

# Table of contents

## Introduction

- [Resources](#)
- [Contact](#)
- [Code in this document](#)

## Getting started

- [Get local copy of this site.](#)
- [Get local copy of the \*exercises and examples\* solutions](#)

## HTML Basics

- [Tags](#)
- [Attributes](#)
- [A valid HTML document](#)
- [The document head](#)
- [The document body](#)
- [Headers: H\\*](#)
- [Containers](#)
- [Inline tags](#)
- [Multi element markup](#)
- [Attributes](#)
- [Exercises](#)

## CSS

- [Include style information](#)
- [CSS selectors](#)
- [CSS properties](#)
- [Exercises](#)

## PHP Basics

- [What makes a PHP-script](#)
- [Types and variables](#)
- [Conditionals](#)
- [Loops](#)
- [Commonly used \(builtin\) functions](#)
- [Exercises](#)

## PHP and HTML

- [PHP Webserver](#)
- [Include / Require](#)
- [Forms](#)
- [PHP special data arrays](#)
- [Exercises](#)

## IO

- [Read a file](#)
- [Create/Write a file](#)
- [Remove a file](#)
- [Rename a file](#)
- [Directories](#)
- [Upload files](#)
- [Exercises](#)

## PHP functions

- [Return a value](#)
- [Exercises](#)

## Beyond this course

- [JavaScript](#)
- [CSS frameworks](#)
- [PHP frameworks](#)

---

This document in one large webpage...

# Introduction

The goal of this course is to provide an introduction to dynamic webpages ( [HTML](#), [CSS](#)) powered by [PHP](#).

During this course the student will learn how to create a static [HTML](#) webpage styled by [CSS](#).

Finally the webpages should become dynamic powered by the [PHP](#) programming language.

## Resources

The main documentation for this course is a site and can be found at <https://asoete.github.io/howest-webtechnology>. The main advantage of 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 will be lost in the conversion to a pdf.

There are no slides available, all the key aspects of this course will introduced via examples during the lessons. These examples and example solutions to the exercises, made during this course, will be published in a git repository on github:

<https://github.com/asoete/howest-webtechnology-code>.

This course is a very practical course. Making exercises and exploring/fiddling with code is the best way to learn and get acquainted 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 the contents of this course or get stuck during a certain exercise, please ask a question via de discussions on LEHO. This way fellow students can help you with your problem or questions.

I will also monitor the discussions page and post answers there.

If you have any remaining questions, please contact me via [arne.soete@howest.be](mailto: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:

```
1 <?php
2
3 echo "<h1>This is an embed example</h1>";
4
5 if( array_key_exists( 'KEY', $_POST ) ) {
6     unset($_POST[ 'KEY' ]);
7 }
8
9
10 ?>
11
12 <p>
13 Markup is interpreted by the browser and formatted accordingly..
14 </p>
```

[introduction/embed\\_example](#) | [src](#)

# 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:

```
1 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**.

This website source can be found at <https://github.com/asoete/howest-webtechnology> and the result viewed at <https://asoete.github.io/howest-webtechnology>

All the code created during the lessons will be made available at <https://github.com/asoete/howest-webtechnology-examples>.

## Init workspace

•

```
1 mkdir ~/webtechnology
2 cd ~/webtechnology
```

- Create your own exercises directory

```
1 mkdir exercises
```

You can store all your scripts in this folder...

## Get local copy of this site.

Although all documents are hosted online (<https://asoete.github.io/howest-webtechnology>) **it is recommended to host the cursus-site locally**.

Github doesn't allow the execution of [PHP](#) scripts, so the exercise solution may not work as they should because Github is preventing [PHP](#)-code execution...

The following steps must be taken to start/open the site locally:

- Get an initial copy of the repository:

```
1 cd ~/webtechnology
2 git clone https://github.com/asoete/howest-webtechnology.git cursus
```

- To get the latest version/updates

```
1 cd ~/webtechnology/cursus
2 git pull origin master
```

- Start a local instance of the site:

```
1 cd ~/webtechnology/cursus
2 make serve
```

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

## Get local copy of the *exercises and examples* solutions

- Get an initial copy of the repository located at <https://github.com/asoete/howest-webtechnology-examples>:

```
1 cd ~/webtechnology
2 git clone https://github.com/asoete/howest-webtechnology-examples.git solutions-and-examples
```

This command will create a `solutions-and-examples`-folder which will contain all the code featured during the lessons:

- \* Example snippet
- \* Exercise solutions

- To get the latest version (aka. update the repository) run:

```
1 cd ~/webtechnology/solutions-and-examples
2 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:

```
1 git fetch --all
2 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...**

## Final result

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

```
~/webtechnology
├── cursus
├── exercises
└── solutions-and-examples
```

## 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 which can contain attributes.

## Tags

An [HTML](#)-tag indicator starts with `<` and ends with `>`, 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 opening and closing tags can contain plain text and/or additional HTML markup.

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

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

The whole (start + content + end) is an HTML element.

## Self-enclosing tags

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

```
1 <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 piece of text into a clickable link.

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

The `href`-attribute defines where the link should point to:

```
1 <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:

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <!-- Your webpages metadata -->
5 </head>
6 <body>
7   <!-- your webpage specific content -->
8 </body>
9 </html>
```

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

## The document head

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

```
1 <head>
2   <!-- meta tags here -->
3 </head>
```

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

Everything defined within the `<head>` element will not be visible in the HTML document. The head content can have an effect on the appearance of the page but its content will not be visualised.

## Title

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

```
1 <head>
2   <title>My web-page's title...</title>
3 </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

```
1 <head>
2   <style type="text/css">
3     /* style information here */
4   </style>
5 </head>
```

### External style sheets

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

```
1 <head>
2   <link href="/link/to/file.css" type="text/css" rel="stylesheet">
3 </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.

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

### Exercise:

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

## HTML: Hello World

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

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

With contents:

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <title>Hello World</title>
5 </head>
6 <body>
7   Hello World from my first web-page.
8 </body>
9 </html>
```

Open the local HTML file in the browser.

```
1 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 `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.

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

[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

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

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


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

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

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

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...

## Pre-formatted text: **pre**

The **pre**  keeps all white-space in the element (in contrast to all the other elements). The text is also displayed in a monospaced font.

```
1 Content before...
2 <pre>
3   Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy
4   eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam
5   voluptua.
6
7   At vero eos et accusam et justo duo dolores et ea rebum. Stet clita
8   kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.
9 </pre>
10 Content after...
```

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



Content before...

output of `html-intro/pre-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.

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

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

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...

## Horizontal line: `hr`

The `hr` tag isn't really a container, it can't contain other elements, but it is a block element. The tag inserts a line into the document. This can be used to split / separate sections.

```
1 Content before...
2 <hr>
3 Content after...
```

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

Content before...

output of `html-intro/hr-tag`

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.

### 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.

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

[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.

```
1 This is the first line.
2 This is the second line, but in html all white space is replaced by a single
3 space...<br> The "br"-tag however instructs the browser to continue on a new
4 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?

### Inline preformat: `code`

The `code` tag is to inline elements what `pre` is to block elements. It will preserve white space and add a monospaced font in an inline way.

```
1 This is <code>preformatted text</code> inline...
```

[html-intro/code-tag / src](#)

This is preformatted text inline...

output of html-intro/code-tag

### Emphasise text: `em`

The `em` tag allows to emphasise certain text.

1 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.

1 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 `span` tag is a generic wrapper. It has no special semantic meaning. The `span` tag is mostly used to style/target some text.

1 This is `<span>span</span>` inline...

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

This is span inline...

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

## Strike text: `strike`

The `strike` tag is used to strike through some text.

1 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 `strong` tag can be used to make text bold.

1 This is `<strong>strong</strong>` inline...

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

This is **strong** inline...

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

## Inline quote text: `q`

The `q` tag is used to indicate certain inline text was quoted from an external source.

1 This is `<q>quote</q>` inline...

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

This is "quote" inline...

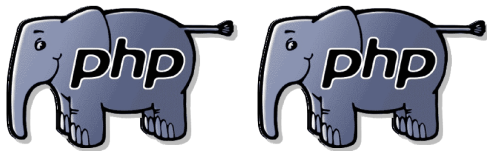
output of [html-intro/quote-tag](#)

## Image `img`

The `img` tag can be used to inert images into the markup. The `src`-attribute is required and specifies the location of the image. The location can be a local path, this means a path form the current script directory to the image location or a full (absolute) URL path.

```
1 
2
3 
```

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



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

### Exercise:

- Make a web-page with links to:
  - [google.com](https://www.google.com)
  - [howest.be](https://www.howest.be)
  - [GitHub.com](https://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 off.

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.

### Fieldset

`fieldset` is a container with an (optional) header ( `legend` ).

```
1 <fieldset>
2   <legend>This is a header / legend </legend>
3   This is the contents of the fieldset container.
4 </fieldset>
```

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

This is a header /  
legend  
This is the contents of the fieldset container.

[output of html-intro/fieldset](#)

This container is often used to group related from items together.

## Lists

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

### Unordered lists `ul`

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

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

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

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

### Ordered lists `ol`

```
1 <ol>
2   <li>list item 1</li>
3   <li>list item 2</li>
4   <li>list item 3</li>
5 </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`

```

1 <table>
2   <tr>
3     <th>Firstname</th>
4     <th>Lastname</th>
5     <th>Age</th>
6   </tr>
7   <tr>
8     <td>John</td>
9     <td>Doe</td>
10    <td>21</td>
11  </tr>
12  <tr>
13    <td>Jane</td>
14    <td>Doe</td>
15    <td>18</td>
16  </tr>
17
18 </table>

```

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

### Firstname Lastname Age

John	Doe	21
Jane	Doe	18

*output of [html-intro/simple-table](#)*

## 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.

## 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...

```

1 <p class="class1" >...</p>
2 <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.

