# Table of contents

This document in one large webpage...

# Introduction

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

## Resources

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

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

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

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

### Additional resources

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

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

## Contact

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

## Code in this document

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

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

Example:

```php
<?php

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

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

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

?>

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

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

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

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

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

## Init workspace

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

- Create your own exercises directory

```
mkdir exercises
```

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

- Get an initial copy of the repository:

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

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

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

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

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

## Get local copy of this site. (Optional)

- Get an initial copy of the repository:

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

- To get the latest version/updates

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

- Start a local instance of the site:

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

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

# Final result

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

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

# HTML Basics

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

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

# Tags

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

There are two types of HTML-tags:

- Non self-enclosing tags
- Self-enclosing tags

# Non self-enclosing tags

Non self-enclosing tags exist out of two parts:

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

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

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

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

# Self-enclosing tags

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

```
<tag>
```

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

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

# Attributes

Attributes modify the behaviour of a tag.

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

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

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

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

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

# A valid HTML document

A valid HTML5 document requires a bit of boilerplate:

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

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

# The document head

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

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

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

# Title

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

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

# Style

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

# Raw style

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

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

## External style sheets

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

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

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

# The document body

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

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

### Exercise:

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

# HTML: Hello World

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

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

With contents:

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

Open the local HTML file in the browser.

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

# Headers: H*

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

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

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

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

*html-intro/headers | src*

*output of html-intro/headers*

# Header 1

## Header 2

### Header 3

#### Header 4

##### Header 5

###### Header 6

Exercise:

- Create a header for each `Hn` -tag

# Containers

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

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

They are also referred to as *block*-elements

# Paragraphs p

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

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

*html-intro/p-tag | src*

Content before...

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

Content after...

## Generic container div

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

It has no special styles by default

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

Content before...

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

## Pre-formatted text: pre

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

```
Content before...
<pre>
    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.
</pre>
Content after...
```

Content before...

    Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy
    eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam
    voluptua.

    At vero eos et accusam et justo duo dolores et ea rebum. Stet clita
    kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

# Blockquote `blockquote`

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

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

## Exercise:

- Create a block of text an 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.

```
<a href="http://google.com">Link to goole</a>
```
*html-intro/a-tag | src*

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

*output of html-intro/a-tag*

## Exercise:

- Create links to
    - google.com

- howest.be
- php.net
- github.com

# A newline: `br`

The `br` insert a newline into the document.

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

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

Cool right?

# Emphasise text: `em`

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

```
This is <em>emphasised</em> inline...
```
*html-intro/em-tag | src*

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

# Small text: `small`

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

```
This is <small>small</small> inline...
```
*html-intro/small-tag | src*

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

# Inline wrap text: `span`

The `a`

```
This is <span>span</span> inline...
```
*html-intro/span-tag | src*

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

# Strike text: `strike`

The `a`

This is **&lt;strike&gt;**strike**&lt;/strike&gt;** inline...

This is ~~strike~~ inline...

## Bold text: strong

The a

This is **&lt;strong&gt;**strong**&lt;/strong&gt;** inline...

This is **strong** inline...

## Inline quote text: q

The a

This is **&lt;q&gt;**quote**&lt;/q&gt;** inline...

This is "quote" inline...

## Exercise:

- Make a web-page with links to:
    - google.com
    - howest.be
    - GitHub.com
- Print the following text so the sentences are broken up as below.

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

- Print the following text so hello world is emphasised.

    > Let's emphasise hello world in this sentence.

- Print the following text so hello world is smaller

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

- Print the following text so hello world is bold

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

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

    > Let's strike hello world in this sentence.

# Multi element markup

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

## Lists

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

## Unordered lists `ul`

```
<ul>
  <li>list item 1</li>
  <li>list item 2</li>
  <li>list item 3</li>
</ul>
```
*html-intro/list-unordered | src*

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

*output of html-intro/list-unordered*

## Ordered lists `ol`

```
<ol>
  <li>list item 1</li>
  <li>list item 2</li>
  <li>list item 3</li>
</ol>
```
*html-intro/list-ordered | src*

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

*output of html-intro/list-ordered*

### Exercise:

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

## Tables

A simple table is composed out of:

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

```
<table>                                                    html-intro/simple-table | src
  <tr>
    <th>Firstname</th>
    <th>Lastname</th>
    <th>Age</th>
  </tr>
  <tr>
    <td>John</td>
    <td>Doe</td>
    <td>21</td>
  </tr>
  <tr>
    <td>Jane</td>
    <td>Doe</td>
    <td>18</td>
  </tr>

</table>
```

| **Firstname** | **Lastname** | **Age** | *output of html-intro/simple-table* |
|---|---|---|---|
| John | Doe | 21 | |
| Jane | Doe | 18 | |

## Exercise:

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

# Attributes

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

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

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

Commonly used attributes:

# Class

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

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

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

# Id

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

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

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

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

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

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

# Style

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

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

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

# See HTML and CSS for more info about styling an element.

# title: PHP basics

# PHP Basics

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

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

## What makes a PHP-script

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

## PHP tags

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

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

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

```php
<?php
echo "Within php-tags\n";
?>
echo "After php-tags";
```

> echo "Before php-tags";
>
> Within php-tags
>
> echo "After php-tags";

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

A valid PHP instruction generally has the form:

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

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

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

# Comments

## Single line comments

PHP will ignore everything behind a # or // .

```php
echo 'hello world';
// ignore this
# and ignore this
echo 'by world';
```

## Multi-line comments

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

```php
echo 'hello world';
/* ignore this
and ignore this */
echo 'by world';
```

# Execute a PHP-script

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

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

Basic invocation:

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

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

# Hello World

The obligatory hello world.

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

```
<?php                                                    php-basics/hello-world | src

echo "Hello World";

?>
```

> Hello World                                            output of php-basics/hello-world

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

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

Run it via:

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

on the command line.

# Types and variables

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

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

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

> **Warning:** It is not allowed to start variable name with a number:
>
> | | |
> |---|---|
> | **$abc** | OK |
> | **$abc123** | OK |
> | ~~$123abc~~ | Not allowed |

PHP knows two main types of variables:

- scalars
- arrays

# Scalars

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

## Types

PHP knows four scalar types:

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

## Declare

Assign a value to a variable:

*Generic syntax:*

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

Examples:

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

## Printing/Displaying scalars

A scalar can be printed via two methods:

- echo
- print

### Echo

*Generic syntax:*

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

Echo outputs the provided scalars.

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

Example:

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

## Print

*Generic syntax:*

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

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

Example:

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

# String concatenation and interpolation

Scalars can be combined, concatenated into larger strings.

The concatenation symbol is a dot: `.` .

