

# An Introduction into modern web



# What we will look at

**HTML** for (structured) content.

Add text, images, files and meta data. *Goal: Know the most important HTML tags.*

**CSS** for the design.

Define spacing, font and colors. *Goal: Know design basics and using a css framework.*

**JavaScript** to add functionality.

Animations! Filters! Sorting! *Goal: Apply jQuery & friends.*

**PHP** to generate content.

Update the webpage more conveniently. *Goal: Generate your webpage based on data on the server.*

*Each of these areas is its own profession!*

*We only briefly touch JavaScript and PHP.*

# Hypertext Markup Language (HTML)

*Slogan: So easy, its not even programming!*

# A HTML document

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport"
      content="width=device-width, initial-scale=1">

    <title>Hello, world!</title>
    <meta name="description" content="A simple HTML document.">
  </head>
  <body>
    <h1>Hello, world!</h1>
    <p>This demonstrates a simple HTML document.</p>
  </body>
</html>
```

*Details on the following Slides.*

# The core constructs

Tags:

```
<tag>  
  content of tag  
</tag>
```

Attributes:

```
<tag attribute="content of attribute"></tag>  
<tag no-content-attribute></tag>
```

Comments:

```
<!-- whatever in here is ignored -->
```

That's all!

# Technical

```
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shr
    <!-- ... -->
  </head>
  <!-- ... -->
</html>
```

**Charset** defines how your bits & bytes are mapped to text.

**Viewport** defines the width of the document shown in the browser.  
Primarily useful for mobile devices.

# Meta data

```
<html>
  <head>
    <!-- ... -->
    <title>Hello, world!</title>
    <meta name="description" content="A simple HTML document.">
  </head>
  <!-- ... -->
</html>
```

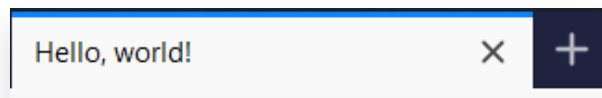
Used by search engines to classify the document and show a preview.

Hello World

 [www.crashcourse.webpage.ch/html/](http://www.crashcourse.webpage.ch/html/)

A simple HTML document.

Used by browsers in the browser tab.



# Body

```
<html>
  <!-- ... -->
  <body>
    <h1>Hello, world!</h1>
    <p>This demonstrates a simple HTML document.</p>
  </body>
</html>
```

The actual, structured content.

Elements:

- Headers: <h1>, <h2>, <h3>, <h4>, ...
- Text: <p>, <br>, <a>, <i>, <u>
- Images: <img>
- Structure: <section>, <article>, <div>



# Headers

H1 for the title (only once)!

H2 for chapter headers.

H3 for subheaders.

`<h1>Header H1</h1>`

`<h2>Header H2</h2>`

`<h3>Header H3</h3>`

`<h4>Header H4</h4>`

# Text

```
<p>Write a paragraph inside p tags.  
Use <br/> to start a new line explicitly</p>
```

Write a paragraph inside p tags. Use  
to start a new line explicitly

```
<i>italic</i> <br/>  
<b>bold</b> <br/>  
<u>underline</u>
```

*italic*, **bold**, underline

```
<a href="https://developer.mozilla.org/">mozilla documentation</a>  
<a ... target="_blank">open in new tab</a>
```

[mozilla documentation](https://developer.mozilla.org/) [open in new tab](#)

# Images

```

```



**University of  
Zurich<sup>UZH</sup>**

describe the image with the alt tag.  
used by search engines, screen readers, ....  
or if the image does not load.

# Files

```
<a href="images/UZH-logo.png" download>download file</a>
```

[download file](#)

add the download attribute to force downloading  
else it opens in browser (which might then display download dialog)

# Misc

```
<audio src="files/audio.mp3">play audio</audio>
```

```
<video src="files/video.mp4">play video</video>
```

```
<ul>  
  <li>Banana</li>  
  <li>Apple</li>  
</ul>
```

```
<ol>  
  <li>Learn HTML</li>  
  <li>Learn CSS</li>  
  ...  
</ol>
```

# (Semantic) Structure

