup label="numbers">
    <option value="1">Option 1</option>
    <option value="2">Option 2</option>
    <option value="3">Option 3</option>
  </optgroup>
  <optgroup label="letters">
    <option value="a">Option a</option>
    <option value="b">Option b</option>
    <option value="c">Option c</option>
  </optgroup>
</select>

```



output of [php-and-html/select-groups](#)

Textarea

To send a block of text back to the server, use a `textarea`.

```

<textarea name="name-sent-to-server">
Write

Multiline content

Here
</textarea>

```

[php-and-html/textarea](#) | [src](#)

Write

output of php-and-html/textarea

Label

The label is not an input/data tag, but a meta-data tag.

This tag is used to add information about a data-tag. For example: label a password field as a password field.

A side benefit of using labels is that clicking a label, will automatically focus it's corresponding data element.

```
<label>
  Name:
  <input type="text" value="jonh Doe" name="name">
</label>

<br>

<label>
  Age:
  <input type="text" value="21" name="age">
</label>

<br>

<label>
  Loves ice-cream:
  <input type="checkbox" name="loves-ice-cream">
</label>
```

[php-and-html/label](#) | [src](#)

Name:

Age:

Loves ice-cream: ☐

output of php-and-html/label

PHP special data arrays

When data is sent to a PHP server, PHP will automatically populate the corresponding special array according/

Special arrays:

- `$_GET` : Holds al the data sent via the GET method.
- `$_POST` : Holds al the data sent via the POST method.
- `$_REQUEST` : Holds al the data sent via GET and POST combined.

- `$_FILES` : Holds all the info about the uploaded files.

Example:

```
<form action="#" method="post">
  <input type="text" value="Hello World" name="name">
  <input type="number" value="21 World" name="age">
  <input type="submit" value="Submit" name="submit">
</form>
```

```
print_r( $_POST );

/*
Array(
    [name] => "Hello World",
    [age] => 21
)
*/
```

This data can no be processed via PHP.

To test if data was submitted, the value of the *submit button* can be used.

In the previous example was the value: `submit` .

```
if( isset( $_POST['submit'] ) ) {

    /* Do stuff with data */

}
```

Exercises

Exercise:

Create a webpage with a login form:

- first name field
- last name field
- gender radio button
- age field
- email address
- password field
- "I want to receive updates" checkbox

Exercise:

Create a webpage that generates a triangle.

- The base of the triangle should be dynamically defined via a form submission or specified in the URL

- The character the triangle is composed of should be dynamically defined via a form submission or specified in the URL

Exercise:

Create a parrot. Everything you submit must be echo-ed back to the screen.

Exercise:

Create a login form with a

- name field
- age field

When the form is submitted the tool should tell if a person is older than 21 an is allowed to enter the website.

- if old enough, print: *access granted for: <name> .Age(<age>) is more than 21.*
- if if to you, print: *access denied for: <name> .Age(<age>) is less than 21.*

Exercise:

Create a tool that validates passwords.

- Password should be entered twice and be the same.
- should be more than 8 characters
- have at least one number and letter.

Exercise:

Create a tool that generates passwords.

- The number of characters should be defined by a form submission or in the URL.
- There should be the possibility (option) to include numbers in the generated password (via form or URL).
- You should be able to specify the number of passwords generated (via form or URL).

Exercise:

Make a web-page where you can paste and upload text and the tool should should:

- calculate and report the number of lines uploaded
- calculate and report the number of words uploaded

- calculate and report the number of characters uploaded
- Report the results in an table.

Exercise:

- header with links
- verify password are the same
- create array from plain text col 1 | col 2 | col 3 --- title: HTML and CSS ---

HTML and CSS

HTML can be styled via CSS. An HTML-element is selected via [CSS selectors](#). Styles/rules are defined per selector block. Each definition is terminated by a `;`. A rules block is enclosed by `{ , }`.