```
<?php                                                    php-basics/concatenate | src

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

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

*output of php-basics/concatenate*

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

```
<?php                                          php-basics/variables-in-single-quotes | src

$variable = 'Hello World';

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

The variable contains: $variable!

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

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

```
<?php                                          php-basics/variables-in-double-quotes | src

$variable = 'Hello World';

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

The variable contains: Hello World!

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

## Special character sequences

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

| Sequence | Result |
|----------|--------|
| \n | New line |

| Sequence | Result |
|---|---|
| \r | New line (Windows) |
| \t | The literal -character |
| \$ | Literal $ (escaping prevents variable interpreation) |
| \" | Literal " (escaping prevents string termination). |

Example:

```php
<?php

$variable = "hello world";

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

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

## Basic arithmetic

Floats an Integers can be used in arithmetic.

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

Example:

```php
<?php                                          php-basics/arithmetic | src

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

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

```
                                               output of php-basics/arithmetic

45.1415
38.8585
131.943
13.369409517746
2
130691232
```

# Arrays

Arrays are able to hold more than one item.

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

## Declare

An array can be declared in two ways:

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

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

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

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

See below for how to print and add values to arrays

```php
<?php                                          php-basics/array-auto-increment | src

$array = [1,2,3];

print_r($array);

$array[] = 'hello';

print_r($array);
```

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

The keys can however be specified manually:

```php
<?php

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

print_r($array);
```

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

## Print/Display arrays

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

*Generic syntax:*

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

```php
<?php

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

print_r( $array );
```

```
Array
(
    [one] => 1
    [two] => three
    [4] => four
    [hello] => world
)
```
*output of php-basics/print_r*

## Get a value from an array

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

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

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

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

Example:

```php
<?php

$array = [1,2,3];

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

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

echo $array_assoc['key1'] . "\n";
echo $array_assoc['key2'] . "\n";
echo $array_assoc['key3'] . "\n";
```
*php-basics/array-get-key | src*

```
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:

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

Example:

```php
<?php                                               php-basics/array-update-value | src

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

print_r($array);

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

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

print_r($array);
```

```
                                               output of php-basics/array-update-value
Array
(
    [0] => value1
    [1] => value2
    [2] => value3
    [100] => hundred
    [key] => value
)
Array
(
    [0] => value1
    [1] => index 1 now points here
    [2] => value3
    [100] => hundred
    [key] => new value for key
)
```

## Manipulating arrays

## Add an item to the end of an array:

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

```php
<?php                                               php-basics/array-append | src

$array = [1,2,3];

print_r( $array );

array_push( $array, 4);

print_r( $array );

// or

$array[] = 5;

print_r( $array );
```

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

## Add an item in front of an array:

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

```php
<?php

$array = [1,2,3];

print_r( $array );

array_unshift( $array, 4 );

print_r( $array );
```

*php-basics/array-prepend | src*

*output of php-basics/array-prepend*

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

## Extract the first element from an array

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

```php
<?php

$array = [1,2,3];

print_r( $array );

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

print_r( $array );
```

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

## Extract the last element from an array

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

```php
<?php

$array = [1,2,3];

print_r( $array );

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

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 .

```php
<?php

$array = [ 1, 2, 3 ];

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

$array[] = "Add item";

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

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

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

```
3
4
2
```

# Special arrays

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

# $argv

This array holds al 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.

```
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 from submission...

# $_SESSION

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

This inter-page data is typically:

- user info
- preferences

# $_FILE

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

Example:

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

# Conditionals

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Conditionals can also be nested:

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

    if( /* condition 2 */ ) {

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

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

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

## Comparison operators

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

| Example | Name | Result |
|---------|------|--------|

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

For example:

```php
<?php

$string = '1 as a string';

var_dump($string);

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

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

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

*php-basics/type-juggling | src*

*output of php-basics/type-juggling*

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

**Info:** `var_dump` prints a variable with type information

# Logical operators

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

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

Example:

Ma_embed_php(php-basics/logical-operators)

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

# 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:

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

    /* execute this block */
}
```

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

**Danger:** The never ending loop:

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

while(1) {

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

```php
<?php

$iterations = 10;

while( $iterations > 0 ) {

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

*output of php-basics/loops-while*

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

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

```php
$var = 1;

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

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

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

## Exercise:

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

Only while loops are allowed.

# For

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

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

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

    /* execute this block */
}
```

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

```php
<?php                                                        php-basics/loops-for | src

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

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

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

## Exercise:

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

Only for loops are allowed.

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

```php
<?php

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

print_r($array);

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

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

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

## Exercise:

- Fill an array with: [ one, two, three, four, five ];
- Print each word on a single line.

- Modify the script to also print the index before the word: `$index: $value`

# Foreach

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

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

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

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

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

```php
<?php                                                    php-basics/loops-foreach | src

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

print_r($array);

foreach( $array as $value ) {

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

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

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

print_r($array);

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

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

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

# Exercises

## Exercise:

Create a script that:

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

**php** count-to-number.php 9

```
Count up from 0 to 9:
0
1
2
3
4
5
6
7
8
9
Count down from 9 to 0:
9
```

Solution (github)

## Exercise:

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

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

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

## Exercise:

Create a script that

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

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

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

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

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

## Exercise:

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

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

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

## Exercise:

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

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

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

# Exercise:

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

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

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

# Exercise:

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

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

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

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

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

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

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

## Exercise:

Create a script that:

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

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

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

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

## Exercise:

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

**php** dna-reverse-complement.php 'ATGCCGATAGGACTATGGACTATCTAGAGATCTATCAGAGAATATATCCGGGATA
ATCGGATATCGGCGATAC'

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

Bonus:

Print bonds:

**php** dna-reverse-complement-with-bonds.php 'ATGCCGATAGGACTATGGACTATCTAGAGATCTATCAGAGAATATATC
CGGGATAATCGGATATCGGCGATAC'

