

Static and dynamic Web pages

Introduction aux technologies Web

+

Initiation au développement Web

Eugen Dedu

Associate professor

Université Marie et Louis Pasteur, IUT Nord Franche-Comté

Montbéliard, France

Sept. 2025

<https://dedu.fr>

eugen.dedu@univ-fcomte.fr

Objectives

- Example with a Web page and its source
- Objectives: develop simple Web pages, both static and dynamic, for all media (PC, smartphone)
- Does not cover: modern Web sites (including mobile phone applications), multimedia (artistic point of view)
- Prerequisites, for part 2 (dynamic Web):
 - programming languages (variable, loop, functions, algorithm, instruction)
 - databases (SQL)

Course overview

Static Web pages (*Introduction aux technologies Web*):

- HTML
- CSS
- FI/FA: 3h CM, 6h TP

- No exam, but marks copied from the technical part of SAÉ14
- Useful also for SAÉ23

Dynamic Web pages (*Initiation au développement Web*):

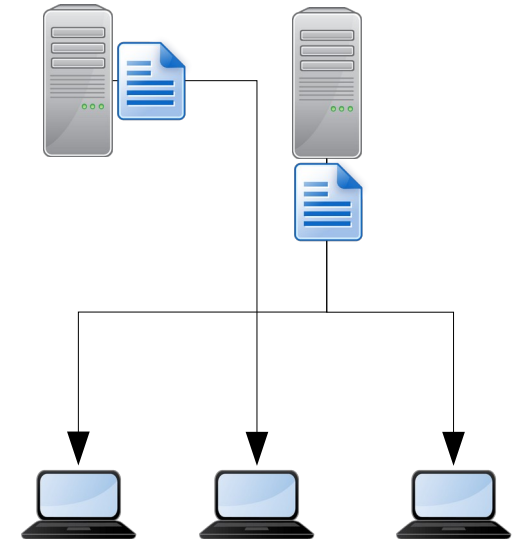
- server-side scripts (back end): PHP
 - database integration (MariaDB/MySQL)
- client-side scripts (front end): JavaScript
- notions of modern dynamic Web
- FI: 7h30 CM, 16h30 TP
- FA: 7h30 CM, 10h30 TP

- Exam on machines of ~1h30

1. Static Web pages

Some definitions

- Internet = the network/infrastructure (e.g. Free, Orange)
- Services on top of Internet: e-mail, WWW (Web), games, videoconference etc.
 - each service uses its specific software: browser for WWW, videoconf client for videoconference etc.
- The Web uses a client-server architecture, where machines are either clients, or servers:
 - server = waits for connections and has the data (Web pages)
 - client = initiates the connection
 - examples of Web servers: Apache, Nginx
 - examples of Web clients (browsers): Firefox, Chrome, Safari, Edge
- HTML (*HyperText Markup Language*) = Web page "description" language
- HTTP (*HyperText Transfer Protocol*) = the protocol defining data exchange between client (browser) and server



1.1 HTML

Bibliography

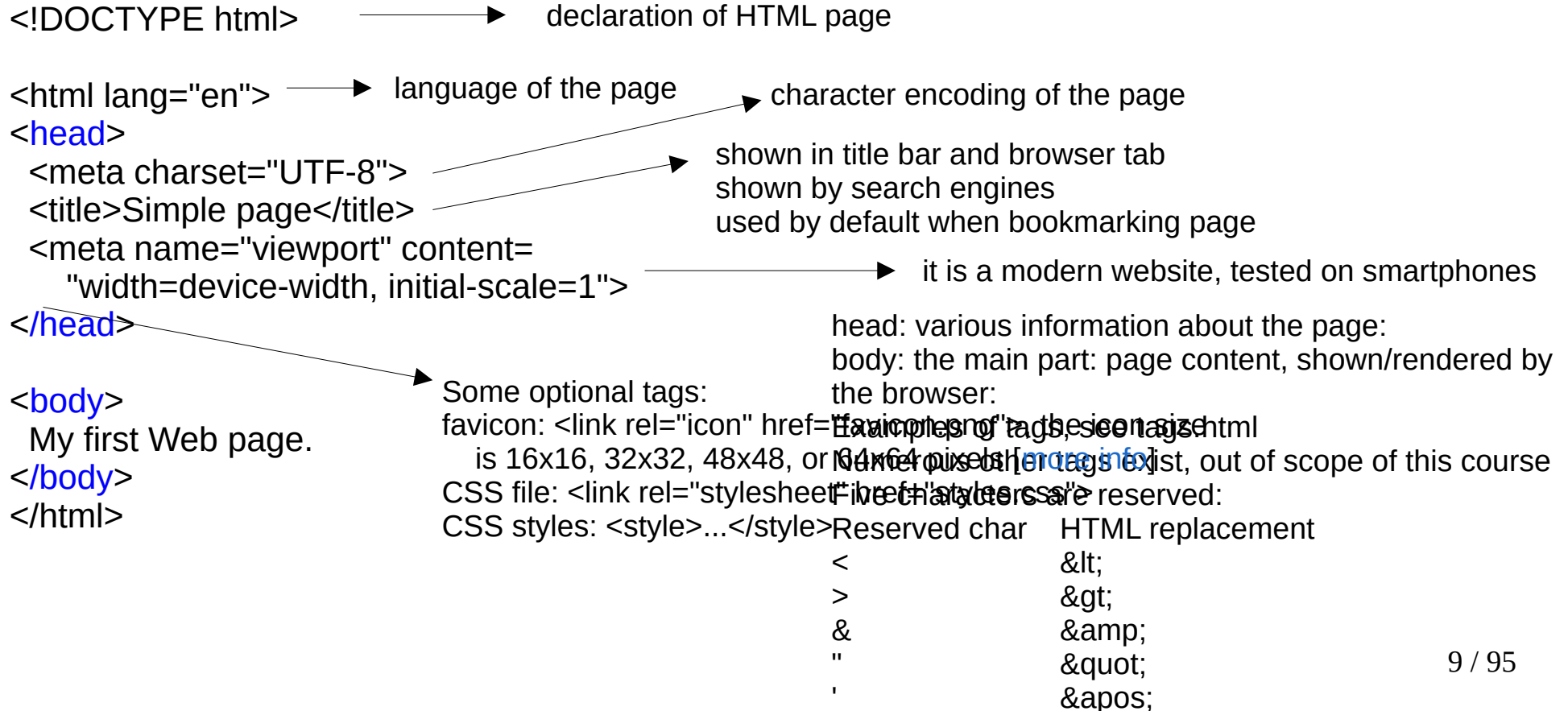
- Tutorial: <https://www.w3schools.com/html/>
- Exercises: <https://www.w3schools.com/html/exercise.asp>

Structure of an HTML page

- An HTML page is a **text** document (composed of visual characters), similar to a C or Python program, and contrary to a Word document or to an executable file for ex.
- In a browser, ctrl-u shortcut shows the page source
- It is recommended to use lower case for file names
- We will validate all the pages we write: <https://validator.w3.org>

The head part of the page

Minimal HTML page:

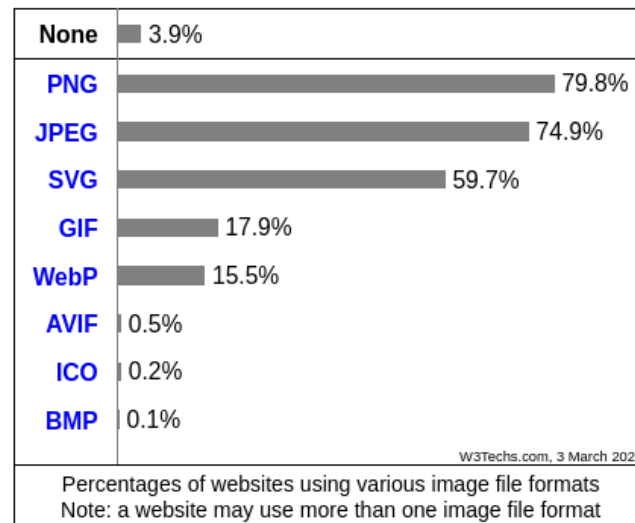


Some details about tags

- HTML element:
`<h1>Title</h1>`
 - h1 is the tag, and Title is its content
 - no space at left and right of the title
- Some elements do not have an end tag, for ex. `<hr>`
- Elements can have attributes
- `Link`
- Attributes are always put in the starting tag
- `name="value"`
- Some *global attributes* (which are available in **all** HTML elements):
 - title – supplementary information, rendered as tooltip
 - lang – the language used (`<p lang="fr">...`, or `Page in French`)
 - and others, see later

