

HTML

bron 1: *HTML, CSS, Bootstrap & Blade. Vormgeving in Laravel. Cursus: Cloud Computing & Toepassingen - 2020/2021 - Kris Aerts*

bron 2: *Responsive Web Design with HTML5 and CSS - 4th edition - Ben Frain*

bron 3: [W3Schools](#)

HyperText Markup language

HTML staat voor ‘HyperText Markup Language’ en is een manier om text content te markeren zodat het door een programmeertaal of webbrowser begrepen kan worden. HTML is essentieel voor menselijk begrijpbare webcontent. Je markeert teksthoud met tags/elements om structuur en inhoud aan te brengen. HTML is dus een standaard voor het structuren van informatie via een markup-taal gebaseerd op XML, en kan in elke browser getoond worden zoals Firefox, Edge, Chrome, Safari, Opera, ...

Het grote voordeel is dat we zelf geen software moeten gaan installeren de computer van gebruikers en dat we voortdurend updates aan de software kunnen aanbrengen zonder dat we deze expliciet moeten distribueren naar de eindgebruiker. Je kan HTML (en CSS) statisch schrijven of dynamisch laten genereren via een programmeertaal en bijhorend framework, zoals PHP en Laravel, Java en Tomcat, C# en ASP, Flask en Jinja, ...

Samengevat is HTML de markup-taal waarmee we via tags informatie kunnen structureren.

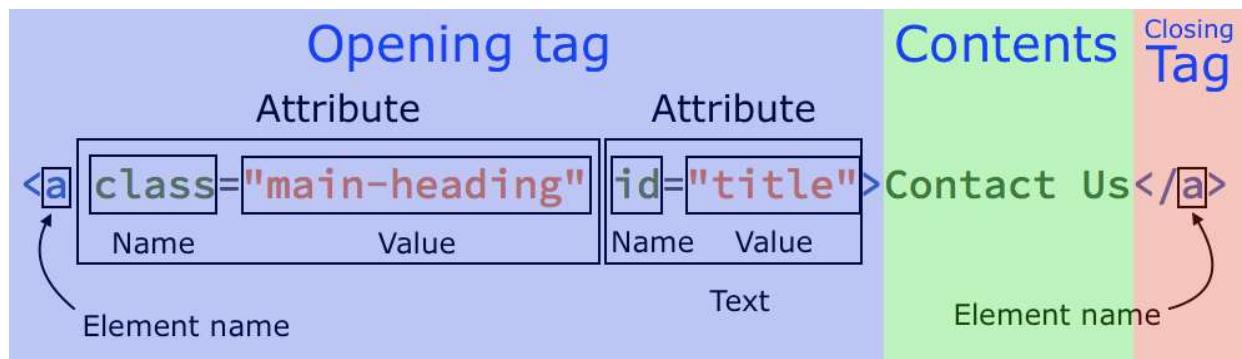
HTML beschrijft enkel de structuur, en géén vormgeving. Dat doen we in CSS.

Merk ten slotte op dat HTML afgeleid is van XML en dat ook CSS een strikte structuur heeft. Om dit te controleren kan je o.a. volgende validators gebruiken:

- Validator voor HTML <http://validator.w3.org/>

Elements, Tags en Attributes

Een **HTML element** wordt gedefinieerd door een open-tag, elementinhoud en een sluit-tag. Een **HTML tag** wordt weergegeven met de naam van het element binnendoor “`< ... >`” en bestaat steeds uit kleine letters bv. `<p>`. Een sluit-tag bevat nog een “/” voor de elementnaam bv. `</p>`. Elke open-tag moet meestal ook gevolgd worden door een sluit-tag van hetzelfde element bv. `<p> ... </p>`. Een uitzondering op deze regel zijn een aantal **self-closing elements** die geen elementinhoud bevatten. Een self-closing element bevat dus maar één tag waarin de “/” na de elementnaam komt en voor de sluitende “>” bv. `` of `
`.



HTML syntax and structure of an HTML element bron

Een element kan verschillende **attributes** bevatten die extra informatie over een HTML element bevatten. Deze attributen kunnen het gedrag of uiterlijk van een element wijzigen, de functionaliteit ervan definiëren of andere details specificeren, zoals de relatie met andere elementen of het gedrag als reactie op gebruikersinteracties.

Attributen worden in de open-tag aan HTML-elementen toegevoegd met behulp van *name-value pairs* binnendoor de open-tag van het element. De belangrijkste attributen die wij gebruiken zijn `id`, `class` en `style`. Enkele andere veelgebruikte HTML-attributen zijn: `href`, `src` ...

Op volgende manier gebruik je attributen in HTML tags (Een element kan meerdere attributen bevatten en je zelf ook eigen attributen toevoegen):

```
<p attributeName="attributeValue">...</p>
```

Twee belangrijke attributen zijn ‘**id**’ en ‘**class**’ die je helpen een specifiek element terug te vinden met behulp van CSS-selectors of JavaScript.

```
<p id="paragraaf1" class="specialeParagraafKlasse" value="1">...</p>
```

- ❶ Een element kan maximaal één **id** hebben maar wel meerdere **class** namen.

Een element kan meerdere andere elementen bevatten. Hier spreken we dan van **nested elements**, bv.:

```
<p id="paragraaf1" value="1">
  <h1>...</h1>
  <a>...</a>
</p>
```

Structuur van een HTML-document

Je kan eender welke simpele tekst- of code-editor gebruiken om HTML bestanden aan te maken of te bewerken (bv. notepad, notepad++, vscode, sublime text, atom, vim, nano ...). Een HTML bestand zou volgende hoofdstuctuur moeten volgen:

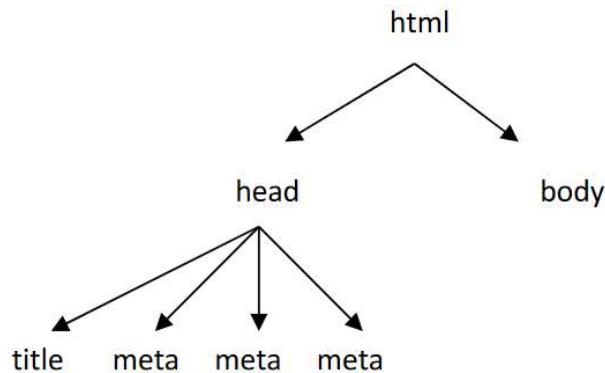
```
<!DOCTYPE html>
<html lang="en">
  <head>
```

```
<meta charset="utf-8" />  
...  
</head>  
<body>  
...  
</body>  
</html>
```

- **Doctype:** Hiermee geven we weer welk type dit document is. In ons geval dus steeds html.
- **html element:** Het HTML element met de naam ‘html’. Dit wordt de **root tag** van het HTML bestand genoemd, want alle html code moet binnen de `<html>` en `</html>` tags staan. Deze tags geven aan waar het html-document begint en eindigt.
 - **lang attribuut:** Met het language attribuut geven we meer informatie mee aan de webbrowser door te vermelden dat de content op deze webpagina in het Engels (`en`) geschreven is. (Nederlands = `nl`)
 - Binnen html zijn er slechts twee tags mogelijk: de `<head>` en de `<body>` voor respectievelijk informatie in de header die niet rechtstreeks voor de lezer bestemd is en anderzijds de body die de feitelijke inhoud van het document bevat.
- **head element:** dit is een container voor de volgende metadata: `<title>`, `<style>`, `<meta>`, `<link>` and `<base>`. (eventueel ook `<script>`)
 - **meta element:** wordt gebruikt om de karakterset/encoding, paginabeschriving, trefwoorden, auteur van het document en viewport settings te declareren.
 - **character encoding:** dit verteld de webbrowser hoe de karakters gecodeerd zijn zodat hij ze correct kan decoderen. Tenzij je een goede reden hebt is de gebruikte codering normaal **utf-8**.
- **body element:** een element container die al de overige inhoud van het HTML bevat. De headers, paragrafen, de tekst ...

Boomstructuur

Bij het werken met deze tags is het **zéér** belangrijk dat we een **boomstructuur** aanhouden: elke tag moet volledig binnen een andere zitten. Anders gezegd betekent dit dat je een omsluitende tag pas mag afsluiten wanneer je alle binnenliggende tags afgesloten hebt. Concreet mag je de `</html>` pas afsluiten na de `</body>`. Het feit dat je een boomstructuur krijgt, betekent dat je gemakkelijk deeltakken van het document kan selecteren.



Types van HTML elements

- **Sectioning elements:** elementen die worden gebruikt om de structuur van een webpagina te definiëren door secties van inhoud te scheiden door de verschillende delen semantische betekenis mee te geven. Bv. `<h1>`, ..., `<h6>`, `<p>`, `<header>` en `<footer>`.

Zoals bv. deze webpagina is opgebouwd uit hoofdstuktitels, paragraaf- en subparagraph-titels van elk een verschillende grootte, of een krant waar je hoofdingen in verschillende groottes hebt afhankelijk van de belangrijkheid van het nieuws.

In HTML hebben we 6 soorten hoofdingen, gaande van `<h1>`, `<h2>`, `<h3>` tot `<h6>` en daarnaast de `<p>` waarin je de eigenlijke paragraaf tekst zet. In principe hoef je niet te weten hoe groot de verschillende headers zijn: de browser is vrij dit zelf te bepalen zolang h1 maar belangrijker is dan h2 (enzoverder), maar sowieso kan je met CSS deze vormgeving nog wijzigen. De `<p>` plaatst de tekst in principe links uitgelijnd en laat steeds een witte regel tussen twee paragrafen, maar ook dit kan je veranderen met CSS.

Visuele onderverdelingen zijn de `
` (break) en de `<hr>` (horizontal ruler). De eerste voegt een blanco regel toe, terwijl de tweede een horizontale scheider plaatst.

- **Grouping elements:** elementen die worden gebruikt om meerdere inhoudsitems te groeperen of te bundelen onder één overkoepelend element. Deze elementen dragen niet echt bij tot de inhoud van het document op zich, maar helpen wel de structuur te verfijnen. Enerzijds gaat het om structurende onderverdelingen: tags die een aantal andere tags samen groeperen tot een nieuwe deelverzameling: een sectie van het

document.

Bv. `<div>` wordt hiervoor het meest gebruikt. Het is wel een eigenschap van de div dat ze zorgt voor het begin van een nieuwe regel. `` heeft dezelfde inhoudelijke betekenis, maar zorgt niet voor een visueel zichtbaar nieuwe regel en wordt daarom eerder binnen tags gebruikt.

- **Opsommingen:** Voor de opsomming hebben we enerzijds de keuze uit de geordende lijst `` (ordered list) of de niet-geordende lijst `` (unordered list), waarbij elk lijst-element op zijn beurt een `` is (listitem).
- **Hypertext:** De tags die we voordien zagen, waren puur text-based en gaan voorbij aan de rijkdom van html, een rijkdom die we onder de noemer Hypertext kunnen plaatsen: html biedt immers de kans om meer dan alleen tekst te tonen: figuren, videos, geluiden en hyperlinks: doorverwijzingen naar andere documenten of naar andere plaatsen binnen het huidige document.
 - `<a>` de anker tag: Bij deze tag heb je enerzijds de linktitel: de tekst die (meestal) in het blauw op je webpagina verschijnt, en anderzijds de link zelf: de pagina waar je naar toe springt wanneer je op de link klikt. De linktitel is de inhoud van de tag, terwijl je de link zelf via het attribuut href moet meegeven,
bv. `De bekendste zoekmachine`
Je kan ook links (ankers) binnen je webpagina maken. Dan moet je het attribuut name gebruiken,
bv. `...`. Om dan naar zo'n anker te verwijzen, moet je als href # gebruiken + de naam van het anker,
bv. `...`
 - `` de image-tag: Dit is een **replaced tag** omdat hij wordt vervangen door de figuur waarnaar verwezen wordt in het `src`-attribuut. Dit attribuut is dan ook verplicht. Daarnaast is ook het attribuut `alt` verplicht voor zoekmachines en blinden of slechtzienden waarbij schermlees software dan de `alt`-tag voorleest.
bv. ``
 - Voor video- en audio-fragmenten heeft men de `<video>`- en `<audio>`-tag voorzien, maar in de praktijk worden echter heel dikwijls iframes gebruikt. Sowieso gebruiken de meeste mensen de embeddable code die krijgen ze van sites zoals youtube of spotify en daarom gaan we daar hier niet dieper op in.
- **Text-level semantics:** dit verwijst naar de manier waarop HTML elements de betekenis en structuur van tekst op een webpagina definiëren. Deze

elementen worden gebruikt om specifieke delen van de tekst te markeren en hun semantische betekenis aan te geven. Bv. `` en ``.

Andere markeringen zijn:

- `<title>` Verschijnt in de titelbalk van de browser en bij de bookmark.
Deze tag moet wel in de head van het html-document.
 - `<cite>` Een citaat uit een andere tekst.
 - `<code>` Voor programmacode.
- **Opgelet: witruimte:** Opvallend binnen HTML is dat eender welke hoeveelheid witruimte beschouwd wordt als het begrip “witruimte” waarvoor de browser slechts **één** spatie zal gebruiken. Of je dus 7 spaties gebruikt of 13 enters of 8 tabs, 3 enters en 22 spaties, dit komt allemaal overeen met “witruimte”.

HTML-entities

Omdat de verschillende talen in de wereld veel verschillende accenten hebben, is er gekozen voor een overdraagbaar systeem van accenten: men doet dit met **HTML-entities**. Dit zijn speciale codes waarmee je speciale tekens kan weergeven. Dit beperkt zich niet alleen tot accenten, maar ook tot tekens zoals `&`, `€`, `<` en `>` (want die laatste worden anders als deel van een tag beschouwd), ... Typisch is dat ze allen **beginnen met & en afgesloten worden met ;**, bv. `&` geeft `&` of `€` geeft `€`.

Voor de accenten heb je het systeem **& + letter + accent + ;**. Het accent is dan een van: grave (à), acute (é), uml (ë), cedil (ç), circ (ê), tilde (ñ).

Enkele interessante zijn voor ampersand (`&` &), euro (`€` €), copyright (`©` ©), reg (`®` ®), trademark (`™` ™), less then (`<` <), greater then (`>` >), less then or equals (`≤` ≤), greater then or equals (`≥` ≥) en non breaking space (` `) bv:

` ` non breaking` `` `` `` ` space
non breaking space

Via volgende link vind je een volledige lijst terug:
<https://www.freeformatter.com/html-entities.html>

List of useful elements

Klik hier om de code te zien/verbergen 

```
<!-- This is a comment in HTML -->
```

```
<!-- SECTIONING ELEMENTS -->
```

```
<nav>
```

The 'nav' element is used to mark up a collection of links to external pages or sections within the current page. As well as being used for the main website navigation, the 'nav' element is also a good fit for things like a table of contents, or a blogroll.

```
</nav>
```

```
<header>Header for webpage</header>
```

```
<aside>
```

The 'aside' element is used to represent content that is tangibly related to the content surrounding it, but could be considered separate. This includes things like sidebars

```
</aside>
```

```
<main>
```

The 'main' element should contain the main content for your web page. All of this content should be unique to the individual page, and should not appear elsewhere on the site. Any content that is repeated on multiple pages (logos, search boxes, footer links, etc.) should not be placed within the 'main' element.

You should only use one 'main' element on a page, and it shouldn't be placed within an 'article', 'aside', 'header', 'footer', or 'nav' element.

```
<article>
```

```
  <header>
```

the 'header' element is used to represent the introductory content to an article or web page. This will usually contain a heading element as well as some metadata that's relevant to the content, such as the post date of a news article for example.

```
    <h1>Largest Header</h1>
```

```
    <h2>Header 2</h2>
```

```
    <h3>Header 3</h3>
```

```
    <h4>Header 4</h4>
```

```
    <h5>Header 5</h5>
```

```
<h6>Smallest Header</h6>

</header>
```

The 'article' element should contain a piece of self-contained content that could be distributed outside the context of the page. This includes things like news articles, blog posts, or user comments.

You can nest 'article' elements within one another. In this case it's implied that the nested elements are related to the outer 'article' element.

```
<aside>Tangibly related content</aside>

</article>
```

```
<section>
```

The 'section' element is used to represent a group of related content. This is similar to the purpose of an 'article' element with the main difference being that the content within a 'section' element doesn't necessarily need to make sense out of the context of the page.

It's advisable to use a heading element ('h1' - 'h6') to define the topic for the section.

If you just need to group content together for styling purposes you should use a 'div' element rather than a 'section'

```
<footer>
```

The 'footer' element is used to represent information about a section such as the author, copyright information, or links to related web pages.

```
<address>
```

This element is not for marking up postal address, but rather for representing the contact information for an article or web page. This could be a link to the author's website or their email address.

```
</address>
```

```
</footer>
```

```
</section>
```

Based on content and design, articles can contain sections and/or sections can contain articles.

```
</main>
```

```
<footer>
```

Footer for webpage

```
<address> Link to email author webpage </address>
```

```
</footer>
```

```
<!-- GROUPING ELEMENTS -->
<div>
    Used for grouping blocks for easy styling. <span>Used for grouping inline
    content</span>
</div>

<p>Defines a paragraph</p>

<pre>Defines preformatted text</pre>

<blockquote cite="citation_source">
    Specifies a section that is quoted from another source
</blockquote>
<q>For inline (short) quotations</q>

<ol type="I" start="1"> <!-- type="1|a|A|i|I" -->
    <!-- Defines an ordered list. An ordered list can be numerical or
    alphabetical. -->
    <li>List Item I</li>
    <li>List Item II</li>
    <li>List Item III</li>
</ol>
<ul>
    <!-- Defines an unordered list. Used the same a 'ol' -->
</ul>

<table> <!-- Defines an HTML table -->
<tr> <!-- Defines a table row -->
    <th>Defines a table head for column</th> <!-- -->
    <th>Header column 2</th>
</tr>
<tr>
    <td>Defines a table cell</td>
    <td>Cell row 2, column 2</td>
</tr>
<tr>
    <td>Cell row 3, column 1</td>
    <td>Cell row 3, column 2</td>
</tr>
</table>

<figure>
    
    <!--src=".//voorbeeld/realtive/path/img.jpg |>
    C://voorbeeld/absolute/path/img.jpg | https://www.voorbeeld-url.com/img.jpg"
```

```
-->
<figcaption>Image caption</figcation>
</figure>

<!-- TEXT-LEVEL SEMANTICS -->

All simple text is displayed without line breaks.
But 'br' creates
<br/> a line break.<br/><br/>

<a href="link_to_webpage/or/local_file" target="_blank">Link text<a> <!--
target=_blank | _self | _parent | _top" -->
<!--href=".//voorbeeld/realtive/path | C://voorbeeld/absolute/path | 
https://www.voorbeeld-url.com" --><br/><br/>

The <em>em-tag</em> is used to define emphasized text. The content inside is
typically displayed in italic.<br/><br/>

The <i>i-tag</i> defines a part of text in an alternate voice or mood. The
content inside is typically displayed in italic.<br/><br/>
<!-- Use the <i> element only when there is not a more appropriate semantic
element -->

The <strong>strong-tag</strong> is used to define text with strong
importance. The content inside is typically displayed in bold.<br/><br/>

The <b>b-tag</b> specifies bold text without any extra importance.<br/><br/>

The <small>small-tag</small> defines smaller text (like copyright and other
side-comments).<br/><br/>

The <s>s-tag</s> specifies text that is no longer correct, accurate or
relevant. The text will be displayed with a line through it.<br/><br/>

The <cite>cite-tag</cite> defines the title of a creative work (e.g. a book,
a poem, a song, a movie, a painting, a sculpture, etc.).<br/><br/>

<p>
The <dfn>dfn-tag</dfn> stands for the "definition element", and it specifies
a term that is going to be defined within the content.
</p>

The <abbr title="abbreviation">ABBR</abbr>-tag defines an abbreviation or an
acronym.<br/><br/>

The <time datetime="2024-02-18 19:00">time-tag</time> defines a specific time
(or datetime).<br/><br/>
```

The `<code>code-tag</code>` is used to define a piece of computer code. The content inside is displayed in the browser's default monospace font.

