



samen sterk voor werk

# JavaScript



Deze cursus is eigendom van de VDAB©

## Inhoudsopgave

<b>1</b>	<b>INLEIDING.....</b>	<b>4</b>
1.1	Vereiste voorkennis.....	4
1.2	Bestanden.....	4
1.3	Editor .....	4
1.4	Browser .....	4
<b>2</b>	<b>DOCUMENT.GETELEMENTBYID .....</b>	<b>5</b>
2.1	DOM .....	5
2.2	document .....	5
<b>3</b>	<b>PLAATS VAN &lt;SCRIPT&gt; .....</b>	<b>6</b>
<b>4</b>	<b>EVENTS .....</b>	<b>7</b>
4.1	click event.....	7
4.2	Anonieme functie .....	7
4.3	keyup event.....	8
4.4	this.....	8
4.5	blur event .....	9
<b>5</b>	<b>GETELEMENTSBYTAGNAME .....</b>	<b>10</b>
5.1	getElementsByTagName op een element.....	11
<b>6</b>	<b>QUERYSELECTORALL .....</b>	<b>12</b>
6.1	forEach .....	12
<b>7</b>	<b>DATA- ATRIBUTEN .....</b>	<b>13</b>
<b>8</b>	<b>DISABLED.....</b>	<b>14</b>
<b>9</b>	<b>CSS .....</b>	<b>15</b>
9.1	classList.....	15
9.2	styles.....	16

<b>10</b>	<b>PARENT, CHILDREN, SIBLINGS .....</b>	<b>17</b>
10.1	parent .....	17
10.2	child elementen.....	17
10.3	siblings.....	18
10.4	nextElementSibling.....	19
10.5	previousElementSibling.....	19
<b>11</b>	<b>BEPERKING .....</b>	<b>20</b>
11.1	Button.....	20
11.2	Hyperlink .....	20
<b>12</b>	<b>ELEMENT TOEVOEGEN .....</b>	<b>21</b>
12.1	createElement en appendChild.....	21
12.2	createTextNode.....	21
12.3	Groter voorbeeld.....	22
12.4	Rijen toevoegen aan een table.....	22
<b>13</b>	<b>ELEMENT VERWIJDEREN .....</b>	<b>23</b>
<b>14</b>	<b>KEUZES .....</b>	<b>24</b>
<b>15</b>	<b>VALIDEREN .....</b>	<b>27</b>
15.1	Methode 1: de validatie helemaal coderen in JavaScript .....	27
15.2	Methode 2: samenwerken met validatie attributen van HTML elementen. ....	28
<b>16</b>	<b>LOCAL STORAGE, SESSION STORAGE .....</b>	<b>30</b>
16.1	Local storage .....	30
16.2	Session storage.....	32
<b>17</b>	<b>JSON.....</b>	<b>33</b>
<b>18</b>	<b>FETCH .....</b>	<b>35</b>
<b>19</b>	<b>GET, POST, PUT, DELETE.....</b>	<b>38</b>
19.1	HTTP .....	38

19.2	POST .....	38
<b>20</b>	<b>HERHALINGSOEFENINGEN .....</b>	<b>40</b>
<b>21</b>	<b>COLOFON.....</b>	<b>41</b>

# 1 INLEIDING

## 1.1 Vereiste voorkennis

Van Blockly naar JavaScript

## 1.2 Bestanden

Je unzippt de bestanden bij de cursus in een directory naar keuze.

## 1.3 Editor


Je kan JavaScript code tikken met veel soorten editors.

Wij raden de editor Visual Studio Code aan. Je vindt die op <https://code.visualstudio.com>.

Met de Visual Studio Code extensie Live Server wordt je werk nog gemakkelijker.

Als je een pagina of een js bestand wijzigt en opslaat, refresht de browser automatisch de pagina.

Je installeert de extensie met volgende stappen:

1. Je kiest links in Visual Studio Code  (Extensions).
2. Je tikt Live Server.
3. Je kiest in de lijst **Install** bij Live Server.

Je gebruikt de extensie als volgt:

1. Je kiest in het menu File de opdracht Open Folder.
2. Je kiest de directory waarin je de bestanden bij de cursus unzipte.
3. Je ziet links de lijst met de bestanden in de directory.
4. Je kiest een HTML bestand in de lijst.
5. Je ziet de source van dit bestand.
6. Je klikt met de rechtermuisknop in de source en je kiest Open with Live Server.
7. Je ziet de pagina in de browser.
8. Als je de source wijzigt in Visual Studio Code en opslaat (bvb. met de sneltoets Ctrl + S) doet de browser automatisch een "refresh" van de pagina.