Images

- ``
- `loading="lazy"` – loads the page when image in or near the viewport
- **Raster (bitmap)** images, storing each pixel separately
 - non compressed: **bmp** – to avoid
 - compressed without information loss (*lossless*): **png** – best for computer-generated images, such as diagrams and text screen captures
 - compressed with loss of information (*lossy*): **jpg** – best for complex images, such as photos
 - **webp** allows compressed lossless and lossy
- **Vectorial** images, described as maths formulas
 - arbitrarily scalable without loss of quality
 - non compressed: **svg**



https://w3techs.com/technologies/overview/image_format

Letter 'e' magnified 7.5 times (12->90 pt)

stored as bitmap



stored as vector



Audio



```
<audio src="radio.ogg" controls>  
  Audio tag not supported.  
</audio>
```

```
<audio controls>  
  <source src="radio.ogg">  
  <source src="radio.mp3">  
  Audio tag not supported.  
</audio>
```

- Attributes: autoplay, controls, loop, src etc.
- Several codecs exist, such as **Opus**, **Vorbis**, and **MP3**; they differ by licence (free or non-free), compression ratio at various bitrates
- Use **Opus**, it is free and the most efficient
- Warning: browsers do not implement all the codecs, see [[wiki pedia](#)] => provide several audio files/formats

Video

```
<video src="movie.ogg" controls>  
  Video tag not supported.  
</video>
```



```
<video controls>  
  <source src="movie.webm"  
    type='video/webm; codecs="av1,opus"'>  
  <source src="movie.mp4" type="video/mp4">  
  <track src="subtitles_en.vtt">  
  Video tag not supported.  
</video>
```

- Attributes: autoplay, controls, height, width, loop, src etc.
- Container = file format; audio/video codec = audio/video compression method
- Several formats for codecs/containers exist, such as **AV1/WebM**, **VP9/WebM**, **HEVC/MP4**, and **H.264/MP4**
- Warning: browsers do not implement all the formats, see [[wiki pedia](#)] => provide several video files/formats

URL

- protocol://host/path/filename
 - protocol : http, https, file, ...
 - if file:, the browser reads itself the file from hard disk, **without** using the Web server
- Relative link: if **host** is not specified: on page <https://dedu.fr>, where does `ParSSAP` point to?
- If **file name** is not specified: what page is fetched when URL is <https://dedu.fr> or <https://dedu.fr/teaching>?
 - index.html (or index.php etc.) will be returned, if it exists; otherwise the server returns the list of files of the directory or an error

Web site hosting

- To host your Web site, you need a machine (server) connected non-stop to Internet, and usually a DNS name too
- **Either** your own machine, that you administer yourself (for ex. security updates)
- **Or** pay for an external hosting server:
 - **free** free for .free.fr, **hosterion** 4 €/month, **o2switch** 5 €HT/month, **ovhcloud** 1 €/month, **elasscloud** 5 €/month etc. etc.
 - usually they provide specific applications (PHP, MariaDB, FTP/SSH access etc.), backups, software updates (for security for ex.), high availability etc.

Web page evolution

- At the beginning (1991), text, links and images – we can click to surf to other pages!!
- CGI, PHP etc. – generate pages on the fly (dynamic pages)
- Forms – generate page based on user input
- CSS – separation of content and presentation
- ~2000: plugin – flash language to show video in browser
- JavaScript – browser animates page, JavaScript frameworks
- 2014 – JavaScript as standard language, audio and video, enhanced forms, event attributes
- 2016 – responsive images, personalised contextual menu etc.
- 2019 – no HTML version anymore, but living standard
- Future: integration with mobile applications, single-page applications?

1.2 CSS

Cascading style sheets (CSS)

- Tutorial: <https://www.w3schools.com/css>
- Exercises: https://www.w3schools.com/css/css_exercises.asp
- CSS describes how HTML elements are to be displayed
- CSS saves a lot of work: it can control the layout of multiple web pages all at once

HTML page:

```
...  
<head>  
  ...  
  <link rel="stylesheet" href="styles.css">  
</head>
```

```
<body>  
<p>A first paragraph.  
<p>A second one.
```

Without CSS:
A first paragraph.

A second one.

styles.css file:

```
body {  
  background-color: blue;  
}  
p {  
  font-style: italic;  
}
```

With CSS:
A first paragraph.
A second one.

CSS selectors

- Selector – specifies the elements to style
 - Examples:
 - p – all p elements
 - p:hover – when the mouse is over them
 - .my-style – all the elements with class="my-style"
 - p.my-style – all p elements with class="my-style"
 - #my-style – the element with id="my-style"
 - (id and class are global attributes)
 - p, h1, h2 – all p, h1, and h2 elements
 - [much more](#)
- ```
styles.css file:
h5, p {
 font-style: italic;
 text-align: center;
}
```

# CSS properties – text and links

- Block declaration – specifies the style to apply to the selector
- One or several declarations of type property: value;
- text-align: center; text-align-last: right;
- text-indent: 2em;
- letter-spacing: .2em;
- line-height: 1.3em; /\* space between lines \*/
- word-spacing: ...;
- text-shadow: ...;
- color: red;
- font-size: x-large; /\* small, large ... \*/
- font-weight: bold; font-style: italic;

```
styles.css file:
h5, p {
 font-style: italic;
 text-align: center;
}
```

```
a {
 color: hotpink;
}
a:link { /* unvisited link */
 color: red;
}
a:visited { /* visited link */
 color: green;
}
a:hover { /* mouse over link */
 color: hotpink;
}
a:active { /* selected link */
 color: blue;
}
```

# CSS properties – tables

- `table, th, td {  
border: 1px solid;  
}`

| Firstname | Lastname |
|-----------|----------|
| Peter     | Griffin  |
| Lois      | Griffin  |

- `table {  
border-collapse: collapse;  
}`

| Firstname | Lastname |
|-----------|----------|
| Peter     | Griffin  |
| Lois      | Griffin  |

- `td {  
text-align: left;  
vertical-align: bottom;  
}`
- `td, th {  
padding: ...;  
border-bottom: ...;  
}`
- See [table borders](#), alignment, style, responsive

# Three places for CSS declarations

## Inline (for a unique element)

- HTML page: `<p style="color: red">...`
- => applies only to this `<p>`

## Internal (for a page with unique style):

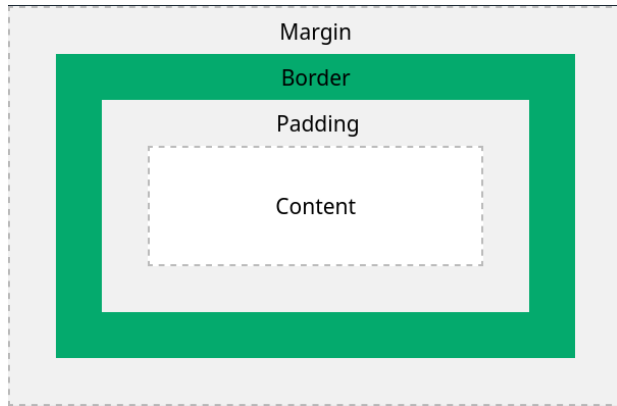
- HTML page: `<head>...<style>p {color: red}</style>...</head>`
- => applies to all `<p>` of the page

## External:

- HTML pages: `<head>...<link rel="stylesheet" href="styles.css">...</head>`
- `styles.css` file: `p {color: red}`
- => applies to all `<p>` of all the pages including `styles.css`
- => pages load faster, because the `.css` file is downloaded only once

Decreasing priority: inline, internal, external, browser

# CSS properties – box model, positioning, and z-index



- `border-bottom: 3px solid red;`
- `border-radius: 8px;`
- `padding: 1em;`
- `margin-top: 3em;`
- width and height apply to Content!

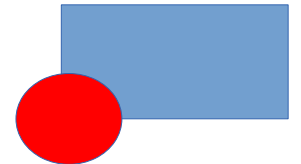
Positioning (`position: ...`):

- `static`
- `relative` (to its static position)
- `fixed` (relative to viewport/window)
- `absolute` (relative to its parent), e.g. a text at bottom right of an image
- `sticky` (stops moving and remains visible when scrolling)

[https://www.w3schools.com/css/css\\_positioning.asp](https://www.w3schools.com/css/css_positioning.asp)

z-index:

- blue rectangle: z-index: 1
- red circle: z-index: 2



# CSS properties – float

- **float: none**



Lorem ipsum dolor sit amet, consectetur adipiscing

elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor. Maecenas nisl est, ultrices nec congue eget, auctor vitae massa. Fusce luctus vestibulum augue ut aliquet.

- **float: left**



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor. Maecenas nisl est, ultrices nec congue eget, auctor vitae massa. Fusce luctus vestibulum augue ut aliquet. Mauris ante ligula, facilisis sed ornare eu, lobortis in odio. Praesent convallis urna a lacus interdum ut hendrerit risus

congue. Nunc sagittis dictum nisi, sed ullamcorper ipsum dignissim ac...