```
<section>
  <div class="container">
    <article>
      <h1>About HTML</h1>

      <h2>Introduction</h2>
      <p>We are going to learn about (...)</p>

      <h2>About Semantic HTML</h2>
      <p>Use semantic tags to (...)</p>
    </article>
  </div>
</section>
```

Create own tags (like `<section>`) without special behaviour (such as `<a>`).  
Give your document semantic meaning to make editing & styling easier.

Have something important? Use `<strong>` instead of `<b>`.  
Writing an article? Wrap it in `<article>` instead of `<div>`.

# Cascading Style Sheets (CSS)

*Slogan: Confused yet?*

<design-crashcourse>

Why? Because:

Users often perceive aesthetically pleasing design as design that's more usable. - <https://www.nngroup.com/articles/aesthetic-usability-effect/>



# Typography - Typeface

**serif** for books (see header).

Line-like Serifs help the eye read.

*Arial, Times New Roman, ...*

**sans-serif** for displays (see body).

No serifs look cleaner.

*Verdana, Courier, Helvetica, ...*

**monospace** for code (like this).

Same width characters help notice patterns faster.

*Consolas, ...*

many more variations...

# Typography - Font

Typefaces may consist of fonts of different weights

thin < light < **regular** < **bold** < **black**

Many free-to-use fonts exist, like [google fonts](#).  
You might want to [pair fonts](#).

# Typography - Size

Adjust font size, line height & line length.

Line height

coordination  
agreement about technicalities  
"cheap talk" is enough (no san

A line should be around 15 words long. If its too long, the eye can not easily find the start of the next line. If its too short, the eye has to move around too much. There are other factors that affect readability, which we will see on the next slide.

# Typography - Balance

Balance font weight, font size, line length and line height.

Improve readability:

- increase line height
- increase font size
- increase font boldness
- decrease line length

coordination

agreement about technicalities without conflict of interest

"cheap talk" is enough (no sanctions needed to enforce)

like on which street side to drive on generally

**coordination**

agreement about technicalities without conflict of interest

"cheap talk" is enough (no sanctions needed to enforce)

like on which street side to drive on generally

# Spacing

Separate what does not belong together.  
Separate little what does.

Emphasize importance with more space around.

You may want to define a spacing scheme for consistency.  
Like 4, 8, 16, 32, 64 (geometric progression).  
Like 5, 8, 13, 21, 34 (Fibonacci numbers / golden cut).

# Spacing - Example

# Spacing - Example

# Colors

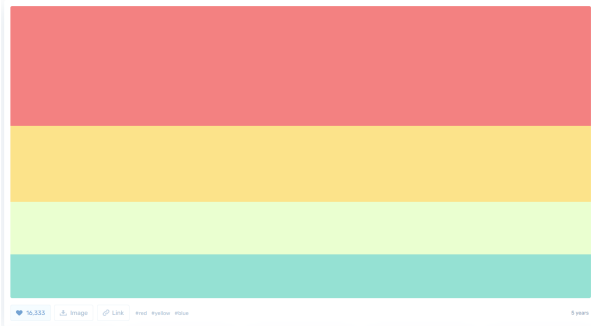
Give your site a distinctive look with your "brand" colors.

Primary color associated with your brand.

Secondary color which contrasts primary nicely (like complement).

Colors for messages (red for errors, orange for warnings, ...).

Pastel colors for a soft look, bright colors for inspiring effects



Many premade color schemes, like [colorhunt](#)

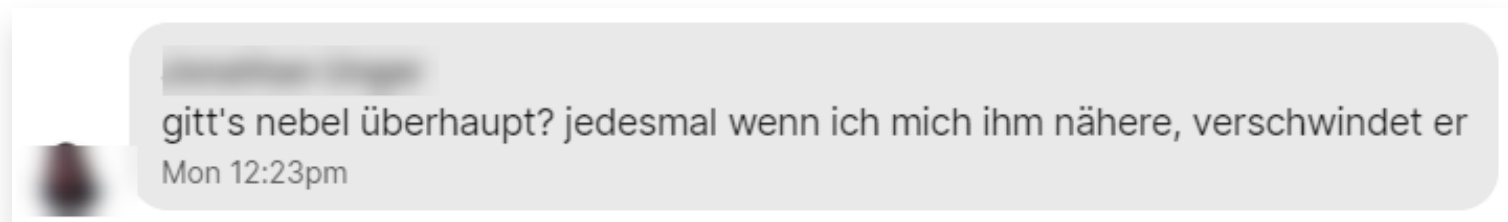
Many color scheme generators, like [paletton](#)



# Cheap Design

"Cheap Design" looks good on the drawing board, but does not work in practice.

No contrast:



Font too light:

## 2. Traktandum 2

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Elit eget gravida cum sociis natoque penatibus et magnis dis. Proin nibh nisl condimentum id venenatis a condimentum vitae sapien. Varius quam quisque id diam vel quam elementum pulvinar etiam. Eget nunc lobortis mattis aliquam faucibus purus. Interdum velit euismod in pellentesque massa placerat. Urna id volutpat lacus laoreet non. Id eu

Accessibility checker for contrast from [WebAIM](#).

Font weight is more [difficult](#).

`</design-crashcourse>`

# CSS basics