## 1.4 Browser

Je kan de programma's in de cursus uitvoeren in elke moderne browser


(Chrome, Firefox, Edge, Safari, ...). Veel programma's werken niet in Internet Explorer.

Dit is een verouderde browser. Hij ondersteunt de moderne mogelijkheden van JavaScript niet.

We gebruiken Chrome.

## 2 DOCUMENT.GETELEMENTBYID

Je opent `copyright.html` in je editor. De pagina verwijst naar `copyright.js`.

Je maakt `copyright.js` in Visual Studio Code met de knop . Je ziet de knop als je links

de muisaanwijzer laat rusten op de directory die je opende:



### 2.1 DOM

DOM is een afkorting: Document Object Model. DOM betekent dat je in je JavaScript code toegang hebt tot alle HTML elementen van de pagina. Elk element wordt je aangeboden als een object. Elk object heeft properties en methods.

### 2.2 document

JavaScript bevat een object `document`. `document` is altijd aanwezig. `document` geeft je toegang tot de DOM. Je kan via `document` HTML elementen lezen, wijzigen en toevoegen. Je leert dit hier.

De eerste stap is één element te zoeken dat in de HTML een `id` attribuut heeft.

`copyright.html` bevat een hyperlink met de `id` `vdab`. Je zoekt de hyperlink met JavaScript code.

Je tikt de code in `copyright.js`:

```
"use strict";
const vdabHyperlink = document.getElementById("vdab");
console.log(vdabHyperlink);
```

❶

- (1) `document` bevat een method `getElementById`. Je geeft als parameter de `id` mee van het element dat je wil zoeken. De method geeft je het gezochte element terug.

Je voert het programma uit. Je ziet dat de variabele `vdabHyperlink` het gevonden element bevat:

```
<a id="vdab" href="https://www.vdab.be/">VDAB</a>
```

Je kan elk attribuut van het element opvragen:

```
console.log(vdabHyperlink.href);
```

❶

- (1) De property `href` van de variabele `vdabHyperlink` bevat het attribuut `href` van het element. Je kan op dezelfde manier andere attributen opvragen.

Je kan dit proberen.

Je kan de tekst tussen de begintag en de eindtag van een element opvragen:

```
console.log(vdabHyperlink.innerText);
```

❶

- (1) De property `innerText` bevat de tekst tussen de begintag en de eindtag.

Je kan dit proberen.

Je kan de attributen ook wijzigen. Je kan de tekst tussen de begintag en de eindtag ook wijzigen.

`copyright.html` bevat een `span` element met de `id` `jaar`.

De tekst tussen de begintag en de eindtag is leeg. Je vervangt de tekst door het huidig jaartal:

```
const vandaag = new Date();
const jaar = vandaag.getFullYear();
const jaarSpan = document.getElementById("jaar");
jaarSpan.innerText = jaar;
```

❶

❷

❸

❹

- (1) JavaScript bevat een class `Date`. `Date` stelt een datum voor. Je maakt een `Date` object.

Je geeft geen parameters mee. Het `Date` object bevat dan de systeemdatum. Opmerking: met `new Date(2019, 12, 31)` maak je een `Date` object met de datum 31/12/2019.

- (2) De method `getFullYear` geeft je het jaartal (bvb. 2019) van de datum.

- (3) Je zoekt het element met de `id` `jaar`. Opgepast, `id`'s zijn hoofdlettergevoelig !

- (4) Je wijzigt de tekst tussen de begintag en de eindtag naar het jaartal van de systeemdatum.

Je kan dit proberen.

Als je wil, kan je de laatste vier opdrachten verkorten tot één opdracht:

```
document.getElementById("jaar").innerText = new Date().getFullYear();
```

### 3 PLAATS VAN <SCRIPT>

In `copyright.html` staat het `<script>` element als laatste binnen de `<body>` tag.

De browser voert het script correct uit. De browser doet hierbij volgende stappen:

1. De browser laadt geleidelijk de HTML elementen van de website tot het `<script>` element.  
De browser onthoudt de elementen in het RAM geheugen.  
Hij toont de teksten van de elementen aan de gebruiker, zodat die snel iets ziet.
2. De browser laadt dan `copyright.js` van de website.  
Dit duurt even bij een trage internetverbinding.
3. De browser voert dan `copyright.js` uit. Je zoekt in de code het element met de id `jaar`.  
De browser geeft je dit element: hij heeft het bij de 1<sup>e</sup> stap in het RAM geheugen geladen.