```
1 <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.

```
1 <!-- http://www.example.com/script.php#chapter -->
2
3 <p id="chapter1">
4   Scroll into view...
5 </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`.

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

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

## Title

The title attribute lets you assign a title to an element. This title will be displayed as a tooltip when hovering with the mouse over this element.

```
1 <p title="Some title for this element">Hover over me...</p>
```

[html-intro/title-attribute](#) | [src](#)

Hover over me...

output of [html-intro/title-attribute](#)

## Exercises

### Exercise:

Create a `HTML` page resembling your CV.

## CV Arne Soete

### Personal Information

Name:	Arne Soete
E-mail address:	<a href="mailto:arnes.soete@howest.be">arnes.soete@howest.be</a>
Birth date:	1988-04-27
Gender:	male
Nationality:	Belgian

[Full Screen](#) | [Solution \(github\)](#)

## Exercise:

Add a table of contents (TOC) to the previous page (cv.html).

The TOC should contain all headers, clicking a header should take the user to this header/section

# CV Arne Soete

1. [Personal Information](#)
2. [Diploma's](#)
3. [Experience](#)
4. [Languages](#)
5. [Hobbies](#)

## Personal Information

[Full Screen](#) | [Solution \(github\)](#)

## Exercise:

Re-create the HTML skeleton of a wiki page: [https://en.wikipedia.org/wiki/FASTA\\_format](https://en.wikipedia.org/wiki/FASTA_format)

# FASTA format

In [bioinformatics](#), FASTA format is a text-based [format](#) for representing either [nucleotide sequences](#) or peptide sequences, in which nucleotides or [amino acids](#) are represented using single-letter codes. The format also allows for sequence names and comments to precede the sequences. The format originates from the [FASTA](#) software package, but has now become a standard in the field of bioinformatics.

[Full Screen](#) | [Solution \(github\)](#)

# 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>;  
}
```

[More info on the CSS syntax at w3schools](#)

## 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



```
1 <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.

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>My pae title</title>
6
7   <style type="text/css">
8     .selector {
9
10       property: value;
11     }
12   </style>
13
14 </head>
15 <body>
16   <!-- the body -->
17 </body>
18 </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.

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>My pae title</title>
6
7   <link rel="stylesheet" type="text/css" href="style.css">
8
9 </head>
10 <body>
11   <!-- the body -->
12 </body>
13 </html>
```

[More info on style inclusion](#)

## 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.

```

1 <style type="text/css">
2     p {
3
4         color: red; /* make text red */
5     }
6 </style>
7
8 <p>
9     P1: Hello world
10 </p>
11
12 <p>
13     P2: Hello world
14 </p>

```

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

P1: Hello world

P2: Hello world

*output of html-and-css/tag-selector*

## 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:

```

1 <style type="text/css">
2     #idname {
3
4         color: red; /* make text red */
5     }
6 </style>
7
8 <p id="idname">
9     P1: Hello world
10 </p>
11
12 <p>
13     P2: Hello world
14 </p>

```

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

P1: Hello world

P2: Hello world

*output of html-and-css/id-selector*

## 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](#).

```

1 <style type="text/css">
2   .classname {
3
4       color: red; /* make text red */
5   }
6 </style>
7
8 <p class="classname">
9   P1: Hello world
10 </p>
11
12 <p class="some-other-class">
13   P2: Hello world
14 </p>
15
16 <div class="classname">
17   P1: Hello world
18 </div>
19

```

[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.

[More info on basic selectors](#)

## 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:

```

1 <style type="text/css">
2   h1, h3, h5 {
3       color: red;
4   }
5 </style>
6
7 <h1>H1</h1>
8 <h2>H2</h2>
9 <h3>H3</h3>
10 <h4>H4</h4>
11 <h5>H5</h5>

```

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

H1

H2

H3

H4

H5

*output of html-and-css/multi-selectors*

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:

```
1 <style type="text/css">
2   p.classname {
3
4       color: red; /* make text red */
5   }
6 </style>
7
8 <p class="classname">
9   P1: Hello world
10 </p>
11
12 <p class="some-other-class">
13   P2: Hello world
14 </p>
15
16 <div class="classname">
17   P1: Hello world
18 </div>
19
```

[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:

```
1 <elem1>
2   <elem2>
3   </elem2>
4 </elem1>
```

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

```
1 <elem1>
2   <another-elem>
3     <elem2>
4     </elem2>
5   </another-elem>
6 </elem1>
```

```

1 <style type="text/css">
2
3   div h3 {
4
5       color: red;
6   }
7
8   .green h3 {
9
10      color: green;
11  }
12
13 </style>
14
15 <h3>H3 outside div</h3>
16
17 <div>
18   <h3>H3 inside div</h3>
19 </div>
20
21 <div class="green">
22   <blockquote class="sub-level">
23     <h3>H3 inside div</h3>
24   </blockquote>
25
26 </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 wrap the child element.

```

1 <parent>
2   <child>
3   </child>
4 </parent>

```

```

1 <style type="text/css">
2
3   div > h3 {
4
5       color: red;
6   }
7
8 </style>
9
10 <h3>H3 outside div</h3>
11
12 <div>
13   <h3>H3 direct child of div</h3>
14 </div>
15
16 <div >
17   <blockquote class="sub-level">
18     <h3>H3 not a direct child of div</h3>
19   </blockquote>
20 </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.

```

1 <style type="text/css">
2
3   .selector1 ~ .selector2 {
4
5       color: red;
6   }
7
8 </style>
9
10 <p class="parent">
11   <div class="selector1">
12     Content of div 1
13   </div>
14
15   <div class="selector2">
16     Content of div 2
17   </div>
18
19   <div class="another-class">
20     Content of div 3
21   </div>
22
23   <div class="selector2">
24     Content of div 4
25   </div>
26 </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.

```
1 <style type="text/css">
2
3   .selector1 + .selector2 {
4
5       color: red;
6   }
7
8 </style>
9
10 <p class="parent">
11   <div class="selector1">
12     Content of div 1
13   </div>
14
15   <div class="selector2">
16     Content of div 2
17   </div>
18
19   <div class="another-class">
20     Content of div 3
21   </div>
22
23   <div class="selector2">
24     Content of div 4
25   </div>
26 </p>
```

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

Content of div 1  
Content of div 2  
Content of div 3  
Content of div 4

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

[More info on selector combinators](#)

## Special selectors

### last-child

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

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

```
1 <style type="text/css">
2
3   .parent div:last-child {
4
5       color: red;
6   }
7
8 </style>
9
10 <div class="parent">
11   <div class="selector1">
12     Content of div 1
13   </div>
14
15   <div class="selector2">
16     Content of div 2
17   </div>
18
19   <div class="another-class">
20     Content of div 3
21   </div>
22
23   <div class="selector2">
24     Content of div 4
25   </div>
26 </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

```
selector: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.



```

1 <style type="text/css">
2
3   .parent div:nth-child( 2n + 1 ) {
4
5       color: red;
6   }
7
8   .parent div:nth-child( 2n ) {
9
10      color: green;
11  }
12
13  .parent div:nth-child( 3 ) {
14
15      background: lightyellow;
16  }
17
18 </style>
19
20 <div class="parent">
21   <div class="selector1">
22     Content of div 1
23   </div>
24
25   <div class="selector2">
26     Content of div 2
27   </div>
28
29   <div class="another-class">
30     Content of div 3
31   </div>
32
33   <div class="selector2">
34     Content of div 4
35   </div>
36 </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

## Hover

```

selector: hover {

    text-decoration: underline;
}

```

Via the **hover selector** allows us to apply styles to an element only when the mouse hovers over the element.

For example: overline a link when the mouse passes over the element:

```
1 <style type="text/css">
2   a: hover {
3
4       text-decoration: overline;
5       color: #888;
6   }
7 </style>
8
9 <a href="http://google.com"> Google </a>
10 <br>
11 <a href="http://http://www.w3schools.com">http://www.w3schools.com </a>
```

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

[Google](#)  
<http://www.w3schools.com>

*output of html-and-css/hover-selector*

## CSS properties

<http://www.w3schools.com> has a very good explanation of most of the [CSS](#) properties. The headers of the listed properties are links to the corresponding <http://www.w3schools.com>-website. These linked web-pages are considered part of the course material!