- **clear: both** – the next element is shown below the floats

- Normally, div elements are displayed one below the other
- With float: left, they float next to each other



# CSS properties – background, visibility

- background-color: blue;
- background-image: url("paper.png");
- opacity: 0.3;
- disabled: true;
- display: none (inline, block, ...), does not take space
  - <https://sources.debian.org/src/ekiga/4.0.1-6>, click to toggle, search for toggle in the source
- visibility: hidden (visible, ...), takes space

# CSS (selector) combinators

- `ul li` – descendant selector = all the `li` elements inside `ul` elements (descendants, i.e. children and children of children etc.)
- `ul > li` – child selector = all the `li` elements which are direct children of `ul`
- `div + p` – adjacent sibling ("frère/sœur") selector = all `li` right after (immediately following) an `ul`
- `div ~ p` – general sibling selector = all `li` after an `ul`

```
div p {
 background-color: yellow;
}
div > p {
 color: red;
}
div + p {
 font-style: italic;
}
div ~ p {
 font-weight: bold;
}
```

```
<p>Start
<div style="height:200px">Un div
 <p>Par inside div

 Alpha<p>Par inside second div
 Beta

</div>
<p>Un
<p>Deux</p>

<code>int a=0;</code>
<p>Trois
<p>Quatre
Alpha<p>Par inside second divBeta
<p>Hi
```

# CSS units

- Used in many cases: width, margin, padding, font-size etc.
- Absolute lengths: cm, px etc.
- Relative lengths:
  - em (= current element's font-size)
  - rem (= root element's font-size)
  - 1vw = 1% of viewport (~browser) width etc.
- Prefer relative lengths
- More info: [https://www.w3schools.com/css/css\\_units.asp](https://www.w3schools.com/css/css_units.asp)

# CSS specificity (rule priority)

- What is the colour of the text?

```
<style>
```

```
 #demo {color: blue}
```

```
 .test {color: green}
```

```
 p {color: red}
```

```
</style>
```

- `<p class="test" id="demo" style="color: pink">Hello!`

# Standard menu (uses <nav> and ul list)

In <body>:

```
<nav>

 Home
 Courses
 Projects
 Hobbies

</nav>
```

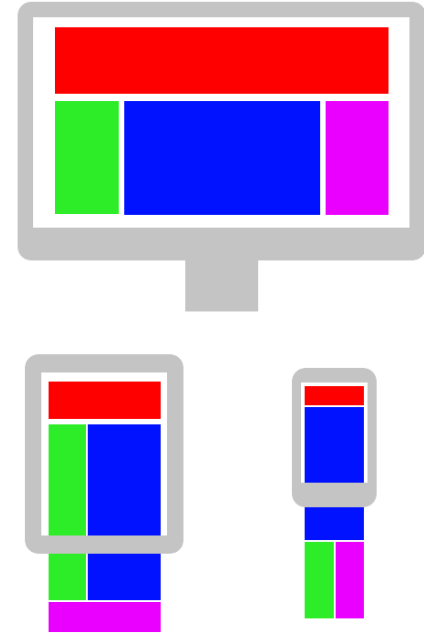
Example of style, in .css:

```
nav ul {
 padding: 0; /* remove space on left */
 list-style-type: none; /* remove bullets */
 overflow: hidden; /* text after menu shown at bottom */
 display: flex; /* show one next to the other */
 flex-wrap: wrap; /* menu on several lines if too long */
 background-color: grey;
}
nav li a {
 display: block;
 color: white;
 padding: 14px 16px;
 text-decoration: none; /* remove bottom blue line */
}
nav li a:hover {
 background-color: black;
}
```

Note that this style applies only to nav tag (the menu)!

# Responsive Web Design

- RWD = techniques to optimise reading and browsing by adapting the page to client characteristics:
  - different supports: screen (PC, tablet, smartphone), printed paper, Braille etc.
  - the same support can have different characteristics: width, height, orientation etc.



Source: [https://en.wikipedia.org/wiki/Responsive\\_web\\_design](https://en.wikipedia.org/wiki/Responsive_web_design)

# Real examples of RWD

- Bad examples: big image shown in small [[page perso](#)]
- `rwd.html`
- Adapt to window width
  - change layout: [meteoblue](#), look on desktop, and choose Tools->BrowserTools->ResponsiveDM for smartphone for portrait and landscape
  - change layout [[vtp](#)]
  - change number of columns in references and notes, shows/hides the table of contents [[wikipedia](#)]
  - change number of columns, replaces several buttons to only one, remove minor information if too small screen (weather), replace text search by a button etc., look at the top of the page and an article [[bostonglobe](#)]
  - affiche ou non des menus, change le nombre de colonnes et leur largeur [[agerpress](#), [tweakers](#), [erc](#)]
- Adapt to media type
  - the printed page does not have the upper, left, and right menus [[wikipedia](#)]
- Use as much as possible the browser's default font size (cf. *Preferences*); if needed, use em (1em = height of current font) instead of pixels or points

# Implementation of RWD: media specification

Present source code of rwd.html and css

- Use *media* (screen, print etc.) with different CSS files:

```
<link href="screen.css" media="screen"
rel="stylesheet">

<link href="print.css" media="print"
rel="stylesheet">
```
- The browser uses the CSS corresponding to the current media
  - note: both desktop and mobile browsers choose *screen*
- Specify media using one/several limiting expressions in parentheses (= media query)
- Limiting expressions: width&height (allows max- and min- too), orientation etc.

## 1. In the Web page:

```
<link rel="stylesheet" href="style.css"
media="(max-width: 80em)">
```

## 2. In the CSS file:

```
@media (max-width: 80em) {
...
}
```

## 3. In a separate CSS file, imported by the main CSS file:

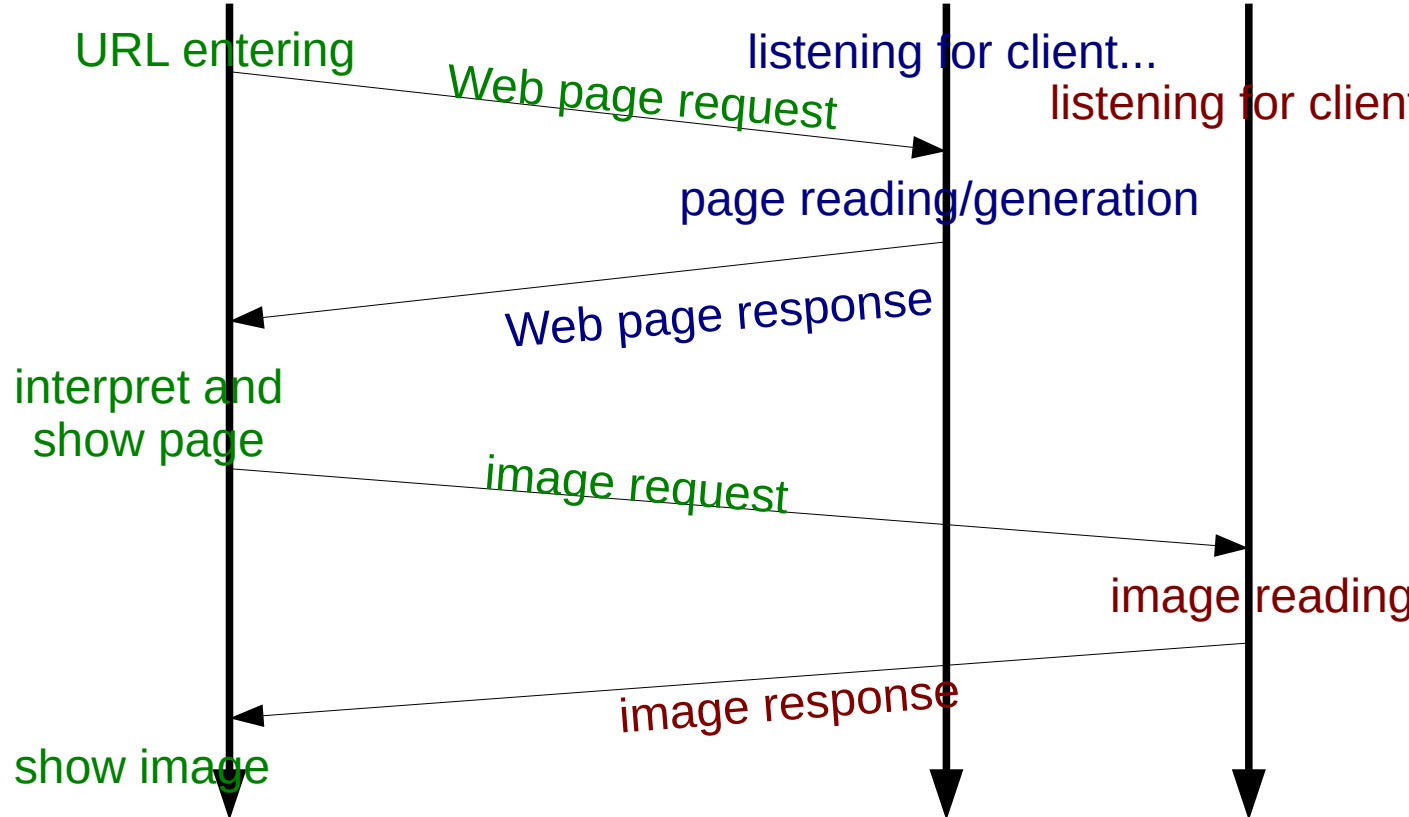
```
@import url("style2.css") (max-width: 80em);
```