Je optimaliseert nu stap per stap het probleem van de traagheid bij stap 2.

Je verplaatst het `<script>` element als eerste element binnen de `<head>` tag.

Je krijgt een fout bij de uitvoering. Je ziet de fout in de console:

**Uncaught TypeError: Cannot read property 'href' of null**

Je krijgt de fout omdat de browser deze keer volgende stappen doet:

1. De browser laadt de HTML elementen van de website tot het `<script>` element.  
Hij heeft de elementen met de id's `jaar` en `vdab` nog niet geladen.
2. De browser laadt dan `copyright.js` van de website.  
Dit duurt even bij een trage internetverbinding.
3. De browser voert dan `copyright.js` uit. Je zoekt in de code het element met de id `jaar`.  
De browser vindt dit element niet en je krijgt een fout.
4. De browser laadt de HTML elementen van de website na het `<script>` element.

Het verplaatsen van het `<script>` element heeft enkel extra problemen veroorzaakt.

Je lost dit op. Je voegt het `defer` attribuut toe aan het `<script>` element:

```
<script src="copyright.js" defer></script>
```

De fout bij de uitvoering verdwijnt en je hebt de snelste oplossing:

1. De browser laadt de HTML elementen van de website tot het `<script>` element.
2. De browser laadt `copyright.js` van de website in de achtergrond en laadt *tegelijk* de resterende HTML elementen. Hij doet deze twee handelingen gelijktijdig.
3. Nadat alle elementen geladen zijn, zijn er twee mogelijkheden.
  - a. `copyright.js` is ondertussen volledig geladen.
  - b. `copyright.js` is nog niet volledig geladen.  
De browser wacht dan tot `copyright.js` volledig geladen is.
4. De browser voert dan `copyright.js` uit. Je zoekt in de code het element met de id `jaar`.  
De browser vindt dit element: hij heeft reeds alle elementen geladen.

We plaatsen vanaf nu het `<script>` element (met `defer`) binnen het `<head>` element.



Begroeting: zie takenbundel

## 4 EVENTS

De pagina's die je tot nu maakte hebben geen interactiviteit.

De gebruiker kan enkel *kijken* naar de pagina maar kan geen *handelingen* doen in de pagina.

Je maakt vanaf dit hoofdstuk pagina's waar de gebruiker *wel* handelingen kan doen.

Voorbeelden van handelingen:

- De gebruiker klikt op een knop.
- De gebruiker beweegt de muis over een deel van de pagina.
- De gebruiker wijzigt de inhoud van een tekstvak.

Men noemt deze handelingen ook events.

Men noemt het "doen van de handeling" ook "het optreden van het event".

De events die optreden in de browser zijn voorgedefinieerd. Er bestaat bijvoorbeeld een click event en een dblclick (dubbele klik) event. Je kan niet zelf een trplclick (driedubbele klik) event uitvinden.

Als een event optreedt kan jij JavaScript code uitvoeren.

In het eerste voorbeeld klikt de gebruiker op een knop. Jij toont dan de systeemtijd.

In technische termen: als het click event optreedt in de knop voer je JavaScript code uit.

Dit gebeurt in drie stappen:

1. Je schrijft de code die de systeemtijd toont in een functie.
2. Je koppelt de functie aan het event.
3. Als het event optreedt, voert JavaScript je functie uit.

### 4.1 click event

Je opent `click.html` in je editor. De pagina verwijst naar `click.js`. Je maakt dit bestand.

Als de gebruiker op de knop klikt, toon je in het `span` element de systeemtijd:

```
"use strict"
```

```
function hoeLaatIsHet() {                                ❶
    document.getElementById("nu").innerText = new Date().toLocaleTimeString(); ❷
}
document.getElementById("zegEens").onclick = hoeLaatIsHet; ❸
```

- (1) Je schrijft de functie die JavaScript moet uitvoeren als de gebruiker op de knop klikt. Je kan de naam van de functie vrij kiezen.
- (2) Je maakt een `Date` object dat de systeemtijd voorstelt. Je voert daarop de method `toLocaleTimeString` uit. Die method geeft een string terug met uren, minuten en seconden. Je vult in het element met id `zegEens` de tekst tussen de begin- en de eindtag met deze string.
- (3) Je koppelt de functie `hoeLaatIsHet` aan het click event van de knop met de id `zegEens`. Je verwijst naar het click event met `onclick`. Bemerkt dat je na `hoeLaatIsHet` geen ronde haakjes tikt. Dan zou JavaScript de functie `hoeLaatIsHet nu` uitvoeren. JavaScript moet de functie `hoeLaatIsHet` niet *nu* uitvoeren, maar *in de toekomst*: bij een klik op de knop.