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

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

## Exercise:

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

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

**php** dna-reverse-complement-robust.php 'ATgCXCgAtAgg  ACTAtgGaCtA X  TCtA g aGaTc TatCAgAgaatAtiXXATCcgg gATAATcggAtATCggCGaTaC'

```
orig.: ATgCXCgAtAgg  ACTAtgGaCtA X  TCtA g aGaTc TatCAgAgaatAtiXXATCcgggATAATcggAtATCggCGaTaC
comp.: TACGGCTATCCTGATACCTGATAGATCTCTAGATAGTCTCTTATATAGGCCCTATTAGCCTATAGCCGCTATG

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

<div align="right">Solution (github)</div>

## Exercise:

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

**php** dna-frequency.php 'ATGCCGATAGGACTATGGACTATCTAGAGATCTATCAGAGAATATATCCGGGATAATCGGAT ATCGGCGATAC'

```
input: ATGCCGATAGGACTATGGACTATCTAGAGATCTATCAGAGAATATATCCGGGATAATCGGATATCGGCGATAC

STATS:
  A: 24 nts -> 32.876712328767 %
  T: 18 nts -> 24.657534246575 %
  G: 18 nts -> 24.657534246575 %
  C: 13 nts -> 17.808219178082 %
```

<div align="right">Solution (github)</div>

## Exercise:

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

- sort by frequency: low to hight + 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 occurences -> 2.3255813953488 %
'e': 1 occurences -> 2.3255813953488 %
'l': 3 occurences -> 6.9767441860465 %
'o': 3 occurences -> 6.9767441860465 %
' ': 7 occurences -> 16.279069767442 %
'w': 1 occurences -> 2.3255813953488 %
'r': 3 occurences -> 6.9767441860465 %
'd': 2 occurences -> 4.6511627906977 %
',': 1 occurences -> 2.3255813953488 %
't': 2 occurences -> 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:

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

Example:

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

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

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

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

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

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

## Making a simple page

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

You should be greeted with Hello world ...

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

# Exercises

## Exercise:

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

**Hello World**

## Exercise:

Create a PHP page that resembles your CV.

# CV

## Personalia

Naam:          Arne Soete

E-mail adres: arnes.soete@howest.be

Leeftijd:      28

Geslacht:      man

## 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

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](#)

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.

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

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

### Exercise:

Create a web-page who includes another file.

# Forms

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

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

## Form tag

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

The form tag has two required attributes:

- action
- method

## Action

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

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

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

## Method

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

There are two main methods:

- GET:
- POST

### GET

`GET` mains appending the data as URL parameters.

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

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

This method has two gotchas:

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

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

### POST

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

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

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

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

# Data tags

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

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

## Input

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

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

Types:

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

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

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

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

*output of php-and-html/input*

- ☐
- ☑
- Choose File  No file selected
- hello.world@mail.com
- 42
- ••••••••••
- ◯
- ◉
- hello world
- hello world

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

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

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

## Select

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

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

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

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

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

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

<br>

<select multiple name="name-sent-backend">
    <option value="1">Option 1</option>
    <option value="2">Option 2</option>
    <option value="3">Option 3</option>
</select>
```
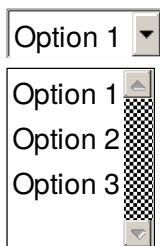
Sub-group can be created via `optgroup`

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

<br>

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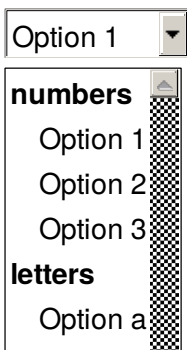
```html
      <option value="b">Option b</option>
      <option value="c">Option c</option>
   </optgroup>
</select>
```

| Option 1 ▼ |
|---|

**numbers**
   Option 1
   Option 2
   Option 3
**letters**
   Option a

## Textarea

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

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

Multiline content

Here
</textarea>
```

| Write |
|---|

## 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 lables is that clicking a label, will automatically focus it's corresponding data element.

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

<br>

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

<br>

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

Name: | jonh Doe |

Age: | 21 |

Loves ice-cream: ☐

# PHP special data arrays

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

Special arrays:

- $_GET : Holds al the data sent via the GET method.
- $_POST : Holds al the data sent via the POST method.
- $_REQUEST : Holds al the data sent via GET and POST combined.
- $_FILES : Holds all the info about the uploaded files.

Example:

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

```
print_r( $_POST );

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

This data can no be processed via PHP.

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

In the previous example was the value: submit .

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

    /* Do stuff with data */
}
```

# Exercises

## Exercise:

Create a webpage with a login form:

- first name field
- last name field
- gender radio button
- age field
- email address
- password field

- *"I want to receive updates"* checkbox

## Exercise:

Create a 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: print how many years the user should wait before resubmitting the form...)

Name: [                    ]

Age: [                  ]

[ submit ]

## Exercise:

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

Extra: append words to previous input...

Enter a word: [              ]  [ submit ]

## 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: 10

Enter character: *

submit

## 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 : 

Validate passwords

## Exercise:

Create a tool that generates passwords.

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

## Exercise:

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

- calculate and report the number of lines uploaded

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

Please paste text here:

[ text area ]

[ submit ]

## 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

```
firt 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:

[ text area ]

☐ First line are headers.

[ submit ]

# HTML and CSS

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

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

# Include style information

Elements can be styled via three methods:

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

# Style attribute

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

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

# Style tag

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

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

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

          property: value;

      }
    </style>

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

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

# External style sheet

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

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

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

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

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

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.

```
<style type="text/css">                             html-and-css/tag-selector | src
    p {

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

<p>
    P1: Hello world
</p>

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

P1: Hello world                                     output of html-and-css/tag-selector

P2: Hello world

# ID

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

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

```
<style type="text/css">                                    html-and-css/id-selector | src
    #idname {

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

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

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

P1: Hello world                                              output of html-and-css/id-selector

P2: Hello world

## Class

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

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

```
<style type="text/css">                                    html-and-css/class-selector | src
    .classname {

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

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

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

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

P1: Hello world                                              output of html-and-css/class-selector

P2: Hello world

P1: Hello world

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

# Combining selectors

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

```
<style type="text/css">               html-and-css/multi-selectors | src
h1, h3, h5 {
  color: red;
}
</style>