The `<var>var-tag</var>` is used to defines a variable in programming or in a mathematical expression. The content inside is typically displayed in italic.


```
<!-- FORMS AND INPUTS -->
<form>
    fieldset
    label
    input text
        type number, password, email, radio, checkbox, submit, button, file
        (accept="image/png, image/jpeg")
        Placeholder
        value
        name
        id
    radiobutton
    checkbox
    dropdown met select en option (met value attr)
    textarea
</form>

<!-- EXTRA -->
-button
-symbols
-Details + summary
-icons
```

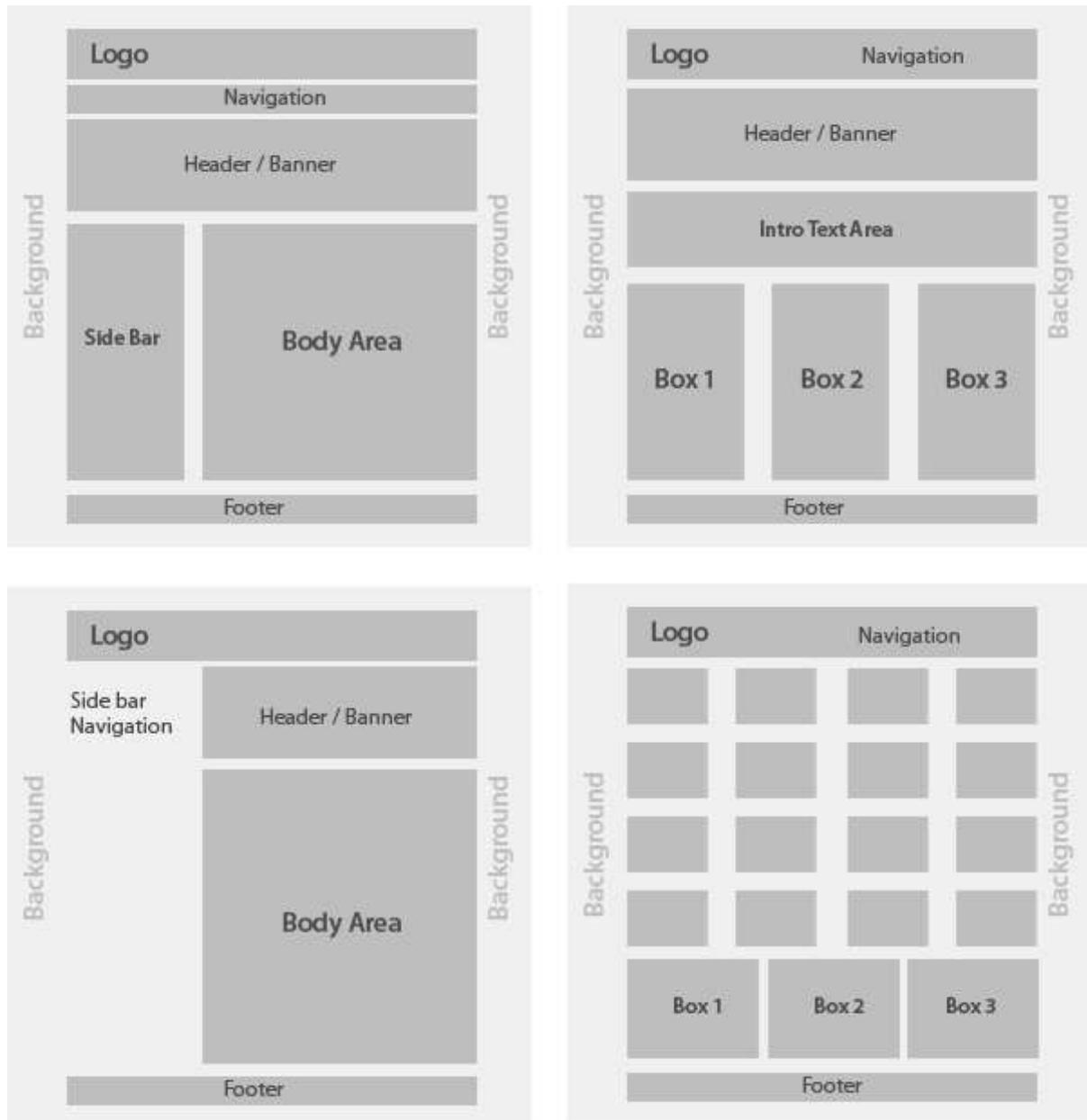
[bron1](#), [bron2](#), [bron3](#)

Een voorbeeld mappenstructuur voor je webpagina's

```
root/
|
+--index.html
|
+--html/
|
```

```
+--- about/
|   |
|   +--- about.html
|
+--- contact/
|   |
|   +--- contact.html
...
+--- assets/
|
+--- images/
|
+--- image1.png
...
...
```

Common website layouts



4 Common website layouts with HTML-elements [bron](#)

HTML vs HTML5

HTML5 is de nieuwere versie van HTML met een aantal nuttige voordelen. Zo is HTML5 zeer vrijgevend gezind in het weglaten van bepaalde attributen en gebruikt HTML5 default waarden voor attributen die nodig zijn maar niet specifiek uitgetyped werden. In XHTML was de basisstructuur ook vrij uitgebreid, maar gelukkig is dit sinds HTML 5 terug sterk vereenvoudigd. Bijvoorbeeld:

```
<link href="CSS/main.css" rel="stylesheet" type="text/css" />
```

HTML5:

```
<link href=CSS/main.css rel=stylesheet >
```

Merk op dat in het HTML5 voorbeeld geen “/” heeft voor het sluitende groter dan teken, geen quotes gebruikt voor de value van het attribuut ‘href’ en ‘rel’ en er geen attribuut ‘type’ gedefinieerd is. HTML5 zal hier echter niet moeilijk over doen.

! In het algemeen is het wel een goede strategie om je HTML bestand zo specifiek mogelijk te coderen om problemen in een later stadium te vermijden.

Aangezien HTML5 dus gewoon de nieuwere versie is van HTML gaan we dit gebruiken.

Formulieren

Met de HTML die we tot nu toe gezien hebben, kan je wel informatieve pagina's maken, maar de bezoeker kan niks anders dan dingen bekijken en links aanklikken. Om reacties of andere gegevens op te vragen, heb je formulieren nodig. In dit stuk behandelen we de opmaak van formulieren, maar we kunnen de ingevoerde gegevens nog niet verwerken.

Formulieropbouw

Een formulier bestaat steeds uit een verzameling **input**-elementen. Omdat er verschillende formulieren op één webpagina kunnen staan, moet je de bij elkaar horende formulierelementen groeperen onder een **form**-tag:

```
...
<form>
  <!-- different form elements -->
</form>
```

...

De **form**-tag heeft normaal ook nog de attributen **method**, **action** en **name**. De elementen binnenin de **form** zijn bijna allemaal “replaced tags”, dus zonder innerHTML en met een />-sluiting van de tag.

Attributen van de form: twee belangrijke, de *method* en de *action*.

- De *method* beschrijft op welke manier de gegevens doorgestuurd worden naar het script dat de formulierinhoud moet verwerken. Er zijn twee mogelijkheden:
 - **GET**: hierbij worden de gegevens in de URL gecodeerd en wordt de formulierinhoud dus zichtbaar in de URL. Een ander belangrijk nadeel is dat de lengte van de data beperkt is.
 - **POST**: bij deze methode worden de gegevens in een body ingepakt en zo doorgegeven. De lengte is nu nagenoeg onbeperkt. Deze methode wordt het meeste gebruikt.
- De *action* bevat de URL van het script dat het formulier zal moeten verwerken. Wanneer je geen action-attribuut definieert, wordt het formulier naar de huidige pagina doorgestuurd. Zolang je puur in HTML werkt, geeft dit de indruk dat het formulier gereset wordt.

Text:

Het eenvoudigste en ook meest gebruikte invoerelement is het één-regel tekstvak. De code hiervoor is `<input type="text" />`. De belangrijkste attributen zijn, naast de **id**, de **size** en de **maxlength**, die respectievelijk het aantal zichtbare letters bevatten en het maximaal aantal letters dat ingegeven kan worden. Je kan ook het type meegeven zoals onder andere: **email**, **password**, **number** ...

Bijvoorbeeld: `<input type="text" />` geeft

Checkbox:

Belangrijk is dat je een attribuut **value** definieert zodat je in je JavaScript kan achterhalen welke waarde geselecteerd is. Wanneer je bovendien **checked="checked"** toevoegt, zal het vakje vanaf het inladen van de pagina aangevinkt zijn.

Bijvoorbeeld: `<input type="checkbox" id="item" value="checkvalue" />` check me geeft

check me

```
en <input type="checkbox" id="item" value="checkvalue" checked="checked" />  
I am checked geeft  
 I am checked
```

Radio:

Bij de radio-button is het belangrijk dat je een radio-groep maakt m.b.v. het attribuut **name**, omdat radio-knopen uit een groep bij elkaar horen en uit die groep mag slechts één radio element geselecteerd worden. Hiervoor moet je binnen de radio-groep bij elk element de **name** dezelfde waarde geven. Anders wordt het beschouwd als een andere radio-groep.

Bijvoorbeeld:

```
Klara <input type="radio" name="station" value="klara" checked="checked" />  
<br />  
Q-Music <input type="radio" name="station" value="q" /> <br />  
MNM <input type="radio" name="station" value="-mnm" /> <br />
```

geeft:

Klara
Q-Music
MNM

(De oude) Button en Submit:

De types **button** en **submit** zien er in de browser hetzelfde uit, maar hebben een heel ander effect. Wanneer je op een **submit**-knop klikt, wordt de formulierinhoud doorgestuurd naar het script dat je in het **action**-attribuut van de form gedefinieerd hebt.

Wanneer je op een **button**-knop klikt, wordt de event-handler (JavaScript) uitgevoerd die bij deze knop hoort. (Hier komen we later op terug in het cursusdeel rond JavaScript)

Bijvoorbeeld

```
<input type="button" value="Click me" /> <br />  
<input type="submit" value="Submit me" />
```

(De nieuwe) Button:

De button uit de paragraaf hierboven had als groot nadeel dat het een *replaced element* was. Technisch betekent dit dat je geen volwaardige tag hebt met een open- en een sluit-tag en daartussen html-code die **in** de tag staat. Visueel houdt het in dat je de browser de html-code vervangt door een standaard knop, maar dat je verder geen structuur kan geven aan de inhoud van de knop. Zo kan je bijvoorbeeld geen figuren op je knop zetten. Daarom vond men het nodig om een volwaardige button-tag toe te voegen (ook al kan je hetzelfde bereiken met de replaced element button en wat specifieke CSS code). Tussen de open- en sluit-tag zet je nu wat op de knop moet verschijnen.

Naast de *name* en *id* is het belangrijkste attribuut van de button het *type: button*, *submit* en *reset*.

Hidden:

Het **hidden** input-element is een element dat niet zichtbaar gemaakt wordt in de browser. Het biedt de mogelijkheid om extra informatie door te geven die onzichtbaar is voor de gebruiker, bv. informatie die we op een vorige pagina binnengehaald hebben of eender welke waarde die relevant is voor de formulieverwerking maar die de gebruiker niet zelf moet invullen.

Bijvoorbeeld: `<input type="hidden" name="geheim" value="317" />` geeft:

Text Area:

De volgende invoerelementen zijn een uitzondering omdat het geen *replaced elementen* zijn. Hier gaat het om volwaardige tags. De **textarea** dient om tekst in te kunnen geven die **meer dan 1 regel** bevat. Met `<input type="text"/>` konden we al wel tekst ingeven, maar slechts op één regel. Met textarea kan dit onbeperkt.

Bijvoorbeeld: `<textarea>typ hier tekst</textarea>` geeft:



Met CSS kunnen we de grootte en vormgeving van de textarea instellen

Select en Option:

Bijvoorbeeld:

```
<select id="station">
    <option value="klara">Klara</option>
    <option value="qmusic">Qmusic</option>
    <option value="mnm">MNM</option>
</select>
```



Hier schrijf je dus binnen je `select` een lijst met `option` tags. Wat tussen de `option` staat, verschijnt in het menu en datgene wat bij de `value` staat, komt terecht in het script dat het formulier verwerkt.

Aan de `select` kan je een attribuut `size` meegeven dat bepaalt hoe groot de niet-uitgeklapte lijst is, en wanneer je `multiple="multiple"` aanduidt, kan je via **Ctrl+klik** verschillende elementen tegelijk selecteren.

Bijvoorbeeld:

```
<select id="station" size="3" multiple="multiple">
    <option value="klara">Klara</option>
    <option value="qmusic" selected="selected">Qmusic</option>
    <option value="mnm" selected="selected">MNM</option>
</select>
```



Tabellen:

Wanneer je gegevens in een tabel wil plaatsen, moet je tabellen gebruiken. Hiervoor bestaan de tags `table`, `tr` en `td`, de afkortingen van *table*, *table row* en *table data*. Merk op dat HTML vooral rijen ziet, terwijl wij eerder kolommen zien. Dat maakt het soms moeilijker om een tabel op te stellen, de structuur blijft wel logisch:

- Eerst maak je een tabel aan: `<table> ... </table>`
- Binnen die tabel definieer je verschillende rijen: `<tr> ... </tr>`
 - Bovenaan kan je ook een hoofding plaatsen met `<th>` i.p.v. een gewone rij.
- En in elke rij kan je verschillende vakjes definiëren met `<td>...</td>`.

Bijvoorbeeld:

```
<table>
  <tr>
    <td>Arne Duyver</td>
    <td>Informatica </td>
    <td>Tessenderlo</td>
  </tr>
  <tr>
    <td>Mark Huybrechts</td>
    <td>Elektromechanica</td>
    <td>Schoten</td>
  </tr>
</table>
```

geeft:

Arne Duyver	Informatica	Tessenderlo
Mark Huybrechts	Elektromechanica	Schoten

Tabellen zijn vooral interessant om formulieren te een layout te geven. In de eerste kolom zet je dan de labels (de uitleg bij wat je moet invullen) en in de tweede kolom de inputs, zodat die verticaal uitgelijnd zijn.

Opdrachten

1. Maak een HTML-bestand en noem het portfolio.html
2. Geef de titel van je webpagina de naam “portfolio”.
3. Voeg jezelf toe als auteur van de webpagina

4. Gebruik het header element om de onderstaande structuur aan te brengen aan je webpagina.

- Expertise
- Over mij
- Mijn projecten
- Technische vaardigheden en CV
- Contact

5. Gebruik de relevante html elementen om het volgende toe te voegen aan je webpagina: voeg boven de header expertise een welkomstbericht toe. (Voor inspiratie voor het tekstje kan je ChatGPT gebruiken, de vormgeving doe je ZELF). Emphasize een aantal inspirerende woorden door ze in het vetgedrukt/schuin te zetten.

6. Voeg ook een mooie afbeelding toe die je online ophaalt. Gebruik hiervoor het ‘figure’ element en voeg een caption toe

7. Plaats de welkomsttekst binnenin een div en geef die div de klasse naam ‘welcometext’.

8. Geef in de Expertise sectie een lijst waarin je je eigen vaardigheden in de verf zet. (je kan ook inspiratie opdoen op andere portfolio websites).

9. Plaats in de Expertise sectie ook een link naar de sectie Technische vaardigheden en CV

10. Plaats in de ‘Over mij’ je favoriete quote van je lievelingsfilm/-boek in de Over mij sectie. Gebruik hier het juiste element voor.

11. Geef ook wat meer informatie over jezelf en plaats minstens één belangrijke zin in een span en geef die de id ‘important-sentence’.

12. Plaats hier ook een afbeelding (de afbeelding moet je lokaal hebben staan)

13. Maak onder de sectie ‘Mijn projecten’ subsecties voor alle projecten die je al eens gemaakt hebt. Bijvoorbeeld je project van ELSY van het eerste jaar, je PES project, eigen andere projecten ...

14. Onder technische vaardigheden maak je een tabel met je verschillende opleidingen in (naam opleiding, startjaar, eindjaar). Voeg ook een lijst met beheerde talen toe en link hier ergens naar volgende webpagina:
<https://detaalbrigade.nl/taalniveaus/>

15. Breng wat meer structuur aan in je teksten met divs, paragrafen en line breaks. Voeg ook eens wat symbolen via hun html-code toe waar nuttig.
16. Gebruik inputs boxen, knoppen, checkboxen, ... om een contactformulier aan te maken. Je vindt hier ontelbare voorbeelden van op het internet.
17. Neem nu je ‘Contact’ sectie en plaats die in een nieuw bestand genaamd ‘contact.html’ en plaats deze in de subfolder genaamd ‘contact’.
 - Maak een gepast formulier aan voor deze webpagina.
18. Maak footer aan waarmee je navigeert naar je ‘contact’ page van je website.
19. Voeg in je head een link toe naar fa-icons zodat je die icoontjes kan gebruiken:

```
<link rel="stylesheet"
      href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.5.1/css/all.min.css">
```
20. Voeg aan je footer ook een link toe via een icoon naar je Github profiel en een icoon voor je linkedIn profiel (indien je dat hebt)
21. Maak een formulier waarin mensen zich kunnen inschrijven voor een nieuwsbrief.
22. Maak een formulier om een bespreking van een optreden van een artiest in te vullen.
23. Maak een formulier waarin je een pizza kan samenstellen.

Test jezelf

Klik [hier](#) om jezelf te testen met de online HTML quiz van w3schools. Of klik [hier](#) voor wat extra oefeningen.