```
body {  
  color: blue;  
}
```

choose a **selector** (like body)  
for **properties** (like color)  
override the default **value** (like blue)

# CSS selector

by element, class or id

```
<section class="content" id="introduction"></section>
```

```
section {  
  background: red  
}  
  
.content {  
  background: blue;  
}  
  
#introduction {  
  background: green;  
}
```

choose the most appropriate selector according to your structure (usually it ends up being the class).

# Typography - Font Family

```
body {  
  font-family: 'Times New Roman', Times, serif;  
  font-weight: bold;  
  font-style: italic;  
}
```

including your own font family with

```
@font-face {  
  font-family: 'Vegur';  
  src: url('Vegur-Bold.ttf') format('ttf');  
  font-weight: bold;  
  font-style: normal;  
}
```

Generate optimized font family files [transfonter.org](https://transfonter.org).

# Typography - Size

```
body {  
  font-size: 16px;  
  line-height: 1.2em;  
}
```

em is a relative unit to the current font size.

rem is a relative unit to the html font size.

You can try out different combinations on [app.typeanything.io](https://app.typeanything.io).



# Color

```
body {  
  color: #FFFFFF;  
  background: rgb(0,0,0);  
}
```

use predefined [colors](#).

or specify own color (using additive light mixing).

use hex (like #rrggbb).

or rgb (like rgb(red, green , blue)).

or rgba (like rgba(red, green , blue, opacity)) representations.



# Display

```
body {  
  display: block;  
}
```

defines how element is placed (and elements around).

display: block renders a full-width container.

display: inline-block renders as wide as the content.

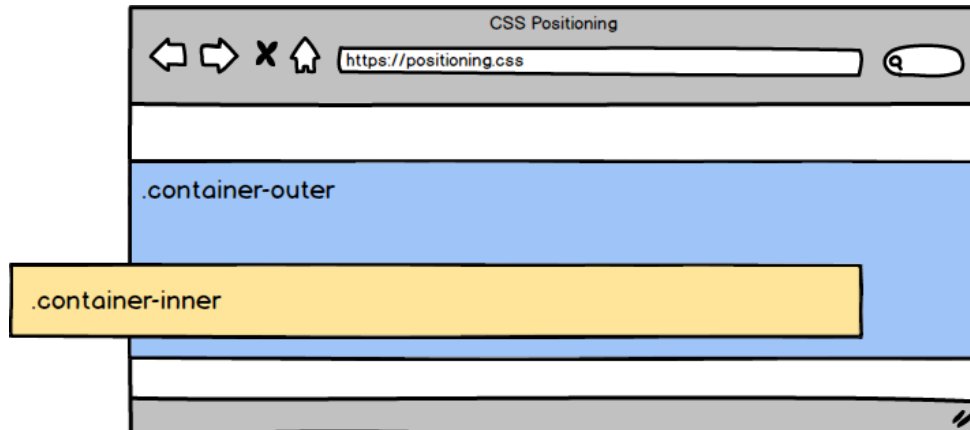
for arranging items along a grid, use display: [grid](#).

for distributing items use display: [flex](#).

# Position

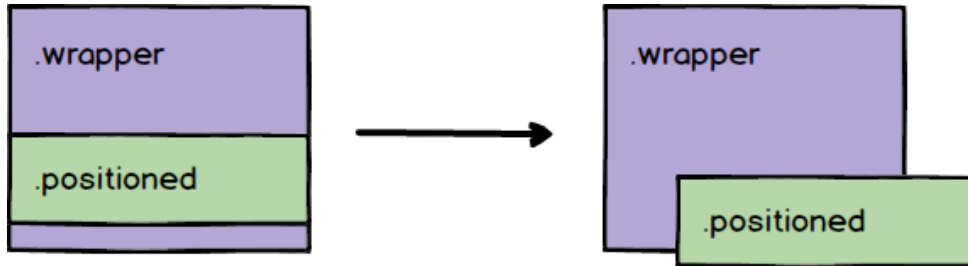
```
.container-outer {  
  position: relative;  
  top: 10px;  
  left: 0px;  
  width: 100%;  
  height: 35px;  
  background-color: blue;  
}
```