# HTTP protocol, Web page transfer with an image

**Web client (browser)**

**Web server** **Web server 2**

HTTP is an application protocol,  
above QUIC/TCP and IP



```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>Simple page</title>
</head>
<body>
 My first page.

</body>
</html>
```

Other embedded objects:  
css, multimedia elements,

# HTTP headers

## Web client request:

GET / HTTP/1.1  
Host: eugen.dedu.free.fr  
User-Agent: Mozilla/5.0 (X11; Linux x86\_64; rv:125.0) Gecko/20100101 Firefox/125.0  
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,\*/\*;q=0.8  
Accept-Encoding: gzip, deflate  
DNT: 1  
Sec-GPC: 1  
Connection: keep-alive  
Upgrade-Insecure-Requests: 1

## Web server response:

HTTP/1.1 200 OK  
Date: Thu, 18 Apr 2024 15:10:53 GMT  
Server: Apache/ProXad [Jan 23 2019 20:05:46]  
Last-Modified: Wed, 17 Apr 2024 12:54:48 GMT  
ETag: "e10130bc-63af-661fc698"  
Connection: close  
Accept-Ranges: bytes  
Content-Length: 25519  
Content-Type: text/html

```
<!DOCTYPE html>
<html lang="en">
<head>
```

...

# Channel security with HTTPS, TLS

- **Security considerations:** HTTP data is transmitted in clear over network
- TLS sits between transport protocol and application
- TLS provides:
  - server authentication: through public key infrastructure
  - cryptography and data integrity: through symmetric cryptography
- Applications:
  - HTTPS = HTTP Secure = HTTP over TLS
    - configure Web server to use, port 443 by default
  - e-mail software, videoconferencing (VoIP), IM etc.
- All browsers have a CA (certificate authority) list, cf. Preferences->Privacy->Certificates
- When it contacts site S, it checks that the certificate given by the site is the same as the certificate of S given by CA
- More information: [[wikipedia](#)]

# Conclusions

- Main skills to have:
  - page skeleton, favicon
  - p, h... headings, hr, a absolute and relative, img, id and internal links, abbr, comment, b, i, small
  - table, list, audio, video
  - css selector and property, menu, rwd @media browser width
  - valid (errorless) page

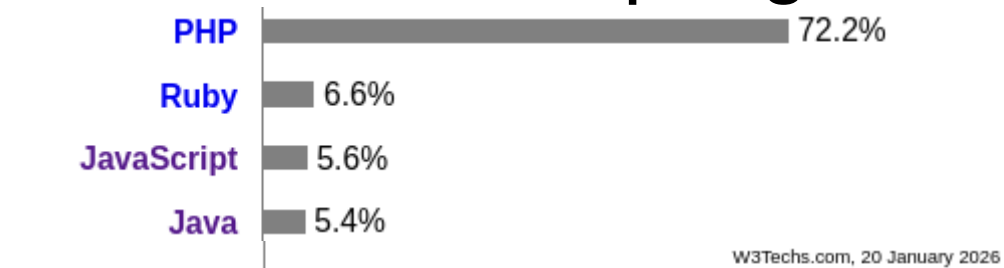
## 2. Dynamic Web pages

*"There are some kernel developers who would not speak to me again if I told them I was playing with web technologies"* (James Bottomley, heard at LinuxCon Japan, 2016)

# Modern Web sites are dynamic

- Nowadays, almost all of the Web pages on Internet are not static, but dynamic:
  - generated on-the-fly, using server-side scripting and a database, known as **back-end** programming
  - **and** animated or modified by the browser, using client-side scripting, known as **front-end** programming

## 2.1 Server-side scripting: PHP



Percentages of websites using various server-side programming languages

Note: a website may use more than one server-side programming language

[https://w3techs.com/technologies/overview/programming\\_language](https://w3techs.com/technologies/overview/programming_language)

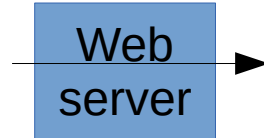
# PHP

- Huge documentation on PHP
- Tutorial: <https://www.w3schools.com/php/>
- Exercises: <https://www.w3schools.com/php/exercise.asp>
- Online Web sites to test simple PHP programs:
  - [https://www.tutorialspoint.com/php\\_webview\\_online.php](https://www.tutorialspoint.com/php_webview_online.php)
  - [https://www.w3schools.com/php/phptryit.asp?filename=tryphp\\_intro](https://www.w3schools.com/php/phptryit.asp?filename=tryphp_intro)
  - replit.com etc.
  - install a Web server and the PHP plugin (e.g. LAMP) on your machine
  - or use the rt-projet server provided by the university
- PHP function reference: <https://php.net/manual/en>

# On-the-fly page generation with PHP

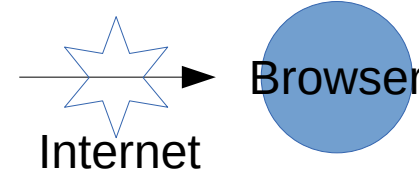
## example.php file:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>Simple page</title>
</head>
<body>
 A table:
 <table>
 <?php
 for ($i=0 ; $i < 4 ; $i ++)
 echo "<tr><td>Iteration $i";
 ?>
 </table>
</body>
</html>
```



## Web page generated:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>Simple page</title>
</head>
<body>
 A table:
 <table>
 <tr><td>Iteration 0
 <tr><td>Iteration 1
 <tr><td>Iteration 2
 <tr><td>Iteration 3
 </table>
</body>
</html>
```



## Rendered page:

A table:  
Iteration 0  
Iteration 1  
Iteration 2  
Iteration 3

- The script is executed upon page request
- It is executed **on the server** (can take time)
- It is executed only **once** per request, to generate the HTML page
- The PHP interpreter **executes** each PHP block and **replaces** it with its output
- The script outputs HTML text

# PHP blocks

- Between `<?php` and `?>`
- `<?= $i ?>` is a shorter form of `<?php echo $i ?>`
- You can include several PHP blocks in the page

## **example.php file:**

```
<table>
<?php
 for ($i=0 ; $i < 4 ; $i ++)
 echo "<tr><td>Iteration $i";
 echo "</table>";
?>
<?php $v = 5; ?>
There are <?= $v ?> lines.
```

## **Web page generated:**

```
<table>
<tr><td>Iteration 0
<tr><td>Iteration 1
<tr><td>Iteration 2
<tr><td>Iteration 3
</table>
There are 5 lines.
```

# Comments, variables and constants

- Comments: `//` and `/* ... */`
- All instructions end with semicolon (`;`)
  - the `;` before closing tag `?>` is optional
- Variables:
  - start with `$`
  - there is no declaration
  - numerical operators: `+`, `-`, `*`, `/`, `%`, `++`, `--`
  - combined operators: `+=`, `-=`, `*=`, `/=`, `%=`, `.=`
  - boolean operators: `<`, `<=`, `>`, `>=`, `==`, `===`, `!=`, `!==` (value and type), or `||`, and `&&`, `xor`, `!`
- Constants:
  - `define ("NB_STUDENTS", 10);`
  - their scope is global (usable everywhere in the script)

# Control structures

- if, switch, while, for, do...while

- break, break n, continue, continue n

```
for ($i=0 ; $i<10 ; $i++)
 if ($i == 4)
 break;
```

```
for ($j=0 ; $j<10 ; $j++)
 if ($j == 4)
 continue;
```

- match (= switch plus facile à utiliser)

```
switch ($statusCode) {
 case 200:
 case 300:
 $message = null;
 break;
 case 400:
 $message = 'not found';
 break;
 case 500:
 $message = 'server error';
 break;
 default:
 $message = 'unknown code';
 break;
}
```

- Ternary operator

```
$type = ($age>=18 ? 'major' : 'minor');
```

```
if ($age>=18)
 $type = 'major';
else
 $type = 'minor'
```

```
// PHP >= 8.0 only
$message = match ($statusCode) {
 200, 300 => null,
 400 => 'not found',
 500 => 'server error',
 default => 'unknown code',
};
```

```
// it doesn't require a break
statement
// it can combine different arms
into one using a comma
// it returns a value, so you only
have to assign value once
```