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

# H1

output of html-and-css/multi-selectors

# H2

### H3

### H4

###### H5

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

## Elements matching multiple rules

```
selector1.selector2 {}
```

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

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

```
<style type="text/css">               html-and-css/selector-chaining | src
  p.classname {

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

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

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

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

P1: Hello world

P2: Hello world

P1: Hello world

In this case `p.classname` means:

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

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

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

## Elements inside another element

```
selector1 selector2 {}
```

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

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

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

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

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

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

  div h3 {

    color: red;
  }

  .green h3 {

    color: green;
  }

</style>

<h3>H3 outside div</h3>

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

<div class="green">
  <blockquote class="sub-level">
    <h3>H3 inside div</h3>
  </blockquote>

</div>
```

## H3 outside div

## H3 inside div

## H3 inside div

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

## Direct children of an element

```
selector1 > selector2 {}
```

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

```
<parent>
  <child>
  </child>
</parent>
```

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

    div > h3 {

        color: red;
    }

</style>

<h3>H3 outside div</h3>

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

<div >
  <blockquote class="sub-level">
      <h3>H3 not a direct child of div</h3>
  </blockquote>
</div>
```

# H3 outside div

# H3 direct child of div

## H3 not a direct child of div

## Siblings

```
selector1 ~ selector2 {}
```

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

```
<style type="text/css">                                              html-and-css/all-siblings-selector | src

  .selector1 ~ .selector2 {

    color: red;
  }

</style>

<p class="parent">
  <div class="selector1">
    Content of div 1
  </div>

  <div class="selector2">
    Content of div 2
  </div>

  <div class="another-class">
    Content of div 3
  </div>

  <div class="selector2">
    Content of div 4
  </div>
</p>
```

Content of div 1                                                    *output of html-and-css/all-siblings-selector*
Content of div 2
Content of div 3
Content of div 4

## Direct siblings

```
selector1 + selector2 {}
```

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

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

  .selector1 + .selector2 {

    color: red;
  }

</style>

<p class="parent">
  <div class="selector1">
     Content of div 1
  </div>

  <div class="selector2">
     Content of div 2
  </div>

  <div class="another-class">
     Content of div 3
  </div>

  <div class="selector2">
     Content of div 4
  </div>
</p>
```

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

More info on selector combinators

# Special selectors

## last-child

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

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

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

  .parent div:last-child {

    color: red;
  }

</style>

<div class="parent">
  <div class="selector1">
    Content of div 1
  </div>

  <div class="selector2">
    Content of div 2
  </div>

  <div class="another-class">
    Content of div 3
  </div>

  <div class="selector2">
    Content of div 4
  </div>
</div>
```

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

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.

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

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

    color: red;
  }

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

    color: green;
  }

  .parent div:nth-child( 3 ) {

    background: lightyellow;
  }

</style>

<div class="parent">
  <div class="selector1">
    Content of div 1
  </div>

  <div class="selector2">
    Content of div 2
  </div>

  <div class="another-class">
    Content of div 3
  </div>

  <div class="selector2">
    Content of div 4
  </div>
</div>
```

Content of div 1

Content of div 2

Content of div 3

Content of div 4

*output of html-and-css/nth-child-selector*

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

## Hover

```
selector:hover {

  text-decoration: underline;
}
```

Via the (hover selector)[http://www.w3schools.com/cssref/sel_hover.asp]{.csstag} 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:

```
<style type="text/css">
  a:hover {

     text-decoration: overline;
     color: #888;
  }
</style>


<a href="http://google.com"> Google </a>
<br>
<a href="http://http://www.w3schools.com">http://www.w3schools.com </a>
```

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

# 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 {

   background: solid red 1px;
   background: 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:



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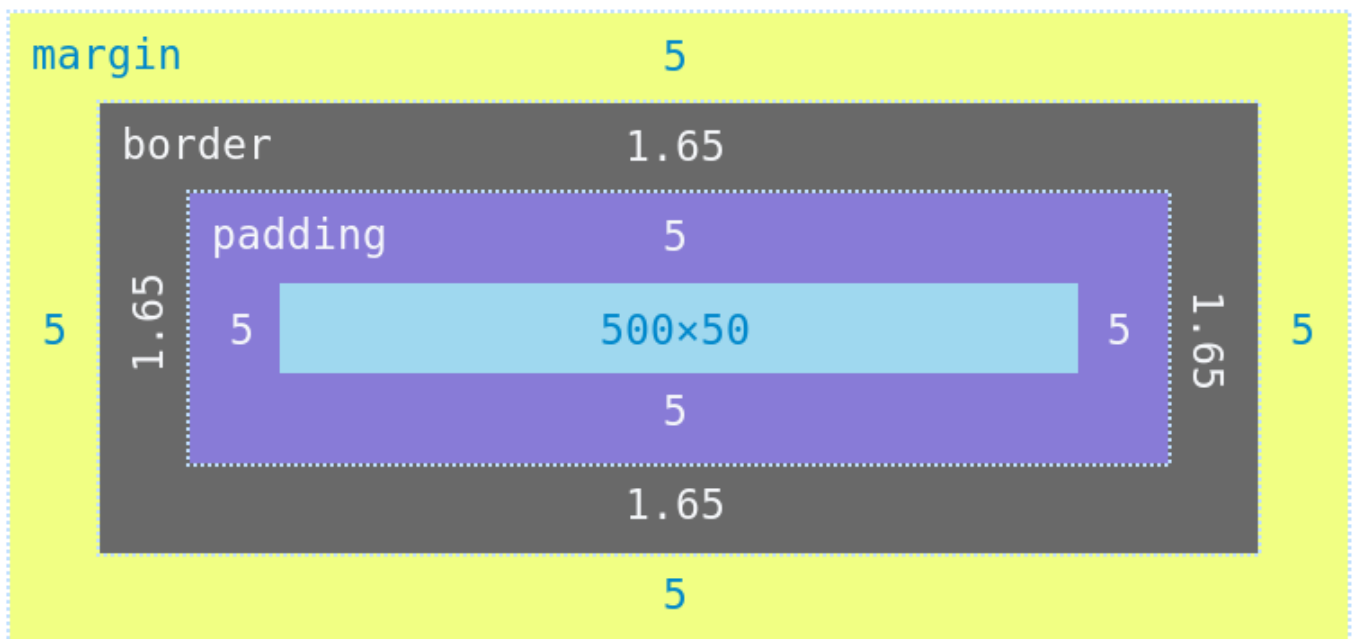
- Content - The content of the box, where text and images appear
- Padding - Clears an area around the content. The padding is transparent
- Border - A border that goes around the padding and content
- Margin - Clears an area outside the border. The margin is transparent