### Colors

Colors in [CSS](#) are most often specified by:

- a valid color name - like "red"
- an RGB value - like "rgb(255, 0, 0)"
- a HEX value - like "#ff0000"

### Background

An element can be given a background:

```
div {
    background: red;
}
```

### Border

An element can be given a border:

```
div {
    border: solid red 1px;
    border: dashed #bbb 1px;
}
```

### Margin

The margin defines how far away other, external, elements are pushed away from the border of the styled element.

```
div {
  margin: 15px; /* All sides the same margin */
  margin: 15px 30px; /* Top and bottom: 15 px, left and right side: 30px*/
  margin: 15px 30px 45px 60px; /* top: 15px, right:30px, bottom:45px, left: 60px */
}
```

## Padding

The padding defines how far away text, and other child, elements should stay away from the border of the element.

```
div {
  padding: 15px; /* All sides: 15px */
  padding: 15px 30px; /* Top and bottom: 15 px, left and right side: 30px*/
  padding: 15px 30px 45px 60px; /* top: 15px, right:30px, bottom:45px, left: 60px */
}
```

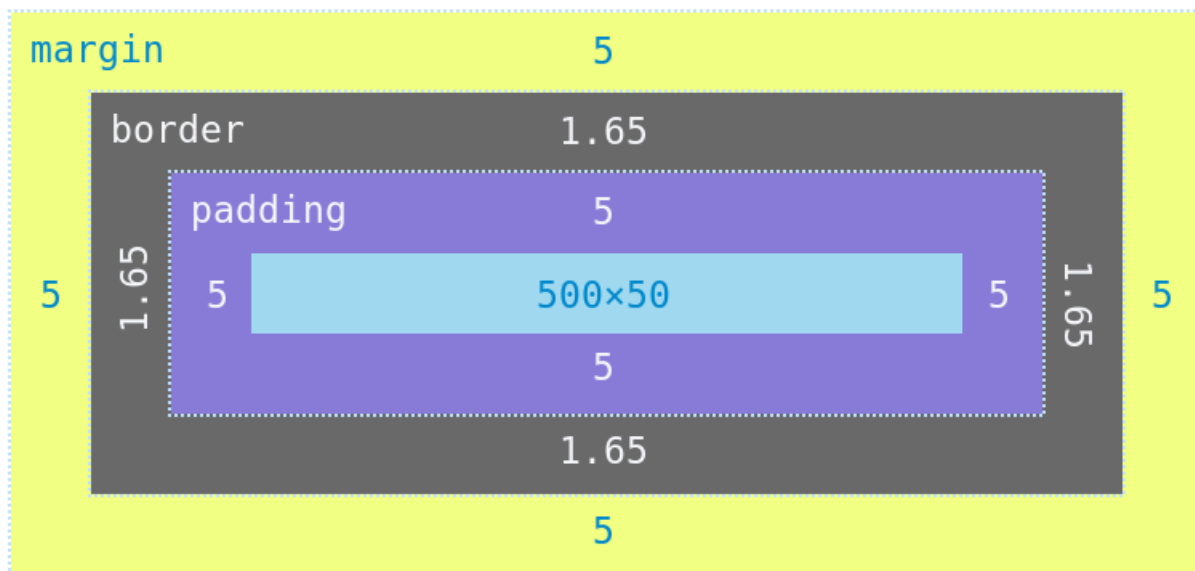
## Height and width

The width and the height of an element can be set via css.

```
div {
  height: 500px;
  width: 150px;
}
```

## Box model

An element is composed out of multiple components which all influence the size of the element:



- Content - The content of the box, where text and images appear
- Padding - Clears an area around the content. The padding is transparent
- Border - A border that goes around the padding and content
- Margin - Clears an area outside the border. The margin is transparent

The [box-sizing](#) property influences how all these components add up to the total size of the element.

- **content-box** : Default. The width and height properties (and min/max properties) includes only the content. Border, padding, or margin are not included
- **border-box** : The width and height properties (and min/max properties) includes content, padding and border, but not the

## Outline

Outline draws a border around the boxmodel.

```
div {  
    outline: solid green 1px;  
}
```

## Font

Customize the appearance of a font:

- font-color
- font-size
- font-weight ( how bold is the font)
- font-style (italic or not)
- font-family (Arial, serif or sans-serif, etc)
- font-variant (small-caps or not)

```
div{  
    font-size: 1.1em;  
    color: blue;  
    font-weight: bold;  
    font-style: italic;  
    font-family: serif;  
    font-variant: small-caps;  
}
```

## Text

We can style more than the appearance of the font, we can also define:

- text-align: left, right, centered or justified
- text-decoration: overline, underline or line-through
- word-spacing
- letter-spacing
- ...

```
div {  
    text-align: center;  
    text-decoration: underline;  
    word-spacing: 5px;  
    letter-spacing: 5px  
}
```

## Link

A link can be in one of four states:

- **a:link** - a normal, unvisited link
- **a:visited** - a link the user has visited
- **a:hover** - a link when the user mouses over it
- **a:active** - a link the moment it is clicked

All these states can be styled independently.

```
a:hover {  
  
    color: pink;  
}
```

## List [↗](#)

The *bullet* of a list ( `ul` or `ol` ) can also styles:

```
div {  
    list-style-type: square;  
    list-style-position: inside;  
    list-style-image: url("custom-bullet.gif");  
}
```

## Table [↗](#)

HTML tables can be styled heavily:

- `border`: style the borders of the table.
- `border-collapse`: merge table cell borders.

Margins, paddings, `nth-child` selectors, etc. can also be applied.

## Display and visibility [↗](#)

The `display` [↗](#)-property defines how an elements behaves:

- `block`: like a `div`, `p`, `pre`, ...
- `inline`: like `span`, `small`, `strong`, ...
- `inline-block` [↗](#): inline, but can be given a width/height, etc
- `hidden`: hide the element.

The `visibility` [↗](#)-property allows one to hide an element from view, but is still occupies space and interacts with the other DOM-elements.

```
div {  
  
    display: none;  
    visibility: hidden;  
}
```

## Position [↗](#)

This property defines how an element behaves in the page flow. There are four possible values:

- `static` : default -> go with the flow
- `relative` : position relative to default position.
- `fixed` : position relative to viewport (eg.: browser window)
- `absolute` : relative to the nearest positioned (= not `static` ) ancestor

```
div {  
  
    position: absolute;  
}
```

## Overflow [↗](#)

The `overflow` [↗](#)-property defines what should happen if the contents of an element is larger than the defined dimensions.

- `visible`: just show the content, don't take the boundaries into considiration.
- `hidden`: hide the overflowing content
- `scroll`: show scollbars

- auto: create scrollbars if needed.

**Info:** The X and Y axis scrollbars can be controlled via `overflow-x` and `overflow-y`

```
div {  
  
    overflow: hidden;  
}
```

## Float

Defining float to `left` or `right` will extract an element out of the normal page flow and *float* all the page content around this element.

```
div {  
  
    float: right;  
}
```

## Align

Elements can be aligned in their parent via three methods. These methods don't always work in all circumstances, so sometimes one must be chosen over the other.

- `text-align`: left, right or centered -> child elements are aligned
- `margin: 0 auto` -> no margin top and bottom, the sides is evenly divided == element is centered...
- `top`, `right`, `bottom`, `left`: specify distance of the side of an element to its parent. Works only on absolute or fixed positions

## Exercises

### Exercise:

Make a web-page with:

- Yellow background
- Red header: `Hello World`

**Hello World**

[Full Screen](#) | [Solution \(github\)](#)

### Exercise:

Make a table with three columns: `first name`, `last name`, `age` where:

- first column text is red
- second column text is blue
- third column text is green
- table caption is underline and overlined
- Extra: add total sum column

MY CLASSMATES

First name	Last name	Age
John	Doe	21
Jaine	Doe	24
Total		21

[Full Screen](#) | [Solution \(github\)](#)

## Exercise:

Make a table with three columns: `first name`, `last name`, `age` where:

- only columns have borders
- all even rows have a gray background
- extra: row where mouse is passing over: darker background

Classmates		
First name	Last name	age
John	Doe	21
Jain	Doe	24
Jake	Doe	18
Lisa	Peters	20
Kim	Jansens	26
Total		63

[Full Screen](#) | [Solution \(github\)](#)

## Exercise:

Create an web-page with:

- A header: `My webpage`
- An image
- Text flowing around the image.
- A copyright footer

# My web-page



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.

*Copyright: 2017*

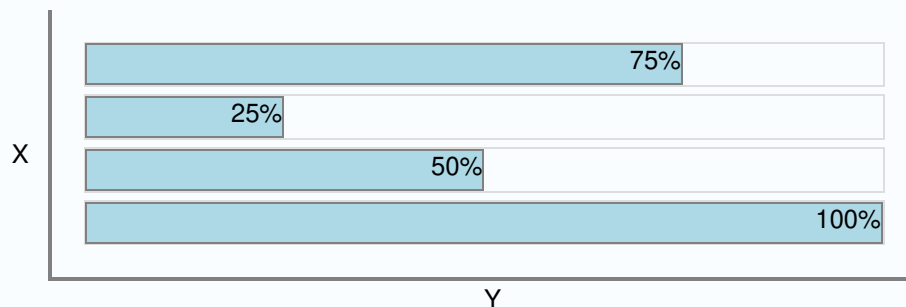
[Full Screen | Solution \(github\)](#)

## Exercise:

Create a horizontal bar-plot with four bars:

- Bar 1: 75%
- Bar 2: 25%
- Bar 3: 50%
- Bar 4: 100%

## Horizontale Barplot



[Full Screen | Solution \(github\)](#)

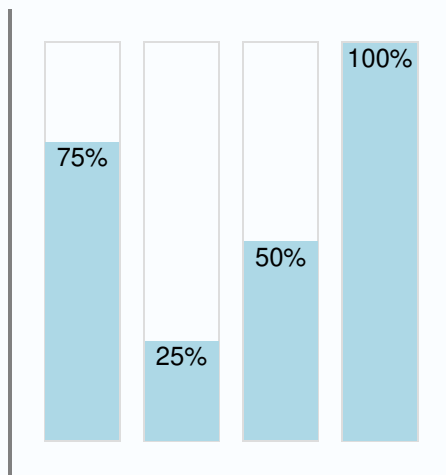
## Exercise:

Create a vertical bar-plot with four bars:

- Bar 1: 75%
- Bar 2: 25%
- Bar 3: 50%
- Bar 4: 100%



## Verticale Barplot



[Full Screen](#) | [Solution \(github\)](#)

### Exercise:

Re-create the HTML skeleton of a wiki page: [https://en.wikipedia.org/wiki/FASTA\\_format](https://en.wikipedia.org/wiki/FASTA_format)

Start from the HTML skeleton created during the [HTML exercises](#)

[Main page](#)  
[Contents](#)  
[Featured content](#)  
[Current events](#)  
[Random article](#)  
[Donate to Wikipedia](#)  
[Wikipedia store](#)

## FASTA format

In [bioinformatics](#), FASTA format is a text-based [format](#) for representing either [nucleotide sequences](#) or peptide sequences, in which nucleotides or [amino acids](#) are represented using single-letter codes. The format also allows for sequence names and comments to precede the sequences. The format originates from the [FASTA](#) software package, but has now become a standard in the field of bioinformatics.