# Strings

- 'A string' or "A string"
- \$v=5; echo "v = \$v"; echo 'v = \$v';
  - " replaces variables and interprets \\, \n, \r, \t
- Concatenation: \$x = 'I live in ' . 'Amman'
- \$x[0]
- Some functions: strtolower, strtoupper, trim, strlen, strpos, substr, nl2br

# Arrays

Stores multiple values (even of different types) in one single variable

**Indexed** arrays: (n,value), index n is an integer number

Compared to C++:

- elements can have different types
- first index can be any positive or negative value (instead of 0)
- dynamic size

```
$t = array (10, 100, "kiwi"); // initialise with several values
```

```
print_r ($t); // Array ([0] => 10 [1] => 100 [2] => kiwi)
```

```
// print_r shows human-readable information
```

```
echo $t[1]; // 100
```

```
echo $t[100]; // nothing shown
```

```
$t[2] = 12; // change type
```

```
print_r ($t); // Array ([0] => 10 [1] => 100 [2] => 12)
```

```
$u[3] = 5;
```

```
$u[] = "hello"; // assign after last index
```

```
print_r ($u); // Array ([3] => 5 [4] => hello)
```

```
echo count ($t) . " " . count ($u); // 3 2
```

**Associative** arrays: (key,value), key is a string

```
$t = array ('kiwi'=>5, 'fig'=>10, 'apple'=>'x');
```

OR

```
$t['kiwi'] = 5; $t['fig'] = 10; $t['apple'] = 'x';
```

```
print_r ($t); // Array ([kiwi]=>5 [fig]=>10 [apple]=>x)
```

```
echo $t['fig']; // 10
```

```
echo $t['peach']; // nothing shown
```

```
echo $t[100]; // nothing shown
```

```
$t['kiwi'] = 12;
```

```
print_r ($t); // Array ([kiwi]=>12 [fig]=>10 [apple]=>x)
```

```
// loops
```

```
foreach ($t as $val)
```

```
 echo $val . " "; // 12 10 x
```

```
foreach ($t as $key=>$val)
```

```
 echo "($key,$val) "; // (kiwi,12) (fig,10) (apple,x)
```

# Function declaration

- `function xyz () {...}`
- `function abc ($param1, $param2) {...}`
- `return ...;`

## Default parameters:

```
function ht ($price, $tax=19.6) {
 return round ($price * (1-$tax/100), 2);
}
...
echo "Price HT = " . ht (100, 5.5);
echo "Price HT = " . ht (100);
```

## Global vs local variables:

```
function show () {
 global $v; // get variable from outside
 echo $u; // shows nothing
 echo $v; // shows 2
}
$u = 1;
$v = 2;
show ();
```

# Miscellaneous functions

- mail (\$to, \$subject, \$message)
- \$scrtdate = date ("Y-m-d");
  - look at the reference of date function
- exit ("Ended") or die ("Ended")

# File inclusion

- You can write some PHP code in a separate file (for ex. the site header, used by all the Web pages), which is included in PHP pages
- `require_once 'header.php';`
  - other methods: `require 'header.php'; include 'header.php'; include_once 'header.php'`
- **Security considerations:** what vulnerability do you see below?
  - `a.php:`

```
if (password != 'pass')
 exit ('Access denied');
else // password ok
 require_once 'header.php';
```
  - `header.php:`

```
echo 'Logged in success';
// remove data etc.
```
- Included files must not be accessible to user using their URL (`http://a.fr/header.php`)
  - solutions: store included files outside the Web hierarchy (`require_once './header.php'`); check if variable is initialised (`if (!isset($inc)) exit()`), etc.

# Forms

[1st page](#), showing the HTML form and allowing user to send data to server

Name:

Password:

I have a bike

I have a car

Yes

No

[2nd page](#), processing user's input and generating the results page:

Your input is: name=Dupont, bike=yes  
Here is the result of the research:

red bike: 200 €  
white bike: 250 €  
...

# Forms, 1st page (showing the form)

```
<form action="processing.php">
Name: <input type="text" name="name">

```

```
<input type="checkbox" name="bike">I have a bike

<label><input type="checkbox" name="car">A car</label>

```

```
<input type="radio" name="cat" value="yes">Yes

<input type="radio" name="cat" value="no">No

```

```
<input type="submit" value="Submit">
</form>
```

- Numerous types:
  - range, password, email, number, date
- Attributes to avoid bad user experience:
  - autofocus
  - placeholder="dd-mm-yyyy"
  - required
  - min, max, step
  - size, maxlength
  - specific validation (e.g. only odd numbers are allowed) can be done using JavaScript, as shown later

# Forms, 2nd page (processing the form)

- Generates the second page on-the-fly
- Use `$_GET` global variable to read user input
  - user types "toto" in form's field with `name="msg"`
  - the 2nd page is called as `.../processing.php?msg=toto`
  - echo `$_GET["msg"]` shows toto

**processing.php** file:

```
<?php
 // show all the parameters
 foreach ($_GET as $key => $val)
 echo "$key = $val
";
?>
```

# GET vs POST method

- Two methods to send parameters from client to server:
  - GET: parameters are appended to the URL: `processing.php?name=John&bike=on`
  - POST: parameters are sent in the HTTP header, not shown by the browser (but still easily found by a malicious user!!): `<form ... method="post">` in HTML, and `$_POST` in PHP
- POST use:
  - when the action must not be repeated, e.g. buying on Internet
  - for large volume of data (> 2kB in practice)
  - other specific cases
- GET use:
  - in all the other cases
  - advantage: allows to save the URL (ADE RT, my OMNI colleagues) and automate tasks (e.g. show map to compute distances `viamichelin` for OM)!

# Form processing security

- **Security considerations (input data):** `$_GET` and `$_POST` get data from users, **never trust user input data!**
  - if only some values are valid (e.g. a student's mark, integer between 0 and 20), then check them before use, for ex:  
`$name = preg_match ('/^[a-zA-Z ]*$/ ', $_GET['name']);`  
`$mark = preg_match ('/^1?[0-9]$/ ', $_GET['mark']);`
  - if all alphanumeric characters are valid (e.g. a user comment), then escape problematic characters, using for ex. `htmlspecialchars` or `htmlentities`

# Client-side storage, cookies

## Generalities about cookies

- Web pages are by default stateless: no information is passed between pages, each page is independent of the other pages, starts from zero; but then, how can the login be passed from one page to another? Examples with DOM, Gmail, FaceBook
- Cookies allow an application to identify a user throughout all the pages of the Web site
- A cookie is a string of characters stored on **client** (browser) but read and written by the **server**
- Cookies are also extensively used to track users, problem with privacy (Google is no 1, it is everywhere!)
- At each HTTP request, the browser sends **all** the cookies associated to that domain

### Cookie table in browser:

domain	name	value
-----		
a.fr	connect	010124
b.ro	name	Peter
c.com	name	John
c.com	age	37
...		

## Working with cookies in PHP

- The PHP program on server can read the cookies (already received by server) and write cookies (to be sent to client along with the generated page)
- Reading a cookie:
  - `echo $_COOKIE['name'];`
- Writing/updating and removing a cookie:
  - `setcookie ("name", "Jean"); // exists only for the session`
  - `setcookie ("age", "37", time()+86400); // exists for 86400s=24h`
  - `setcookie ("age", "38", time()-3600); // remove cookie (older date)`
  - **no** HTML character must be sent before calling setcookie!!
- More information: [https://www.w3schools.com/php/php\\_cookies.asp](https://www.w3schools.com/php/php_cookies.asp)
- Server-side storage: session? TODO

# Cookies, security

- **Security considerations (input data):** Cookies are stored on browsers, so user can modify them => never trust cookie values
- Check validity of cookie value with `preg_match`, as shown previously
- Think of what you store in a cookie: is it a good idea to store the login (ededu or jvaljean) in a cookie?
  - by changing the cookie in his browser (for ex. using "Web developer Tools"->Storage in firefox) a malicious user can pretend being any user
  - how do you fix this vulnerability?