```
<selector> {  
  <property> : <value>;  
  <property> : <value> <value>;  
}
```

Include style information

Elements can be styled via three methods:

- style-attribute (discouraged)
- a style tag in the head (less discouraged)
- an external stylesheet (encouraged)

Style attribute

```
<element style="/* my styles */"></element>
```

Use this method to quickly test some rule. Not as a permanent style. **This way of managing styles is discouraged** because it is a less maintainable way of styling web-pages. For example: styles can not be shared by elements...

Style tag

The `style`-tag that should be defined in the head of the document. The styles defined in this tag apply to the complete page.

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>My pae title</title>

  <style type="text/css">
    .selector {

      property: value;
    }
  </style>

</head>
<body>
  <!-- the body -->
</body>
</html>

```

Even though the styles are defined only once, elements can share them via selectors (tag-name, classes, ...)

External style sheet

The CSS-rules can also be defined in a dedicated CSS-file. This file can be included in a web-page via the link-tag

The rules defined in the file can be included in as many HTML-pages as you want. This makes it the most scalable method of defining and including CSS-rules.

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>My pae title</title>

  <link rel="stylesheet" type="text/css" href="style.css">

</head>
<body>
  <!-- the body -->
</body>
</html>

```

CSS selectors

In order to apply rules to a certain element, the element must be targeted, selected.

CSS has the notion of selectors to target elements.

Tag-name

The tag-name can be used to style all the same tags the same way.

```
<style type="text/css">
  p {

    color: red; /* make text red */
  }
</style>

<p>
  P1: Hello world
</p>

<p>
  P2: Hello world
</p>
```

[html-and-css/tag-selector](#) | [src](#)

P1: Hello world

output of html-and-css/tag-selector

P2: Hello world

ID

The id attribute can be used to give an element an unique identifier. This id can be selected via CSS.

A pound symbol `#` indicates the following string is an id-name:

```
<style type="text/css">
  #idname {

    color: red; /* make text red */
  }
</style>

<p id="idname">
  P1: Hello world
</p>

<p>
  P2: Hello world
</p>
```

[html-and-css/id-selector](#) | [src](#)

P1: Hello world

output of html-and-css/id-selector

P2: Hello world

Class

Multiple elements can have the same class-name set. Elements with a certain class can be targeted/selected via this class-name.

Strings prefixed with a dot `.` are considered class-names in CSS.

```
<style type="text/css">
  .classname {
    color: red; /* make text red */
  }
</style>

<p class="classname">
  P1: Hello world
</p>

<p class="some-other-class">
  P2: Hello world
</p>

<div class="classname">
  P1: Hello world
</div>
```

[html-and-css/class-selector](#) | [src](#)

P1: Hello world

P2: Hello world

P1: Hello world

output of html-and-css/class-selector

Info: The elements sharing a class-name can be different tags.

Combining selectors

If multiple selectors are provided, separated by a comma, `,`, the defined rules will apply for all the elements matching any of the selectors:

```
<style type="text/css">
h1, h3, h5 {
  color: red;
}
</style>

<h1>H1</h1>
<h2>H2</h2>
<h3>H3</h3>
<h4>H4</h4>
<h5>H5</h5>
```

[html-and-css/multi-selectors](#) | [src](#)

H1

output of [html-and-css/multi-selectors](#)

H2

H3

H4

H5

CSS selector rules can also be composed out of multiple selectors. This allows for a more detailed/specific selection.

Elements matching multiple rules

```
selector1.selector2 {}
```

Selectors can be chained concatenated into a longer selection to make the selection more specific.

For example select only the `p`-tags with a certain class:

```
<style type="text/css">
  p.classname {
    color: red; /* make text red */
  }
</style>

<p class="classname">
  P1: Hello world
</p>

<p class="some-other-class">
  P2: Hello world
</p>

<div class="classname">
  P1: Hello world
</div>
```

[html-and-css/selector-chaining](#) | [src](#)

P1: Hello world

output of [html-and-css/selector-chaining](#)

P2: Hello world

P1: Hello world

In this case `p.classname` means:

- Select all elements with class `classname`
- From all the elements with class `classname` , select all `p` -tags.

You can make these selectors as long and as complex as you want:

`div.class1.class2.class3 ...`

Elements inside another element