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 margin

## 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 of 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 styles 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: hidden;
    visibility: none;
}
```

# 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 : postion 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 boundries 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 choosen 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 devided == 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

## 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

M̲Y̲ ̲C̲L̲A̲S̲S̲M̲A̲T̲E̲S̲

| First name | Last name | Age |
|------------|-----------|-----|
| John | Doe | 21 |
| Jaine | Doe | 24 |
| **Total** | | **21** |

## 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 |

## Exercise:

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

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

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

## 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: 2016*

## Exercise:

Create a horizontal bar-plot with four bars:

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

### Horizontale Barplot
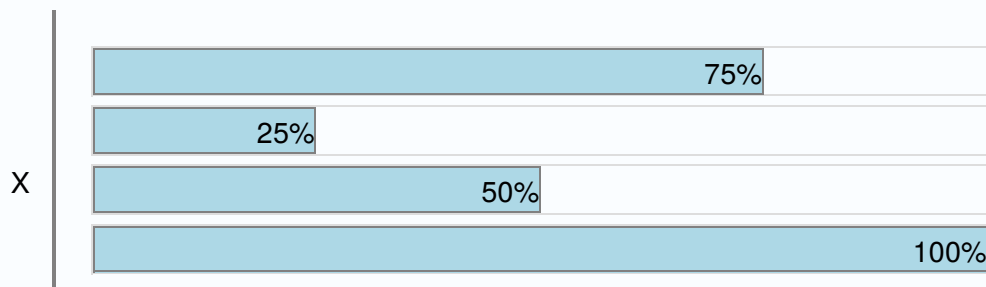
## Exercise:

Create a vertical bar-plot with four bars:

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

## Verticale Barplot



75%

25%

50%

100%
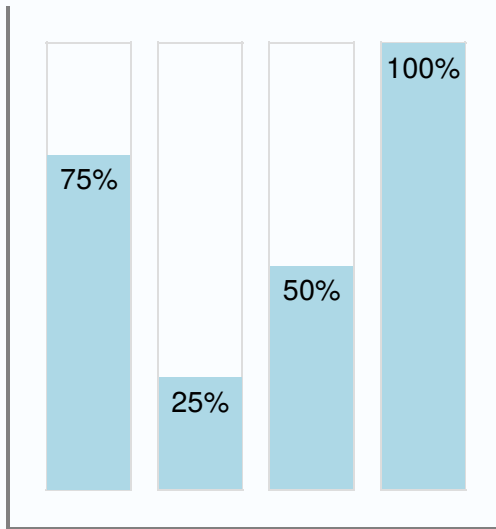
## Exercise:

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: [Dr. ▼]

Comment:
[                                        ]
[                                        ]
[                                        ]
[                                        ]
[                                        ]
[                                        ]

☐ Post anonymous

# IO

By IO we mean working with files...

# Read a file

There are multiple way 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.

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

# File_get_contents

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

```
$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.

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

## Write

`fwrite` can be used to write contents to a file handle.

A handle can be created via `fopen` .

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

This sequence of opening, writing and closing a file is quiet cumbersome, therefor a shorthand is also available: `file_put_contents` .

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

## Remove a file

A file can be deleted via `unlink` .

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

## Rename a file

A file can be deleted via `unlink` .

```
$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
```

```php
$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.

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

# Upload files

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

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

> **Info:** In order to use the `input[type=file]` , the form **must** specify the `enctype` -attribute:
>
> ```html
> <form action="#" method="post" enctype="multipart/form-data">
>
>     <input type="file" name="uploaded-file">
>
>     <input type="submit" value="submit" name="submit">
> </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:

```php
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

## Exercise:

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

- report line-, word-a 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

## 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

## 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%, ...)

## 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.

# 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:

```php
function <name> ( <arguments> ) {

    /* function code here */
}
```

Example: function without arguments

```php
function greet() {
    echo "Hello!\n";
}

greet(); // Hello!
greet(); // Hello!
```

Example: function with arguments

```php
function greet( $name ) {
    echo "Hello $name!\n";
}

greet( 'john' ); // Hello john!
greet( 'jane' ); // Hello jane!
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

```php
function greet( $name = 'anonymous' ) {
    echo "Hello $name!\n";
}


greet( 'john' ); // Hello john!
greet( 'jane' ); // Hello jane!
greet(); // Hello anonymous
```

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

```php
function greet_both( $first, $second = '' ) {

    echo "Hello $first";
    echo "Hello $second";
}


greet_both( 'john', 'jane' );
// Hello john
// Hello jane


greet_both('sam');
// Hello sam
// Hello
```

# Return a value

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

```php
function fn() {
    return "Hello World";
}


$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:

```php
function fn() {

    if( true ) {

        return "was true";
    }

    echo "This will never execute because the function returned from the if statement";
}
```

# 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
```

## Exercise:

Create a function which checks if a number is:

- positive
- even
- and smaller then 100

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

## Exercise:

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

Abstract the bars away in a function.

```
print_bar('25%');
print_bar('80%');
print_bar('15%');
```

## Exercise:

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

Example:

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

## 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, retrun this value.

## 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
```

# Git basics

Git is a version control system. It is designed to track and store revisions/versions of files.

This way a developer can save snapshots of the project and revert back to them if needed. Git also enables branching of some work, this means saving the changes in a separate store allowing the developer to test some feature and discard or merge the changes back into the main code depending on whether the experiment worked.

# Setup

To start tracking the contents of a directory, git must be initialised in this directory. This is done via the `git-init` command

```
# Setup directory:
# mkdir /tmp/git-playground
# cd /tmp/git-playground

git init
# Initialized empty Git repository in /tmp/git-playground/.git/
```

Git tells you is initialised the git data store in a hidden `.git` folder in the root of our project directory.

```
.git/
├── branches
├── config
├── description
├── HEAD
├── hooks
│   ├── applypatch-msg.sample
│   ├── commit-msg.sample
│   ├── post-update.sample
│   ├── pre-applypatch.sample
│   ├── pre-commit.sample
│   ├── prepare-commit-msg.sample
│   ├── pre-push.sample
│   ├── pre-rebase.sample
│   └── update.sample
├── info
│   └── exclude
├── objects
│   ├── info
│   └── pack
└── refs
    ├── heads
        └── tags
```

For the moment this is just a skeleton without any data e.g. we are not tracking any files. The status of our repository can always be obtained bu running the `git-status` -command:

```
git status
# On branch master
#
# Initial commit
#
# nothing to commit (create/copy files and use "git add" to track)
```

So we should start adding files to our git repository...