```
.container-inner {  
  position: absolute;  
  top: 20px;  
  right: 20px;  
  width: 100%;  
  height: 10px;  
  background-color: yellow;  
}
```

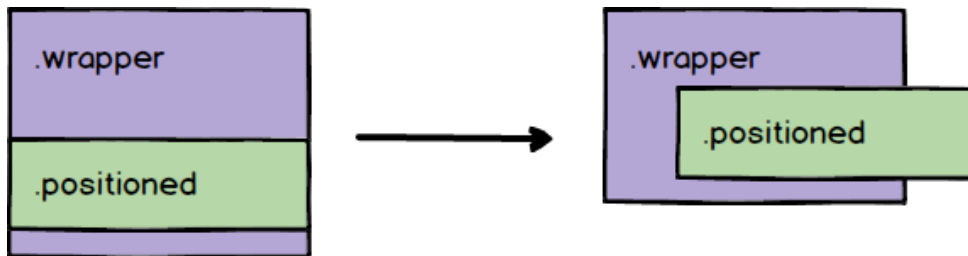


# Position

relative to original position.



absolute to its *closest positioned ancestor*: next parent with `position: relative`.  
Removed from the document flow.



# (Good) design & CSS is hard!

Use proper resources:

- <https://developer.mozilla.org/en-US/docs/Web/CSS>

Use a CSS framework:

- [Bootstrap](#)
- [Foundation](#)
- [Bulma](#)

Getting it right is *extremely* hard!

# Opiniated advice what to do now

You know how to create a webpage now (without functionality)!  
To tame the complexity, just start :)

Setup steps:

- Pick a CSS framework (like [Bootstrap](#))
- Look for the "starter template" (in HTML) or something similar
- Create a .html (like index.html) and paste the template
- Look at it in the browser (double-click)

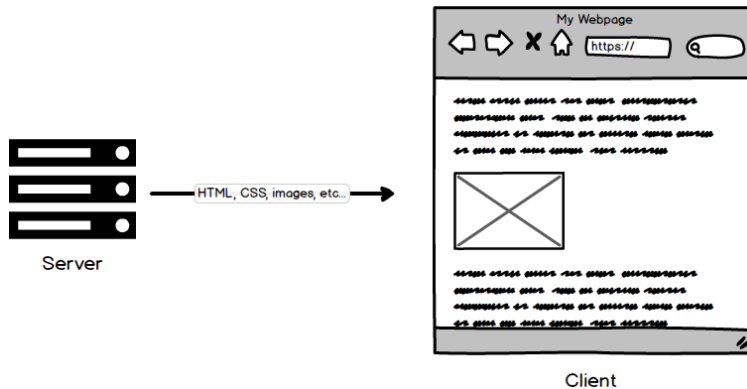
Create your webpage:

- Add your own HTML to this .html file
- Add styling using the framework provided CSS classes

# JavaScript

*Slogan: Absolutely no similarities with Java AT ALL.*

# Why?



Execute code on client for interactivity.

Examples:

- Upon scrolling, content fades in (design)
- Button is disabled as long as user must not click it (usability)
- Table entries can be filtered and searched for (functionality)

# About JavaScript

General-Purpose programming language.

Dynamically typed (like python).

Curly brackets to designate blocks of code (like C#, Java).

Prototype-based inheritance (like Lua; rarely used concept).

In browsers, has access to DOM (representation of HTML).

Can rewrite the content of the page!

Essentially two concepts on how to use JavaScript:

- **to enhance server-generated HTML ("traditional")**
- to generate HTML ("webapps")



# Webapps

heavy client-side functionality.

communicates with the server using mostly using "AJAX" requests: Content is loaded continuously in the background without the user noticing.

examples:

- google docs
- spotify
- facebook

frameworks:

- [Angular](#)
- [React](#)
- [VueJS](#)
- many, MANY more

# Traditional websites

content is generated by the server, but some small interactivity is added.

the webpage is mostly complete & usable without JavaScript; its a "Nice-to-have".

examples:

- ETH, UZH homepages
- this presentation
- most KMU webpages

by combining multiple JavaScript libraries (for tables, for animations, for modals, ...) and pointing them to right DOM element to "enhance".

# JavaScript Samples

## Hello world