Je kan dit proberen. Telkens je op de knop klikt, voert JavaScript de functie `hoeLaatIsHet` uit.

### 4.2 Anonieme functie

Je kan aan een event ook een anonieme functie koppelen. Dit is een functie die geen naam heeft.

Je wijzigt `click.js`:

```
"use strict"
```

```
document.getElementById("zegEens").onclick = function() { ❶
    document.getElementById("nu").innerText = new Date().toLocaleTimeString();
};
```

- (1) Je koppelt aan het click event een functie die je ter plaatse schrijft. De functie heeft geen naam. Het is daarom een anonieme functie.

Je kan dit proberen.



### 4.3 keyup event

Het keyup event treedt op als de gebruiker een teken (letter, cijfer) tikt.

Je opent `paswoorden.html` in je editor. De pagina verwijst naar `paswoorden.js`.

Je maakt dit bestand.

Telkens de gebruiker een teken tikt in het eerste invoervak controleer je hoe lang het ingetikte paswoord reeds is.

- Als het minstens 6 tekens bevat, toon je de tekst OK.
- Anders toon je de tekst Te kort.

```
"use strict";
document.getElementById("paswoord").onkeyup = function () {           ❶
    const paswoord = document.getElementById("paswoord").value;      ❷
    const paswoordFeedbackSpan = document.getElementById("paswoordFeedback"); ❸
    if (paswoord.length >= 6) {                                       ❹
        paswoordFeedbackSpan.innerText = "OK";                       ❺
    } else {                                                           ❻
        paswoordFeedbackSpan.innerText = "Te kort";
    }
};
```

- (1) Je zoekt het vak met de id `paswoord`. Je koppelt een anonieme functie aan het keyup event.
- (2) Je zoekt het vak met de id `paswoord`.  
`value` bevat de tekst die gebruiker in dit vak tikte.
- (3) Je zoekt de span met de id `paswoordFeedback`.
- (4) Als het ingetikte paswoord minstens 6 tekens bevat
- (5) toon je in de span de tekst OK.
- (6) Anders toon je de tekst Te kort.

Je kan dit proberen.

### 4.4 this

Je kan de code bij ❷ korter schrijven:

```
const paswoord = this.value;                                         ❶
```

- (1) `this` verwijst, in een functie die aan een event gekoppeld is, naar het element waarin het event optrad. De functie is gekoppeld aan vak waarin de gebruiker het paswoord tikt.  
`this` verwijst dus naar dit vak. `this.value` bevat dus het ingetikte paswoord.

Je kan dit proberen.

Je kan de code in de anonieme functie korter schrijven:

```
const paswoord = this.value;
const paswoordFeedbackSpan = document.getElementById("paswoordFeedback");
paswoordFeedbackSpan.innerText =
    paswoord.length >= 6 ?                                           ❶
    "OK" :
    " Te kort";
```

- (1) Je gebruikt voor het eerst de conditionele operator. Dit is een verkorte vorm van `if ... else`.  
De operator bevat drie delen: Het deel voor `?` is een voorwaarde die `true` of `false` oplevert.  
De operator geeft het deel tussen `?` en `:` terug als de voorwaarde `true` is.  
De operator geeft het deel na `:` terug als de voorwaarde `false` is.  
Hier geeft de operator dus "OK" terug als de lengte van het paswoord minstens 6 is.  
De operator geeft "Te kort" terug als de lengte van het paswoord kleiner is dan 6.

Je kan dit proberen.

## 4.5 blur event

Het blur event treedt op als de gebruiker een vak verlaat (door met de muis buiten het vak te klikken of door de TAB toets in te drukken). Dit event wordt regelmatig gebruikt om de inhoud van het vak te valideren (controleren).