# Track files

Let us first create a file we want to track and run `git-status` again.

```
echo "Let's track this file..." > file.txt

git status
# On branch master
#
# Initial commit
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       file.txt
#
# nothing added to commit but untracked files present (use "git add" to track)
```

So git is telling us we are not tracking any files, but that an untracked file is present in our project.

To start tracking a file two actions must be performed:

- `git-add`
- `git-commit`

`git-add` will add the file to the staging area. This means group/tag the files so future commands can operate on them.

```
git add file.txt
git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#   new file:   file.txt
```

The status command tells us the file is added to the staging area and is ready for the next step: committing.

`git-commit` will take the files in the staging area and create a new snapshot of their current contents. This commit takes a commit message: a description of the changes to the project via the modifications of those staged files.

```
git commit # a text editor will open, type your message and save + quit the file

git commit -m "start tracking file.txt" # the -m options allows us to specify the commit message on the command line

# [master (root-commit) 13ce453] start tracking file.txt
#  1 file changed, 1 insertion(+)
#  create mode 100644 file.txt
```

If we run `git-status` again, we see that all our changes are saved in a snapshot. This is what git calls a *clean working directory*.

```
git status
# On branch master
# nothing to commit, working directory clean
```

# Git internals

So what actually happened when we added a file?

First of all lets have a look at the git store:

```
  tree .git
  .git
  ├── branches
  ├── COMMIT_EDITMSG
  ├── config
  ├── description
  ├── HEAD
  ├── hooks
  │   ├── applypatch-msg.sample
  │   ├── commit-msg.sample
  │   ├── post-update.sample
  │   ├── pre-applypatch.sample
  │   ├── pre-commit.sample
  │   ├── prepare-commit-msg.sample
  │   ├── pre-push.sample
  │   ├── pre-rebase.sample
  │   └── update.sample
  ├── index
  ├── info
  │   └── exclude
  ├── logs
  │   ├── HEAD
  │   └── refs
  │       └── heads
  │           └── master
  ├── objects
  │   ├── 13
  │   │   └── ce45372a70dba8b87d4612952d0f4c1775960b
  │   ├── 3e
  │   │   └── 20527aac2a95acab281585e2ec2015b7e8a77c
  │   ├── 77
  │   │   └── 25275230928119f8acf11ee297e559f7e4ce56
  │   ├── info
  │   └── pack
  └── refs
      ├── heads
      │   └── master
      └── tags
```

The most important thing is the `objects` directory. In git everything is an object.

If we add a file git will create a checksum of this file. Git uses `sha-1` to create the checksum. `sha-1` will reduce the complete contents of the file to a 40-character unique key. A slight modification to the file will result in a completely different key.

For example:

```
echo "Hello World"  | sha1sum # 648a6a6ffffdaa0badb23b8baf90b6168dd16b3a
echo "Hello World." | sha1sum # b924c2f360b572e17c971f1b1b667e0732944df7
```

By adding a dot to our sentence, the hash has a completely different value...

Git stores the contents of each file committed file under the checksum name.

For example our file.txt is stored as `.git/objects/77/25275230928119f8acf11ee297e559f7e4ce56` where `7725275230928119f8acf11ee297e559f7e4ce56` is the checksum of this file...

The other two files are the commit message and a list of all the tracked files...

Via this checksumming, git knows which files are modified and which are not because each change results in another checksum...

```
echo "Modify the file" >> file.txt
git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#   modified:   file.txt
#
# no changes added to commit (use "git add" and/or "git commit -a")
```

Let's commit (save) this change:

```
# add the file to the staging area:
git add file.txt

# save all staged files
git commit -m "modify file.txt"
# [master 1de3a69] modify file.txt
#  1 file changed, 1 insertion(+)

git status
# On branch master
# nothing to commit, working directory clean
```

# Git log

To get a list of all commits (snapshots/return points) use the `git-log` -command:

```
git log
# commit 1de3a699080e8df52f3f12af988152208c632b00
# Author: Arne Soete <arne.soete@irc.vib-ugent.be>
# Date:   Sun Nov 6 18:50:29 2016 +0100
#
#     modify file.txt
#
# commit 13ce45372a70dba8b87d4612952d0f4c1775960b
# Author: Arne Soete <arne.soete@irc.vib-ugent.be>
# Date:   Sun Nov 6 18:28:06 2016 +0100
#
#     start tracking file.txt
```

# Checkout a commit

Via `git-checkout` we can jump back in time to another snapshot:

```
git checkout 1de3a
```

**Info:** Note that instead of typing the complete checksum, only a fraction of the key was specified. Git is smart enough to find out which commit we meant en checked this revision out. We just have to ensure that the provided part is unique in the project...

# Git diff

`git-diff` allows to show the differences between two revisions.

```
git diff 13ce4 1de3a
# diff --git a/file.txt b/file.txt
# index 7725275..817f387 100644
# --- a/file.txt
# +++ b/file.txt
# @@ -1 +1,2 @@
#  Let's track this file...
# +Modify the file
```

We see that between the two commits the line `Modify the file` was added (indicated by the `+` sign).

Is is however cumbersome to look up the hashes for a certain commit. Fortunately git has the notion of *refs*. Refs are named commits e.q. a commit can be addressed by a name.

For example `HEAD` always points to the last commit on the current branch. `HEAD^` points to the second to last commit. `HEAD^^` the one before that etc...

Thus the previous diff can be written as:

```
git diff HEAD^
# diff --git a/file.txt b/file.txt
# index 7725275..817f387 100644
# --- a/file.txt
# +++ b/file.txt
# @@ -1 +1,2 @@
#  Let's track this file...
# +Modify the file
```

> **Info:** If only one commit is provided, the other commit is assumed HEAD

# Branching and merging

## Branch

As mentioned before, git allows us to branch of you code, test some things an merge this new code back in to the main branch or discard the changes.