```
selector1 selector2 {}
```

Multiple selectors separated by spaces indicate nesting. The last selector should be found inside the previous, inside the previous, ...

Info: Inside in this context means, the element must be wrapped by the other element:

```
<elem1>
  <elem2>
  </elem2>
</elem2>
```

Is doesn't matter how many other tags are in-between the parent and the nested element:

```
<elem1>
  <antoher-elem>
    <elem2>
    </elem2>
  </antoher-elem>
</elem2>
```

```
<style type="text/css">
```

[html-and-css/selector-nesting](#) | [src](#)

```
div h3 {
```

```
    color: red;
}
```

```
.green h3 {
```

```
    color: green;
}
```

```
</style>
```

```
<h3>H3 outside div</h3>
```

```
<div>
```

```
  <h3>H3 inside div</h3>
```

```
</div>
```

```
<div class="green">
```

```
  <blockquote class="sub-level">
```

```
    <h3>H3 inside div</h3>
```

```
  </blockquote>
```

```
</div>
```

H3 outside div

output of html-and-css/selector-nesting

H3 inside div

H3 inside div

Info: If two selectors target the same element and the same property, the last one encountered takes precedence

Direct children of an element

```
selector1 > selector2 {}
```

The **>** symbol between two selectors indicates an direct parent -> child relationship. By this we mean the second element must be an immediate child of the first selector. No other tags may warp the child element.

```
<parent>
```

```
  <child>
```

```
  </child>
```

```
</parent>
```

```
<style type="text/css">
```

[html-and-css/direct-child-selector](#) | [src](#)

```
div > h3 {
```

```
    color: red;
}
```

```
</style>
```

```
<h3>H3 outside div</h3>
```

```
<div>
```

```
  <h3>H3 direct child of div</h3>
```

```
</div>
```

```
<div >
```

```
  <blockquote class="sub-level">
```

```
    <h3>H3 not a direct child of div</h3>
```

```
  </blockquote>
```

```
</div>
```

H3 outside div


output of html-and-css/direct-child-selector

H3 direct child of div

H3 not a direct child of div

Siblings

```
selector1 ~ selector2 {}
```

The  symbol between two selectors, selects all the elements that match the second selector who exist after the first selector and have the same parent.

```
<style type="text/css">
```

[html-and-css/all-siblings-selector](#) | [src](#)

```
.selector1 ~ .selector2 {
```

```
    color: red;
}
```

```
</style>
```

```
<p class="parent">
```

```
  <div class="selector1">
```

```
    Content of div 1
```

```
  </div>
```

```
  <div class="selector2">
```

```
    Content of div 2
```

```
  </div>
```

```
  <div class="another-class">
```

```
    Content of div 3
```

```
  </div>
```

```
  <div class="selector2">
```

```
    Content of div 4
```

```
  </div>
```

```
</p>
```

Content of div 1

Content of div 2

Content of div 3

Content of div 4

output of [html-and-css/all-siblings-selector](#)

Direct siblings

```
selector1 + selector2 {}
```

The **+** symbol between two selectors, selects the first element that matches the second selector, is located directly after the first selector and have the same parent.

```
<style type="text/css">
```

[html-and-css/direct-siblings-selector](#) | [src](#)

```
.selector1 + .selector2 {
```

```
  color: red;
}
```

```
</style>
```

```
<p class="parent">
```

```
  <div class="selector1">
```

Content of div 1

```
  </div>
```

```
  <div class="selector2">
```

Content of div 2

```
  </div>
```

```
  <div class="another-class">
```

Content of div 3

```
  </div>
```

```
  <div class="selector2">
```

Content of div 4

```
  </div>
```

```
</p>
```

Content of div 1

Content of div 2

Content of div 3

Content of div 4

output of [html-and-css/direct-siblings-selector](#)

Special selectors

last-child

```
selector:last-child {}
```

This selection modifier targets the last element matching the given selector:

```
<style type="text/css">
```

[html-and-css/last-child-selector](#) | [src](#)