The simplicity of FASTA format makes it easy to manipulate and parse sequences using text-processing tools and [scripting languages](#) like [R](#), [Python](#), [Ruby](#), and [Perl](#).

**Contents**

- [History](#)
- [Description Line](#)
  - [Sequence Representation](#)
- [Sequence Identifiers](#)
  - [Compression](#)

[Full Screen](#) | [Solution \(github\)](#)

### Exercise:

Create a webpage with information about the four DNA nucleotides:

- A
- T
- C
- G

Each of this pages should contain:

- a header
- an image of the molecule
- some info about the nucleotide

All of this can be fetched from wikipedia... <https://en.wikipedia.org/wiki/Adenosine>, <https://en.wikipedia.org/wiki/Thymidine>, <https://en.wikipedia.org/wiki/Cytosine>, <https://en.wikipedia.org/wiki/Guanine>.

- The text should flow around the image.
- Create a landing page. This page will welcome the user and link to the other pages.
- Display a navigation bar on top of each page.

[Adenosine](#)

[Thymidine](#)

[Guanine](#)

[Cytosine](#)

## The four basic DNA nucleotides

1. [Adenosine](#)
2. [Thymidine](#)
3. [Guanine](#)
4. [Cytosine](#)

Full Screen | Solution (github)

**Info:** In the next chapter we will see how to prevent code duplication. So don't duplicate code if there is another way ( Keep it DRY)

## 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](http://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: `?>`

```
1 echo "Before php-tags";
2
3 <?php
4
5 echo "Within php-tags\n";
6
7 ?>
8
9 echo "After php-tags";
```

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

```
echo "Before php-tags";

Within php-tags

echo "After php-tags";
```

*output of [php-basics/php-tags](#)*

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

A valid PHP instruction generally has the form:

```
1 {{ 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 // .

```
1
2 echo 'hello world';
3
4 // ignore this
5
6 # and ignore this
7
8 echo 'by world';
```

### Multi-line comments

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

```
1
2 echo 'hello world';
3
4 /* ignore this
5
6 and ignore this */
7
8 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:

```
1 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:

```
1 <?php
2
3 echo "Hello World";
4
5 ?>
```

[php-basics/hello-world | src](#)

```
Hello World
```

*output of php-basics/hello-world*

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

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

Run it via:

```
1 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.

```
1 $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:	
<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	42	Whole 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:

```
1 $varname = 'value to store';
```

Examples:

```
1 $int    = 123;
2 $float  = 4.56;
3 $string = 'Hello World';
4 $true   = true;
5 $false  = false;
```

## Printing/Displaying scalars

A scalar can be printed via two methods:

- `echo` 
- `print` 

### Echo

Generic syntax:

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

Echo outputs the provided scalars.

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

Example:

```
1 echo 123;
2 echo 4.56;
3 echo 'Hello World';
4 echo true;
5 echo false;
```

### Print

Generic syntax:

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

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

Example:

```
1 print( 123 );
2 print( 4.56 );
3 print( 'Hello World' );
4 print( true );
5 print( false );
```

# String concatenation and interpolation Or single vs. double quotes

Scalars can be combined, concatenated into larger strings.

The concatenation symbol is a dot: `.`.

1

<?php

2

3

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

php-basics/concatenate | src

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.

1

<?php

2

3

\$variable = 'Hello World';

4

5

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

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

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: `"`.

1

<?php

2

3

\$variable = 'Hello World';

4

5

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

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

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:

1

<?php

2

3

\$variable = "hello world";

4

5

echo "1. The value of the variable is: \$variable.";

6

echo "2. The value of the variable is: \$variable.\n";

7

echo "\t3. The value of the variable is: \$variable.\n";

8

echo "4. The value of the variable is: \\$variable.\n";

9

echo "5. The value of the variable is: \"\$variable\".\n";

php-basics/escape-sequences | src

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 and 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-basics/arithmetic | src](#)

```

1 <?php
2
3 $a = 42;
4 $b = 3.1415;
5 $c = 5;
6
7 echo $a + $b . "\n";
8 echo $a - $b . "\n";
9 echo $a * $b . "\n";
10 echo $a / $b . "\n";
11 echo $a % $c . "\n";
12 echo $a ** $c . "\n";

```

[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:

```

1 $array = array( /* list of array items */ );
2
3 $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

```
1 <?php
2
3 $array = [1,2,3];
4
5 print_r($array);
6
7 $array[] = 'hello';
8
9 print_r($array);
```

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

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

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

The keys can however be specified manually:

```
1 <?php
2
3 $array = [
4     'key1' => 'value1',
5     'two'   => 2,
6     3       => 'hello world',
7 ];
8
9 print_r($array);
```

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

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

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

## Print/Display arrays

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

Generic syntax:

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



```

1 <?php
2
3 $array = array(
4     'one' => 1,
5     'two' => 'three',
6     4     => 'four',
7     'hello' => 'world'
8 );
9
10 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 wish to retrieve enclosed in square brackets:

```
1 $array[<key>;
```

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

```
1 $array['<key>'];
```

Example:

```

1 <?php
2
3 $array = [1,2,3];
4
5 echo $array[0] . "\n";
6 echo $array[1] . "\n";
7 echo $array[2] . "\n";
8
9 $array_assoc = [
10     'key1' => "value1",
11     'key2' => "value2",
12     'key3' => "value3",
13 ];
14
15 echo $array_assoc['key1'] . "\n";
16 echo $array_assoc['key2'] . "\n";
17 echo $array_assoc['key3'] . "\n";
18

```

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

```

1
2
3
value1
value2
value3

```

*output of php-basics/array-get-key*

## 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:

```
1 $array[<key>] = <new value>;
```

Example:

```
1 <?php
2
3 $array = [
4     "value1",
5     "value2",
6     "value3",
7     100 => "hundred",
8     'key' => "value",
9 ];
10
11 print_r($array);
12
13 $array['key'] = "new value for key";
14
15 $array[1] = 'index 1 now points here';
16
17 print_r($array);
```

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

*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 at the end of an array can be accomplished by the function `array_push` or by the square brackets notation (`$array[] = $value;`).

```
1 <?php
2
3 $array = [1,2,3];
4
5 print_r( $array );
6
7 array_push( $array, 4);
8
9 print_r( $array );
10
11 // or
12
13 $array[] = 5;
14
15 print_r( $array );
```

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

```

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 in front of an array can be accomplished by the function `array_unshift` [🔗](#).

```

1 <?php
2
3 $array = [1,2,3];
4
5 print_r( $array );
6
7 array_unshift( $array, 4 );
8
9 print_r( $array );

```

```

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` [🔗](#).

```

1 <?php
2
3 $array = [1,2,3];
4
5 print_r( $array );
6
7 echo array_shift( $array ) . "\n";
8
9 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`.

```

1 <?php
2
3 $array = [1,2,3];
4
5 print_r( $array );
6
7 echo array_pop( $array ) . "\n";
8
9 print_r( $array );

```

```

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`.

```

1 <?php
2
3 $array = [ 1, 2, 3 ];
4
5 echo count($array) . "\n";
6
7 $array[] = "Add item";
8
9 echo count($array) . "\n";
10
11 array_shift( $array );
12 array_shift( $array );
13
14 echo count($array) . "\n";

```

```
3
4
2
```

## 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
```

```
Array
(
    [0] => print_r-argv.php
    [1] => arg1
    [2] => arg2
    [3] => 123
    [4] => --options
)
```

[Solution \(github\)](#)

**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.

```
1 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

### \$\_FILES

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 **{ }**).