## 2.2 Accessing databases using PHP

# Usefulness of DB

- Dynamic pages generally take data from DB, examples: wikipedia, qwant, covidtracker
- Provides persistent, and optimised, storage of data, in files
- Several standards to access a DB, we will use SQL, the most popular
- Several incompatible SQL server implementations exist, we will study MariaDB/MySQL, the most popular
  - SQL exercises: <https://www.w3schools.com/sql/exercise.asp>

# SQL DB organisation

- A DB server manages several DB
- A small Web site uses one DB, e.g. students
- A DB has a **structure** and **data**
- Structure:
  - a DB contains tables, but also indexes etc.
  - a table has columns
  - a column has a name and a type, e.g. VARCHAR(30), CHAR(30), INT, FLOAT, DATE
- Data:
  - data is stored as rows of tables, called **records**

**Database** students

Table marks                      Table ...

name	mark
Rami	10
Raed	9
Jane	3
Peter	7

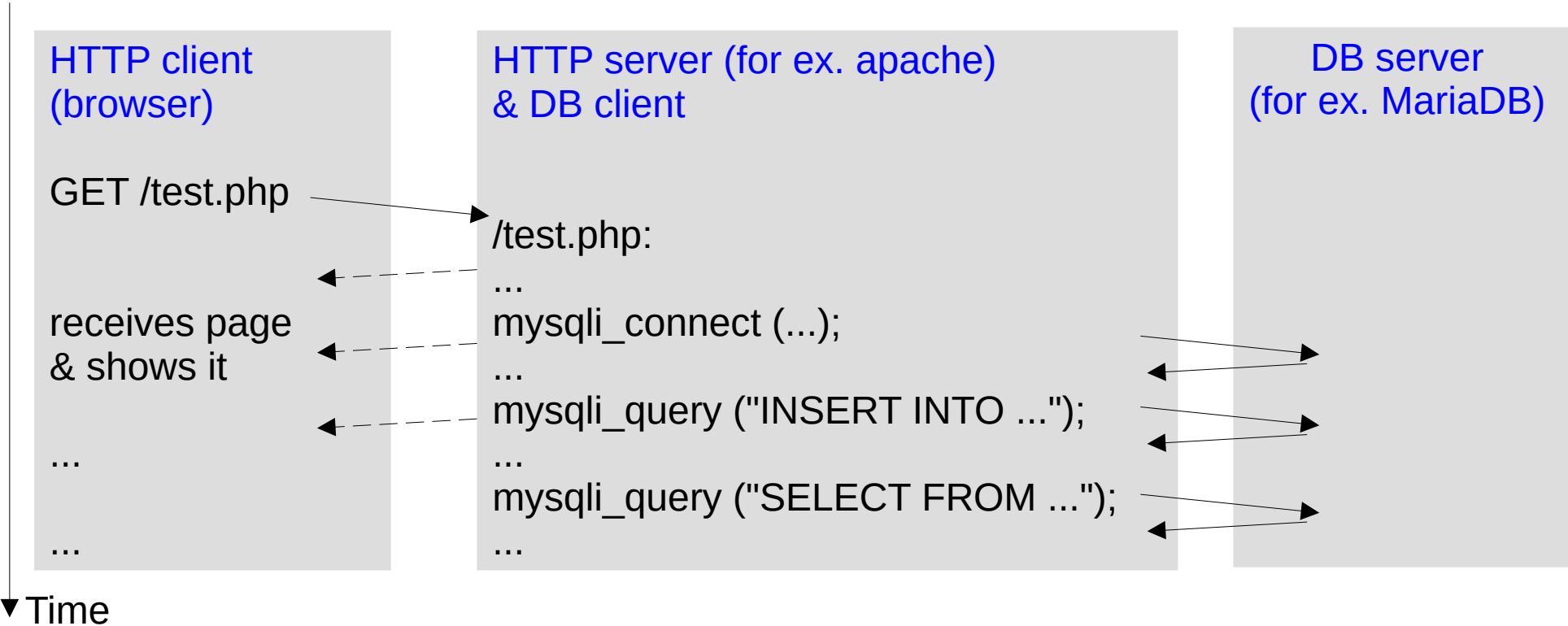
# DB structure creation

- Using phpmyadmin Web application (visual) <-- we will use this, because it is simpler
  - Using SQL queries in PHP or using mysql command (in GNU/Linux):
    - SHOW DATABASES;
    - DROP DATABASE dbname;
    - DROP TABLE table\_name;
    - NOT NULL
    - UNIQUE
    - primary key – uniquely identifies a row
    - index – speeds up data SELECT
    - AUTO\_INCREMENT
- ```
CREATE DATABASE students;
CREATE TABLE marks (
    name VARCHAR(30) UNIQUE NOT NULL,
    mark INT
);
```

Accessing DB using PHP

- Tutorial: https://www.w3schools.com/php/php_mysql_intro.asp
- PHP provides two API (ways) to access a DB:
 - mysqli extension:
 - procedural style <-- we will use this
 - object-oriented style
 - PHP Data Objects (PDO)
- Usually, we execute SQL queries from PHP
 - `mysqli_query (... , $query);`
 - four main SQL operations on data: select, insert, update, delete

3-tier architecture, PHP – DB exchanges



Data – record insertion, update and removal in PHP/MariaDB

```
// open DB
$ct = mysqli_connect ('localhost', 'login', 'pwd', 'dbname');
if ($ct == null) exit ("Unable to connect");
```

```
... // work with DB data
```

```
// close DB
mysqli_close ($ct);
```

the two columns
in the order in DB

| name | mark |
|-------|------|
| Rami | 10 |
| Raed | 9 |
| Jane | 3 |
| Peter | 7 |

Record insertion:

```
$ret = mysqli_query ($ct, "INSERT INTO marks VALUES ('Raed', '9')");
if ($ret == null) exit ("Unable to execute query: " . mysqli_error ($ct));
```

Record update:

```
$ret = mysqli_query ($ct, "UPDATE marks SET mark='10' WHERE name='Raed'");
if ($ret == null) exit ("Unable to execute query: " . mysqli_error ($ct));
```

Record removal:

```
$ret = mysqli_query ($ct, "DELETE FROM marks WHERE name='Raed'");
if ($ret == null) exit ("Unable to execute query: " . mysqli_error ($ct));
```

Data – record retrieval in PHP/MariaDB

Data retrieval and processing (the most used operation!):

```
$result = mysqli_query ($ct, "SELECT * FROM marks WHERE mark>='9'");  
if ($result == null) die ("Unable to execute query: " . mysqli_error ($ct));  
while ($row=mysqli_fetch_assoc ($result)) // associative array  
    echo $row['name'] . " " . $row['mark'] . "<br>";
```

Functions similar to `mysqli_fetch_assoc`:

```
// indexed-only array (faster I think)  
// $nbLines = mysqli_num_rows ($result);  
while ($row=mysqli_fetch_row ($result))  
    echo $row[0] . " " . $row[1] . "<br>";
```

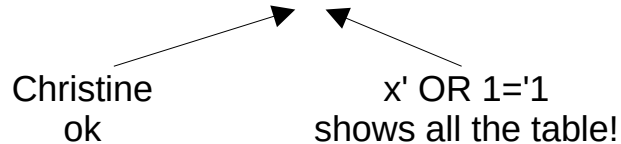
```
// indexed and associative array (slower I think)  
while ($row=mysqli_fetch_array ($result))  
    echo $row['name'] . " " . $row[1] . "<br>";
```

| name | mark |
|-------|------|
| Rami | 10 |
| Raed | 9 |
| Jane | 3 |
| Peter | 7 |

DB security, SQL injection

- **Security considerations (damage):**
- Attack (if input data is not checked before use):
 - let's have a page with a text field called name for the name to search
 - the attacker's goal is to make the text he enters interpreted as a command
 - the following PHP code is vulnerable:

```
$req = "SELECT * FROM student WHERE name=' " . $_GET['name'] . " '";
```



- **Solutions:**
 - use preg_match if only some characters should be allowed
 - replace `$_GET['name']` by `mysqli_real_escape_string (... , $_GET['name'])`
 - prepared statements

2.3 Client-side scripting: JavaScript



Percentages of websites using various client-side programming languages

Note: a website may use more than one client-side programming language

https://w3techs.com/technologies/overview/client_side_language

Client-side use cases

- Example: a PHP page showing current time does not update each second
Current hour is: `<?= date("H:i:s") ?>`.
Current hour is: 11h20'25".
- Loading new page content or submitting data to the server via Ajax without reloading the page (for example, a social network might allow the user to post status updates without leaving the page)
- Animation of page elements, fading them in and out, resizing them, moving them, etc.
- Interactive content, for example games
- Validating input values of a Web form to make sure that they are acceptable before being submitted to the server
- Transmitting information about the user's reading habits and browsing activities to various websites. Web pages frequently do this for Web analytics, ad tracking, personalization or other purposes

JavaScript

- Huge documentation on JavaScript
- Tutorial: <https://www.w3schools.com/js/>
- Exercises: https://www.w3schools.com/js/exercise_js.asp

Example

- Inline code

```
<script>  
  "use strict"; // perform additional JS checks  
  alert ("Hi!");  
</script>
```

- External file:

```
<script src="a.js"></script>
```

a.js file:

```
alert ("Hi!");
```

- To see **errors**, show the console:
Tools->BrowserTools->WebDeveloperTools (or F12)

- Code is executed at its position, when reading the page

- to avoid errors, declare functions **before** calling them
- to execute code at the end of page loading, simply place it at the very end of the page:

```
...  
<script>  
...  
</script>  
</body>  
</html>
```