CSS

bron 1: *HTML, CSS, Bootstrap & Blade. Vormgeving in Laravel. Cursus: Cloud Computing & Toepassingen - 2020/2021 - Kris Aerts*

bron 2: *Responsive Web Design with HTML5 and CSS - 4th edition - Ben Frain*

bron 3: [W3Schools](#)

Cascading Style Sheet

HTML alleen maakt een website niet visueel aantrekkelijk. CSS is dan nodig voor de vormgeving. Met CSS-regels kun je ontwerpen hoe de afzonderlijke componenten die je eerder in HTML hebt gedefinieerd, moeten worden weergegeven. Je kan CSS dus gebruiken om het ontwerp en de lay-out van een webpagina te definiëren. Je kan bijvoorbeeld tekstkleuren, tekstgroottes, randen, achtergrondkleuren, kleurverlopen enzovoort definiëren.

Vooraleer we dit doen en de algemene regels bespreken, willen we wel vermelden dat we slechts een beperkt deel van CSS zien. Het doel van deze cursus is enkel individuele pagina's toch wat vormgeving te kunnen geven zodat we op zijn minst wat accenten kunnen leggen en de verschillende onderdelen van een webpagina kunnen positioneren. We gaan geen professioneel ogende sites maken met complexe layoutschema's of uitgebreide websites met uitgebreide navigatie. Daarom mag en zal onze portie CSS relatief beperkt blijven.

CSS definiëren op verschillende niveaus



CSS kan op verschillende manieren worden gedefinieerd: in een apart CSS-bestand, in de `<head>` van je HTML-document, of inline in de open-tag van een HTML-element. Elk van deze methoden heeft zijn eigen voor- en nadelen, afhankelijk van de situatie en het doel van de styling.

Een apart CSS bestand

Het gebruik van een apart CSS-bestand is de meest gebruikelijke en aanbevolen methode. Dit zorgt voor een *duidelijke scheiding tussen de structuur (HTML) en de opmaak (CSS)*, wat de *leesbaarheid* en *onderhoudbaarheid* van de code bevordert. Bovendien kunnen meerdere HTML-pagina's dezelfde CSS-bestand gebruiken, wat consistentie in de styling garandeert en de laadtijd van de website kan verbeteren door caching.

Je gebruikt hiervoor het `<link>`-element in de `head` van je HTML-bestand. Als `href`-attribuut geef je het pad naar je `.css`-bestand mee:

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

In het HTML-element `<style>` in de head van je HTML-bestand

Het definiëren van CSS in de `<head>` van een HTML-document, ook wel interne of embedded CSS genoemd, kan handig zijn voor *kleinere projecten* of wanneer *specifieke stijlen alleen voor een enkele pagina gelden*. Dit kan echter de leesbaarheid van de HTML-code verminderen en maakt het moeilijker om stijlen te hergebruiken op andere pagina's.

```
<head>
  <style>
    /*Write your CSS code here*/
  </style>
</head>
```

Met `/*` en `*/` kan je commentaar schrijven bij je CSS-code. Bv. `/* Dit is een CSS comment */`

Inline CSS: het `style`-attribuut

Met inline CSS wordt de styling direct in de open-tag van een HTML-element geplaatst. Dat kan nuttig zijn voor *snelle, eenmalige aanpassingen* of voor het

toepassen van *unieke stijlen op specifieke elementen*. Deze methode wordt echter over het algemeen afgeraden omdat het de HTML-code rommelig maakt en de scheiding tussen structuur en opmaak vervaagt. Bovendien is inline CSS moeilijker te onderhouden en te updaten.

```
<htmlElement style="property1:value1 ; property2:value2;">  
...  
</htmlElement>
```

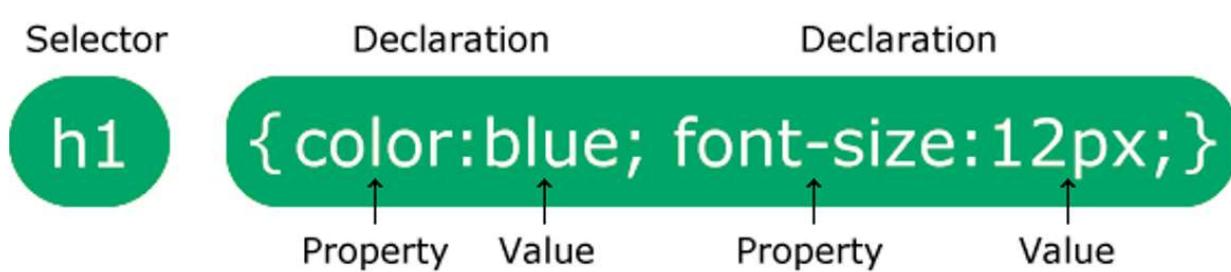
Info

Met een goede code-editor krijg je vaak hints over welke css-properties en corresponderende values allemaal mogelijk zijn. Dat versnelt de productie significant.

De plaats van het definiëren van de CSS-code is dus een afweging tussen de specificiteit, leesbaarheid, onderhoudbaarheid en herbruikbaarheid van de code

Syntax

Een CSS-rule (CSS-regel) bestaat steeds uit een **selector** en één of meerdere **declaraties**. De declaraties moeten tussen accolades geplaatst worden: { en }. Elke declaratie bestaat op zijn beurt uit een **property** gevolgd door een :, een geschikte **value** en wordt afgesloten met een ;.



Selectors, Pseudo-classes en Pseudo-elements

Een selector is een patroon dat wordt gebruikt om HTML-elementen te selecteren die je wilt stylen. Je kan op verschillende manieren elementen selecteren, met behulp van:

- **een element selector**: kan je alle HTML-elementen van een bepaald type selecteren en stylen. Bijvoorbeeld, de selector `p` selecteert alle paragraafelementen (`<p>`) op een webpagina.
- **een klasse selector**: kan je all HTML-elementen van een bepaalde klasse selecteren en stylen. Bijvoorbeeld, de selector `.mijnKlasseNaam` selecteert alle elementen (`<element class="mijnKlasseNaam">`) op een webpagina.
- **een id selector**: kan je het HTML-element met een bepaalde id selecteren en stylen. Bijvoorbeeld, de selector `#mijnIdNaam` selecteert het element (`<element id="mijnIdNaam">`) op een webpagina.
- **een samengestelde/complexe selector**: bv. `li.belangrijk`. Dit betekent dat elke `li` met als klasse `belangrijk` de bijhorende vormgeving moet krijgen, en de andere `li`'s niet, evenmin als de elementen van klasse `belangrijk` die géén `li` zijn. Wanneer je een spatie laat tussen de combinatie, bv. `p .belangrijk`, betekent dit “met daarbinnen”, dus elk element met klasse `belangrijk` binnen een `p` krijgt de vormgeving, maar andere elementen binnen `p` niet, evenmin als elementen met klasse `belangrijk` die niet binnen `p` staan.
 - **een attribuut selector**: je kan dan ook nog dieper gaan selecteren op attributen (`[attribuut="waarde"]`). Bijvoorbeeld, de selector die de `<p>`-elementen selecteert die als waarde `yes` van het attribuut `mijnAttribuut` hebben ziet er als volgt uit: `p[mijnAttribuut=yes]`.

```
elementType {...;}  
.className {...;}  
#idName {...;}  
/*Dit is een comment*/  
  
element:pseudo-classname {...;}  
/*e.g.: a:hover{...;}*/  
  
element::pseudo-elementname {...;}  
/*e.g.: h1:before{...;}*/  
  
/*Attribute selectors*/  
element[attribute] {...;}  
element[attribute="value"] {...;}  
element[attribute~="value"] {...;} /*contains specific words*/  
element[attribute|= "value"] {...;} /*specific value or followed by - */  
element[attribute^="value"] {...;} /*starts with specific value*/
```

```
element[attribute$="value"] {...;} /*ends with specific value*/  
element[attribute*="value"] {...;} /*contains a specified value*/
```

Types of combinators:

- descendant selector (space) specifies all descendants
- child selector (>) only goes one deep
- adjacent sibling selector (+) selects an element that is directly after another specific element.
- general sibling selector (~) selects all elements that are next siblings of a specified element.

Margin, padding en outline



Sizing

- **Absolute**: px, pt, pc, in, cm, mm
- **Relative**: %, em, rem, ex, ch, fr
- **Viewport** (define in head): vw, vh, vmin, vmax

We gaan in 99% van de gevallen enkel gebruik maken van `px`, `rem` en `%` (af en toe `em` relatief t.o.v. `font-size` eigen element)!!!

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

Positioning

- **Static:** An element with `position: static;` is not positioned in any special way; it is always positioned according to the normal flow of the page.
Static positioned elements are not affected by the top, bottom, left, and right properties.
- **Relative:** An element with `position: relative;` is positioned relative to its normal position.
Setting the top, right, bottom, and left properties of a relatively-positioned element will cause it to be adjusted away from its normal position. Other content will not be adjusted to fit into any gap left by the element.
- **Fixed:** An element with `position: fixed;` is positioned relative to the viewport, which means it always stays in the same place even if the page is scrolled. The top, right, bottom, and left properties are used to position the element.
A fixed element does not leave a gap in the page where it would normally have been located.
- **Absolute:** An element with `position: absolute;` is positioned relative to the nearest positioned ancestor (instead of positioned relative to the viewport, like fixed).
Absolute positioned elements are removed from the normal flow, and can overlap elements.
- **Sticky:** An element with `position: sticky;` is positioned based on the user's scroll position.
A sticky element toggles between `relative` and `fixed`, depending on the scroll position. It is positioned relative until a given offset position is met in the viewport - then it "sticks" in place (like `position:fixed`).

List of usefull properties

```
#elementId {  
    background-color: red; /*rgb(255,0,0) , #FF0000, hsl(0, 100%, 50%) ,  
    rgba(255,0,0,0.5), hsla(0, 100%, 50%, 0.5)*/  
    opacity: 0.3; /*Doorschijnbaarheid 0-1*/  
    background-image: url("image.jpg");  
    background-repeat: repeat-x; /*repeat-y, no-repeat*/  
    background-position: right top;  
    background-attachment: fixed; /*scroll*/  
  
    border-style: dotted; /*dashed, solid, double, groove, ridge, inset,  
    outset, none, hidden*/  
    /*border-top-style, border-right-style, border-bottom-style, border-left-  
    style*/  
    border-width: 5rem; /*top right bottom left*/  
    border-width: black;  
    border-radius: 5px;  
  
    margin: 10px; /*top right bottom left*/  
    /*margin-top, margin-right, margin-bottom, margin-left*/  
    padding: 10px; /*top right bottom left*/  
    /*padding-top, padding-right, padding-bottom, padding-left*/  
  
    height: 200rem;  
    max-height: 20px;  
    width: 80%; /*relative to parent*/  
    max-width: 60ch;  
  
    outline: 1px solid red; /*width style color*/  
    outline-offset: 15px;  
  
    color: black; /*Text color*/  
    text-align: center; /*left, right, justify*/  
    text-align-last: center; /*right, justify*/  
    direction: rtl; /*ltr*/  
    vertical-align: baseline; /*text-top, text-bottom, sub, super*/  
    text-decoration-line: overline; /*line-through, underline, overline  
    underline*/  
    text-decoration-color: red;  
    text-decoration-style: solid; /*double, dotted, dashed, wavy*/  
    text-decoration-thickness: 5px;  
    text-transform: uppercase; /*lowecase, capitalize*/  
    text-indent: 5px; /*first line indent*/  
    letter-spacing: -2px;  
    line-height: 0.8;  
    word-spacing: 5rem;  
    white-space: nowrap;  
}
```

```
text-shadow: 1px 1px 2px black, 0 0 25px blue, 0 0 5px darkblue;
/*horizontal vertical blur color, ...*/
font-family: "Times New Roman", Times, serif; /*desired, fallback1,
fallback2*/
font-style: normal; /*italic, oblique*/
font-weight: normal; /*bold*/
font-size: 0.5rem;
font-variant: normal; /*small-caps*/

overflow-x: scroll; /*hidden, auto, visible*/
overflow-y: scroll; /*hidden, auto, visible*/

display: inline; /*block, inline-block, contents, flex, grid*/
position: static; /*relative, fixed, absolute, sticky (to parent)*/
bottom: 0;
right: 0;
z-index: -1; /*lower = further in background*/

float: right; /*left, none, inherit*/
/*in parent*/ clear: left; /*right, none, inherit*/

box-sizing: content-box; /*border-box*/

/*IF display: flex*/
flex-direction: row; /*column, column-reverse, row-reverse*/
flex-wrap: wrap; /*nowrap, wrap-reverse*/
/*flex-flow: row wrap; */ /*direction wrap*/
justify-content: flex-start; /*center, flex-end, space-around, space-
between*/
align-items: stretch; /*baseline, flex-start, center, flex-end*/
align-content: space-between; /*space-around, stretch, center, flex-start,
flex-end*/

/*IF display: grid*/

}

ul {
list-style-type: circle; /*square, upper-roman, lower-alpha, none*/
list-style-image: url('sqpurple.gif');
list-style-position: outside; /*inside*/
}

table {
border-collapse: collapse;
}

tr:nth-child(even) {background-color: #f2f2f2;}
```

```

input {
    outline: none;
}

a:link {...}
a:visited {...}
a:hover {...}
a:active {...}

/*COUNTERS*/
#containerElement {
    counter-reset: section;
}
element::before{
    counter-increment: section;
    content: "Section " counter(section) ": ";
}

```

Icons

```

<script src="https://kit.fontawesome.com/yourcode.js"
crossorigin="anonymous"></script> <!-- In the head -->
<i class="fas fa-cloud"></i>

```

Text effects

text-overflow: hoe moet verborgen overflowing content weergegeven worden?

- clip
- ellipsis *overflow moet hidden zijn (werkt niet bij overflow visible)*

```

#textOverflow {
    text-overflow: clip;
    overflow: hidden;
}
#textOverflow2 {
    text-overflow: ellipsis;
}

```

```
    overflow: hidden;  
}
```

word-wrap: om lange woorden op te breken en te wrappen naar de volgende regel

```
#wordWrap {  
    word-wrap: break-word;  
}
```

word-break: hoe moeten lijnen text gebroken worden.

```
#wordWrap {  
    word-break: keep-all;  
}  
#wordWrap2 {  
    word-break: break-all;  
}
```

writing-mode: horizontaal of verticaal.

```
#writingMode {  
    writing-mode: horizontal;  
}  
#writingMode {  
    writing-mode: vertical;  
}
```

Calculations

e.g.:

- width: calc(100% - 100px);
- width: max(50%, 300px);
- width: min(50%, 300px);

CSS variables

```
:root {  
    --blue: #1e90ff;  
    --white: #ffffff;  
}  
  
body { background-color: var(--blue); }
```

Extra

Andere CSS files importeren in de main.css

```
@import url('./animations.css');
```

Moet helemaal in het begin van je CSS-file staan

!important: om alle andere styling te overschrijven.

```
element {  
    background-color: red !important;  
}
```

simple linear gradient: gebruik background-image property en niet background-color.

```
#grad {  
    background-image: linear-gradient(to right, red, yellow); /*direction,  
    color-stop1, color-stop2, ...*/  
}
```

Divs, columns, User Interface

```
div {  
    column-count: 3;  
    column-gap: 40px;  
    column-width: 100px;  
    column-rule-style: solid;  
    column-rule-width: 1px;  
    column-rule-color: lightblue;  
    /*element inside the div*/ column-span: all;  
  
    resize: horizontal; /*vertical, both*/  
    overflow: auto;  
}
```

img:

```
img {  
    border-radius: 8px;  
    opacity: 0.5;  
    filter: grayscale(100%);  
    box-shadow: 0 0 2px 1px rgba(0, 140, 186, 0.5);  
    -webkit-box-reflect: below; /*above, left, right*/  
    object-fit: cover; /*contain, fill, none, scale-down*/  
    object-position: 80% 100%;  
    -webkit-mask-image: url(img1.png);  
    mask-image: url(img1.png);  
    -webkit-mask-repeat: no-repeat;  
    mask-repeat: no-repeat;  
}
```

Opdrachten

1. Maak je footer fixed onderaan de pagina
2. Voeg een afbeelding van jezelf (of) een stockfoto toe aan je over mij sectie en zorg ervoor dat de afbeelding steeds rechts staat
3. Maak een nav sectie die 2 anchor elementen heeft: 1 voor de home page en een voor de contact page.
4. Gebruik volgende **bron** om je nav sectie te stylen zodat je een verticale navigatie sectie hebt aan de linkerkant van je pagina die 100% van de hoogte

in beslag neemt.

5. Gebruik een input checkbox die je gefixed houd in de linkerboven hoek. Zorg ervoor dat je navigatie sectie verborgen wordt wanneer de checkbox niet is aangevinkt en getoond wordt wanneer je de checkbox aanduidt.
6. Gebruik volgende [bron](#) om je formulier in je contact pagina te stijlen..
7. Gebruik volgende [bron](#) om de globale layout van je site te updaten.
8. Zorg al voor een responsive design door al je font-sizes aan te passen aan een vaste font-size die zich aanpast aan de grootte van de pagina.
9. Kies een leuke font voor je pagina en eventueel complementaire fonts voor speciale secties zoals quotes. (zorg ook voor fallback fonts)
10. Gebruik icons in plaats van tekst in je navigation sectie.
11. Geef je creativiteit de vrije loop om je site zo mooi mogelijk te maken.

Test jezelf

Klik [hier](#) om jezelf te testen met de online HTML quiz van w3schools. Of klik [hier](#) voor wat extra oefeningen.

Libraries en CDN

Je kunt **libraries** zoals CSS-frameworks toevoegen aan je websitecode om je ontwikkelproces te versnellen en consistente, professionele ontwerpen te creëren. Dit doe je door de benodigde bestanden te downloaden en lokaal op te slaan, waarna je ze via `<link>`-tags in de `<head>` van je HTML-document opneemt. Bijvoorbeeld, voor Bootstrap, een populaire CSS-library, zou je een link naar het gedownloade CSS-bestand toevoegen:

```
<link rel="stylesheet" href="path/to/bootstrap.min.css">
```

Een eenvoudigere manier om libraries toe te voegen is via een **Content Delivery Network (CDN)**. CDNs hosten de bestanden van de library op servers over de hele wereld, waardoor je website sneller kan laden doordat de bestanden van een server dichtbij de gebruiker worden opgehaald. Om een library via een CDN toe te

voegen, hoef je alleen maar een <link>-tag met de URL van de CDN op te nemen in je HTML-document. Voor Bootstrap zou dit er als volgt uitzien:

```
<link rel="stylesheet"  
      href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
```

Het gebruik van een CDN heeft als voordeel dat je geen bestanden lokaal hoeft op te slaan en te beheren, en dat de laadtijd van je website kan verbeteren door de geografische spreiding van de CDN-servers.

Bootstrap

Met het beperkte stuk HTML en CSS dat we hierboven hebben behandeld, kun je functionele HTML-pagina's maken die met CSS worden gestyled. Veel ingenieurs zijn echter niet altijd bedreven in het creëren van aantrekkelijke layouts. Daarom kan het een goed idee zijn om gebruik te maken van kant-en-klare layouts en deze aan te passen naar je eigen smaak. Omdat er veel vraag is naar dergelijke oplossingen, zijn er verschillende frameworks ontwikkeld waarmee je de layout van webpagina's kunt instellen. Wij richten ons hier op Bootstrap, een simpel en beginnersvriendelijk framework voor responsive designs.