```
1  if( /* <condition> */ ) {
2
3      /* execute this code here */
4  }
```

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

```
1  if( /* condition */ ) {
2
3      /* execute when condition is true */
4  }
5  else {
6
7      /* execute when condition is false */
8  }
```

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

```
1  if( /* condition 1 */ ) {
2
3      /* execute when condition 1 is true */
4  }
5  elseif( /* condition 2 */ ) {
6
7      /* execute when condition 2 is true */
8  }
9  elseif( /* condition 3 */ ) {
10
11      /* execute when condition 3 is true */
12  }
13  else {
14
15      /* execute when conditions 1, 2 and 3 are false */
16  }
```

Conditionals can also be nested:

```

1  if( /* condition 1 */ ) {
2
3      if( /* condition 2 */ ) {
4
5          /* execute when condition 1 and 2 evaluate to true */
6      }
7      else {
8
9          /* execute when conditions 1 evalutes to true and condition 2 to false */
10     }
11 }
12 else {
13
14     /* execute when condition 1 evaluates to false*/
15 }

```

## Comparison operators

| Example                       | Name                     | Result   |
|-------------------------------|--------------------------|--|
| <code>\$a == \$b</code>       | Equal                    | <b>true</b> if <code>\$a</code> is equal to <code>\$b</code> after type juggling.                    |
| <code>\$a === \$b</code>      | Identical                | <b>true</b> if <code>\$a</code> is equal to <code>\$b</code> , and they are of the same type.        |
| <code>\$a != \$b</code>       | Not equal                | <b>true</b> if <code>\$a</code> is not equal to <code>\$b</code> after type juggling.                |
| <code>\$a &lt;&gt; \$b</code> | Not equal                | <b>true</b> if <code>\$a</code> is not equal to <code>\$b</code> after type juggling.                |
| <code>\$a !== \$b</code>      | Not identical            | <b>true</b> if <code>\$a</code> is not equal to <code>\$b</code> , or they are not of the same type. |
| <code>\$a &lt; \$b</code>     | Less than                | <b>true</b> if <code>\$a</code> is strictly less than <code>\$b</code> .                             |
| <code>\$a &gt; \$b</code>     | Greater than             | <b>true</b> if <code>\$a</code> is strictly greater than <code>\$b</code> .                          |
| <code>\$a &lt;= \$b</code>    | Less than or equal to    | <b>true</b> if <code>\$a</code> is less than or equal to <code>\$b</code> .                          |
| <code>\$a &gt;= \$b</code>    | Greater than or equal to | <b>true</b> 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:


```

1  <?php
2
3  $string = '1 as a string';
4
5  var_dump($string);
6
7  # $string to int = 1 the '+' triggers the type juggling
8  var_dump( $string + 0);
9
10 # -----
11
12 var_dump( '1' == 1, 1 == true, 'abc' == true );
13 var_dump( '1' === 1, 1 === true, 'abc' === true );

```

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

```
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 &amp;&amp; \$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:

```
1 <?php
2
3 if( true or false ) {
4     echo "`true or false` evaluated to TRUE\n";
5 }
6 else {
7     echo "`true or false` evaluated to FALSE\n";
8 }
9
10 echo "-----\n";
11
12 if( !true ) {
13     echo "`!true` evaluated to TRUE\n";
14 }
15 else {
16     echo "`!true` evaluated to FALSE\n";
17 }
18
19 echo "-----\n";
20
21 if( true and false ) {
22     echo "`true and false` evaluated to TRUE\n";
23 }
24 else {
25     echo "`true and false` evaluated to FALSE\n";
26 }
27
28 ?>
```



```
`true or false` evaluated to TRUE
-----
`!true` evaluated to FALSE
-----
`true and false` evaluated to FALSE
```

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

```
1 <?php
2
3 if( ( 1 and 0 ) and true ) {
4     echo "`( 1 and 0 ) and true` --> true\n";
5 }
6 else {
7     echo "`( 1 and 0 ) and true` --> false\n";
8 }
9
10 if( 1 and ( 0 and true ) ) {
11     echo "`1 and ( 0 and true )` --> true\n";
12 }
13 else {
14     echo "`1 and ( 0 and true )` --> false\n";
15 }
```

```
`( 1 and 0 ) and true` --> false
`1 and ( 0 and true )` --> false
```

## 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:

```
1 while( /* <condition> */ ) {
2
3     /* execute this block */
4 }
```

**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:

```
1 # This will run until interrupted by the user.
2
3 while(1) {
4
5     echo "Use `Ctrl`+`c` to abort this loop\n";
6 }
```

```

1 <?php
2
3 $iterations = 10;
4
5 while( $iterations > 0 ) {
6
7     echo "Countdown finished in $iterations iterations\n";
8     $iterations = $iterations - 1;
9 }

```

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 of this, and similar operations, are available:

```

1 $var = 1;
2
3 # Add or subtract by 1:
4 $var++; // increment by 1
5 $var-- // decrement by 1
6
7 # Add or subtract by n:
8 # $var += n;
9 # $var -= n;
10
11 $var += 3;
12 $var += 100;
13 $var -= 42;
14 $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 `false`. 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.

```

1 for( <init counter>; <condition>; <increment counter> ) {
2
3     /* execute this block */
4 }

```

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

```
1 <?php
2
3 for( $counter = 0; $counter < 10; $counter++ ) {
4
5     echo "Loop interation: $counter\n";
6 }
```

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

```
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
```

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

## 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:

```
1 <?php
2
3 $array = [
4     1,
5     2,
6     'three',
7     'value'
8 ];
9
10 print_r($array);
11
12 for( $i = 0; $i < count($array); $i++ ){
13
14     echo "\$array has value: '". $array[$i] . "' at index $i\n";
15 }
```

[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.

```
1 foreach( <array> as [<key-placeholder> =>] <value-placeholder> ) {  
2  
3     /* use key and value here*/  
4 }
```

**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...

```
1 <?php  
2  
3 $array = [ 1, 2, 'three', 'value' ];  
4  
5 print_r($array);  
6  
7 foreach( $array as $value ) {  
8  
9     echo "The obtained value is: `$value`\n";  
10 }  
11  
12 # ----- #  
13  
14 $array = [  
15     1 , 2, 3,  
16     'key1' => 'value1',  
17     100 => 'hello'  
18 ];  
19  
20 print_r($array);  
21  
22 foreach( $array as $key => $value ) {  
23  
24     echo "Key: `$key` has value: `$value`\n";  
25 }  
26
```

php-basics/loops-foreach | src


```

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`

```

## Commonly used (builtin) functions

### isset

The `isset`  function checks whether a PHP variable was created / assigned.

This function returns false if the tested variable isn't declared before or is equal to `null`

```

1 <?php
2
3 echo "test undeclared variable\n";
4 var_dump( isset($some_variable) );
5 echo "\n";
6
7 echo "test declared variable\n";
8 $some_variable = "hello world";
9 var_dump( isset($some_variable) );
10 echo "\n";
11
12 echo "test null variable\n";
13 $some_variable = null;
14 var_dump( isset($some_variable) );

```

php-basics/isset | src

```

test undeclared variable
bool(false)


test declared variable
bool(true)

test null variable
bool(false)

```

output of php-basics/isset

### empty

The `empty`  function checks whether a PHP variable has an empty value.

Examples of empty values: `null`, `""`, `0`, ...

```
1 <?php
2
3 echo "test undeclared variable\n";
4 var_dump( empty($some_variable) );
5 echo "\n";
6
7 echo "test declared variable\n";
8 $some_variable = "hello world";
9 var_dump( empty($some_variable) );
10 echo "\n";
11
12 echo "test empty variables\n";
13 $some_variable = "";
14 var_dump( empty($some_variable) );
15 $some_variable = 0;
16 var_dump( empty($some_variable) );
17 $some_variable = null;
18 var_dump( empty($some_variable) );
```

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


```
test undeclared variable
bool(true)

test declared variable
bool(false)

test empty variables
bool(true)
bool(true)
bool(true)
```

*output of php-basics/empty*

## strlen

The `strlen`  function returns the length (number of characters) in a string.


```
1 <?php
2
3 $string = "The length of this string is: ";
4 echo $string;
5 echo strlen( $string );
```

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

```
The length of this string is: 30
```

*output of php-basics/strlen*

## str\_split

The `str_split`  function parses a string into individual characters and returns an array where each item in the array corresponds to one character in the original string.


```
1 <?php
2
3 $string = "split into chars";
4
5 $array_of_chars = str_split( $string );
6
7 print_r( $array_of_chars );
```

[php-basics/str\\_split](#) | [src](#)

Array

```
(
    [0] => s
    [1] => p
    [2] => l
    [3] => i
    [4] => t
    [5] =>
    [6] => i
    [7] => n
    [8] => t
    [9] => o
    [10] =>
    [11] => c
    [12] => h
    [13] => a
    [14] => r
    [15] => s
)
```

## explode

The `explode`  function parses a string into individual pieces based on a delimiter and returns an array where each item in the array corresponds to a section in the original string separated by the delimiter.

php-basics/explode | src

```
1 <?php
2
3 $string = "This is the first section of a semicolon separated string;This is the second section;And this
4
5 $array_of_sections = explode(";", $string);
6
7 print_r( $array_of_sections );
```

output of php-basics/explode

Array

```
(
    [0] => This is the first section of a semicolon separated string
    [1] => This is the second section
    [2] => And this the third!
)
```

This function can be used to convert the lines in a file (retrieved via `file_get_contents` ) into an array of lines.

php-basics/explode-newline | src


```
1 <?php
2
3 $string = ">Random Sequence (250 nts sampled from ATGC)
4 GCCAATGGTATGTCAACTCAGTCTCCCCAACTCTCGTAACTGGTAGAAGG
5 GTGGAGCATGAATCATGCCATCCATCTCGTATTGGAATCCGTGCGCGCGG
6 AGCCGAATATTTAAGTATACACACCCGCAGCACGAGCATCGTACTTTGCC
7 TTTCCAGCATAATTATGCATGAGGTGATAGCGAGATGATTCGGGGTAATG
8 CCGTTGTACCGCGCTCGGTCTTTCCGACGACGTCCCAACGCCGACACCA";
9
10 $lines = explode("\n", $string);
11
12 print_r( $lines );
```