Comments, variables, control structures, operators, and strings

- Comments: `//` and `/* ... */`
- Like in PHP, all instructions end with semicolon (`;`)
 - right before `</script>` it is optional
- Variables: `let val = 5;`
- Constants (can be initialised only once): `const maxMark = 20;`
- Control structures (if, while, ...): like in PHP/C++
- Operators (+, -, ...): like in PHP/C++
+

Strings:

```
let s = "hello world"; // or 'hello world'
alert (s.length); // 11
alert (s.charAt(1)); // "e"
alert (s.substring(3,7)); // "lo w"
alert (s.indexOf("o")); // 4
alert (s.toUpperCase()); // HELLO WORLD
alert (s + "!"); // hello world!
```

Arrays and functions

Arrays:

- index is integer, starting with 0
- values can be of different type
- dynamic size

```
let colours = ["red", "blue", "green"];
colours[0] = "yellow";
alert (colours.length); // 3
// add 4th colour
colours[colours.length] = "black";
```

Functions

- Like in PHP:
function xyz (param1, param2) {...}
- Default parameters: like in PHP

Scope of variables: global visibility, except if declared inside a function

```
function test () {
    let vlocal = "hi"; // exists only in this function
    vglobal = "hi"; // ok, global variables are visible
}
let vglobal; // global variable
test ();
alert (vlocal); // error
alert (vglobal); // hi
```

BOM: window, timers, dialogue boxes

Browser Object Model: interaction with browser

Dialogue boxes:

```
alert ('Hi!');
```

```
if (confirm("Are you sure?"))  
    alert ("I am glad");  
else  
    alert ("I am sorry");
```

```
let yourname = prompt ("Your name?", "abc");  
if (yourname != null)  
    alert ("Welcome " + yourname);
```

Console:

```
console.log (); // debugging message, press F12  
to show console
```

Timers:

```
function hello () {  
    alert ("Hello world");  
}  
setTimeout (hello, 1000);  
  
setInterval (hello, 500);
```

```
let num=0;  
let i = setInterval (repeatCall, 500);  
function repeatCall () {  
    if (num++ < 4)  
        alert (num);  
    else  
        clearInterval(i);  
}
```

DOM: selecting HTML elements, changing their style and content

Document Object Model: API to access HTML page

```
document.title = "New title";  
alert (document.URL);
```

```
<p id="x">A paragraph.  
<p>A second paragraph.
```

```
<script>  
// select a unique HTML element, by its id  
let p = document.getElementById ("x");  
  
// change HTML elements  
p.style.color = "blue";  
// compare to background-color in CSS  
p.style.backgroundColor = "red";  
p.innerHTML = 'Modified paragraph';  
</script>
```

```
<ul>  
<li>One  
<li class="col">Two  
<li class="col">Three  
</ul>
```

```
<script>  
// select multiple elements, by their tag name / class  
let list = document.getElementsByTagName ("li");  
// let list = document.getElementsByClassName ("col");  
  
// change HTML elements  
for (let i=0; i<list.length; i++)  
    list[i].style.color = "green";  
</script>
```

DOM: adding and removing HTML elements

Adding a text:

```
let txt = document.createTextNode ("Some text");  
document.body.appendChild (txt);
```

Adding a table:

```
let table = document.createElement ("table");  
for (let i=1; i<=10; i++) {  
  let row = document.createElement ("tr");  
  for (let j=1; j<=10; j++) {  
    let cell = document.createElement ("td");  
    cell.appendChild (document.createTextNode (i*j));  
    row.appendChild (cell);  
  }  
  table.appendChild (row);  
}  
document.getElementById("x").appendChild (table);
```

Removing an element:

```
document.getElementById("x").remove ();
```

DOM: events

- Event = something provoked by user (see below) or browser itself (timer)
- Examples of events: window (onload, onresize), form (onchange, onfocus, onblur, onsubmit), keyboard (onkeypress), mouse (onclick, ondblclick, onmouseover/onmouseenter, onmouseleave), media (onplay, onpause, onended)
- When an event happens (is triggered), its handler function is called
- DOM level 0, inline model:
 - uses element's attributes
`<h1 onclick="alert('Click')">...</h1>`
`<video onpause="f()" ...>...</video>`

Two other ways to specify event handler:

DOM level 1, traditional model:

- the handler is added or removed by scripts

```
<h1 id='p'>...
```

```
document.getElementById('p').onclick = myfct;
```

```
window.onload = myfct;
```

DOM level 2:

- allows several handlers for a same event

DOM: event handling function, form validation

- Some events lead to actions
- For them, if the handling function returns false, then the action is not carried/followed
- Examples of events with action:
 - onclick – does not follow the link
`Disabled text`
 - onkeypress – ignores character
`<input type="text" onkeypress="if (event.key == 'x') return false" ...>`
 - onsubmit – validates form, does not go to the 2nd page (does not submit form to server)

Example of form validation (validate input data before leaving form page):

```
<script>
function validation (form) {
    let age = form.age.value;
    if (age != "" && age>=0 && age<=120)
        return true;
    return false;
}
</script>
```

```
<form onsubmit='return validation(this)';>
    Your age: <input type="text" name="age">
    <input type='submit'>
</form>
```



Graphics on a Web page



Rectangular regions can be defined in HTML, allowing to **draw** various graphics on it (text, line, rectangle, circle/ellipse, gradient, colour, image) and animate or interact with them using JavaScript, CSS etc.

As **bitmap** (using canvas)

```
<canvas id="x" width="100" height="100"></canvas>
```

```
<script>
```

```
let example = document.getElementById ("x");
```

```
let context = example.getContext ("2d");
```

```
context.fillStyle = "red";
```

```
context.fillRect (0, 0, 100, 100);
```

```
context.strokeStyle = "blue";
```

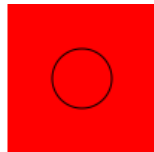
```
context.beginPath();
```

```
// x centre, y centre, radius, starting angle, ending angle
```

```
context.arc (50, 50, 20, 0, 2*Math.PI);
```

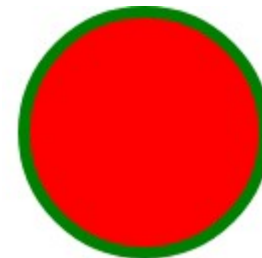
```
context.stroke ();
```

```
</script>
```



As **vectors** (using SVG)

```
<svg width="100" height="100"  
  xmlns="http://www.w3.org/2000/svg">  
  <circle cx="50" cy="50" r="40" fill="red"  
    stroke="green" stroke-width="4" />  
</svg>
```



Client-side storage: cookies and Web storage

Cookies, found on **client**, but usually used **server-side** and hence transmitted server<-->client, max around 4 kB; JavaScript can access them:

- `document.cookie = "name=John"; // create/modify a cookie`
- `alert (document.cookie);`
- to get a specific cookie:
`a = explode (";", document.cookie);`
`alert (a['name']);`
- new way, using asynchronous CookieStore API:
 - `cookieStore.set("name","John")`
 - `cookieStore.get("name")`
 - `cookieStore.delete("name")`

Web storage, found on **client** and read only **client-side**, allows much more space for storage:

- `localStorage.name = "John"; // create/modify an item`
- `localStorage.removeItem ("name"); // remove item`
- `sessionStorage` works similarly, but is removed by browser when tab closes

IndexedDB – store (large) data locally in browser as a DB

Web page updates

- Examples:
 - a search engine which shows completion during user input [google] – is it client-driven or server-driven?
 - what is interesting about the live update [match scores at [sofascore](#), [congés](#)], and what new notions does it use? – is it client-driven or server-driven?
 - real-time notifications (such as a chat), live data updates, progress of an action, IoT device status etc.
- Describes what happens **after** loading the page: client asks/gets new data from server, and updates part of the page (without reloading the whole page)
- Two ways, depending on which end initiates the update:
 - client-initiated (data **pull**, asks the server about new data)
 - server-initiated (data **push**, once new data is available, send it to client)
- Browser security:
 - what happens when a script asks a Web page in another domain (to make a bank transfer for example) and the user is logged in?
 - answer: same-origin policy (SOP), CORS

Client-initiated updates (data pull): AJAX

- AJAX, Asynchronous JavaScript and XML, is a method to use JavaScript in Web pages to **send HTTP requests** to server and use the response to **update the page**

Script in the Web page, executed by the client:

```
// using promises
setTimeout (a, 3000);
async function a() {
  let resp = await fetch ('ajax.php');
  let user = await resp.text ();
  alert (user); // Returned text
}

// using callbacks
let req = new XMLHttpRequest ();
req.onload = function () {
  // callback function
  alert (this.responseText);
  // 'Returned text'
};
req.open ('get', 'ajax.php');
req.send ();
```

ajax.txt file, on the server:

Returned text

Possible evolutions:

- ajax.php?param=value
- richer data formats: text, XML, JSON, ...