By default all code lives in the *master*-branch. A new branch can be created via `git-branch`.

When no arguments are passed to `git-branch`, the command will list all the available (local) branches.

```
git branch
# * master
```

The asterisks indicates which branch you are currently on.

If we pass git branch a name, a new branch with this name will be created:

```
git branch 'my-new-branch'

git branch
# * master
#   my-new-branch
```

`git-checkout` is used to switch branches:

```
git checkout 'my-new-branch'
# Switched to branch 'my-new-branch'

git branch
#   master
# * my-new-branch
```

Lets add a change in this branch:

```
echo "Add line from 'my-new-branch!'" >> file.txt

git add file.txt

git commit -m "add change from my-new-branch"
# [my-new-branch d5136c8] add change from my-new-branch
#  1 file changed, 1 insertion(+), 2 deletions(-)
```

Similar to `HEAD` before, `master` and `my-new-branch` these names are *refs* pointing to a certain commit.

So we can also take a *diff* between two branches:

```
git diff master my-new-branch
# diff --git a/file.txt b/file.txt
# index 817f387..082bfe1 100644
# --- a/file.txt
# +++ b/file.txt
# @@ -1,2 +1,3 @@
#  Let's track this file...
#  Modify the file
# +Add line from 'my-new-branch'

git diff master
# diff --git a/file.txt b/file.txt
# index 817f387..082bfe1 100644
# --- a/file.txt
# +++ b/file.txt
# @@ -1,2 +1,3 @@
#  Let's track this file...
#  Modify the file
# +Add line from 'my-new-branch'
```

> **Info:** If a second argument is omitted, HEAD is used (and HEAD in this case is last commit on my-new-branch)

# Merge

If we want to include the changes we introduced in the *my-new-branch*-branch into another branch, say *master*, we have to merge in these changes via `git-merge`.

First we have to position ourselves at the *master*-branch:

```
git checkout master
# Switched to branch 'master'
```

```
git diff my-new-branch
# diff --git a/file.txt b/file.txt
# index 082bfe1..817f387 100644
# --- a/file.txt
# +++ b/file.txt
# @@ -1,3 +1,2 @@
#  Let's track this file...
#  Modify the file
# -Add line from 'my-new-branch'
```

Via the *diff* we notice we are missing the `Add line from my-new-branch` line.

The include these changes:

```
git merge my-new-branch
# Updating 1de3a69..e336094
# Fast-forward
#  file.txt | 1 +
#  1 file changed, 1 insertion(+)
```

Git tells us the merge was a success, no merge conflicts where encountered and git was able to auto-merge the branches...

# Merge conflict

Not all merges will go as smooth as the previous example. Lets create a merge conflict.

```
# Create two branches
git checkout master
git branch conflicting-branch

# modify file on master
echo "Add line line from master" >> file.txt
git add file.txt
git commit -m "add line from master"

# modify file on other branch
git checkout conflicting-branch
echo "Add line line from conflicting-branch" >> file.txt
git add file.txt
git commit -m "add line from conflicting-branch"
```

Now we have a merge conflict. The same file was modified on two separate branches en git has no way to tell which version of the file is the correct one, or how to combine the introduced modifications...

The commit graph looks something like:

```
* 42a0cbc (conflicting-branch) add line from conflicting-branch
| * eae4aa2 (HEAD -> master) add line from master
|/
* e336094 add line form my-new-branch
* 1de3a69 modify file.txt
* 13ce453 start tracking file.txt
```

Lets run a merge and see what happens:

```
git checkout master
# Switched to branch 'master'

git merge conflicting-branch
# Auto-merging file.txt
# CONFLICT (content): Merge conflict in file.txt
# Automatic merge failed; fix conflicts and then commit the result.
```

To resolve a merge conflict via a third party application (GUI?) use `git-mergetool`

```
git mergetool
# Merging:
# file.txt
#
# Normal merge conflict for 'file.txt':
#   {local}: modified file
#   {remote}: modified file
# file.txt seems unchanged.
# Was the merge successful? [y/n] y
```

In my case the default merge tool is meld:

- the left pane is a branch (master)
- the middle pane is the new (to combine) file
- the right pane is the other branch (conflicting-branch)
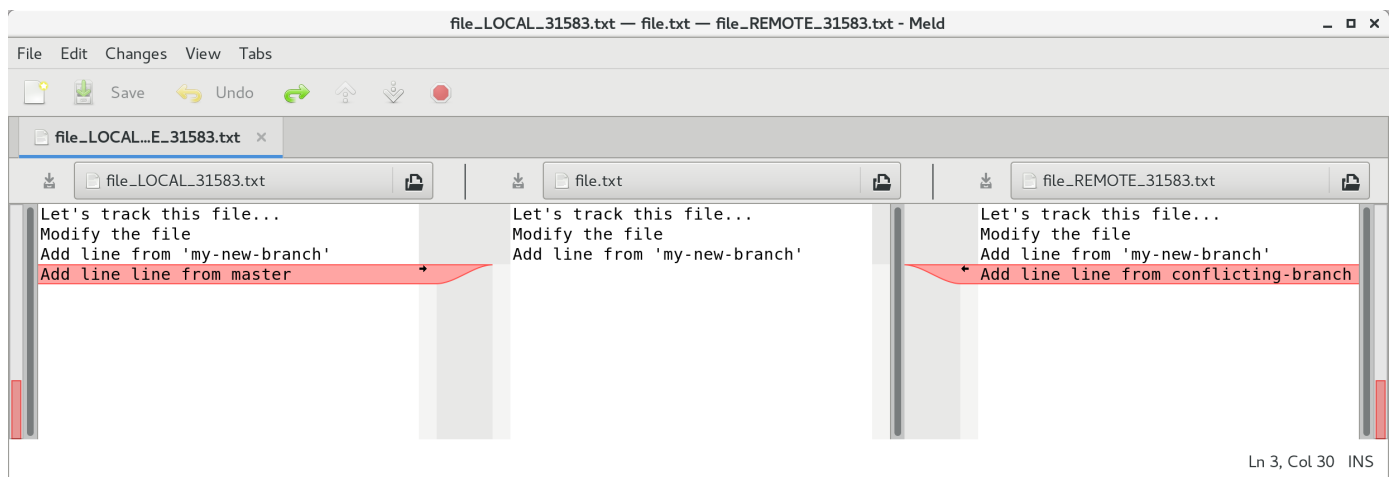


git-mergetool: meld

Once your merges are made, the files saved and the application quit, git will ask if the merge was a success.