Je werkt verder in `paswoorden.js`. Nadat de gebruiker het vak "Herhaal paswoord" verlaat, controleer je of hij hetzelfde paswoord tikte als in het vak "Paswoord".

Je geeft de gebruiker feedback over deze controle. Je voegt daartoe code toe:

```
document.getElementById("herhaal").onblur = function () { ❶  
  const herhaalFeedbackSpan = document.getElementById("herhaalFeedback");  
  herhaalFeedbackSpan.innerHTML =  
    this.value === document.getElementById("paswoord").value ? ❷  
    "OK" :  
    "Verschilt van paswoord";  
};
```

- (1) Je zoekt het vak met de id herhaal. Je koppelt een anonieme functie aan het blur event.
- (2) Je vergelijkt het paswoord ingetikt in het huidige vak (herhaal) met het paswoord ingetikt in het vak met de id paswoord.

Je kan dit proberen.

Je leert verder in de cursus nog andere events kennen.



Spaties: zie takenbundel



Klinkers: zie takenbundel



Palindroom: zie takenbundel

## 5 GETELEMENTSBYTAGNAME

Je leert een tweede manier (naast `document.getElementById`) kennen om elementen te zoeken: `document.getElementsByTagName`.

Je maakt een pagina eerst met `getElementById`. Je leert de nadelen van `getElementById`.

Je lost de nadelen op: je vervangt `getElementById` door `getElementsByTagName`.

Je opent `standbeelden.html` in je editor. De pagina verwijst naar `standbeelden.js`.

Je maakt dit bestand.

`standbeelden.html` bevat 3 hyperlinks. Je zal met JavaScript het volgende doen:

- Als de gebruiker op de 1° hyperlink klikt, toon je in het `img` element `vrijheidsbeeld.jpg`.
- Als de gebruiker op de 2° hyperlink klikt, toon je in het `img` element `moederland.jpg`.
- Als de gebruiker op de 3° hyperlink klikt, toon je in het `img` element `mannekepis.jpg`.

De hyperlinks hebben niet hun standaard gedrag: een andere pagina openen. Ze dienen om JavaScript code uit te voeren als je er op klikt. Je vult dan het `href` attribuut met de waarde `#`.

Je tikt code in `standbeelden.js`:

```
"use strict";
function toonStandbeeld() {
    document.getElementById("afbeelding").src=`${this.id}.jpg`;
}
document.getElementById("vrijheidsbeeld").onclick=toonStandbeeld;
document.getElementById("moederland").onclick=toonStandbeeld;
document.getElementById("mannekepis").onclick=toonStandbeeld;
```

- (1) Je koppelt verder in de code deze functie aan het click event van de hyperlinks.
- (2) Je maakt een string. Die begint met de id van de hyperlink waarin het click event optrad. Daarna bevat de string `.jpg`. Als de gebruiker op de 1° hyperlink klikt bevat deze string `vrijheidsbeeld.jpg`. Je vult met deze string het `src` attribuut van het `img` element. De browser laadt dan de afbeelding `vrijheidsbeeld.jpg` en toont die in het `img` element.
- (3) Je koppelt je functie aan de 1° hyperlink.
- (4) Je koppelt je functie aan de 2° hyperlink.
- (5) Je koppelt je functie aan de 3° hyperlink.

Je kan dit proberen. De pagina werkt, maar de code heeft nadelen:

- Als je nog een hyperlink toevoegt in `standbeelden.html` moet je ook een opdracht zoals ❸ toevoegen aan `standbeelden.js`. Als je dit vergeet werkt de hyperlink niet.
- De code bij ❸, ❹ en ❺ is bijna altijd dezelfde. Meerdere keren dezelfde code schrijven leidt tot een te groot, onoverzichtelijk, moeilijk onderhoudbaar programma.

Je lost dit probleem op met `document.getElementsByTagName`. Je vervangt ❸, ❹ en ❺:

```
const hyperlinks = document.getElementsByTagName("a");
for (const hyperlink of hyperlinks) {
    hyperlink.onclick = toonStandbeeld;
}
```

- (1) Je zoekt met `document.getElementsByTagName` alle elementen met een bepaalde tag name. Je zoekt hier alle hyperlinks: de elementen met de tag name `a`: `<a ...>`. De method geeft je een verzameling terug met de 3 hyperlinks in het document.
- (2) Je kan met `for ... of` itereren over de hyperlinks in de verzameling, zoals je met `for ... of` kan itereren over de elementen van een array.
- (3) Je koppelt je functie aan het click event van de hyperlink van de huidige iteratie.