Bootstrap werkt door een verzameling van CSS- en JavaScript-componenten te voorzien die je kunt gebruiken om je webpagina's snel te stylen en functionaliteit toe te voegen via class names.

1. Voeg Bootstrap via CDN of link naar gedownloade bestanden toe aan je HTML-bestand.
2. Bootstrap biedt een breed scala aan vooraf ontworpen componenten zoals knoppen, formulieren, navigatiebalken en kaarten. Je kunt deze componenten eenvoudig in je HTML-code opnemen door de juiste **Bootstrap-classes** toe te voegen.

...

```
<div class="container mt-5">  
  <!-- Bootstrap-knop -->  
  <button type="button" class="btn btn-primary">Klik hier</button>
```

```
</div>
```

...

3. Bootstrap heeft een krachtig **grid-systeem** waarmee je de layout van je pagina kunt structureren. Het grid-systeem is gebaseerd op **rows** en **columns**, en maakt het eenvoudig om responsieve layouts te creëren die zich aanpassen aan verschillende schermgroottes.

...

```
<div class="container mt-5">
  <div class="row">
    <div class="col-md-4">
      <div class="p-3 border bg-light">Kolom 1</div>
    </div>
    <div class="col-md-4">
      <div class="p-3 border bg-light">Kolom 2</div>
    </div>
    <div class="col-md-4">
      <div class="p-3 border bg-light">Kolom 3</div>
    </div>
  </div>
</div>
```

...



Info

Hoewel Bootstrap veel standaardstijlen biedt, kun je deze altijd aanpassen door je **eigen CSS** toe te voegen of de bestaande Bootstrap-klassen te **overschrijven**.

Een aantal handige links:

- [Bootstrap documentatie](#)
- [w3schools tutorial](#)
- [tutorialrepublic bootstrap](#)

ADVANCED CSS

bron 1: *Responsive Web Design with HTML5 and CSS - 4th edition* - Ben Frain

bron 2: [W3Schools](#)

Quicktip

Je css code spreiden over meerdere stylesheets is mogelijk door in je `main.css` andere CSS-bestanden te importeren. Je kan bijvoorbeeld al je animation klassen in een aparte `animations.css` plaatsen en dat bestand dan importeren met onderstaande code in je `main.css`:

```
@import url('./animations.css');
```

Moet helemaal in het begin van je CSS-file staan

Transformations

CSS-transformations zijn krachtige tools die developers in staat stellen om HTML-elementen visueel te manipuleren zonder hun oorspronkelijke structuur te veranderen. Met CSS-transformations kunnen elementen worden verplaatst, gedraaid, geschaald en scheefgetrokken, waardoor dynamische en interactieve ontwerpen mogelijk worden.

Deze transformaties worden toegepast met de `transform`-property en kunnen meerdere transformaties combineren voor complexe effecten. Bijvoorbeeld, een

element kan tegelijkertijd worden verplaatst en gedraaid, wat zorgt voor vloeiende animaties en visuele aantrekkingskracht.

Met CSS-transformaties kun je elementen dus verplaatsen, roteren, schalen en scheef trekken. Volgende 2D-transformaties zijn beschikbaar:

- **translate()**: Verplaatst een element van zijn huidige positie.

```
transform: translate(50px, 100px); /* Verplaatst het element 50px naar rechts  
en 100px naar beneden */
```

- **rotate()**: Draait een element rond een vast punt.

```
transform: rotate(45deg); /* Draait het element 45 graden met de klok mee */
```

- **scaleX()**: Schaal een element horizontaal.

```
transform: scaleX(1.5); /* Vergroot de breedte van het element met 50% */
```

- **scaleY()**: Schaal een element verticaal.

```
transform: scaleY(0.5); /* Verkleint de hoogte van het element met 50% */
```

- **scale()**: Schaal een element zowel horizontaal als verticaal.

```
transform: scale(2); /* Verdubbelt de grootte van het element in beide  
richtingen */
```

- **skewX()**: Scheeftrekken van een element langs de X-as.

```
transform: skewX(30deg); /* Scheeftrekt het element 30 graden langs de X-as */
```

- **skewY()**: Scheeftrekken van een element langs de Y-as.

```
transform: skewY(20deg); /* Scheeftrekt het element 20 graden langs de Y-as */
```

- **skew()**: Scheeftrekken van een element langs zowel de X- als de Y-as.

```
transform: skew(30deg, 20deg); /* Scheeftrekt het element 30 graden langs de X-as en 20 graden langs de Y-as */
```

- **matrix()**: Combineert meerdere transformaties in één.

```
transform: matrix(1, 0.5, -0.5, 1, 100, 50); /* Voert een combinatie van translaties, rotaties, schalingen en scheeftrekkingen uit */
```

Extra Examples:

```
div {  
    transform: translate(50px, 100px);  
    transform: rotate(20deg);  
    transform: rotate(-20deg);  
    transform: scale(2, 3);  
    transform: scale(0.5, 0.5);  
    /* transform: scaleX(2);  
    transform: scaleY(3); */  
    transform: skew(20deg, 10deg);  
    /* transform: skewX(20deg);  
    transform: skewY(20deg); */  
}
```

Matrix

De methode `matrix()` combineert alle 2D-transformatiemethoden in één. De `matrix()` methode neemt zes parameters, die wiskundige functies bevatten, waarmee je elementen kunt roteren, schalen, verplaatsen (vertalen) en schuin houden.

De parameters zijn als volgt: `matrix(scaleX, skewY, skewX, scaleY, translateX, translateY)`

```
div{  
    transform: matrix(1, 0.5, -0.5, 1, 100, 50);  
}
```

De `matrix(1, 0.5, -0.5, 1, 100, 50)`-transformatie voert een combinatie van schalen, scheeftrekken en verplaatsen uit op een element. Hier is wat elke parameter doet:

- `scaleX (1)`: Schaal de breedte van het element met een factor van 1 (geen verandering).
- `skewY (0.5)`: Scheeftrek het element langs de Y-as met een hoek van ongeveer 26,57 graden (0.5 radian).
- `skewX (-0.5)`: Scheeftrek het element langs de X-as met een hoek van ongeveer -26,57 graden (-0.5 radian).
- `scaleY (1)`: Schaal de hoogte van het element met een factor van 1 (geen verandering).
- `translateX (100)`: Verplaats het element 100 pixels naar rechts.
- `translateY (50)`: Verplaats het element 50 pixels naar beneden.

Net zoals het werken met andere CSS-code is “doing it yourself” de boodschap om meer feeling te krijgen.

Transitions

CSS-transitions zijn een manier om geleidelijke veranderingen in de stijl van een element te creëren wanneer een eigenschap wijzigt. Ze maken het mogelijk om

animaties te maken die soepel en visueel aantrekkelijk zijn, zonder dat er complexe JavaScript nodig is. Met transitions kun je specificeren **welke eigenschappen** moeten veranderen, de **duur van de verandering**, de **timing-functie** (zoals lineair of versneld) en **eventuele vertragingen**. Bijvoorbeeld, je kunt een knop laten veranderen van kleur wanneer de muis eroverheen beweegt, of een afbeelding laten vergroten wanneer deze wordt aangeklikt.

Met andere woorden kan je **CSS-transitions** kun je de waarden van eigenschappen soepel veranderen gedurende een bepaalde tijd. We bespreken de volgende overgang properties:

- `transition`
- `transition-property`
- `transition-duration`
- `transition-timing-function`
- `transition-delay`

Om een transition effect te maken, moet je minstens twee dingen specificeren:

1. de CSS-property waaraan je een effect wilt toevoegen
2. de duur van het effect *Opmerking: Als het onderdeel duur niet wordt opgegeven, heeft de overgang geen effect, omdat de standaardwaarde 0 is.*

Om een transition te laten plaatsvinden, moet een element een verandering in state hebben en moeten voor elke state verschillende styles worden bepaald. De eenvoudigste manier om styles voor verschillende states te bepalen is door gebruik te maken van de `:hover`, `:focus`, `:active` en `:target` pseudo-klassen.

Example:

```
div {  
    width: 100px;  
    height: 100px;  
    background: red;  
    transition: width 2s;  
}  
div:hover {  
    width: 300px;  
}
```

Je kan de transition properties apart definiëren of allemaal samen in een shorthand:

```
div {
  transition-property: width;
  transition-duration: 2s;
  transition-timing-function: linear;
  transition-delay: 1s;
}
/* Shorthand */
div {
  transition: width 2s linear 1s;
}
```

Voor de transition-timing-function zijn er een aantal mogelijkheden. (**In de developper tools van je browser kan je met deze waarden spelen en de gewenste beziercurve kopiëren**):

- **ease** - specifies a transition effect with a slow start, then fast, then end slowly (this is default)
- **linear** - specifies a transition effect with the same speed from start to end
- **ease-in** - specifies a transition effect with a slow start
- **ease-out** - specifies a transition effect with a slow end
- **ease-in-out** - specifies a transition effect with a slow start and end
- **cubic-bezier(n,n,n,n)** - lets you define your own values in a cubic-bezier function

Je kan transitions voor meerdere elementen definiëren door ze te splitsen met een < ; >: bv.

```
/* multiple transitions */
div {
  transition: width 2s linear 1s, padding-top 1s ease-out;
}
```

Animations

Met **CSS-animations** laat je een element geleidelijk veranderen van de ene stijl naar de andere. Je kunt zoveel CSS-properties wijzigen als je wil, zo vaak je wil. We bespreken de volgende animation properties:

- `animation`
- `@keyframes`
- `animation-name`
- `animation-duration`
- `animation-delay`
- `animation-iteration-count`
- `animation-direction`
- `animation-timing-function`
- `animation-fill-mode`
- `animation`

Om CSS-animations te gebruiken, moet je eerst een aantal keyframes opgeven voor de animatie. Keyframes geven aan welke stijlen het element op bepaalde momenten zal hebben.

Je kan ook meerdere keyframes meegeven zodat je meerdere states tussen de animation hebt. Deze tussenstates geef je dan aan met percentages, %.

```
/* The animation code */
@keyframes example {
  from {background-color: red;}
  to {background-color: yellow;}
}

/* The element to apply the animation to */
div {
  width: 100px;
  height: 100px;
  background-color: red;
  animation-name: example;
  animation-duration: 4s;
}
```

Shorthand:

```

div {
    animation-name: example;          /*keyframe name*/
    animation-duration: 5s;
    animation-timing-function: linear; /*linear, ease, ease-in, ease-out,
ease-in-out*/
    animation-delay: 2s;
    animation-iteration-count: infinite;
    animation-direction: alternate;    /*normal, reverse, alternate,
alternate-reverse*/
    animation-fill-mode: forwards;     /*none, forwards, backwards, both*/
}

/* Shorthand */
div {
    animation: example 5s linear 2s infinite alternate forwards;
}

```

animation-timing-function: zie transitions

animation-direction: **-normal** - The animation is played as normal (forwards). This is default
-reverse - The animation is played in reverse direction (backwards)
alternate - The animation is played forwards first, then backwards
-alternate-reverse - The animation is played backwards first, then forwards

animation-fill-mode: **-none** - Default value. Animation will not apply any styles to the element before or after it is executing
-forwards - The element will retain the style values that is set by the last keyframe (depends on animation-direction and animation-iteration-count)
-backwards - The element will get the style values that is set by the first keyframe (depends on animation-direction), and retain this during the animation-delay period
-both - The animation will follow the rules for both forwards and backwards, extending the animation properties in both directions

Verschil met transitions

CSS-animations definiëren complexe bewegingen met keyframes, zoals **rotate** of **fade**, terwijl **CSS-transitions** soepele veranderingen maken in elementeneigenschappen, zoals grootte of kleur, tijdens gebeurtenissen zoals **hover**. Animaties gebruiken keyframes en kunnen oneindig herhalen, terwijl overgangen optreden bij eigenschapsveranderingen en meer geschikt zijn voor subtiele

effecten. Beiden voegen interactiviteit en aantrekkelijkheid toe aan webpagina's, maar hebben verschillende toepassingen.

Animation utility classes en dubbele classes

Je gebruikt **animation utility classes** om de relatie tussen animatie en element te ontkoppelen en voor herbruikbaarheid. Je kan op die manier ook verschillende animatie eigenschappen opsplitsen in verschillende klassen en dan via de juiste klassennamen snel gaan combineren in je html.

```
.fade-in {  
    animation-name: fadeIn;  
}  
  
.animate {  
    animation-duration: 1s;  
    animation-fill-mode: both;  
}  
  
.animate.animate--infinite {  
    animation-iteration-count: infinite;  
}  
  
.animate.animate--delay-1s {  
    animation-delay: 1s;  
}  
  
@keyframes fadeIn {  
    from {  
        opacity: 0;  
    }  
    to {  
        opacity: 1;  
    }  
}  
  
@keyframes fadeAnimation {  
    0%, 50%, 100%   {opacity: 1;}  
    25%, 75%   {opacity: 0.5;}  
}
```

met HTML

```
<div class="animate fade-in animate--delay-1s">
  ...
</div>
```

Je gebruikt **dubbele klasse** notaties ter bescherming:

```
.animatie.fade-in {
  animation: fadeIn 0.5s ease-in forwards;
}

@keyframes fadeIn {
  from {
    opacity: 0;
  }
  to {
    opacity: 1;
  }
}
```

Dit biedt verschillende voordelen:

1. **Specifieke Stijlen Toepassen:** Door dubbele klassen te gebruiken, kun je zeer specifieke stijlen toepassen op elementen. Dit helpt bij het verfijnen van je CSS en voorkomt conflicten tussen stijlen.
2. **Herbruikbaarheid van CSS:** Je kunt algemene stijlen in één klasse definiëren en specifieke animaties of effecten in een andere. Dit maakt je CSS herbruikbaarder en gemakkelijker te onderhouden.
3. **Bescherming tegen Stijlconflicten:** Door specifieke combinaties van klassen te gebruiken, verminder je de kans op stijlconflicten. Dit is vooral handig in grotere projecten met veel CSS.

Animation play state

Door gebruik te maken van de **animation-play-state** property kun je eenvoudig animaties pauzeren en hervatten. De mogelijke values zijn:

- **paused**: De animatie stopt op het huidige frame
- **running**: De animatie wordt afgespeeld.
- **initial**: De animatie keert terug naar de standaardwaarde ‘running’.
- **inherit**: De animatie neemt de waarde over van het bovenliggende element.

```
div:hover {  
    animation-play-state: paused; /*running, initial, inherit*/  
}
```

Animatie klasse dynamisch toevoegen met JavaScript

Dit zien we in het deel rond JavaScript.

Opdrachten

Exercise 1, 2, 3, 4: Er worden drie **transitions** getoond. Je hebt ongeveer 8 minuten om de transities zo goed mogelijk te evenaren. Daarna wordt de oplossing overlopen. (De exacte pixel afstanden worden niet verwacht, het is voldoende wanneer de transitie gelijkaardig is).

Exercise 5, 6, 7, 8: Er worden drie **animations** getoond. Je hebt ongeveer 8 minuten om de animations zo goed mogelijk te evenaren. Daarna wordt de oplossing overlopen. (De exacte pixel afstanden worden niet verwacht, het is voldoende wanneer de transitie gelijkaardig is).

Exercise 9, 10, 11, 12, 13, 14, 15: Er worden 7 **transitions of animations** getoond. Je hebt ongeveer 15 minuten om de voorbeelden zo goed mogelijk te evenaren. Daarna wordt de oplossing overlopen. (De exacte pixel afstanden worden niet verwacht, het is voldoende wanneer de transitie gelijkaardig is).

De oplossingen vind je [hier](#)

Portfolio website

Fleuren nu je eigen portfolio website op door wat transitions en animaties toe te voegen waar nuttig.

JAVASCRIPT

bron 1: *Rich(er) client met Javascript: Cloud Computing & Toepassingen - 2020/2021* - Kris Aerts

bron 2: *Responsive Web Design with HTML5 and CSS - 4th edition* - Ben Frain

bron 3: [W3Schools](#)

JavaScript

Met HTML en CSS maak je in feite statische webpagina's in de zin dat de inhoud en de vormgeving vast zijn: de HTML en CSS blijven hetzelfde, ook wanneer je de pagina ververst. Als je bijvoorbeeld een klok wil maken waar de tijd live tickt en waar de achtergrond donkerder wordt naargelang de nacht valt, dan heb je nog iets extra nodig buiten HTML en CSS. Elke web-browser-bouwer kan hiervoor vrij een aantal talen kiezen, maar in de praktijk wordt énkel JavaScript (officieel ECMAScript) gebruikt.

Met JavaScript kan je direct reageren wanneer de gebruiker een veld van dat formulier invult, maar je kan ook veel andere dingen doen afhankelijk van de actie(s) van de gebruiker, tot zelfs volledige games spelen. Javascript zorgt er voor dat je allerlei dynamische effecten in webpagina's kan bekomen en hiermee verkrijg je volwaardige interactiviteit en dynamisch ogende webapplicaties.

JavaScript wordt dus gebruikt om interactie en dynamische functionaliteit aan je website toe te voegen. Het stelt ontwikkelaars in staat om webpagina's te manipuleren, gebruikersacties te verwerken en te communiceren met servers zonder de pagina opnieuw te laden. JavaScript is veelzijdig en wordt ondersteund door alle belangrijke webbrowsers. Het maakt deel uit van de frontend is vaak niet of minimaal aanwezig op statische websites.

Wat nog belangrijk is om te weten, is dat de JavaScript standaard (**ECMAScript**) beheerd wordt door *Ecma International*, vroeger ook wel de *European Computer Manufacturers Association (ECMA)* genoemd. Een overzicht van de historische details zijn terug te vinden op de JS [wikipage](#).

Info

We gaan er vanuit dat je al een basiskennis programmeren bezit. In deze sectie zoomen we vooral in op de interactie die JavaScript ons biedt met HTML-elementen en andere functionaliteiten die specifiek zijn voor het maken van websites.

JavaScript is geen Java

Hoewel de namen vergelijkbaar zijn, zijn JavaScript en Java twee heel verschillende programmeertalen met verschillende toepassingen en eigenschappen. JavaScript is een scripttaal die voornamelijk wordt gebruikt voor het toevoegen van interactieve elementen aan webpagina's en het manipuleren van de DOM (Document Object Model). Het wordt direct in de browser uitgevoerd en is essentieel voor het creëren van dynamische en responsieve webapplicaties. Java daarentegen is een objectgeoriënteerde programmeertaal die vaak wordt gebruikt voor het ontwikkelen van server-side applicaties, mobiele apps (vooral Android), en enterprise-software. Java-code wordt gecompileerd naar bytecode die draait op de Java Virtual Machine (JVM), waardoor het platformonafhankelijk is. Ondanks de naamovereenkomst zijn de syntaxis, het gebruik en de onderliggende architectuur van JavaScript en Java dus fundamenteel verschillend.