The merge can also be made manually by editing the file directly:

```
gedit file.txt
```

```
Let's track this file...
Modify the file
Add line from 'my-new-branch'
<<<<<<< HEAD
Add line line from master
=======
Add line line from conflicting-branch
>>>>>>> conflicting-branch
```

Notice the added `<<<<<<< HEAD` , `=======` , `>>>>>>> conflicting-branch` lines.

These indicators show the division between the two branches.

We are currently at the master branch, so HEAD refers to the changes made at *master*, the second line holds the

changes introduced in *conflicting-branch*.

We can now modify the file, add it and commit the merge. This will resolve the merge conflict

- Modify file:

      Let's track this file...
      Modify the file
      Add line from 'my-new-branch'
      Add line line from master
      Add line line from conflicting-branch

- Save changes

      **git** add file.txt
      **git** commit -m "merge: conflicting-branch into master"

Our commit graph now looks like:

```
*   e5bbaef (HEAD -> master) merge: conflicting-branch into master
|\
| * 42a0cbc (conflicting-branch) add line from conflicting-branch
* | eae4aa2 add line from master
|/
* e336094 add line form my-new-branch
* 1de3a69 modify file.txt
* 13ce453 start tracking file.txt
```

> **Info:** To create a similar graph, run: `git log --oneline --decorate --graph`

# Delete branch

Once a branch is merged in we can delete the branch via:

```
git branch -d my-new-branch
# Deleted branch my-new-branch (was e336094).
```

# Remotes

We are now able to create, merge and delete local branches, the same can be done with remote branches.

Git is a distributed system this means we can clone, sync and push with other repositories. These repositories can live in the same machine but in another directory or on another computer (server).

Remote repositories can be accessed via SSH or HTTP if the remote is configured correctly.

github.com allows anyone to create remote repositories. These can be used to collaborate with classmates or colleagues or as a remote backup.

The only downside is that github only offers free hosting for public repositories, so everyone can view your code...

In the following examples we will use another directory as a remote, but the same principles apply to remoter servers.

# Clone

`git-clone` allows us to fetch a local copy of a remote repository...

```
git clone /tmp/git-playground /tmp/git-clone
# Cloning into '/tmp/git-clone'...
# done.

cd /tmp/git-clone
ls
# file.txt

git log
# commit e5bbaef29c5c6d3d874d0f92b9b4be72fe007dca
# Merge: eae4aa2 42a0cbc
# Author: Arne Soete <arne.soete@irc.vib-ugent.be>
# Date:   Sun Nov 6 21:51:53 2016 +0100
#
#     merge: conflicting-branch into master
#
# commit 42a0cbce88d3724603c975005488b30dcdf67141
# Author: Arne Soete <arne.soete@irc.vib-ugent.be>
# Date:   Sun Nov 6 21:31:55 2016 +0100
#
#     add line from conflicting-branch
#
# commit eae4aa2178acba0125d3c23adf77d9791a802ddf
# Author: Arne Soete <arne.soete@irc.vib-ugent.be>
# Date:   Sun Nov 6 21:31:31 2016 +0100
#
#     add line from master
#
# commit e336094c2f404534ccf47e058cbce8ef53911c15
# Author: Arne Soete <arne.soete@irc.vib-ugent.be>
# Date:   Sun Nov 6 20:41:50 2016 +0100
#
#     add line form my-new-branch
#
# commit 1de3a699080e8df52f3f12af988152208c632b00
# Author: Arne Soete <arne.soete@irc.vib-ugent.be>
# Date:   Sun Nov 6 18:50:29 2016 +0100
#
#     modify file.txt
#
# commit 13ce45372a70dba8b87d4612952d0f4c1775960b
# Author: Arne Soete <arne.soete@irc.vib-ugent.be>
# Date:   Sun Nov 6 18:28:06 2016 +0100
#
#     start tracking file.txt
```

A complete copy of the repository was fetched from the remote, we have all the history..

## Update

To update a repository to the latest version we can use `git-pull`.

By default `git-pull` needs two arguments:

- the name of the remote
- the name of the branch to fetch

If a repository is cloned, by default the name of the remote is `origin`.

A list of all the remotes can be obtained via `git-remote`:

```
git remote
# origin

git remote -v # -v -> get more info about the remote
# origin  /tmp/git-playground (fetch)
# origin  /tmp/git-playground (push)
```

Lets see a `git-pull` in action

Make some changes to original repository:

```
cd /tmp/git-playground
echo "add another file" > file2.txt
git add file2.txt
git commit -m "add a second file"
```

Fetch the latest changes into clone:

```
cd /tmp/git-clone
git pull origin master
# remote: Counting objects: 3, done.
# remote: Compressing objects: 100% (2/2), done.
# remote: Total 3 (delta 0), reused 0 (delta 0)
# Unpacking objects: 100% (3/3), done.
# From /tmp/git-playground
# * branch        master    -> FETCH_HEAD
#   e5bbaef..b7e2e69  master   -> origin/master
# Updating e5bbaef..b7e2e69
# Fast-forward
#  file2.txt | 1 +
#  1 file changed, 1 insertion(+)
#  create mode 100644 file2.txt
```

A merge conflict can occur similar to `git-merge`. The methods apply to resolve the merge conflict.

1. Modify the conflicting files via `git mergetool` or manually via the editor
2. Add en commit the changes

# Push

To submit changes back to a remote, `git-push` is used.

Similar to `git-pull`, `git-push` accepts two parameters:

1. the name of the remote
2. the remote branch to push to

```
git push origin master
# Counting objects: 3, done.
# Delta compression using up to 8 threads.
# Compressing objects: 100% (2/2), done.
# Writing objects: 100% (3/3), 318 bytes | 0 bytes/s, done.
# Total 3 (delta 0), reused 0 (delta 0)
# To /tmp/git-playground/
#   b7e2e69..2038622  master -> master
```

> **Info:** You cannot pus to a normal *checked out* repository. Only to bare repositories. Create a bare repository:
>
> ```
> git clone --bare /tmp/git-playground /tmp/git-bare
> ```
>
> See `git-clone` for more info

# Additional exercise

## 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

---

Last modified by Arne Soete at 2016-11-07