```

Array
(
    [0] => >Random Sequence (250 nts sampled from ATGC)
    [1] => GCCAATGGTATGTCAACTCAGTCTCCCAACTCTCGTAACTGGTAGAAGG
    [2] => GTGGAGCATGAATCATGCCATCCATCTCGTATTCGAATCCGTGCGCGCGG
    [3] => AGCCGAATATTTAAGTATACACACCCGACGACGAGCATCGTACTTTGCC
    [4] => TTTCCAGCATAATTATGCATGAGGTGATAGCGAGATGATTTCGGGGTAATG
    [5] => CCGTTGTACCCGGCCTCGGTCTTTCCGACGACGTCCCAACGCCGACACCA
)

```

## implode

The `implode`  function takes an array and *glue* as input. The items in the array will be *glued* together into a new string. Each of the sections in the new output string will be separated by the *glue*

php-basics/implode | src

```

1 <?php
2
3 $array_of_sections = [
4     "section 1",
5     "section 2",
6     "section 3",
7     "section 4",
8 ];
9
10 echo "Create a new string delimited by semicolons: \n";
11 echo implode(";", $array_of_sections);
12
13 echo "\n";
14 echo "Create a new string delimited by newlines: \n";
15 echo implode("\n", $array_of_sections);

```

output of php-basics/implode

```

Create a new string delimited by semicolons:
section 1;section 2;section 3;section 4
Create a new string delimited by newlines:
section 1
section 2
section 3
section 4

```

## preg\_match

*Regular expressions* are a very powerful way of detecting patterns in a string.

The complete depth and power of *regular expressions* are out of the scope of this course but we will use them to detect header lines in multifasta sequences.

php-basics/preg\_match | src

```

1 <?php
2
3 echo "We want to match a leading > symbol:\n";
4 //      Pattern Find pattern in this
5 var_dump( preg_match("/^>/", ">MultiFasta Sequence Header") );
6
7 var_dump( preg_match("/^>/", "Hello World") );

```

output of php-basics/preg\_match

```

We want to match a leading > symbol:
int(1)
int(0)

```



# 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
```

[Solution \(github\)](#)

## Exercise:

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

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

```
*****
```

[Solution \(github\)](#)

## Exercise:

Create a script that

- prints a square of 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
```

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

[Solution \(github\)](#)

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

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

[Solution \(github\)](#)

## 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
```

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

[Solution \(github\)](#)

## 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
```

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

[Solution \(github\)](#)

## 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
```

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

[Solution \(github\)](#)

## Exercise:

Create all 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
```

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

[Solution \(github\)](#)

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

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

[Solution \(github\)](#)

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

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

[Solution \(github\)](#)

## Exercise:

Create a script that counts the number of parameters provided

```
php count-argv.php 9 12 3 5 4 1 8 5
```

```
8 arguments where provided
```

[Solution \(github\)](#)

## 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
- Print how many times a number occurs in the list
- (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
```

[Solution \(github\)](#)

## Exercise:

Create a script which stores the keys of an array as values of another array. (Extract the keys from an array)

```
1 $array = array(
2     'position 1' => 'hello',
3     'position 2' => 'world',
4     3           => 'three',
5     'four'      => 4
6 );
```

php array-keys.php

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

[Solution \(github\)](#)

## Exercise:

- Create a script which reverses an array.
- Create a script which reverses an array and preserves the keys

```
1 $array = array(
2     'position 1' => 'hello',
3     'position 2' => 'world',
4     3           => 'three',
5     'four'      => 4
6 );
```

php array-reverse.php

```
Array
(
    [0] => 4
    [1] => three
    [2] => world
    [3] => hello
)
Array
(
    [four] => 4
    [3] => three
    [position 2] => world
    [position 1] => hello
)
```

[Solution \(github\)](#)

## Exercise:

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

php dna-reverse-complement.php 'ATGCCGATAGGACTATGGACTATCTAGAGATCTATCAGAGAATATATCCGGGATAATCGGATATCGGCGATAC

```
orig.: ATGCCGATAGGACTATGGACTATCTAGAGATCTATCAGAGAATATATCCGGGATAATCGGATATCGGCGATAC
comp.: TACGGCTATCCTGATACCTGATAGATCTCTAGATAGTCTCTTATATAGGCCCTATTAGCCTATAGCCGCTATG
```

## Solution (github)

### Bonus:

Print bonds:

```
php dna-reverse-complement-with-bonds.php 'ATGCCGATAGGACTATGGACTATCTAGAGATCTATCAGAGAATATATCCGGGATAATCGGAT
```

```

orig.: ATGCCGATAGGACTATGGACTATCTAGAGATCTATCAGAGAATATATCCGGGATAATCGGATATCGGCGTAC
|||||
comp.: TACGGCTATCCTGATACCTGATAGATCTCTAGATAGTCTCTTATATAGGCCCTATTAGCCTATAGCCGCTATG

```

### Solution (github)

**Info:** The PHP functions: `str_split` and `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 TcTtA g aGaTc TatCAgAgaatAtiXXATCggggA

orig.: ATGcXcGAtAgg ACTAtgGaCtA X TcTt g aGaTc TatCagAgaatAtiXXATCcgggAtAATcggAtATCggCGaTaC  
comp.: TACGGCTATCCTGATACCTGATAGATCTCTAGATAGTCTCTTATATAGGCCCTATTAGCCTATAGCCGCTATG

```
Invalid NT characters:
X: 4 occurrences
i: 1 occurrences
```

## Solution (github)

## Exercise:

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

Additional: Create a simple bar plot to visualise the percentages.

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 | % |

GRAPH:

A: =====

T: =====

G: =====

C: =====

## Solution (github)

## 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 %  
,': 1 occurrences -> 2.3255813953488 %  
't': 2 occurrences -> 4.6511627906977 %
```

[Solution \(github\)](#)

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:

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

Example:

```
1 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

```
1 mkdir my-website
2 cd my-website
3 echo "Hello world" > index.html
4 php -S localhost:8080
5 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

**Hello World**

[Full Screen](#) | [Solution \(github\)](#)

### Exercise:

Create a PHP page that resembles your CV.

## CV Arne Soete

### Personalia

Naam: Arne Soete  
E-mail  
adres: [arnes.soete@howest.be](mailto:arnes.soete@howest.be)  
Leeftijd: 28  
Geslacht: man  
Nationaliteit: Belg

[Full Screen](#) | [Solution \(github\)](#)

### Exercise:

Create a web-page that prints

- an ordered list of three your three favourite dishes (dynamically)
- a list of (three) hobbies

1. fries
  2. spaghetti
  3. pizza
- basketball
  - music
  - games

[Full Screen](#) | [Solution \(github\)](#)

## Exercise:

Create three web-pages that interlink to one another.

- Home
  - Title
  - Name
  - Age
- hobbies
  - Title
  - list
- Favourite dishes
  - Title
  - list

## Home

Hello, I'm arne and I'm 28 years old.

- [Home](#)
- [Hobbies](#)
- [Favourite dishes](#)

[Full Screen](#) | [Solution \(github\)](#)

## 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.

```
1 include('path/to/file.php');  
2  
3 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

```
1 <form action="<action>" method="<method>">
2   <!-- content -->
3 </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.

```
1 <form action="#"></form>
2 <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 in 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

```
1 <form action="#" method="get"></form>
2 <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

```
1 <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 attribute holds the default value of the element. If not defined, the element will be empty.

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

```
1 <ul>
2   <li><input type="checkbox" checked name="input-type-checkbox"></li>
3   <li><input type="checkbox" name="input-type-checkbox"></li>
4   <li><input type="email" value="hello.world@mail.com" name="input-type-email"></li>
5   <li><input type="file" value="Hello World" name="input-type-file"></li>
6   <li><input type="hidden" value="Don't show this input to the user" name="input-type-hidden"></li>
7
8   <li><input type="number" value="42" name="input-type-number"></li>
9   <li><input type="password" value="hello world" name="input-type-password"></li>
10  <li><input type="radio" checked name="input-type-radio"></li>
11  <li><input type="radio" name="input-type-radio"></li>
12  <li><input type="submit" value="hello world" name="input-type-submit"></li>
13  <li><input type="text" value="hello world" name="input-type-text"></li>
14 </ul>
15
16
```

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

- ☒
- ☐
- 
- Choose File No file selected
- 
- 
- 
- ☒
- ☐
- 
- 

*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`

```
1 <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:

```

1 <select name="name-sent-backend">
2   <option value="1">Option 1</option>
3   <option value="2">Option 2</option>
4   <option value="3">Option 3</option>
5 </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.