De namen JavaScript en Java lijken op elkaar vanwege marketing en historische redenen. Toen JavaScript werd ontwikkeld door Brendan Eich bij Netscape in 1995, was Java al een populaire programmeertaal. Netscape zag een kans om mee te liften op het succes van Java en besloot de naam van hun nieuwe scripttaal te veranderen van "LiveScript" naar "JavaScript". Dit zorgde voor verwarring, maar het hielp ook om JavaScript snel bekendheid te geven. [\[1\]](#), [\[2\]](#)

Een Rich client side experience

JavaScript speelt een cruciale rol in de ontwikkeling van Rich Internet Applications (RIA's). RIA's zijn webapplicaties die dezelfde functionaliteit en interactiviteit bieden als desktopapplicaties, maar toegankelijk zijn via een webbrowser. JavaScript maakt het mogelijk om dynamische en interactieve gebruikersinterfaces te creëren door het manipuleren van de DOM, het verwerken van gebruikersinvoer, en het communiceren met servers via AJAX (Asynchronous JavaScript and XML). Hierdoor kunnen RIA's snel reageren op gebruikersacties zonder de hele pagina te vernieuwen, wat resulteert in een vloeiendere en meer responsieve gebruikerservaring. Bekende voorbeelden van RIA's zijn *webmaildiensten, online tekstverwerkers, en interactieve dashboards*.

Hoe JavaScript toevoegen aan je website

Je kan JavaScript rechtstreeks in je html-document toevoegen via het `<script>` element. *Deze manier wordt echter niet aangeraden.* Je kan het `<script>-element` toevoegen in de sectie `head` of in de `body` van je HTML-document. Het is echter aan te raden om scripts onderaan het `<body>-element` toe te voegen, omdat dit de laadtijd van een webpagina kan verbeteren. [3]

Om de code leesbaar en herbruikbaar te houden en te maken, kan je er ook voor kiezen om de functie declaraties toch in de `head` te doen omdat die gebruikt wordt voor algemene definities.

```
...
<script type="text/javascript">
    //Jouw code komt hier
</script>
...
```

Je kan commentaar schrijven in JS met // zoals in Java of / */ zoals in CSS*

JavaScripts worden echter beter verzameld in een `scripts`-directory. Op die manier heb je betere scheiding van verantwoordelijkheden tussen HTML en JavaScript (JS). Je kan ook zeer makkelijk verschillende JS-scripts aanmaken voor je verschillende webpagina's. Je kan hier weer het `<script>-element` voor gebruiken, maar dit keer laten we de content leeg en verwijzen we met behulp van het `src`-attribuut naar de juiste JS-file.

```
...
<script type="text/javascript" src="./path_to_your/javascript_file.js">
</script>
...
```

Zo importeren we ook voorgemaakte libraries van JS of importeren we JS code vanaf een DNS

Een veelgebruikte directory structuur voor het beheren van al je client-side files ziet er zo uit.

```
root/
|
+--- index.html
|
+--- scripts/
|   |
|   +--- main.js
|   ...
+--- styles/
|   |
|   +--- main.css
|
...
```

Basis JavaScript syntax

Variabelen

In talen zoals Java, C++, C#, ... moet je elke variabele die je gebruikt, verplicht declareren. In JS moet dit niet: je kan gewoon ter plekke een nieuwe variabele beginnen gebruiken. **Moderne stijlgidsen raden dit echter ten zeerste af!** In Javascript heeft het echter geen zin om het type van een variabele te declareren omdat een variabele geen type heeft. De variabele kan in het begin een **integer** bevatten, even later een **String**, nog wat later een HTML-node, ... Je moet in feite dus alleen aangeven dat je een variabele gaat gebruiken. Tot ECMAScript 2015 kon

je dit alleen met de `var` doen, bv. `var getal;` Sinds 2017 ondersteunen echter alle moderne browsers **ECMAScript 2015** en heb je ook de keuze uit `let` en `const`:

- `const` gebruik je om aan te geven dat de inhoud van een variabele constant is en dus niet mag veranderen.
- `let` is ingevoerd om beter aan te sluiten bij de praktijk van zowat alle andere programmeertalen. De `let` heeft **block scope**. Dat betekent dat de `let` enkel gekend is in de lexicaal blok omsloten door accolades. In het voorbeeld hieronder is de `i` na de lus niet meer gekend.
- `var` heeft **function scope**. Dat betekent dat de variable “gehoist” wordt tot het niveau van de functie en dat je een herdeclaratie kan doen die dan direct impact heeft op functieniveau.

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

console.log(i); // i is unknown here, because i defined with block scope in
the loop

// VERSUS

for (var i=0; i<10; i++) {
    console.log(i);
}

console.log(i); // prints 10, because i has function scope.
```

Omdat het gedrag van de `let` dus perfect overeenstemt met wat we in andere talen gewoon zijn, raden we aan om overal te declareren met `let`.

Andere voorbeelden:

```
var x = 1;                      // met 'var' definiëer je een variabele die
beschikbaar blijft binnen de functie scope.
let y = "let";                  // met 'let' definiëer je een variabele die
beschikbaar blijft binnen de block scope. (bv enkel binnenin die if-
statement)
const z = "Ik verander niet"; // met 'const' definiëer je een variabele die
niet meer verandert
```

Dankzij onze kennis van Java en aanverwante talen is beginnen programmeren in Javascript enorm eenvoudig: de syntax met zijn accolades en punt-komma's en de belangrijkste controlestructuren zoals de toekenning, de `++`, `if-then-else`, de `for`, de `while`, ... zijn allemaal exact hetzelfde als in Java.

Wil je toch even een herhaling in JavaScript klik dna hier om de code te bekijken 

Types

```
// Numbers:  
let length = 16;  
let weight = 7.5;  
  
// Strings:  
let color = "Yellow";  
let lastName = "Johnson";  
  
// Booleans  
let x = true;  
let y = false;  
  
// Object:  
const person = {firstName:"John", lastName:"Doe"};  
const person = {  
    firstName: "John",  
    lastName : "Doe",  
    id       : 5566,  
    fullName : function() {  
        return this.firstName + " " + this.lastName;  
    }  
};  
  
// Array object:  
const cars = ["Saab", "Volvo", "BMW"];  
  
// Date object:  
const date = new Date("2022-03-25");
```

If, switch, for, while en try-catch

```
//IF-STATEMENT
if (condition1) {
    // block of code to be executed if condition1 is true
} else if (condition2) {
    // block of code to be executed if the condition1 is false and condition2
    is true
} else {
    // block of code to be executed if the condition1 is false and condition2
    is false
}

//SWITCH
switch(expression) {
    case x:
        // code block
        break;
    case y:
        // code block
        break;
    default:
        // code block
}

//FOR-LOOP
for (let i = 0; i < cars.length; i++) {
    text += cars[i] + "<br>";
}

//WHILE
while (condition) {
    // code block to be executed
}

//TRY CATCH
try {
    //Block of code to try
}
catch(err) {
    //Block of code to handle errors
}
finally {
    //Block of code to be executed regardless of the try / catch result
}
```

Functies en klassen definiëren

```
//FUNCTIE
function(parameter) {
    //mijn code
    return result;
}

//EXAMPLE Class
class Car {
    constructor(name, year) {
        this.name = name;
        this.year = year;
    }
    age(x) {
        return x - this.year;
    }
}
const myCar = new Car("Ford", 2014);
```

Developer tools and logging

Om onze JS code te debuggen gaan we veel gebruik maken van de developer tools die beschikbaar zijn in je browser. In google chrome kan je de developer tools openen met **F12**. De plaats waar we de meeste tijd gaan doorbrengen in de **console**. We kunnen namelijk allerhande informatie laten printen naar de console. Dit kan op volgende manieren:

```
console.log("Ik ben een normaal log bericht");
console.debug("Ik ben een debug bericht");
console.error("Ik ben een error");
console.info("Ik ben info");
console.warn("Ik ben een warning");
```

In de developer console kan je dan ook filteren op de verschillende soorten logberichten.

Webcontent inspecteren

Om informatie over bepaalde content op te vragen moeten we eerst weten hoe we HTML-elementen kunnen opvragen om ze daarna te inspecteren.

Omdat HTML gebaseerd is op XML, heb je een boomstructuur en kan je relatief – gemakkelijk individuele takken opvragen. Stel dat je binnen het HTML-document twee divs hebt. Met `document.div[0]` en `document.div[1]` kan je dan de 1e en 2e div opvragen. De eerste div binnen de tweede div zou dan `document.div[1].div[0]` zijn. Op analoge wijze kan je eender welke blok opvragen. Deze techniek van het document doorlopen verwijst naar het **DOM – Document Object Model**.

Het grote nadeel aan deze techniek is dat je sterk afhankelijk bent van de opbouw van je HTMLpagina, terwijl dit net iets is wat gemakkelijk wijzigt: je gaat blokken toevoegen, verwijderen of op andere plaatsen zetten, of zelfs bijna alles herstructureren. In zo een gevallen zou je al je verwijzingen naar elementen moeten wijzigen in je code en dat is niet houdbaar.

Daarom is één van de gemakkelijkste manieren om een specifiek element op te vragen gebaseerd op de `id`:

```
document.getElementById("idName"); //returns an HTMLObject
```

Als je een element in een HTML-pagina wilt opvragen, begin je altijd met het refereren naar het `document-object`.

Omdat een `id` uniek is (of moet zijn) geeft deze functie één `HTML-node` terug. Het is een goed idee om die in een variabele op te slaan (`let mijn_html_object = document.getElementById("idName");`), omdat we dikwijls verschillende dingen met die node gaan doen, bijvoorbeeld de inhoud ervan aanpassen of de stijl op verschillende manieren aanpassen.

Andere manieren om een element te krijgen is via de klassennaam of de tagnaam. Hier wordt dan echter steeds een lijst greturnde op basis van de volgorde in het HTML-bestand.

```
document.getElementsByClassName("className");           //returns an  
HTMLCollection  
document.getElementsByName("valueOfAttributeName"); //returns an  
HTMLCollection  
document.getElementsByTagName("tagName");            //returns an  
HTMLCollection
```

Verder kan je ook **CSS-selector syntax** gebruiken om een element op te vragen. De **querySelector**-functie geeft telkens het eerste **HTMLObject** terug dat voldoet aan de query. De **querySelectorAll**-functie geeft een **NodeList** met alle objecten terug die voldoen aan de query.

```
document.querySelector("#idName");      //returns an HTMLObject  
document.querySelector(".className");   //returns an HTMLObject  
document.querySelector("tagName.className:not(.className)  
tagName[attributeName='attributeValue']"); //returns an HTMLObject  
  
document.querySelectorAll("#idName");    //returns a NodeList
```

Sommige functies geven een **HTMLCollection** terug terwijl anderen een **NodeList** teruggeven. Een **NodeList** bevat bovenop descendant HTML-elementen ook de stukken tekst die tussen andere eventuele HTML-elementen gebruikt worden. Dit wordt belangrijk bij het opvragen van **children** (**HTMLCollection**) of **childNodes** (**NodeList**)

Wil je over zo een lijst elementen itereren dan kan je de **for(let ... of ...)**-syntax gebruiken.

Informatie over elementen inspecteren

Nu we een element kunnen opvragen, kunnen we meer specifieke informatie over dat element inspecteren zoals de waarde van attributen, textcontent, descendants ...

```
console.log(html_object.innerText);  
console.log(html_object.textContent);  
console.log(html_object.innerHTML);  
console.log(html_object.children);  
console.log(html_object.childNodes);  
  
console.log(html_object.getAttributeNames());  
console.log(html_object.getAttribute("attributeName"));  
console.log(html_object.getAttributeNode("attributeName"));
```

```
console.log(html_object.style.fontSize); //Dit werkt niet altijd op deze  
manier wanneer er specifieke CSS styling werd toegepast.  
//Gebruik dan volgende methode in de plaats:  
console.log(window.getComputedStyle(html_object,  
null).getPropertyValue('font-size'));
```

Info

Je kan de console gebruiken om te inspecteren welke attributen een HTML-element allemaal bevat. Bovendien kan je in de documentatie steeds terugvinden wat de return-typen zijn van de verschillende functie. Je kan echter ook zeer snel het type controleren met de JS-functie `typeof naamObject`.

Warning

Wanneer een attribuut of object niet gevonden wordt, wordt meestal `undefined` gereturned.

Webcontent manipuleren

Je kan nu ook snel content aanpassen door de attribuut waarde te veranderen:

```
html_object.innerText = "nieweTekst";  
html_object.innerText = '<h1 id="nieuweId" > Nieuwe header </h1>'; // Je kan dus zelfs dynamisch HTML-elementen toevoegen of verwijderen  
html_object.style.fontSize = "15px";  
  
html_object.setAttribute("attributeName", "newValue");  
  
//Examples  
voornaamInput_object.setAttribute("value", "Arne");  
inputCheckbox_object.setAttribute("checked", "true");
```

Gebruiker interactie verwerken

Je kan rechtstreeks in een HTML-element functies (of zelfs gewoon JS code) toevoegen om uitgevoerd te worden bij bepaalde acties van de gebruiker.

```
<button onclick="functionName">Click me</button>
<button onclick="console.log('button clicked')">Click me</button>
```

Dit doe je echter beter in de JavaScript file zelf door **eventListeners** te koppelen aan de gewenste HTML-elementen. Een aantal voorbeelden vind je hieronder. Je kan onder andere een listener toevoegen voor **click**, **mouseover**, **mouseout**, **mousemove** ...

```
html_object.addEventListener("click", functionName);
// Je kan ook rechtrstreeks functies definiëren als parameter:
html_object.addEventListener("click", function(){alert("I was clicked")});
// Of door gebruik te maken van de pijl notatie:
html_object.addEventListener("click", () => {alert("I was clicked")});

//Je kan ook eventListeners toevoegen aan je window
window.addEventListener("resize", function() { console.log("resized
window");});
```

Event-handlers in JavaScript

In plaats van zelf te kiezen wanneer code uitgevoerd wordt, willen we meer controle leggen bij de gebruiker, net zoals het in “echte” programma’s gebeurt: de gebruiker kan hierbij beslissen wanneer er iets moet gebeuren, bv. wanneer hij op een toets drukt, met het gamepad speelt of met de muis een knop indrukt. Dit kunnen we ook in JS doen door code te laten reageren op **events**, de verzamelnaam voor alle gebeurtenissen die externe ‘gebruikers’ kunnen veroorzaken.

Een volledig overzicht van events kan je [hier terugvinden](#)

Niet alle events zijn van toepassing op alle html-elementen, maar welke events op wat van toepassing zijn, is vrij vanzelfsprekend en daar gaan we dus niet dieper op in.

Het Event object

Net zoals in Java weet je op basis van de event-handler wel welk event er plaats vond: een muisklik, een toets ingedrukt, de muis bewogen, ... maar niet welke muistoets, op welke coördinaat, welke toets, ... Willen we dat toch weten, dan moeten we het **Event**-object gebruiken. Zo een object wordt automatisch aangemaakt in elke **event-handler** en krijgt daar de naam **event**. In de praktijk gaan we dit event doorgeven aan de functie die we in de event-handler oproepen, bv.

```
//html
<div id="naam" onclick="kijkEven(event); ">blabla</div>
//javascript
function kijkEven(event) { alert(event); }
```

Je kan nu zien dat de browser in zijn `alert [object MouseEvent]` toont, wat aangeeft dat er zo een object is voor het muisevent, en dat je daarvan verschillende **eigenschappen** kan opvragen. Soortgelijk heb je ook een `KeyEvent` e.d. met zijn eigenschappen. Met deze eigenschappen kan je extra informatie opvragen, zoals de positie van de muisklik, de muistoets enzoverder. **Dergelijke eigenschappen kan je oproepen door achter de object-naam een punt te typen en dan de naam van de eigenschappen.**

Online kan je meer info vinden over de verschillende eigenschappen van de events. Op w3schools halen we al een lijst voor de [MouseEvent](#) en de [KeyboardEvent](#).

Met alert, confirm en prompt kan je snel pop-up berichtjes toevoegen

De `alert()` is de `MsgBox()` van Javascript. Met deze functie kan je een boodschapvenster op het scherm toveren, bv. `alert("Hoe gaat het er mee?");`. Dit is een manier om (vrij opdringerig) met de gebruiker te communiceren, maar het kan ook handig zijn als foutopsporingshulpje. Je kan immers op verschillende plaatsen in je code een `alert()` zetten, wat een soort ‘pauze’-functie verzorgt en waarmee je dus kan zien of en hoe lang je programma correct verloopt.

```
//alert
alert("Alert message, click 'ok' to continue.");
//confirm
```

```
var confirmOutput = confirm("Confirm message, click 'ok' or 'cancel' to
continue.");
console.log(confirmOutput)
//prompt
var promptOutput = prompt("The prompt message");
console.log(promptOutput)
```

Forms and the value property

Zoals we eerder al zagen, is het formulier de plaats in HTML waar de gebruiker input kan geven over wat er moet gebeuren. In JS gaan we via een **event-handler** definiëren welke functie uitgevoerd moet worden. Dikwijls gaan we die aan een knop hangen, maar we moeten er dan wel voor opletten dat de knop van het **type="button"** is, en NIET van het **type="submit"**, want die tweede knop gaat de **action** uitvoeren en een volledig nieuw document inladen (zie PHP). Bovendien gaan we de formulierelementen met de naam via het attribuut **id**, bv. **id="tekst"** werken. Met dit **id** en de functie **document.getElementById()** kunnen we dan de form ophalen. Via **de eigenschap .value** kan je vervolgens de inhoud van het formulieveld ophalen.

*Bij PHP gaat het zo zijn dat je in het formulier een attribuut **action** moet gebruiken om aan te geven welk script de form moet verwerken. Met het attribuut **name** kies je een naam die je in PHP kan gebruiken in tegenstelling to JS waar we de naam van opgegeven in de **id** gebruiken.*

De optelling en parseInt()

De **.value** property geeft altijd een **String** terug. Wanneer je dan twee invoerwaarden ‘optelt’, wordt de **+** van de **String** gebruikt, en dit is, net als in Java, de concatentatie: “Johnny” + “ en ” + “Marina” is dan “Johnny en Marina”; “33” + “4” is “334”. Het eerste vindt iedereen logisch; het tweede veel minder. Om JS te dwingen de tekst als een getal te beschouwen, moet je de functie **parseInt()** gebruiken. Deze verwacht als parameter een tekst en zal zijn uiterste best doen om uit die tekst een geheel getal te halen. Wanneer dit niet lukt, zal hij de waarde **NaN** geven: **Not A Number**.

*Deze **parseInt()** geeft dus (meestal) een geheel getal terug. Wil je een niet-geheel getal, dan gebruik je **parseFloat()**.*

Om te controleren of er effectief een getal **geparsed** kon worden, kan je de functie **isNaN(x)** gebruiken. Deze geeft een **boolean** terug met **true** als de parameter NaN is.

Om even terug te komen op de optelling: wanneer je getallen en teksten combineert in een optelling, doet Javascript dat van links naar rechts. "De som van " + 3 + " en " + 4 + " is " + 3 + 4 geeft dan "De som van 3 en 4 is 34", omdat je begint met tekst, daar dan 3 bij optelt, vervolgens " en ", 4, en " is ". Dit tussenresultaat is "De som van 3 en 4 is". Hier 3 bij optellen plaatst 3 erachter, en vervolgens komt achter die tekst nog eens 4. "De som van " + 3 + " en " + 4 + " is " + (3 + 4) geeft wel "De som van 3 en 4 is 7". **Haakjes spelen dus een belangrijke rol!**

De style property

Om veranderingen te beklemtonen, kan het interessant zijn om hoofdingen te markeren, lettertypes en/of achtergronden te veranderen, ... In een static page gebruiken we CSS om die vormgeving in te stellen met de property **style** doen we dat in JavaScript. Deze eigenschap **style** is dus **de directe link tussen CSS en Javascript**. In feite is **style** niet zomaar een eigenschap, maar een echt object. Het heeft zelf verschillende eigenschappen die je stuk voor stuk kan veranderen. Zo zal volgende code de achtergrondkleur van de paragraaf veranderen wanneer je er op klikt.