Je kan dit proberen.

Als je een hyperlink toevoegt aan `standbeelden.html` moet je niets wijzigen in `standbeelden.js`.

```
<li><a href="#" id="verlosser">Christus de verlosser</a></li>
```

Je kan dit proberen.

## 5.1 getElementsByTagName op een element

`document.getElementsByTagName("a")` zoekt *alle* hyperlinks in het document. Dit benadeelt de uitbreidbaarheid van het programma. Je leert dit met een voorbeeld.

Je voegt in `standbeelden.html` regels toe na het `img` element:

```
<div>
  <a href="#" id="isHetWeekend">Is het weekend</a>
  <span id="weekend" class="feedback"></span>
</div>
```

Als de gebruiker op de hyperlink klikt, toon je de tekst `Het is weekend` of de tekst `Het is geen weekend`.

Je voegt in `standbeelden.js` regels toe na `"use strict";`:

```
document.getElementById("isHetWeekend").onclick = function () {
  const dagInWeek = new Date().getDay();
  document.getElementById("weekend").innerText =
    dagInWeek === 6 || dagInWeek === 0 ? "Ja" : "Nee";
};
```

- (1) De method `getDay` geeft je de "dag in de week" van de datum als een getal. Zondag is 0, maandag is 1, ... zaterdag is 6.

Je kan dit proberen, maar een klik op Is het weekend geeft niet het juiste resultaat. De reden: Je koppelt in de code bij ❶ aan het click event van de hyperlink de functie die de tekst toont. Je zoekt daarna met `document.getElementsByTagName("a")` *alle* hyperlinks. De zoek operatie vindt ook de hyperlink Is het weekend. Je koppelt aan het click event van alle gevonden hyperlinks een functie die een foto toont. Als de gebruiker klikt op Is het weekend probeer je ook een foto te tonen. Dit was niet de bedoeling.

Je lost dit op. Je vervangt de regels

```
const hyperlinks = document.getElementsByTagName("a");
for (const hyperlink of hyperlinks) {
  door
  const standbeeldenUl = document.getElementById("standbeelden");
  for (const hyperlink of standbeeldenUl.getElementsByTagName("a")) {
```

- (1) Je zoekt het element met de id `standbeelden`. Dit is het `ul` element. De hyperlinks die bij een klik een foto tonen zijn child elementen van dit `ul` element.  
 (2) Je roept `getElementsByTagName` op op dit `ul` element. JavaScript zoekt enkel hyperlinks in de child elementen van dit `ul` element en verbindt enkel aan het click event van die hyperlinks je code die een foto toont. JavaScript verbindt die code dus niet aan de hyperlink Is het weekend.

Je kan dit proberen.

Opmerking:

je kan over de elementen die `getElementsByTagName` teruggeeft *niet* itereren met `forEach`.

Conclusie:

- Als je code hebt die je aan een event van *slechts één* element koppelt, zoek je dit element met `getElementById`.
- Als je code hebt die je aan een event van *meerdere* elementen koppelt, zoek je die elementen met `getElementsByTagName`.



Dubbelspel: zie takenbundel

## 6 QUERYSELECTORALL

Je leert een derde manier kennen om elementen te zoeken: `querySelectorAll`.

Je kan `querySelectorAll` (zoals `getElementsByTagName`) uitvoeren

- op document  
JavaScript doorzoekt dan *alle* elementen.
- op een element  
JavaScript doorzoekt dan enkel de *child elementen* van dit element.

Je geeft aan `querySelectorAll` een CSS selector mee. De CSS selector geeft de elementen aan die je wil zoeken. Een CSS selector is het onderdeel in CSS waarmee je aangeeft op welke elementen je CSS wil toepassen. In `h1 {color:red;}` is `h1` een CSS selector.

Je vindt een overzicht van CSS selectors op [https://www.w3schools.com/cssref/css\\_selectors.asp](https://www.w3schools.com/cssref/css_selectors.asp).

Enkele voorbeelden van `querySelectorAll`:

- `document.querySelectorAll("div, p")`  
Geeft een verzameling met de `<div>` elementen en de `<p>` elementen.
- `document.querySelectorAll("#voornaam, #familienaam")`  
Geeft een verzameling met de elementen met de id's `voornaam` en `familienaam`.
- `document.querySelectorAll(".feedback, .fout")`  
Geeft een verzameling met de elementen die de class `feedback` of de class `fout` hebben.
- `document.querySelectorAll("[autofocus]")`  
Geeft een verzameling met de elementen die een attribuut `autofocus` hebben.