```
.parent div:last-child {
```

```
    color: red;
}
```

```
</style>
```

```
<div class="parent">
```

```
  <div class="selector1">
```

```
    Content of div 1
```

```
  </div>
```

```
  <div class="selector2">
```

```
    Content of div 2
```

```
  </div>
```

```
  <div class="another-class">
```

```
    Content of div 3
```

```
  </div>
```

```
  <div class="selector2">
```

```
    Content of div 4
```

```
  </div>
```

```
</div>
```

Content of div 1

Content of div 2

Content of div 3

Content of div 4

output of html-and-css/last-child-selector

In the example is the div coloured red, the last div in the `.parent` tag.

nth-child

```
select:nth-child( n ... ) {}
```

The `nth-child` modifier targets the elements matching a simple equation where n is the position of the element in the list.


```
<style type="text/css">
```

[html-and-css/nth-child-selector](#) | [src](#)

```
.parent div:nth-child( 2n + 1 ) {
```

```
    color: red;
}
```

```
.parent div:nth-child( 2n ) {
```

```
    color: green;
}
```

```
.parent div:nth-child( 3 ) {
```

```
    background: lightyellow;
}
```

```
</style>
```

```
<div class="parent">
```

```
  <div class="selector1">
```

```
    Content of div 1
```

```
  </div>
```

```
  <div class="selector2">
```

```
    Content of div 2
```

```
  </div>
```

```
  <div class="another-class">
```

```
    Content of div 3
```

```
  </div>
```

```
  <div class="selector2">
```

```
    Content of div 4
```

```
  </div>
```

```
</div>
```

Content of div 1

Content of div 2

Content of div 3

Content of div 4

output of [html-and-css/nth-child-selector](#)

- `2n + 1` : all odd elements (the `odd` keyword can also be used: `:nth-child(odd)`)
- `2n` : all even elements (the `even` keyword can also be used: `:nth-child(even)`)
- `3` : the third element

Colours

The appearance of text can be modified heavily via CSS

Colours can be defined in multiple ways:

- Colour name: the most common colours like red, blue, green, yellow,... can be specified via their names.
- RGB notation: `rgb(red-value, green-value, blue-value)` .
- HEX notation: `#RRGGBB` where `RR` is the hexa-deciaml notation of the red value, `GG` for green and `BB` for blue...

See [background](#) for an example.

Background

See [background](#) for detailed information about the background.

Set the background of an element.

```
div {  
  
    background: steelblue; /* named colour */  
    background: rgb( 70, 130, 180 ); /* Red green and blue value */  
    background: #4682B4; /* hexa decimal notation */  
}
```

Font and text.

In depth info about fonts can be found [here](#) and about text [here](#).

The appearance of text can be heavily modified:

Colour

Set the [color](#) of the text.

```
div {  
  
    color: steelblue;  
}
```

Font-family

Set a font.

Specify the most fonts from most desired font first to least desired (and more generic) font last.

```
div {  
    font-family: "Times New Roman", Times, serif;  
}
```

Font-size

This property determines the [font](#) -size of the text.

font-size: (xx-small | x-small | small | medium | large | x-large | xx-large)

font-size: (smaller | larger)

font-size: <value>

The size can be defined as a predefined word/value or as an action.

smaller goes a step down in the rank, for example: **small** -> **x-small** . **larger** goes a step down in the rank, for example: **large** -> **x-large** .

```
div {  
  
  font-size: 25px;  
  font-size: 1.2em;  
}
```

- **px** : set the height of the font to this amount o pixels;
- **em** : Set the size of the font relative the its parent: For example:
 - **1em** : no changes = 100%
 - **1.5em** : 150% the size of the parent
 - **0.9em** : 90% the size of the parent

Font-style

font-style: (normal | italic | oblique | inherit)

- **normal** : normal text.
- **italic** : slanted text (a custom designed slanted font set.)
- **oblique** : slanted text (the normal font slanted by software)
- **inherit** : inherit the value from the parent element

```
div {  
  
  font-style: italic;  
}
```

Font-variant

Visualise the font in all caps. Real capitals are still a bit larger than the normal letters.