```
let message = "Hello world";  
console.log(message);
```

## Loops

```
let sum = 0  
for (let i = 0; i < 10; i++) {  
    sum += i  
}  
console.log(sum)
```

## Interact with the DOM and browser

```
document.getElementById("wrapper").innerHTML = "Hello world";  
window.location.href = "https://developer.mozilla.com";  
alert("This is annoying");
```

# jQuery

library which simplifies interacting with the DOM.

DOM manipulation

```
$(".button").click()
```

Event handling

```
$(document).ready(() => { $(".button").click() });
```

Ajax

```
$.ajax({  
  type: "POST",  
  url: 'index.php',  
  data: $('form').serialize()  
});
```

# DataTable example

```
<!doctype html>
<html lang="en">
  <head>
    <!-- ... -->
    <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>

    <link rel="stylesheet" href="https://cdn.datatables.net/1.10.23/css/jqu
    <script src="https://cdn.datatables.net/1.10.23/js/jquery.dataTables.mi
    <script>
      $(document).ready(function () {
        $('.table-interactive').DataTable();
      });
    </script>
  </head>
  <body>
    <table class="table table-interactive">
      <!-- ... -->
    </table>
  </body>
</html>
```

# Masonry example

```
<!doctype html>
<html lang="en">
  <head>
    <!-- ... -->
    <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>

    <script src="https://unpkg.com/masonry-layout@4/dist/masonry.pkgd.min.js"></script>
    <script>
      $(document).ready(function () {
        $('.grid').masonry({
          itemSelector: '.grid-item',
        });
      });
    </script>
  </head>
  <body>
    <div class="grid">
      <div class="grid-item ..."></div>
      <div class="grid-item ..."></div>
      <!-- ... -->
    </div>
  </body>
</html>
```

# Development tools

In browser, do right click and select "Inspect element" or similar.

HTML view:

- view & modify the HTML
- investigate applied css properties

Console view:

- view errors if they occurred
- directly execute JavaScript

Useful to try out / investigate when something does not work.

# JavaScript summary

General-Purpose language. Many read-to-use libraries for beginners.

General steps to use a library:

- Include required JavaScript / css files
- Add initialization code hooking library and DOM
- Debug using the Development tools

Resources:

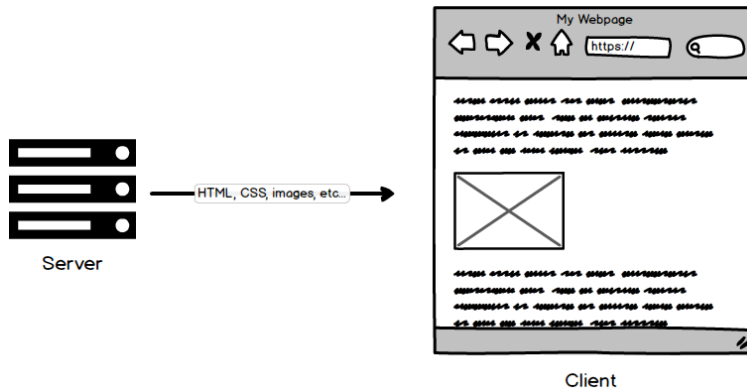
- [jQuery libraries](#)
- [Package manager introduction](#)
- [JavaScript Developer Roadmap](#)



# PHP: Hypertext Preprocessor (PHP)

*Slogan: It is getting better. Or: [PHP: a fractal of bad design](#).*

# Why?



Generate HTML on server (for example using data from a database).

Examples:

- A table of all upcoming events is generated
- An E-Mail is sent upon contact form submission

# About PHP

General-Purpose programming language.

Dynamically typed (like python).

Curly brackets to designate blocks of code (like C#, Java).

"Normal" inheritance (like C#, Java).

Developments towards more "typing".

Multiple approaches:

- **Inject into HTML to generate only (dynamic) parts of page**
- Configure pre-made content management system (CMS)
- Use fully-fledged framework for webapps which generates HTML
- Provide only data over API to be consumed by client-side code

# Executing PHP

PHP is a server-side language. You need a PHP server!

ETH student?

- Login per SSH into slab1.ethz.ch with kürzel & password.
- Place files in homepage folder.
- Access your personal webpage under <https://n.ethz.ch/~kürzel/>

Many commercial products like [cyon](#), [metanet](#), [hostpoint](#), ...

Test locally with [XAMPP](#) or read the documentation of your linux distribution how to setup Apache with PHP.

You can use [FileZilla](#) to access the server via FTP / SSH.

# "Hello World" in PHP