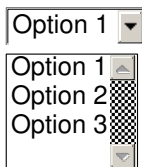
This can be altered via the `multiple`-attribute.

```

1 <select name="name-sent-backend">
2   <option value="1">Option 1</option>
3   <option value="2">Option 2</option>
4   <option value="3">Option 3</option>
5 </select>
6
7 <br>
8
9 <select multiple name="name-sent-backend">
10  <option value="1">Option 1</option>
11  <option value="2">Option 2</option>
12  <option value="3">Option 3</option>
13 </select>

```

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



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

Sub-group can be created via `optgroup`

```

1 <select name="name-sent-backend">
2   <optgroup label="numbers">
3     <option value="1">Option 1</option>
4     <option value="2">Option 2</option>
5     <option value="3">Option 3</option>
6   </optgroup>
7   <optgroup label="letters">
8     <option value="a">Option a</option>
9     <option value="b">Option b</option>
10    <option value="c">Option c</option>
11  </optgroup>
12 </select>
13
14 <br>
15
16 <select multiple name="name-sent-backend">
17   <optgroup label="numbers">
18     <option value="1">Option 1</option>
19     <option value="2">Option 2</option>
20     <option value="3">Option 3</option>
21   </optgroup>
22   <optgroup label="letters">
23     <option value="a">Option a</option>
24     <option value="b">Option b</option>
25     <option value="c">Option c</option>
26   </optgroup>
27 </select>

```

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

Option 1 ▾

**numbers**  
Option 1  
Option 2  
Option 3

**letters**  
Option a  
Option b  
Option c ▾

output of php-and-html/select-groups

## Textarea

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

```
1 <textarea name="name-sent-to-server">
2 Write
3
4 Multiline content
5
6 Here
7 </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.

```
1 <label>
2   Name :
3   <input type="text" value="jonh Doe" name="name">
4 </label>
5
6 <br>
7
8 <label>
9   Age:
10  <input type="text" value="21" name="age">
11 </label>
12
13 <br>
14
15 <label>
16   Loves ice-cream:
17   <input type="checkbox" name="loves-ice-cream">
18 </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:

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

```
1 print_r( $_POST );
2
3 /*
4 Array(
5     [name] => "Hello World",
6     [age] => 21
7 )
8 */
```

This data can now 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` .

```
1 if( isset( $_POST['submit'] ) ) {
2
3     /* Do stuff with data */
4 }
```

## 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

## Fill in the form to subscribe to the service

First name:

Last name:

Male: ☐ Female: ☐

E-mail address:

Enter/Choose a password:

[Open in new tab](#) | [Solution \(github\)](#)

## Exercise:

Print a rainbow pyramid, each column should have it's own color

- red
- orange
- yellow
- green
- blue
- indigo
- violet

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

[Full Screen](#) | [Solution \(github\)](#)

## Exercise:

Create a login form with a

- name field
- age field

Please validate if a user is older than 21.

Print an *access granted* or *denied* accordingly.

(Extra: Show error when a field is not filled...)

(Extra II: print how many years the user should wait before resubmitting the form...)

Name:

Age:

[Open in new tab](#) | [Solution \(github\)](#)

## Exercise:

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

Extra: append words to previous input...

Enter a word:

[Open in new tab](#) | [Solution \(github\)](#)

## Exercise:

Create a webpage that generates a triangle.

- The base of the triangle should be dynamically defined via a from submission or specified in the URL
- The character the triangle is composed of should be dynamically defined via a from submission or specified in the URL
- Extra: re-fill fields with previously entered values

Enter base size:   
Enter character:

[Open in new tab](#) | [Solution \(github\)](#)

## Exercise:

Create a tool that validates passwords.

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

Password 1 :

Password 2 :

[Open in new tab](#) | [Solution \(github\)](#)

## 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:

- 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.

Please paste text here:

[Open in new tab](#) | [Solution \(github\)](#)

## Exercise:

Make a webpage where you can upload comma separated data and convert it into a table.

- Extra: add option: make the first row headers
- Extra: throw error if number of fields is incorrect

```
first name, last name, gender, age
john, doe, male, 21
jane, doe, female, 18
jake, smith, male, 20
joan, d'arc, female, 33
```



Please paste CSV here:

☐ First line are headers.

submit

[Open in new tab](#) | [Solution \(github\)](#)

## Exercise:

Create a webtool which generates random sequences. The specifics of the sequence should be configurable:

Allow the user to specify:

- Sequence header (if not defined use: **Random sequence #1** )
- The alphabet the sequence should be composed of (if not defined use **ATGC** )
- The total number of nucleotides (default: **250** )
- The number of nucleotides per line (default: **50** )
- (Extra: Add the option (specify the number of sequences to generate) to generate multi-fasta e.g. generate multiple sequences: **Random sequence #1** , **Random sequence #2** , ...)

[Open in new tab](#) | [Solution \(github\)](#)

## Exercise:

Create a *post comment* form with fields:

- name
- email
- title options: (Dr, Mr, Ms, Ir)
- comment box
- post anonymous checkbox.

Validate:

- Name and/or email are not empty (unless *post anonymous* was checked)
- Comment has max 500 characters
- Print a red error messages if a validation failed + indicate which field failed validation, in red.

- If an error occurred pre-fill the elements with the valid data
- If no errors occurred,
  - ( Mr. or Ms. ) (name or *anonymous*) posted:
  - print *message posted* in gray.

Name:

Email:

Title:  ▼

Comment:

☐ Post anonymous



[Open in new tab | Solution \(github\)](#)

## IO

By IO we mean working with files...


## Read a file

There are multiple ways to read a file:

- `file` 
- `file_get_contents` 

are most important ones.

## File

The `file` -function takes a filename as argument and returns an array where each array item corresponds to a line in the file.

```
1 $lines = file('/path/to/file.txt');
```

## File\_get\_contents

This function takes a filename as argument and returns a string containing the complete file contents (hence the name).

```
1 $complete_contents = file_get_contents('path/to/file.txt');
```

## Create/Write a file

# Touch

`touch` [🔗](#) can be used to update the modification date of a file. If the file doesn't exist yet, it will be created.

```
1 touch('new-file.txt'); // create new-file.txt
2 touch('new-file.txt'); // update modification timestamp of new-file.txt
```

# Write

`fwrite` [🔗](#) can be used to write contents to a file handle.

A handle can be created via `fopen` [🔗](#).

```
1 $handle = fopen('file.txt', 'w'); // 'w' indicates open file for writing
2 fwrite($handle, 'Hello world'); // Write hello world to opened file
3 fclose($handle); // close file
```

This sequence of opening, writing and closing a file is quite cumbersome, therefore a shorthand is also available: `file_put_contents` [🔗](#).

```
1 $nr_bytes_written = file_put_contents('/path/to/file.txt', $file_contents );
```

# Remove a file

A file can be deleted via `unlink` [🔗](#).

```
1 $delete_ok = unlink('/path/to/file.txt');
```

# Rename a file

A file can be renamed via `rename` [🔗](#).

```
1 $rename_ok = rename('/path/to/file.txt', '/path/to-new-filename.txt');
```

# Directories

Similar functions exist for directories:

- `mkdir` [🔗](#): Create directory

```
mkdir('/path/to/dir');
```

- `rmdir` [🔗](#): Delete directory

```
rmdir('/path/to/dir');
```

- `rename` [🔗](#): Rename directory

```
rename('/path/to/dir', '/path/-to-new-dirname');
```

# List all files in a directory:


- `readdir` [🔗](#): Iterate over the files in a directory-handle

```
$handle = opendir('/path/to/dir');

while (false !== ($file = readdir($handle))) {

    echo "$file\n";

}
```

- **glob** : List all files in a directory (as array) matching a certain pattern.

```
$files = glob('/path/to/dir/*.php');
```

## Upload files

File can be uploaded via a form-submission. The tag used to specify files is:

```
1 <input type="file" name="uploaded-file">
```

**Info:** In order to use the `input[type=file]`, the form **must** specify the `enctype`-attribute:


```
1 <form action="#" method="post" enctype="multipart/form-data">
2
3     <input type="file" name="uploaded-file">
4
5     <input type="submit" value="submit" name="submit">
6 </form>
```

When a form containing files is submitted the `$_FILES` special PHP-variable is populated with the file(s) currently uploading:

```
$_FILES = Array
(
    [uploaded-file] => Array
        (
            [name] => MyFile.txt           // name of the file
            [type] => text/plain           // filetype
            [tmp_name] => /tmp/php/php1h4j1o // php stores the file in a randomized tmp location
            [error] => 0                   // 0 = UPLOAD_ERR_OK --> no errors
            [size] => 123                   // the size in bytes
        )

    [file2] => Array
        (
            [name] => MyFile.jpg
            [type] => image/jpeg
            [tmp_name] => /tmp/php/php6hst32
            [error] => UPLOAD_ERR_OK
            [size] => 98174
        )
)
```

PHP stores the uploaded file in a temporary location. We must use the files contents or move the file to a different location. At the next request this `tmp_name` will be something else. So keep this in mind.

If we opt not to read the file and process it, but move the file instead, the `move_uploaded_file`  should be used to move a file to a new destination.

This function has some additional checks builtin and is thus the recommended way to move uploaded files:

```
1 move_uploaded_file('/tmp/php/php1h4j10', '/home/me/some-file.txt')
```

## Exercises

## Exercise:

Create a web-page which allows to upload a file, and print the contents of the file.

Choose File No file selected

submit

[Open in new tab](#) | [Solution \(github\)](#)

## Exercise:

Create a web-page which allows to upload a file, and:

- report line-, word- and character-count in a table
  - nicely spaced cells
  - lines, words, characters as rows but bold (headers)
- list the top 10 most frequent words.
  - use a list but show all items aside each other (one line).