```
div {  
  
  font-variant: small-caps;  
}
```

Font-weight

Make the font bold or reset the boldness.

font-weight: (normal | bold | inherit)

```
div {  
  
    font-weight: bold;  
}
```

Text-decoration

This property determines if text is:

- underlined
- line-through
- overlined

Text-align

Define how the text should be aligned inside an element:

Example

```
<div style="color:red;">Hello World</div>  
<div style="font-family: 'Times New Roman', Times, serif;">Hello World</div>  
<div style="font-size:large;">Hello World</div>  
<div style="font-size:small;">Hello World</div>  
<div style="font-size:10px;">Hello World</div>  
<div style="font-size:1.2em;">Hello World</div>  
<div style="font-style:italic;">Hello World</div>  
<div style="font-variant:small-caps;">Hello World</div>  
<div style="font-weight:bold;">Hello World</div>  
<div style="text-decoration:underline;">Hello World</div>  
<div style="text-decoration:line-through">Hello World</div>  
<div style="text-decoration:overline;">Hello World</div>
```

[html-and-css/font-and-text](#) | [src](#)

Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
HELLO WORLD
Hello World
Hello World
Hello World

output of [html-and-css/font-and-text](#)

Border

Draw a border around an element.

```
border: <style> <color> <width>;
```

Types:

- dotted
- dashed
- solid
- double
- groove
- ridge
- inset
- outset
- none
- hidden

```
<p style="border: solid red 1px"> Hello World </p>  
<p style="border: dashed green 2px"> Hello World </p>  
<p style="border: dotted blue 3px"> Hello World </p>
```

[html-and-css/border](#) | [src](#)

Hello World

output of [html-and-css/border](#)

Hello World

Hello World

In depth info about borders can be found [here](#)

Margin

Sometimes elements stick too close to each other. The margin specifies how far other elements should stay away from an element

The margin property accepts one, two or four values.

```
margin: 1px; /* set a margin of 1px for all four sides */  
margin: 1px 2px; /* set a margin of 1px for the top and bottom and 2px for the left and right side */  
margin: 1px 2px 3px 4px ; /* top: 1px, right side: 2px, bottom: 3px, left side: 4px */
```

```
<style type="text/css">  
  div { border: solid blue 1px;}  
</style>
```

[html-and-css/margin](#) | [src](#)

```
<div>Hello World</div>  
<div style="margin: 5px;">Hello World</div>  
<div>Hello World</div>  
<div style="margin: 5px 10px;">Hello World</div>  
<div>Hello World</div>  
<div style="margin: 5px 10px 15px 20px;">Hello World</div>  
<div>Hello World</div>
```

Hello World

output of html-and-css/margin

Hello World

Hello World

Hello World

Hello World

Hello World

Hello World

All sides can also be specified separately:

```
margin-top: 5px;  
margin-right: 5px;  
margin-bottom: 5px;  
margin-left: 5px;
```

In depth info about margins can be found [here](#)

Padding

The padding defines the distance the content inside the element should stay away from its border.

The padding property accepts one, two or four values.

```
padding: 1px; /* set a padding of 1px for all four sides */  
padding: 1px 2px; /* set a padding of 1px for the top and bottom and 2px for the left and right side */  
padding: 1px 2px 3px 4px ; /* top: 1px, right side: 2px, bottom: 3px, left side: 4px */
```

```
<style type="text/css">  
  div { border: solid blue 1px;}  
</style>
```

html-and-css/padding | src

```
<div>Hello World</div>  
<div style="padding: 5px;">Hello World</div>  
<div>Hello World</div>  
<div style="padding: 5px 10px;">Hello World</div>  
<div>Hello World</div>  
<div style="padding: 5px 10px 15px 20px;">Hello World</div>  
<div>Hello World</div>
```

| | |
|-------------|---|
| Hello World | <i>output of html-and-css/padding</i> |
| Hello World | |
| Hello World | |
| Hello World | |
| Hello World | |
| Hello World | |
| Hello World | |

All sides can also be specified separately:

```
padding-top: 5px;  
padding-right: 5px;  
padding-bottom: 5px;  
padding-left: 5px;
```

In depth info about paddings can be found [here](#)

Box-model

Exercises