```
<!doctype html>
<html lang="en">
  <head>
  </head>
  <body>
    <?php
      echo "<h1>Hallo Welt</h1>";
    ?>
  </body>
</html>
```

Placed as `index.php` on your root directory, it generates:

```
<!doctype html>
<html lang="en">
  <head>
  </head>
  <body>
    <h1>Hallo Welt</h1>
  </body>
</html>
```

# Upcoming events in table (1/2)

Dynamic data generation requires:

- data source (like database or simply a file)
- transformation (transform result of query into usable php variables)
- generation (generate html out of the variables)

For this example, we place `events.json` in the same directory as our script:

```
[
  {
    "title": "Web",
    "date:" "30.08.2020"
  },
  {
    "title": "Project Management",
    "date:" "10.09.2020"
  },
  ...
]
```

# Upcoming events in table (2/2)

create one row in the table for each event

```
<?php
$eventsJson = file_get_content("events.json");
$events = json_decode($eventsJson);
?>
...
<table>
  <tbody>
    <?php foreach ($events as $event) { ?>
      <tr>
        <td><?= $event->title ?></td>
        <td><?= $event->date ?></td>
      </tr>
    <?php } ?>
  </tbody>
</table>
...
```

# Upcoming events in table result

HTML that is delivered to the browser

```
...  
<table>  
  <tbody>  
    <tr>  
      <td>Web<td>  
      <td>30.08.2020</td>  
    </tr>  
    <tr>  
      <td>Project Management<td>  
      <td>10.09.2020</td>  
    </tr>  
  </tbody>  
</table>  
...
```



# Contact form submission

We require:

- get user input (like using a form)
- send it to the server (HTTP GET / POST / PUT / ...)
- process it on the server (query global php variables)

Get user input with HTML forms:

```
<!-- ... -->
<form method="post">
    <input name="name" type="text"></input>
    <textarea name="message">
    <input name="submit" type="submit"></input>
</form>
<!-- ... -->
<?php
if (isset($_POST["name"]) && isset($_POST["message"])) {
    $body = $_POST["name"]." has message ".$_POST["message"];
    mail("info@thealternative.ch", "Contact request", $body);
}
?>
<!-- ... -->
```

# Develop PHP snippets

Beginning is simple, mastery is hard.

Challenges:

- Subtle bugs as PHP very permissive
- Many security challenges (especially with user-input)
- Many beginners = much wrong information

Resources:

- Use `var_dump` function to debug variables
- <https://www.php.net/> for the documentation
- <https://stitcher.io/> for news about php

# Content Management System (CMS)

After installation, non-technical users can modify content.

Easy to setup marketing webpage, blog, small online shops, ...

Examples:

- [Wordpress](#) as most used, very good usability
- [Joomla](#) as another much used alternative
- [Drupal](#) for more complex applications

Popular Webpage-As-A-Service:

- [Wix](#)
- [Squarespace](#)

# Frameworks

PHP is good for beginners, but scaling is difficult.  
Use one of the well-designed, stable frameworks.

Recommended Frameworks:

- [Symfony](#) as the most used framework
- [Laravel](#) uses symfony concepts, more "hip"
- [Slim](#) as a minimal framework
- [ApiPlatform](#) which speeds up API development

Mastery of Frameworks takes *years*,  
but reduces development / maintenance by *order of magnitudes*.

# Next steps

Summary:

- **HTML** for (structured) content.
- **CSS** for the design.
- **JavaScript** to add client functionality.
- **PHP** to generate content on the server.

For developers:

- Choose focus (Frontend? Backend? Marketing? Engineering?)
- Learn! Useful: [Roadmap](#)
- Ask questions on [stackoverflow](#) once you can [ask good questions](#)
- High investment = high reward

For researcher webpages:

- Connect to the provided ETH server
- Write HTML / CSS; add PHP / JavaScript only when necessary
- Use CSS frameworks if you want to have it look good

Many, many more use cases. This presentation is technically also a webpage :)