```
<p id="par" onclick="veranderParagraaf()">blablabla</p>

<script type="text/javascript">
    function veranderParagraaf() {
        var paragraaf = document.getElementById("par");
        paragraaf.style.backgroundColor = "red";
    }
</script>
```

Merk op dat het liggend teken in CSS niet gebruikt kan worden in Javascript (omdat het als ‘min’ geïnterpreteerd zou worden). Daarom wordt het vervangen door een hoofdletter van het volgende woord: **background-color** is **backgroundColor** geworden. Een volledig overzicht van de eigenschappen die via het **.style**-element van JS aangepast kunnen worden, vind je op [hier](#).

Je kan ook `.style.cssText` gebruiken en dan de CSS-stijl als string meegeven, bv.

```
element.style.cssText = 'color:red;backgroundColor:yellow';
```

De oude property opvragen

Bij het veranderen van de properties kan je in principe de oude waarde hergebruiken door de oude property op te vragen, bv.

```
alert(document.getElementById('par').style.fontSize);
```

Zo zou je bijvoorbeeld de margin kunnen verdubbelen of het lettertype **10%** groter zetten. Spijtig genoeg zijn er twee problemen:

1. de waarde die je in je CSS definieert komt niet zomaar in je JS terecht. Het is pas nadat je die met JS veranderd hebt, dat je de correcte waarde terug krijgt.
2. de waarde die je terug krijgt heeft ook een eenheid, bv. 14px of 200%. Dat zijn tekstwaarden en die kan je niet zomaar vermenigvuldigen met 1,1 of 2 bij optellen.

Het eerste probleem kan je oplossen door de waarde met JS te initialiseren op het einde van de pagina, bv

```
document.getElementById('par').style.fontSize = '14px';
```

Voor het tweede probleem gebruiken we de `parseInt` van hierboven, doen een bewerking en zetten er dan de tekst "px" achter, zoals in volgend voorbeeld:

```
<p id="par" onClick="veranderParagraaf()">blablabla</p>

<script language="javascript">
    function veranderParagraaf() {
        var paragraaf = document.getElementById("par");
        paragraaf.fontSize = (parseInt(paragraaf.fontSize) + 3) + "px";
    }
</script>
```

De className property

Wanneer je met CSS je stijlen in `classes` hebt ingedeeld kan je ook rechtstreeks de property **className** van het HTML-element gebruiken. Dan kan je in één keer een heel nieuwe stijl of animaties toekennen aan een specifiek HTML-element, bv.

```
document.getElementById('par').className = "red";
```

Content wegschrijven naar localStorage of sessionStorage

Je kan met JavaScript ook data opslaan in je browser. Dit is steeds een **key-value pair** waarbij beide waarde **Strings** zijn. Wegschrijven naar **localStorage** behoudt de data tussen sessies (sluiten en heropenen van de webpagina). Wegschrijven naar **sessionStorage** behoudt de data tussen sessies **NIET**. Maak gebruik van **JSON** om gemakkelijk data objecten op te slaan:

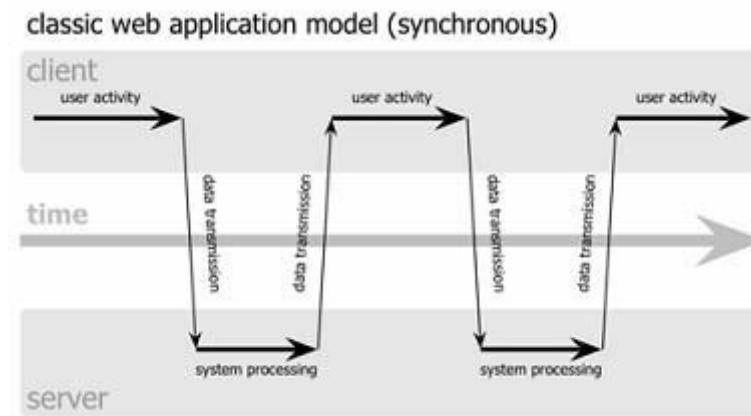
```
//Het op te slagen object in JSON formaat
var testObject = {
  'one': 1,
  'two': 2, 'three': 3
};
//Opslaan in localStorage
localStorage.setItem('testObject', JSON.stringify(testObject));
//Inladen vanuit localStorage
var retrievedObject = localStorage.getItem('testObject');
var retrievedTestObject = JSON.parse(retrievedObject);

//idem voor sessionStorage.
```

Vergeet je objecten dus niet te **JSON.stringify**-en om ze op te slaan en te **parse**-n om ze van text terug om te zetten naar waardige JS objecten.

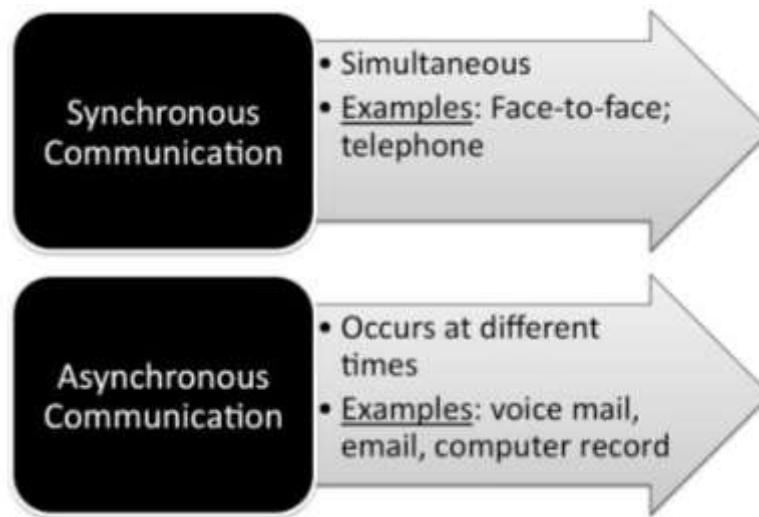
Synchroon vs Asynchroon

In het **synchrone** model stuur je **een request voor een nieuwe pagina** (via een link, een URL of een formulier) en vervangt de binnenkomende response de hele pagina. De server kan tijdens de uitvoering van zijn script (andere) webservices synchroon oproepen. De responses voor die requests moeten helemaal binnen zijn vooral het script verder gaan en op het einde de nieuwe pagina terug geeft. Dit systeem zie je hieronder:



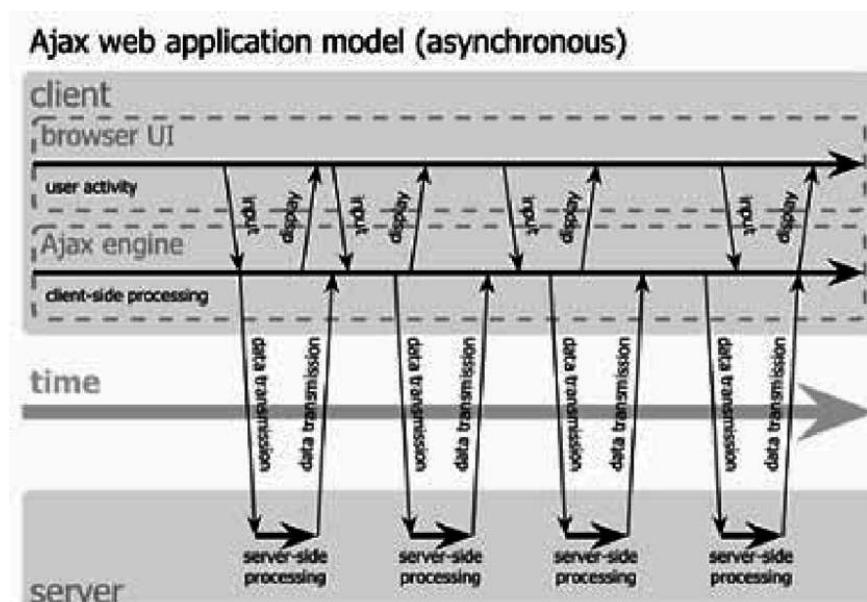
Voorbeeld Synchroon [[bron](#)]

Dit werkt goed voor een aantal toepassingen, zoals telefonie, maar voor de hedendaagse, veeleisende internetgebruiker is het asynchrone model v  l gepaster. Hierbij wordt de request verstuurd en wacht de client niet op het antwoord van de server vooraleer verder te gaan. Ondertussen wacht er op de achtergrond een call-back tot de request terug is.



Voorbeelden Synchroon vs Asynchroon

In het **asynchrone** model stuur je dus **een request voor nieuwe data** eventgebaseerd via Javascript: de gebruiker klikt op een knop, selecteert tekst, beweegt over een figuur, ... en als gevolg daarvan gaan we een request sturen. We wachten echter **niet** tot de response helemaal binnen is vooraleer de gebruiker nieuwe acties mag ondernemen, maar zorgen er voor dat de gebruiker verder kan blijven werken en bijkomende requests kan sturen. Het binnenkomen van de responses kan nu in een andere volgorde gebeuren dan het versturen van de request omdat de ene request meer tijd nodig kan hebben dan de andere.



Voorbeeld Asynchroon [bron]

Een response die binnenkomt, zal er nu voor zorgen dat slechts een deel van de pagina aangepast wordt, bijvoorbeeld door tekst toe te voegen aan een `<div>`, door een formulierveld inhoud te geven, door menu-opties te veranderen, door bepaalde blokken van kleuren te veranderen, ... Hiervoor moeten we zowel de inhoud als de vormgeving van HTML-elementen op de huidige pagina kunnen veranderen. Hoe we dat laatste stuk doen, hebben we hierboven al verteld, maar de asynchrone communicatie met de server moeten we nog bespreken. Dit proces zie je hierboven. In deze voorbeeldfiguur komen de verschillende antwoorden nog in dezelfde volgorde binnen als de requests verstuurd zijn, maar het kan perfect zijn dat de eerste response pas binnenkomt als de response op de derde request al lang binnen is. We hebben dan een volledig **asynchrone** model.

Het grote voordeel van het asynchrone model is dat je veel directere interactie met de gebruiker kan krijgen. Het nadeel is dat het iets moeilijker is om te implementeren en dat je een extra technologie nodig hebt. Dat je hiervoor Javascript nodig hebt, zal je al wel doorhebben, want anders zouden we dit niet bespreken in het hoofdstuk Javascript. Concreet praten we dan over **Ajax**, wat meestal in combinatie met REST gebruikt wordt.

REST met Javascript ≈ Ajax

De volgende stap is om de HTML-elementen te wijzigen, niet alleen maar op basis van vaste code of formulierinvoer, maar op basis van externe webpagina's/webservices. Hiervoor moeten we dus een (asynchrone) request kunnen sturen, en **wanneer we de response ontvangen** deze verwerken of rechtstreeks in een div plaatsen. Omdat de request dikwijls als XML terug komt, heb je 3 technologieën die samenkommen: Asynchrone, Javascript en XML. Dit wordt afgekort tot Ajax met de A van Asynchrone, de JA van Javascript en de X van XML. Tot enkele jaren geleden had je gespecialiseerde bibliotheken nodig zoals Prototype of jQuery om dit in verschillende browsers op dezelfde manier te kunnen doen, maar nu kan dit gemakkelijk met fetch de bijhorende Promises en de functionele programmeerstijl.

Basisprincipe = request + response opvangen via call back functie

In zijn meest eenvoudige vorm ga je een webpagina opvragen door rechtstreeks zijn URL op te geven en plaatsen we de response rechtstreeks in een div. Deze

vorm klinkt heel simpel, maar wordt toch heel veel gebruikt omdat je in die URL heel gemakkelijk GET-parameters mee kan geven. En wanneer de webservice direct HTML terug geeft, kan je die zonder problemen in een div plakken.

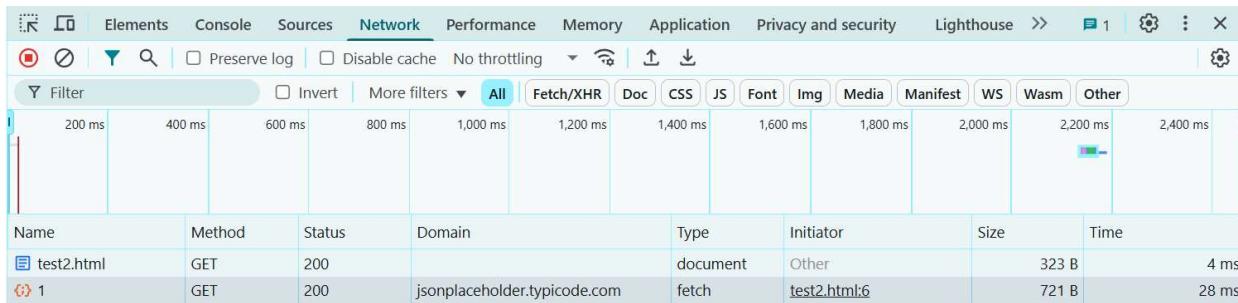
Het moeilijke is echter dat je die response asynchroon wil opvangen. Hiervoor moet je een `call back` functie definiëren die opgeroepen wordt (“call”) wanneer de response terug komt (“back”). Voor de komst van `fetch` had je hiervoor redelijk veel boilerplate-code nodig en stond de callback redelijk ver van de oproep. Met `fetch` wordt het allemaal veel compacter. Hieronder overlopen we in kleine stappen het hele proces tot en met het opvangen van fouten.

Een request versturen

De request versturen is extreem eenvoudig: gebruik de `fetch`-functie met als enige parameter de URL van de service. Hieronder vind je een minimalistisch voorbeeld:

```
<html>
  <head>
    <script type="text/javascript">
      function fetchDemo() {
        fetch("https://jsonplaceholder.typicode.com/posts/1");
      }
    </script>
  <body>
    <div id="feedback"></div>
    <div id="input">
      <button onclick="fetchDemo()" type="button">fetch</button>
    </div>
  </body>
</head>
</html>
```

Wanneer je dit uitprobeert in je favoriete browser zie je niets omdat we nog een call-back gedefinieerd hebben. Als je echter de ontwikkelaaropties activeert en je bij het netwerkverkeer gaat kijken, kan je echter zien dat er een request verstuurd is.



Je kan de request dan openklikken en de response bekijken, bijvoorbeeld als volgt:

The screenshot shows the Response tab for the 'test2.html' request. The response body contains a JSON object:

```

1  {
2   "userId": 1,
3   "id": 1,
4   "title": "delectus aut autem",
5   "completed": false
6 }

```

Controleren of de response OK is

Met dergelijke request kunnen verschillende dingen mis gaan: het kan zijn dat de URL niet bestaat, dat je verkeerde argumenten meegeeft, dat je niet de juiste rechten hebt, ... Het is dan ook zaak eerst te controleren of de response ok is. Hiervoor ga je een eerste call-back definiëren. Hiervoor gebruik je de **then**-handler. De eerste then-handler neemt **als argument de response** en kan ofwel dat argument helemaal consumeren; ofwel een nieuwe Response terug geven. In de eerste versie hieronder geven we niks meer terug, maar vanaf de volgende sectie geven we een nieuwe response terug om zo het werk beter te verdelen.

De then-handler verwacht een functie-expressie: dit is ofwel een verwijzing naar een bestaande functie (gewoon de naam van de functie) ofwel een **lambda-expressie**. In Javascript heeft zo'n lambda-expressie de vorm parameter => return-waarde. Zo is `x => x * 2` een functie in de vorm van een lambda-expressie die zijn parameter verdubbelt. Je mag ook verschillende statements samenzetten achter de =>, maar dan moet je die in een blok met accolades zetten en explicit return gebruiken wanneer je iets wil terug geven.

In de eerste versie gebruiken we een lambda-expresie die de response in een alert plaatst.

```
function fetchDemo() {
  fetch("https://jsonplaceholder.typicode.com/posts/1")
    .then(response => alert(response));
}
```



Je ziet duidelijk in je alert dat je een object van het type Response terug krijgt uit de fetch. Met `console.log(response)` kan je het volledige object in de console bekijken.

```
▼ Response ⓘ
  body: (...)

  bodyUsed: false
  ▶ headers: Headers {}
  ok: true
  redirected: false
  status: 200
  statusText: ""
  type: "cors"
  url: "https://jsonplaceholder.typicode.com/todos/1"
  ▶ [[Prototype]]: Response
```

Om te controleren of de response in orde is, moeten we de `ok` eigenschap opvragen met voor de hand liggende betekenis. De `then`-handler wordt dan als volgt:

```
function fetchDemo() {
  fetch("https://jsonplaceholder.typicode.com/posts/1")
    .then(response => {
      if (response.ok) alert("Alles ok");
      else alert("Niet ok");
    });
}
```

De response converteren naar het juiste formaat

Herinner je dat de laatste letter van de afkorting Ajax verwijst naar XML, maar dat er in de praktijk veel meer JSON wordt gebruikt. XML wordt hierbij als gewone tekst beschouwd, terwijl er voor JSON extra verwerking voorzien is. Bij JSON heb je immers een reeks velden met telkens een waarde. Door de response als JSON te verwerken kom je dan een object met als data members de velden in de JSON-teks. Om dit te faciliteren heeft men in fetch voor objecten van het type **Response** een methode **.text()** die de zuivere tekst-waarde terug geeft; of beter gezegd een **Promise** met als parameter een zuivere tekst; en een methode **.json()** die een **Promise** terug geeft met als parameter een JSON-object.

Omdat we in de Response van hierboven konden zien dat de service op <https://jsonplaceholder.typicode.com/posts/1> een JSON terug geeft, gebruiken we de **.json()**-methode.