`querySelectorAll` is de krachtigste manier om elementen te zoeken.

Je verkort met `querySelectorAll` de code in `standbeelden.js`.

Je vervangt de opdrachten

```
const standbeeldenUl = document.getElementById("standbeelden");
for (const hyperlink of standbeeldenUl.getElementsByTagName("a")) {
  door
  for (const hyperlink of document.querySelectorAll("#standbeelden a")) { ❶
```

- (1) Je zoekt de hyperlinks (a) die child elementen zijn van het element met de id `standbeelden` (`#standbeelden`).

Je kan dit proberen.

Er bestaat ook een method `querySelector`. `querySelector` werkt zoals `querySelectorAll` maar geeft enkel het *eerst gevonden* element terug, terwijl er mogelijk meerdere gevonden zijn. `document.querySelector("#standbeelden a")` geeft enkel de hyperlink Vrijheidsbeeld terug.

### 6.1 forEach

`querySelectorAll` geeft je een verzameling elementen terug.

Je itereert tot nu over de verzameling met `for ..of`.

Je kan de iteratie ook doen met `forEach`:

```
document.querySelectorAll("#standbeelden a")
  .forEach(hyperlink => hyperlink.onclick = function () {
    document.getElementById("afbeelding").src = `${this.id}.jpg`;
  }); ❶
```

- (1) JavaScript roept per hyperlink de lambda op die je meegeeft aan `forEach`.  
De parameter `hyperlink` van de lambda bevat de hyperlink.

Je kan dit proberen.



Plaats: zie takenbundel

## 7 DATA- ATRIBUTEN

Je werkt verder in `standbeelden.html` en `standbeelden.js`.

Het `id` attribuut van elke standbeeld hyperlink bevat de naam van het afbeeldingsbestand van het standbeeld. Dit heeft beperkingen:

- Een `id` mag niet beginnen met een cijfer en mag geen spatie bevatten.  
De `id` kan dus niet verwijzen naar een bestandsnaam met een spatie.
- Misschien wil je over het standbeeld nog andere data bijhouden, zoals de plaats van het standbeeld. Je kan deze extra data moeilijk ook in het `id` attribuut proppen.

HTML heeft een oplossing. Je kan in een element extra data bijhouden. Je plaats die data in attributen waarvan je de naam zelf verzint, op voorwaarde dat die naam begint met `data-`.

Je wijzigt de regels met de standbeeld hyperlinks:

```
<li><a href="#" data-foto="vrijheidsbeeld"
  data-plaats="New York">Vrijheidsbeeld</a></li>
<li><a href="#" data-foto="moederland"
  data-plaats="Volgograd">Het moederland roept</a></li>
<li><a href="#" data-foto="mannekepis"
  data-plaats="Brussel">Manneke pis</a></li>
<li><a href="#" data-foto="verlosser"
  data-plaats="Rio de Janeiro">Christus de verlosser</a></li>
```

①  
②

(1) Je onthoudt in het `data-foto` attribuut de naam van het fotobestand van het standbeeld.

(2) Je onthoudt in het `data-plaats` attribuut de plaats van het standbeeld.

Je leest zo'n attribuut in JavaScript code op een speciale manier.

- Je leest het `data-foto` attribuut als de `property dataset.foto`.
- Je leest het `data-plaats` attribuut als de `property dataset.plaats`.

Je past dit toe in `standbeelden.js`.

Je vervangt de opdracht

```
document.getElementById("afbeelding").src = `${this.id}.jpg`;
```

door

```
const img = document.getElementById("afbeelding");
img.title = this.dataset.plaats;
img.src = `${this.dataset.foto}.jpg`;
```

①

(1) Je leest de inhoud van het attribuut `data-plaats` van de aangeklikte hyperlink.

Je vult daarmee het `title` attribuut van het `img` element.

De gebruiker ziet de tekst `title` als hij de muisaanwijzer laat rusten op de afbeelding.

Je kan dit proberen.

Opmerking: Je kan in je JavaScript code de `data-` attributen niet enkel lezen, maar ook wijzigen:


```
this.data.plaats = "...";
```



Landen: zie takenbundel

## 8 DISABLED

Een disabled element is een element dat de gebruiker op dat moment niet kan gebruiken.