Server-initiated updates (data push): Server-Sent Events (SSE)

- Problem of AJAX with the bouncing ball from machine to machine (clients have to ask regularly, too many connections)
- How do AI models answer to your Web page?
- Clients get automatic updates from server as soon as the new data exists (real-time notifications)
- Describes how servers can initiate data transmission towards clients **once an initial client connection has been established**

On client:

```
let source = new EventSource ('updates.php');  
// listen to updates  
source.onmessage = function (event) {  
    alert (event.data);  
};
```

On server, file updates.php:

```
<?php  
header ('Content-Type: text/event-stream');  
header ('Cache-Control: no-cache');  
  
while (true) {  
    $nb = rand (1, 10);  
    // send data: start with "data:" and end with "\n\n"  
    echo "data: The random nb is $nb\n\n";  
    flush ();  
    sleep (5);  
    // if (connection_aborted()) break; ????  
}  
?>
```

JavaScript security

- **Security considerations (damage):** XSS (Cross-Site Scripting)
 - inject client-side scripts written by a user in the page viewed by other users, to be interpreted by them
- **Example:**
 - a site where the users, after login, can add comments
 - M notices that if he adds the comment **I like flowers!**`<script src="...">`, the text is shown and the scripts **is executed**
 - each user who looks at the page will see the text, and the script will execute, even without noticing it
 - the script obtains the authentication cookie for ex. and sends it to M's server, allowing him to be like the user
- **Real case:** [bugzilla](#)
- **Problem:** the browser/server executes a user's **code** in the context of **another** user
- **Solution:** escape special characters: <, >, ", &, ', but others too, in all user entries (URL included)

Client-side vs server-side scripting

- Tic-tac-toe in the same computer or in network, calculator, currency converter, chess against computer – which side is better?
- Code visibility: if client-side, **all code** is visible to the user; if server-side, code is invisible (e.g. passwords, high quality algorithm, safe access to DB and sensitive files on server)
- Server-side consumes resources on server
- Client-side – beware of client incompatibility

2.5 Web security

Reiteration on **Web security**

- Internet is a very dangerous place!!
- Internet is used by all kinds of people, from best to worst
- Where can user input data, and what damage can he do?
- **Input data:** Golden rule: always check and sanitise data from user!
 - input data: form parameters (GET/URL and POST data), cookies, even database
- **Damage:** Where does the problem show up?
 - at other users (e.g. their password discovery)
 - on the server (e.g. database data removal)

2.6 Notions of modern dynamic Web

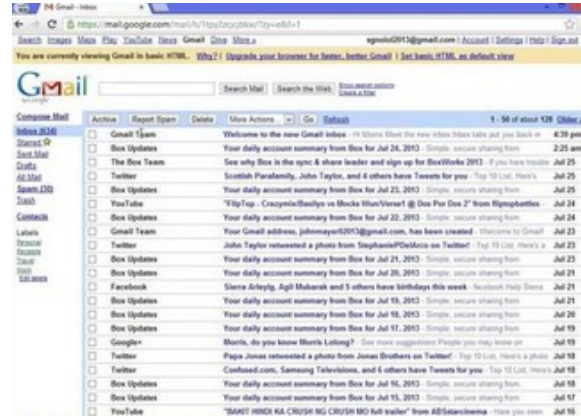
Need of (higher-level) frameworks

- Web applications have lots of various parts: form handling, HTML template processing, session management, database access, authentication, internationalisation, and so on
- Frameworks provide solutions for some or all of those parts

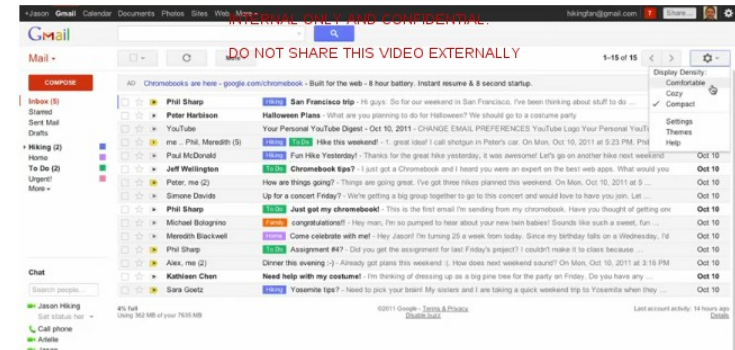
Server-side scripts

- The past: CGI, JSP, ASP.NET
- The current: PHP with frameworks: symfony, zend
- Modern framework: Node.js
- Evolves much slower than client-side scripting

Client-side scripts –the past–



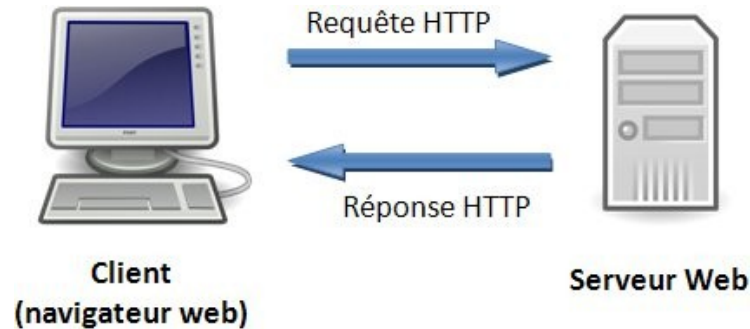
As of today



- Vanilla JS (pure JS)
- Web 2.0 (mi-2000) used massively JS (ex.: google docs)
- JS standard evolved slowly => creation of TypeScript (generates JS), Flash, Silverlight
- Since 10 years ago, JS has been promoted by W3C and evolves very fast

Client-side scripts

–asynchronous communication–



- Classical HTTP: client-initiated data, replaces current Web page
- HTTP with AJAX: client-initiated data, does not replace current page
- WebSocket: both server and client-initiated data
 - better reactivity (chat, sensors, games, ...)

Client-side scripts –past present–



```
$(document).ready(function(){  
  $("button").click(function(){  
    $("#div1").fadeIn();  
    $("#div2").fadeIn("slow");  
    $("#div3").fadeIn(3000);  
  });  
});
```

Bootstrap

- Responsive
- CSS with a little bit of JS (for visual rendering)

jQuery

- General-purpose library
- Eases DOM update, CSS animations and AJAX

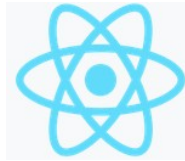


Client-side scripts

–future present–

- Single-page applications (SPA) : google/qwant maps, [example](#) et [un autre](#)

- React



- easy to start coding
- additional libraries are needed for complex sites

- Angular

- Java-like coding



- Vue.js :

- declarative



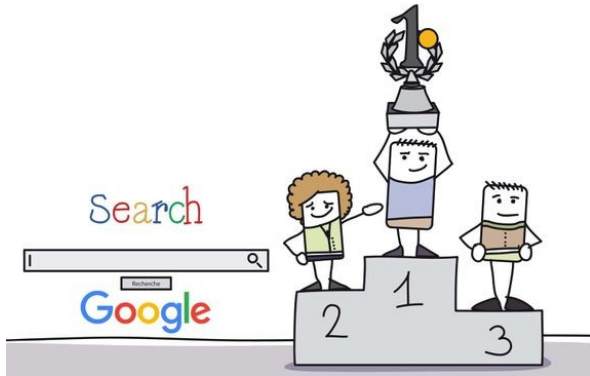
```
new Vue({  
  el: '#app',  
  data: {  
    message: 'Hello Vue.js!'  
  }  
})
```

```
ReactDOM.render(  
  <h1>Hello world!</h1>,  
  document.getElementById('root')  
)
```

```
import { Component } from '@angular/core';  
@Component({  
  selector: 'app-root',  
  templateUrl: './app.component.html',  
  styleUrls: ['./app.component.css']  
})  
export class AppComponent {  
  title = 'Hello World';  
}
```

Client-side scripts –difficulties–

- Given the high number of frameworks, which one is appropriate to my application?
 - the frameworks are not simply libraries (providing an API), but structure/define the application (organisation and coding manner)
 - sometimes the compatibility breaks (Angular.js -> Angular)



- Correct Web indexing needs care:
 - for complex and public Web sites, pages generated on-the-fly and single-page applications (SPA) break Web indexing (e.g. onclick vs classical link)

Client-side scripts –the future–

- Web development is not yet mature, because frameworks evolve, and fast (fierce competition)
- Lack of consensus: they answer different needs; different ways of working
- Link to mobile applications (progressive web applications)
- Interfaces evolve: keyboard-mouse, tactile screen, smartphone, virtual reality?



Conclusions

- This course is only an introduction to Web technologies
- It is easy to write static HTML pages with some CSS, and simple dynamic pages in PHP/MariaDB and JavaScript
- Modern Web sites written using JavaScript frameworks can be complex and evolve very fast
 - the Web technology is not yet mature
- Beautiful (successful) Web sites need artistic skills