Choose File No file selected

submit

[Open in new tab](#) | [Solution \(github\)](#)

## Exercise:

Create a web-page which allows to upload or paste multi-fasta data and:

- report number of sequences
- print the sequences:
  - fasta header is bold
  - nucleotides are monospaced, 80 characters wide
- report GC-content per sequence

☒ Select fasta file:

Choose File No file selected

☐ Paste fasta

[Open in new tab](#) | [Solution \(github\)](#)

## Exercise:

Create a web-page which allows to upload or paste multi-fasta data and:

- print the sequences:
  - fasta header is header

- (inter)link to fasta blocks
- nucleotides are monospaced, 80 characters wide
- each nucleotide should have its own color:
  - A: red
  - T: yellow
  - G: green
  - C: blue
- Display the nucleotide frequencies in a bar graph. (A: xx%, T: xx%, ...)

☒ Select fasta file:

No file selected

☐ Paste fasta

[Open in new tab](#) | [Solution \(github\)](#)

## Exercise:

Create a web-page which allows to upload a file or paste some text, and specify a list of terms.

- Print the uploaded/pasted text
- Highlight all the specified terms in the uploaded/pasted text. (for example each term starts at a new line in a textarea)
- Extra: allow the user to specify the color per term. For example: `the|red a|green an`

Select file:

No file selected

Paste words to highlight

[Open in new tab](#) | [Solution \(github\)](#)

## Exercise:

Create a paste-bin web-app. The app allows the user to create a new snippet and save it (eq. write to file) via a webpage. The user can browse all snippets and view them by clicking a link...

This app is composed out of three pages:

1. index: list all the uploaded snippets
2. upload: a form:
  - snippet name
  - contents Allow the user to specify a new snippet to store. Make sure:
    - no empty snippets or snippets with no name can be uploaded.
    - snippet name doesn't already exists.
3. snippet contents: show the contents of a snippet

# Welcome to snippets

[+ Create new snippet](#)

## All snippets

- [Hello-world.txt](#)
- [Lorem-ipsum](#)

[Open in new tab](#) | [Solution \(github\)](#)

## PHP functions

Sometimes you use a piece of code over and over again. This is not optimal and this code should be extracted to its own function.

A function is a block of code with a given name and can be called (executed) by this name and optionally passed arguments to change the behaviour/output of the function.

We have already used a lot of PHP builtin functions. For example: `print`, `isset`, `empty`, `array_pop`, `array_push`, ...

As mentioned we can define our own functions.

Syntax:

```
1 function <name> ( <arguments> ) {  
2  
3     /* function code here */  
4 }
```

Example: function without arguments

```
1 function greet() {  
2     echo "Hello!\n";  
3 }  
4  
5 greet(); // Hello!  
6 greet(); // Hello!
```

Example: function with arguments

```
1 function greet( $name ) {  
2     echo "Hello $name!\n";  
3 }  
4  
5 greet( 'john' ); // Hello john!  
6 greet( 'jane' ); // Hello jane!  
7 greet(); // - ERROR -
```

If arguments are present in the function definition, these arguments must be passed when the function is invoked. Otherwise an error is thrown...

Arguments can however be defined with a default value, if the argument is not passed at invocation (or the argument value is `NULL`) the default value will be used instead.

Example: function with arguments

```

1 function greet( $name = 'anonymous' ) {
2     echo "Hello $name!\n";
3 }
4
5 greet( 'john' ); // Hello john!
6 greet( 'jane' ); // Hello jane!
7 greet(); // Hello anonymous

```

Multiple arguments can be passed, separated by comma's `,`.

```

1 function greet_both( $first, $second = '' ) {
2
3     echo "Hello $first";
4     echo "Hello $second";
5 }
6
7 greet_both( 'john', 'jane' );
8 // Hello john
9 // Hello jane
10
11 greet_both('sam');
12 // Hello sam
13 // Hello

```

## Return a value

The `return` keyword must be used to return a value from an array:

```

1 function fn() {
2     return "Hello World";
3 }
4
5 $string = fn();

```

Whenever a return statement is encountered, the function will stop and return the specified value. Any code defined after the return statement will not be executed...

Example:

```

1 function fn() {
2
3     if( true ) {
4
5         return "was true";
6     }
7
8     echo "This will never execute because the function returned from the if statement";
9 }

```

## Exercises

### Exercise:

Create a function which accepts two arguments:

- name
- sentence

Output: `<name> -> <sentence>`

php chat.php



```
john -> @jane: Hello
jane -> Hello john
john -> Nice weather...
jane -> Yep is is
```

[Solution \(github\)](#)

## Exercise:

Create a function which checks if a number is:

- positive
- even
- and smaller then 100

```
1 if( check_number( $nr ) ) { echo "number ($nr) passed tests"; }
2 else { echo "number ($nr) failed tests"; }
```

```
php validate-number.php -5
```

```
number (-5) failed tests
```

[Solution \(github\)](#)

```
php validate-number.php 7
```

```
number (7) failed tests
```

[Solution \(github\)](#)

```
php validate-number.php 22
```

```
number (22) passed tests
```

[Solution \(github\)](#)

```
php validate-number.php 11
1
```

```
number (111) failed tests
```

[Solution \(github\)](#)

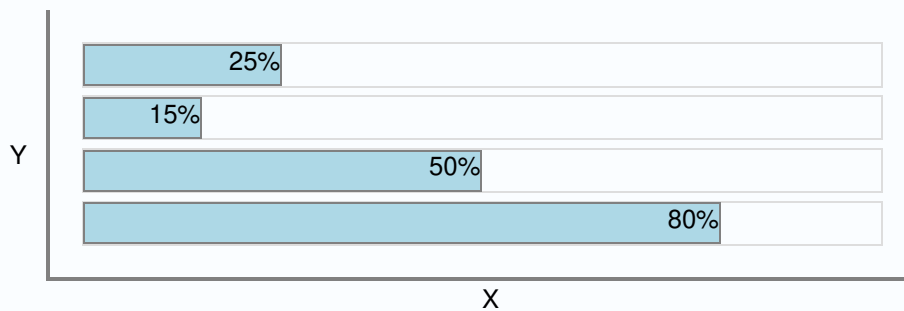
## Exercise:

Create a barplot: 25%, 80%, 15%.

Abstract the bars away in a function.

```
1 print_bar('25%');
2 print_bar('80%');
3 print_bar('15%');
```

## Horizontale Barplot



[Full Screen](#) | [Solution \(github\)](#)

## Exercise:

Create a function for each arithmetic operator: `+`, `-`, `*`, `/`. The function should accept an array of values and apply the operations in sequence:

Example:

```
1 plus([1,2,3, 4]); // -> 10
2 subtract([10,5,1]); // -> ( 10 - 5 ) - 1
3 divide([100, 10, 5]) // -> ( 100 / 10 ) / 5
4 multiply([7,9,8,4]) // -> 7 * 9 * 8 * 4
```

- Extra: accept two values: `plus( 1, 2 )` or array: `plus([1,2])`

Info: Use `is_array` to check if a variable is an array...

php arithmetic-operators.php

```
10
4
2
2016
```

[Solution \(github\)](#)

## Exercise:

Create a function which can retrieve and validate data from an array (ex.: `$_POST`)

- retrieve key from array
- check if value is not empty (otherwise show error)
- check if value is not longer than 50 characters (otherwise show error)
- modify the function so a default value can be passed to the function, if the key is not found in the array, return this value.

php retrieve-key.php

```
value1
default value
default value
value1
`keyx` not found in array
```

[Solution \(github\)](#)

## Exercise:

**Extra:** Create a function that can calculate the factorial of a number.

Try not to use loops but recursion...

```
php factorial.php 5
```

```
120
```

[Solution \(github\)](#)

## Beyond this course

As you can imagine, there is more to be learned about Web Technologies than we are able to cover during this course.

## JavaScript

Modern web applications make heavy use of JavaScript to obtain high user interactivity. This course skipped the user side interactivity via JavaScript because in order to create a JavaScript application you need to know about HTML and CSS for the frontend and a serverside language to create the backend. Although JavaScript (nodejs) is used more and more as the serverside backend, PHP is still the more broadly used language...

## CSS frameworks

It can become quite cumbersome to (re)create base styles from scratch. This is an issue many developers run into and for this purpose CSS frameworks were created. These frameworks ensure an identifiable style uniform across browsers. They also aid in responsive design etc...

Some popular choices:

### (Twitter) Bootstrap

Build fast, responsive sites with Bootstrap

Quickly design and customize responsive mobile-first sites with Bootstrap, the world's most popular front-end open source toolkit, featuring Sass variables and mixins, responsive grid system, extensive prebuilt components, and powerful JavaScript plugins.

### Tailwind CSS

Rapidly build modern websites without ever leaving your HTML.

A utility-first CSS framework packed with classes like flex, pt-4, text-center and rotate-90 that can be composed to build any design, directly in your markup.

### Bulma

Bulma: the modern CSS framework that just works.

Bulma is a free, open source framework that provides ready-to-use frontend components that you can easily combine to build responsive web interfaces.

# PHP frameworks

Many PHP applications also share common functionality: user authentication, receiving, sanitizing and processing data etc.

To prevent developers of reinventing the wheel over and over again, PHP frameworks where created.

## Laravel

The PHP Framework for Web Artisans

Laravel is a web application framework with expressive, elegant syntax. We've already laid the foundation — freeing you to create without sweating the small things.

## Symfony

Symfony is a set of reusable PHP components and a PHP framework for web projects

The standard foundation on which the best PHP applications are built. Choose any of the 50 stand-alone components available for your own applications.

Speed up the creation and maintenance of your PHP web applications. End repetitive coding tasks and enjoy the power of controlling your code.