```
function fetchDemo() {
  fetch("https://jsonplaceholder.typicode.com/posts/1")
    .then(response => {
      if (response.ok) return response.json();
      else alert("Niet ok");
    })
    .then(json => console.log(json));
}
```

Een aparte functie om de JSON te verwerken

Om de JSON verwerken, schrijf je best een aparte functie zodat je aparte verantwoordelijkheden mooi in aparte functies steekt. In het voorbeeld hieronder hebben we een (lege) functie `showData` gemaakt en die opgeroepen i.p.v. de `alert(json)`.

```
function fetchDemo() {
  fetch("https://jsonplaceholder.typicode.com/posts/1")
    .then(response => {
      if (response.ok) return response.json();
      else alert("Niet ok");
    })
    .then(json => showData(json));
}
function showData(json){}
```

Dat is correct, maar het kan compacter door i.p.v. `json => showData(json)` gewoon de verwijzing naar de functie te vermelden. Dat zie je hieronder:

```
function fetchDemo() {
  fetch("https://jsonplaceholder.typicode.com/posts/1")
    .then(response => {
      if (response.ok) return response.json();
      else alert("Niet ok");
    })
    .then(showData);
}
function showData(json){}
```

Om de functie `showData` nuttig in te vullen gebruiken we de `querySelector`, de eigenschap `.innerHTML` en de data members van het JSON-object. Bijkomend hebben we ook nog de stijl van de div aangepast.

```
function showData(json){
  console.log(json);
  let fb = document.querySelector('#feedback');
```

```
fb.innerHTML = json.title;
fb.style.backgroundColor = "#ae8";
}
```

Fouten opvangen via catch

Het kan nog een tikkeltje beter: net als in Java kan je fouten opvangen via een try-catch-achtige structuur. Hiervoor moet je natuurlijk wel eerst een gepaste Exception gooien voor je die kan opvangen. Zo'n Exception gooi je door een Response.reject terug te geven met als parameter een passende foutmelding, bijvoorbeeld: `return Promise.reject("Wrong address");`

Het opvangen van de exception doe je door een .catch toe te voegen aan de ketting van handlers, bijvoorbeeld als volgt: `.catch(err => alert(err));`

Samen wordt dit als volgt:

```
function fetchDemo() {
  fetch("https://jsonplaceholder.typicode.com/posts/1")
    .then(response => {
      if (response.ok) return response.json();
      else return Promise.reject("Wrong address");
    })
    .then(showData)
    .catch(err => alert(err));
}
```

Get-Parameters toevoegen

In het voorbeeld hierboven zag je dat de laatste parameter van de GET-request **1** was (het laatste achter de / elke / staat voor een parameter die je meegeeft. In bovenstaand voorbeeld vragen we dus de **/posts** op van de api en daarvan de eerste **/1**). Wanneer je dit getal niet hard wil coderen, maar laten afhangen van de waarde die de gebruiker in een tekstvak invult, kan je op eenvoudige wijze met de querySelector het desbetreffende input-veld ophalen, met **.value** de waarde van het veld ophalen en dan een URL genereren.

Post-Parameters toevoegen

Soms gebruik je geen GET-request om dingen op te vragen (hierbij staat alle info in de url), maar gebruik je een POST-request, hierbij moet je extra info dan in de

Post-parameters toevoegen. Dit is een tikkeltje moeilijker. De functie fetch krijgt nu als tweede parameter een object met daarin minstens 3 velden:

- **method**: om te kunnen zeggen dat we de gegevens met POST versturen
- **headers**: om het formaat te beschrijven waarin we de gegevens versturen. JSON is hierbij het gemakkelijkste.
- **body**: om de gegevens te versturen. Dit is een tikkeltje moeilijker omdat we eerst een gewoon object moeten maken, en dan dat object omzetten in JSON. Dat laatste kan je met `JSON.stringify(jsonObject)`.

Op <https://css-tricks.com/using-fetch/> vind je hiervan volgend voorbeeld:

```
let content = {some: 'content'};

// The actual fetch request
fetch('some-url', {
  method: 'post',
  headers: {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify(content)
})
// .then()...
```

Een voorbeeld voor onze eigen demo:

```
function fetchDemo() {
  fetch('https://jsonplaceholder.typicode.com/posts', {
    method: 'POST',
    body: JSON.stringify({
      title: 'My post for FSWEB',
      body: 'My super great post about what I learned during the FSWEB
courses.',
      userId: 1,
    }),
    headers: {
      'Content-type': 'application/json; charset=UTF-8',
    }
  })
  .then(response => response.json())
  .then(data => console.log(data))
}
```

```
    },
  })
  .then(response => {
    if (response.ok) return response.json();
    else return Promise.reject("Wrong address");
  })
  .then(showData)
  .catch(err => alert(err));
}

}
```

CSRF en CORS

In de beginjaren van het internet heerde er een zeer open filosofie: iedereen mocht probleemloos elkaars services gebruiken. Omdat er echt ook minder betrouwbare sujetten op het internet vertoeven, kwam men snel in de problemen: malafide lieden misbruikten andermans services voor eigen geldgewin en/of om mensen af te zetten of te haken. Dit concept noemt men **Cross Site Request Forgery**, of afgekort CSRF: een valse of ongewenste request op je site.

Tegenwoordig is het standaard ingesteld dat services niet vanop andere computers opgeroepen kunnen worden. Laravel is hier zelfs nog extra streng in, want niemand, ook je eigen site niet, kan een POST-service van Laravel oproepen tenzij je het csrf-token meegeeft. Dat is veilig, maar tegelijk ook restrictief: het hele concept van Service Oriented Architectures vervalt dan. Gelukkig kan je dat dus ook terug open zetten. Dit noemt men dan **Cross-Origin Resource Sharing** of afgekort CORS. Je kan met CORS specifieke IP-adressen toegang geven of ook iedereen toegang geven, maar dat is dikwijls toch weer een tikkeltje te vrij. Daarom voorzien de meeste service-aanbieders nog een extra authenticatie, maar daar komen we laten op terug.

Het is afhankelijk van de programmeertaal waarin de service geschreven is, hoe je die CORS wordt open gezet.

Extra voorbeelden op de async functions, promises, await & fetch from api's

Soms wil je dingen opvragen of kan het een tijd duren voordat een functie een return geeft. Om te voorkomen dat je gedurende die tijd niet kan interageren met de website, moet je gebruik maken van de **async** functions. Ergens in je functie gebruik je dan het woord **await** zodat de rest van je functie wacht en pas verder gaat wanneer je een return waarde hebt ontvangen. Tijdens het wachten kan dan andere code uitgevoerd worden.

```

function resolveAfter2Seconds() {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve('resolved');
    }, 2000);
  });
}

async function asyncCall() {
  console.log('calling');
  const result = await resolveAfter2Seconds();
  console.log(result);
  // Expected output: "resolved"
}

// MET: <button onclick="asyncCall()" type="button">async call</button>

```

Een voorbeeld waarvoor je vaak de **async function** gebruikt is bij het gebruik van api's om data op te halen/weg te schrijven zoals hierboven al uitgebreid aangehaald:

```

<html>
<head>
  <script type="text/javascript">
    async function cuteDogPicture() {
      fetch('https://dog.ceo/api/breeds/image/random')
        .then((response) => {           // Parameter 'response' refers to the
          return of the function above this (fetch)
          return response.json();
        })
        .then((myContent) => {           // Parameter 'myContent' refers to
          the return of the function above this
          let fb = document.querySelector('#feedback');
          fb.innerHTML = '<img src=' + myContent['message'] + '/>';
        });
    }
</script>

```

```

        }
    </script>

<body>
    <div id="feedback"></div>
    <div id="input">
        <button onclick="cuteDogPicture()" type="button">fetch</button>
    </div>
</body>
</head>

</html>

```

Opdrachten reeks 1

- Breed onderstaand voorbeeld, waar we de invoer in een tekstvak in een alert laten verschijnen, uit tot een formulier met twee velden: een voor de tekst en een voor kleur. Wanneer de gebruiker op de knop klikt, moet de tekst in een tweede div verschijnen met als achtergrondkleur de kleur die gebruiker ingegeven had.

```

// met invoer een html-input element van het type `text`, met als `id` "tekst"
function toonInvoer() {
    var invoer = document.getElementById("tekst").value;
    alert(invoer);
}

```

- Maak een opteller in HTML en JavaScript: een simpel formulier met 2 tekstvakken waar men een getal moet invullen en een div waar je de uitkomst in zet.
- Maak een quizje met drie meerkeuzevragen. Als men op het foute antwoord klikt, moet dat antwoord een rode achtergrond krijgen. Klikt men op het juiste antwoord, dan moet dit een groene achtergrond krijgen.

- Voeg op minstens 2 plaatsen JavaScript toe aan je portfolio website. Ten eerste zorg je ervoor dat wanneer mensen op de ‘submit’-knop drukken in je contactenformulier dat je de data van de inputvelden op een correcte manier in een object variabele opslaat. Sla daarna de data op in je **sessionStorage** door gebruik te maken van JSON. Zorg er ook voor dat wanneer je je contactformulier opent het formulier al is ingevuld met de laatst opgeslagen gegevens indien ze bestaan.
- Zoek een interessante API die je kan oproepen met AJAX en kan gebruiken in je website. Maak het zo dat je verschillende oproepen kan doen op basis van de input van de gebruiker.

Test jezelf

Klik [hier](#) om jezelf te testen met de online JS quiz van w3schools. Of klik [hier](#) voor wat extra oefeningen.

RESPONSIVE DESIGN

Starting fresh

Start je CSS file met volgende code om enkel “ongewenst” default gedrag van de browser te overschrijven:

```
*,
*:before,
*:after {
    box-sizing: border-box; /* Zorgt ervoor dat default padding en border worden meegerekend met breedte en hoogte */
}

* {
    margin: 0; /* Zorgt ervoor dat elk element default geen margin heeft */
    padding: 0; /* Zorgt ervoor dat elk element default geen padding heeft */
}
```

Positioning (herhaling)

- `position: absolute;` item is verwijderd van het document.
 - `top, right, bottom, left` unlocked.
 - zet `position: relative;` in parent om het element relative te positioneren t.o.v. van parent i.p.v. hele document.
 - gebruik `z-index` om elementen naar voor te halen of naar achter te brengen.

- (Wanneer je elementen met negatieve waarden niet meer passen op het scherm< verschijnt een scroll bar.Verwijder de scroll bar met **overflow: hidden;** in parent or body)
- **position: relative;** position relative t.o.v. parent en blijven in de normal flow van het document.
 - **top, right, bottom, left** unlocked.
- **position: static;** default, er gebeurt niks.
- **position: fixed;** gelijkaardig aan absolute en dus ook verwijderd uit document, maar item volgt nu met het scrollen en altijd relative t.o.v. het html element. (ideaal voor headers and footers)
- **position: sticky;** werkt alleen wanneer **top** of **bottom** property is meegegeven.
 - blijft enkel sticky binnen de parent.

Voorbeeld zie [demo 1](#)

Flexbox

Geeft het element 2 assen om elementen op te plaatsen en de default as is horizontaal. Dus onze items worden niet meer boven elkaar getoond maar naast elkaar. Bijvoorbeeld:

```
<div class="container">
  <div class="item item1">Item 1</div>
  <div class="item item2">Item 2</div>
  <div class="item item3">Item 3</div>
</div>

.containter {
  display: flex;

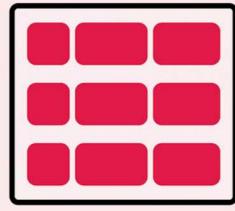
  /*Verander de main axis: column , row (default)*/
  flex-direction: row;

  /*Hoe moeten de elementen geplaatst worden op de main axis: flex-start
  (default) , flex-end , center , space-between , space-around , space-evenly*/
  justify-content: flex-start;
```

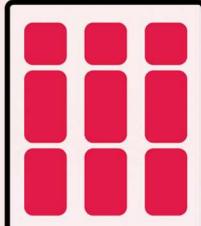
```
/*Hoe moeten de elementen geplaatst worden op de cross axis: flex-start  
  (default) , flex-end , center , baseline*/  
align-items: flex-start;  
  
/*Hoe omgaan met overflow: nowrap (default) , wrap*/  
flex-wrap: wrap;  
/*Wrap unlocks align-content zelfde opties als `justify-content`*/  
align-content: flex-start;  
  
/*Plaats tussen elementen aanpassen: 0px (default)*/  
gap: 1em;  
}  
  
.item.item1{  
/* flex-grow: 1; */  
flex-shrink: 5; /*item 1 shrinkt 5x sneller dan de andere (0 item shrinkt  
niet)*/  
flex-basis: 300px; /*overschrijft breedte item in flex container*/  
  
/*SHORTHAND: grow shrink basis*/  
flex: 1;  
  
align-self: center; /*overschrijf align van container*/  
order: 2; /*verander de volgorde waarin elementen in flex container getoond  
worden t.o.v. volgorde in html bestand. (beter via HTML aanpassen)*/  
}  
  
.item.item2{  
flex-grow: 2; /*neemt dubbel zoveel plaats in als 1 en 3*/  
}  
  
.item.item3{  
flex-grow: 1;  
}
```

CSS Flexbox

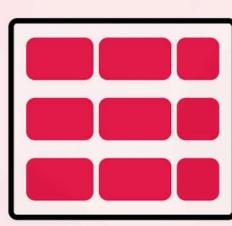
flex-direction



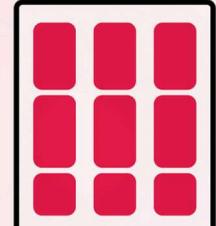
row



column

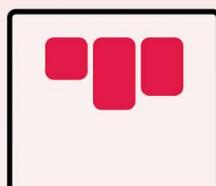


row-reverse

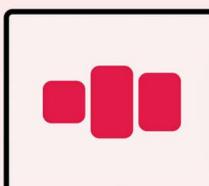


column-reverse

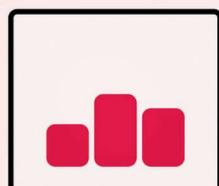
align-items



flex-start



center



flex-end



stretch

justify-content



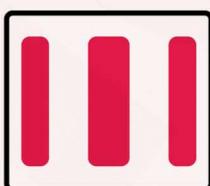
flex-start



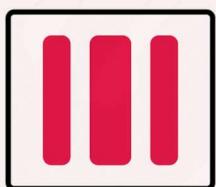
center



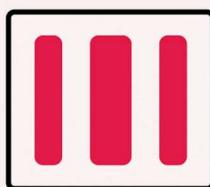
flex-end



space-between



space-around

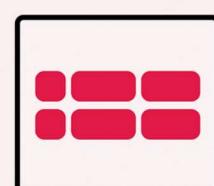


space-evenly

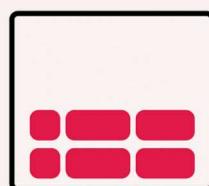
align-content



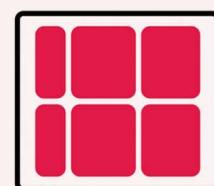
flex-start



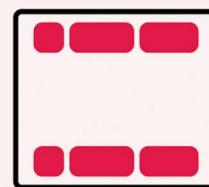
center



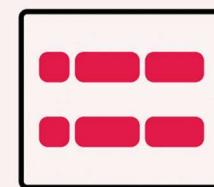
flex-end



stretch



space-between



space-around

@eludadev

Flex info [bron](#)

Voorbeeld zie [demo 2](#)

Grid

Met behulp van **Grid** kan je items plaatsen op basis van een soort grid coördinaten: Bijvoorbeeld:

```
<div class="container">
  <div class="item item1">Item 1</div>
  <div class="item item2">Item 2</div>
  <div class="item item3">Item 3</div>
</div>

.containter {
  display: grid;
  /*Geef het aantal rijen en kolommen mee*/
  grid-template-rows: 100px 100px 100px; /*3 rijen van 100px*/
  grid-template-columns: 100px 100px 100px; /*3 kolommen van 100px*/

  grid-gap: 1em 2em; /*eerst gap tussen rows dan columns*/

  /*Naamgeving grid vakken*/
  grid-template-areas:
    'header header header'
    'vak4 vak5 vak6'
    'vak7 vak8 vak9';
}

/*Items positioneren*/
.item.item1{
  grid-row-start: 1;
  grid-row-end: 2;
  /*SHORTHAND*/
  /* grid-row: 1 / 2; */
  grid-column-start: 1;
  grid-column-end: 4;
  /*SHORTHAND*/
  /* grid-column: 1 / 4; */

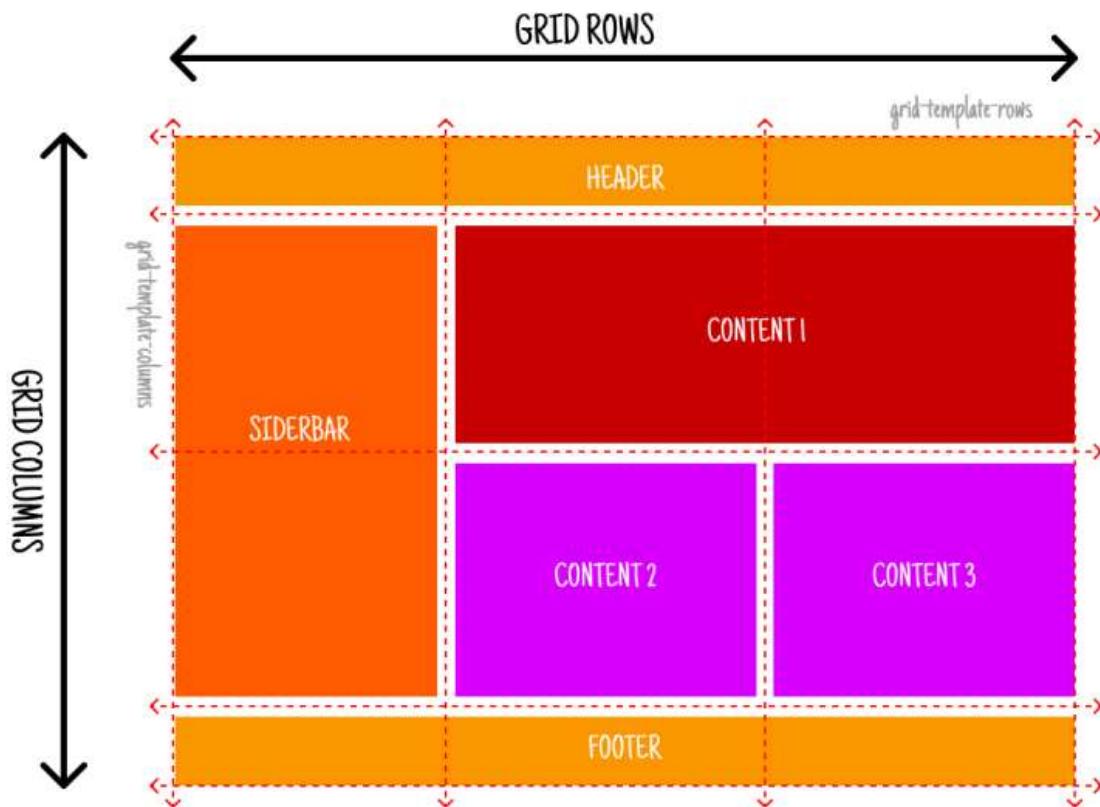
  /*Via area naam*/
  /* grid-area: header; */

  z-index: 1; /*to show on top, if overlapping*/
}

.item.item2{
  grid-row: span 2;
  grid-column: span 2;
}
```

```
.item.item3 {  
    grid-area: 2 / 3 / 4 / 4;  
}
```

- **Extra items toevoegen:** Als je extra items toevoegd terwijl je grid al vol is, dan krijg je een impliciet grid dat automatische een rij bijmaakt bijvoorbeeld. Met `grid-auto-rows: 100px` kan je een grootte van `100px` geven aan de automatisch gegenereerde rijen (idem `grid-auto-columns`). Met de `grid-auto-flow` property kan je de overflow op `columns` toepassen i.p.v. `rows`.
- **Sizing van de templates:** Je kan alle gebruikelijke units gebruiken maar ook een extra speciale unit, specifiek voor grids: `fr`. Deze **fractional** unit gebruikt dan een fractie van de ruimte beschikbaar door de container. Je kan ook de `minmax()` functie gebruiken bv. `minmax(100px, 3fr)`
- **De repeat() function:** met de repeat functie kan je snel meerdere waarden herhalen bv `repeat(3, 100px)`
- **Gridception:** Je hoeft je niet te beperken tot 1 grid om je website op te stellen. Je kan een grid onder een grid, in een grid, naast een grid ... gebruiken. Think, trail, error, repeat is hier de boodschap.
- **justify-items en align-items:** deze properties werken gelijkaardig aan de flexbox, maar dan binnenin 1 vak van je grid. (`stretch` = default, `start`, `end`, `baseline`, `center`)
 - Gebruik de **justify-self** en **align-self** properties in de individuele elementen om ze apart te stylen. **-Plaatsing grid binnenin container:** Gebruik `justify-content` en `align-content` properties van de container. (`start` , `end` , `center` , `baseline` , `space-between` , `space-around`, `space-evenly`)



Grid info [bron](#)

Voorbeeld zie [demo 3](#)

Responsive grid zonder media queries

Example non responsive:

```
.container {
  display: grid;
  grid-template-rows: repeat(4,100px);
  grid-template-columns: repeat(4, minmax(100px,1fr));
}
```

Example responsive:

```
.container {  
    display: grid;  
    grid-template-rows: repeat(4, 100px);  
    /*Wrap item to next row if it doesn't fit*/  
    grid-template-columns: repeat(auto-fit, minmax(100px, 1fr));  
}
```

Oefening

Hieronder vind je een HTML-bestand inclusief CSS en JavaScript code voor de oefeningen. De gewenste positie en methode is steeds weergegeven als de tekst die je moet stylen. We gebruiken klassen om de gewenste styling te krijgen. De klassennamen zijn al ingevuld in de HTML, je moet dus enkel nog de CSS klassen aanvullen:

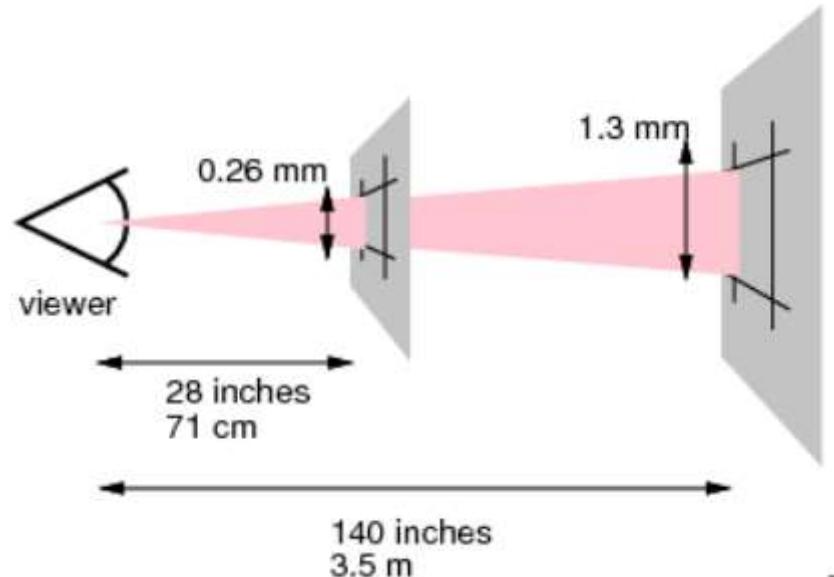
Klik hier om de start code te voor de oefening te zien/verbergen ▾

Solution: Klik hier om de code te zien/verbergen ▾

Let's talk about Units

Zoals al eerder vermeld bestaan er verschillende soorten units in HTML/CSS, maar waarvoor gebruik je welke unit nu het beste? Hieronder vind je een aantal tips.

Maar eerst volgt er nog een kleine uitleg over de abstracte unit **pixel** of **px**. Er bestaat namelijk wel wat verwarring op het internet over wat hoe groot **1px** juist is. Allereerst is het de kleinste unit die je kan gebruiken, maar hoe groot is het juist. We gaan geen exacte meting meegeven maar vertellen je wel waarvoor de pixel dient. De pixelgrootte is verschillend voor verschillende schermgrootten en met goede reden. De grootte van de pixel heeft te maken met de aangeraden kijkafstand ten opzichte van het scherm. De pixel unit zorgt er namelijk voor dat **10px** op een klein scherm er even groot zou moeten uitzien als **10px** op een groot scherm wanneer je rekening houdt met de kijkafstand tot de twee schermen. In werkelijkheid is **10px** op het grote scherm groter, maar als je op de correcte afstand van het scherm zit zouden de groottes overeen moeten komen. Dit maakt de **pixel** de ideale basis unit voor je design.



Schema werking pixel unit [bron](#)

Browser default font-size

De browser default font-size is **16px** voor de meeste moderne browsers. Gebruikers kunnen dit echter aanpassen naar hun accessibility noden. Daarom is het belangrijk specifieke absolute font-sizing te vermijden. Op die manier blijven de accessibility features bruikbaar.

Gebruik daarom steeds de **rem** unit om font-size mee te geven zodat alles mee scaled met de default waarde gebruikt door browser. We gebruiken **rem** omdat die steeds relatief is t.o.v. de root font-size. **em** is scaling t.o.v. de eigen parent maar dit kan snel de leesbaarheid van je code verminderen. De **em** unit is wel handig om bijvoorbeeld breedte, hoogte, padding, margins ... van een element in te stellen omdat je op die manier je grootte baseert op font-size van zijn content.

% vh vw

- **Percentage** units zijn altijd relative t.o.v. hun parent.
- **Viewport** units zijn altijd relative t.o.v. de schermgrootte.

Met **vh** kan je er bijvoorbeeld voor zorgen dat content steeds 80% van de hoogte van je scherm inneemt (`min-height: 80vh;`). **Let op** wanneer je een **vw** van 100

gebruikt, aangezien de viewport geen rekening houdt met scrollbars, zal je bij 100 altijd een zeer kleine “irritante” horizontale scrollbar krijgen.

ch

Een laatste unit die je voor een specifiek geval kan gebruiken is de **ch** (of character width) unit. Een bepaalde design filosofie stelt dat het een good practice is om textcontainers niet breder te maken dan 60 karakters. Door de styling `max-width: 60ch;` toe te passen, wordt automatisch deze good practice toegepast.

99% van de tijd zal je dus enkel met de hierboven vermelde units werken

Voorbeeld zie [demo 4](#)

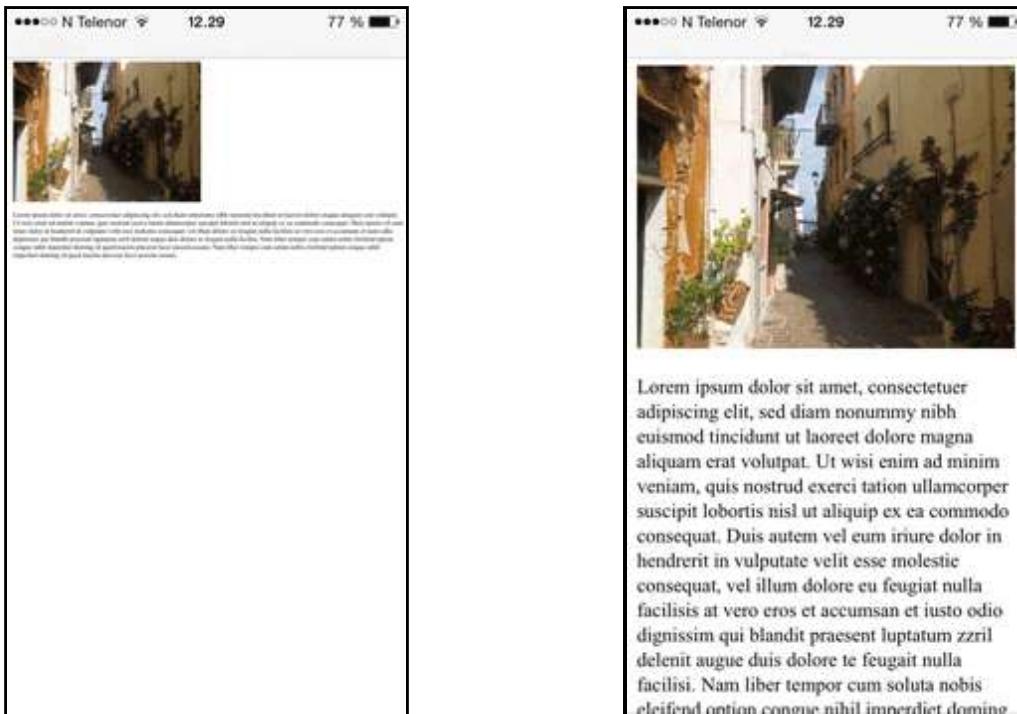
Viewport

To create a responsive website, add the following tag to all your web pages:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

Deze tag zorgt ervoor dat de viewport, het gebied waarin webinhoud wordt weergegeven, zich aanpast aan de breedte van het apparaat waarop de website wordt bekeken. Door `width=device-width` in te stellen, wordt de breedte van de viewport gelijk aan de breedte van het apparaat, wat betekent dat de website correct wordt geschaald op verschillende schermformaten, van smartphones tot desktops. De `initial-scale=1.0` zorgt ervoor dat de inhoud niet wordt ingezoomd of uitgezoomd bij het laden van de pagina.

Hier is een voorbeeld van een webpagina zonder de viewport-metatag en dezelfde webpagina met de viewport-metatag:



Zonder viewport-metatag (links), met viewport-metatag (rechts) [bron](#)

Media Queries

Media queries in CSS zijn een krachtige tool waarmee je de stijl van een website kunt aanpassen aan verschillende schermgroottes en apparaattypen. Ze maken het mogelijk om specifieke **CSS-regels toe te passen afhankelijk van de eigenschappen van het apparaat**, zoals de breedte, hoogte, resolutie en oriëntatie van het scherm. Dit is essentieel voor responsive webdesign, waarbij een website er goed uitziet en functioneert op zowel desktops, tablets als smartphones. Door media queries te gebruiken, kun je bijvoorbeeld een andere lay-out toepassen voor mobiele apparaten dan voor grotere schermen, waardoor de gebruikerservaring op elk apparaat geoptimaliseerd wordt.



Media query syntax bron

- **Media-type:** kies tussen `screen`, `print`, `speech` of `all` (default)
- **Expression to test:** verschil `min-width`, `max-width`. Wanneer je je website voor desktop designed dan gebruik je media queries met `max-width` om ze ook responsive te maken op kleinere viewports. Vice versa als je je website designed voor mobile, gebruik dan `min-width` op responsive te zijn op grotere viewports.
- **Conditional CSS:** style je elementen zoals ze er moeten uitzien op de gequeriede media.

Je kan dus voor veel verschillende screen sizes custom CSS schrijven.

Voorbeeld:

```

@media (min-width: 500px){
  html {
    color: red;
  }
}
  
```

Voorbeeld zie [demo 5](#)

Extra media queries

Je kan ook verschillende CSS files gebruiken om de verschillende layouts te definiëren. Via het `<link>` element kan je dan alle CSS bestanden laten laden bij de verschillende omstandigheden:

```
<!-- Load in `portrait-screen.css` when screen is in portrait mode -->
<link rel="stylesheet" media="screen and (orientation: portrait)"
      href="portrait-screen.css"/>
```

Je kan hetzelfde breiken met imports in je `main.css` file:

```
@import url("portrait-screen.css") screen and (orientation: portrait);
```

Je kan media queries ook **inverteren** met behulp van het keyword `not`:

```
@media not (orientation: portrait) { ... }
```

Je kan media queries ook **combineren** met behulp van het keyword `and`:

```
@media screen and (orientation: portrait) and (min-width: 500px) { ... }
```

Voorbeeld zie [demo 6](#)

Lijst met veelgebruikte eigenschappen waar media queries op kunnen testen:

- `width`: The viewport width.
- `height`: The viewport height.
- `orientation`: This capability checks whether a device is portrait or landscape in orientation.
- `aspect-ratio`: The ratio of width to height based on the viewport width and height. A 16:9 widescreen display can be written as `aspect-ratio: 16/9`.

- **color:** The number of bits per color component. For example, min-color: 16 will check that the device has 16-bit color.
- **color-index:** The number of entries in the color lookup table (the table is how a device changes one set of colors to another) of the device. Values must be numbers and cannot be negative.
- **monochrome:** This capability tests how many bits per pixel are in a monochrome frame buffer. The value would be a number (integer), for example, monochrome: 2, and cannot be negative.
- **resolution:** This capability can be used to test screen or print resolution; for example, min-resolution: 300dpi. It can also accept measurements in dots per centimeter; for example, min-resolution: 118dpcm. 74 Media Queries and Container Queries
- **scan:** This can be either progressive or interlace, features largely particular to TVs. For example, a 720p HD TV (the “p” part of 720p indicates “progressive”) could be targeted with scan: progressive, while a 1080i HD TV (the “i” part of 1080i indicates “interlaced”) could be targeted with scan: interlace.
- **grid:** This capability indicates whether or not the device is grid bitmap-based.
- **prefers-color-scheme:** The theme selected by the user in the browser settings (“light” or dark).

Alle bovenstaande functies, met uitzondering van scan, raster en prefers-color-scheme, kunnen voorafgegaan worden door min- of max- om bereiken te maken.

Workflow

Workflow versie 1: ontwerp je website voor een specifieke viewport, maar laat zo veel mogelijk aan de defaults van de browser over. Test daarna voor andere viewports en pas eventueel je CSS code aan zodat ze automatisch meer responsive is. Schrijf ten slotte specificieke media queries voor die dingen die niet automatisch aangepast kunnen worden.

Workflow versie 2: Bepaal op voorhand een aantal media query breakpoints (XL, L, M, S, XS) en maak elk ervan responsive.

UI design mock ups

Wanneer je start aan het ontwerp van je website is HTML-code schrijven niet de eerste stap. In de eerste plaats ga je nadenken **wat** je wil bereiken met je website, **wie** je wil bereiken met je website. Al je design keuzes moeten een antwoord geven op die vraag. In de eerste plaats ga je nadenken over wat de **content** van je website gaat zijn (wat moet er allemaal op mijn website beschikbaar zijn, wat zijn de functionaliteiten). Ten tweede ga je proberen die content op de beste manier te **presenteren**. Voor deze laatste stap kan je gebruik maken van een UI design tool. Daarmee kan je snel UI mock ups maken die je dan als referentie kan gebruiken bij het implementeren van je website.

Bekende voorbeelden van zo een design tools zijn: [Figma](#) (free), [Sketch](#), [Invision](#), [Adobe XD](#), [Proto](#) ...

Other practical responsive tips

- Gebruik container utility classes om je media queries simpel te structureren. (eventueel met max-width container gelijk aan min-width van query)
- Snapping vs resizing (eigen keuze)
- Maak **img** responsive met `display: block ; max-width: 100%;`
- Gebruik **min-height** i.p.v. **height** (idem **max-width** and **width**)
- Center met `margin-left: auto ; margin-right: auto;`
- Pas geen CSS aan dat niet nodig is. Zo behoud je het default gedrag opgelegd door de browser.
- `Width: auto;` werkt soms beter dan `width: 100%;`
- Zet de **flex-wrap** property op **wrap**, wanneer je een flexbox gebruikt.
- Vermijd media-queries waar mogelijk. (uitzondering bv. echt grote layout wijzigingen voor andere schermtypes)

Easy dark mode example the right way

Een goede manier om een darkmode toe te voegen aan je website is gebruik te maken van CSS-variabelen voor je kleurenpaljet. Dit is echter niet voldoende

aangezien sommige HTML-elementen dan nog niet de correcte styling hebben. Hiervoor moeten we dus ook de color-scheme zelf aanpassen.

```
html {  
  color-scheme: dark light;  
}
```

Voorbeeld zie [demo 7](#)

Simple interactive design examples

- Voorbeeld van een sliding menu met pure CSS zie [demo 8](#)
- Voorbeeld van een hamburger menu met pure CSS zie [demo 9](#)
- Voorbeeld van een hamburger menu met bootstrap zie [demo 10](#)

Opdrachten

Update je portfolio website

1. Maak een keuze of je je website ontwikkeld voor desktop first of mobile first. En voeg de [clean slate code](#) toe bovenaan je CSS. (Pas nu dan eventuele padding/margins toe op de juiste plaatsen)
2. Bepaal 5 sizes waarvoor je media queries gaat schrijven en maak correct gebruik van min- of max-width op basis van je keuze in opdracht 1.
3. Gebruik flexboxes waar nodig.
4. Gebruik grid om de globale layout van je webpagina te bepalen. Tips voor layouts vind je [hier](#)
5. Voorzie een checkbox en gebruik variables, CSS en javascript om een darkmode toe te voegen.
6. Animeer je navigatiebar en voeg en de checkbox toe om te switchen tussen light- en darkmode.

7. Experimenteer met een UI/UX tool. (Maak jouw ideale portfolio website in deze tool. Maak een mock up voor desktop en mobile)

Demos

Demo 1: positioning

Klik hier om de code te zien/verbergen 

Demo 2: flexbox

Klik hier om de code te zien/verbergen 

Demo 3: grid

Klik hier om de code te zien/verbergen 

Demo 4: units

Klik hier om de code te zien/verbergen 

Demo 5: media query

Klik hier om de code te zien/verbergen 

Demo 6: media queries

Klik hier om de code te zien/verbergen 

Demo 7: light dark theme

Klik hier om de code te zien/verbergen 

Demo 8: interactive example

Klik hier om de code te zien/verbergen 

Demo 9: Hamburger menu

Klik hier om de code te zien/verbergen 

Demo 10: Hamburger menu bootstrap

Klik hier om de code te zien/verbergen 