Dit is een gewone (enabled) knop: 

Dit is dezelfde knop, maar disabled: 

Je maakt een element disabled in HTML met het disabled attribuut:

```
<button disabled>Inschrijven</button>
```

Je leest in JavaScript dit attribuut als de disabled property van een element.

- Als de property true bevat, is het element disabled.  
Je kan de property zelf op true plaatsen om het element te disable.
- Als de property false bevat, is het element enabled.  
Je kan de property zelf op false plaatsen om het element te enable.

Je opent partner.html in je editor. De pagina verwijst naar partner.js. Je maakt dit bestand.

Als de gebruiker Alleenstaande aanvinkt, disable je het vak voor de naam van de partner.

Als de gebruiker Alleenstaande uitvinkt, enable je het vak voor de naam van de partner:

```
"use strict";  
document.getElementById("alleenstaande").onchange = function () {  
    document.getElementById("naamPartner").disabled = this.checked;  
};
```

❶  
❷

- (1) Je koppelt een anonieme functie aan het change event van het vinkje.  
Dit event treedt op als de gebruiker dit vinkje wijzigt (af → aan of aan → af).
- (2) this.checked bevat true als het vinkje aan staat. Je disable dan het vak met de partner.  
this.checked bevat false als het vinkje uit staat. Je enable dan het vak met de partner.

Je kan dit proberen.



Disabled: zie takenbundel

## 9 CSS

Je leert hoe je met JavaScript code leest welke CSS op een element is toegepast.

Je leert de CSS ook wijzigen met JavaScript code.

Je opent pannenkoek.html in je editor. De pagina verwijst naar pannenkoek.js.

Je maakt dit bestand.

### 9.1 classList

De classList property geeft je de verzameling CSS classes die toegepast zijn op een element:

```
"use strict";
for (const eenClass of document.getElementById("beschrijving").classList) { ❶
  console.log(eenClass); ❷
}
```

(1) Je zoekt het element met de id beschrijving.

Je itereert over de verzameling CSS classes (classList) van dit element.

(2) Je toont elke CSS class in de console.

Je kan dit proberen. Je ziet in de console zachteAchtergrond en dunneKader:

de CSS classes die in pannenkoek.html zijn toegepast op het element.

Je kan in je code een CSS class toevoegen aan de classList. De browser past die CSS class onmiddellijk toe op het element. Je past dit toe. Als de gebruiker op een ingrediënt klikt, toon je alle ingrediënten met een opvallende achtergrond. Als de gebruiker op een werkwijze klikt, toon je alle werkwijzen met een opvallende achtergrond. Je vervangt de bestaande code:

```
"use strict";
for (const element of document.querySelectorAll("#ingrediënten, #werkwijze")) { ❶
  element.onclick = function () { ❷
    this.classList.add("opvallend"); ❸
  };
}
```

(1) Je itereert over de verzameling elementen met de ids ingrediënten en werkwijze.

(2) Je koppelt aan het click event van zo'n element een anonieme functie.

(3) Je voegt de CSS class opvallend toe aan het element.

Je kan dit proberen. Als je *meerdere keren* in een ingrediënt of werkwijze klikt, probeer je *meerdere keren* de CSS class opvallend toe te voegen. Dat is niet erg: als je aan classList een CSS class toevoegt die reeds voorkomt in classList voegt classList de CSS class niet nog eens toe.

Je kan een CSS class verwijderen uit een element. De browser past die CSS class dan niet meer toe op dat element. Bij een 1° klik in een ingrediënt of werkwijze voeg je de CSS class opvallend toe.

Bij een 2° klik verwijder je die CSS class. Bij een volgende klik voeg je ze weer toe ...

Je wijzigt de code in de anonieme functie:

```
if (this.classList.contains("opvallend")) { ❶
  this.classList.remove("opvallend"); ❷
} else {
  this.classList.add("opvallend"); ❸
}
```

(1) Je geeft aan de method contains een CSS class mee.

contains geeft true terug als de classList deze CSS class bevat.

Dit is bijvoorbeeld het geval bij de 2° klik, omdat je de CSS class bij de 1° klik toevoegde.

(2) Als de classList de CSS class opvallend bevat, verwijder je deze CSS class.

(3) Als de classList de CSS class opvallend niet bevat, voeg je deze CSS class terug toe.

Je kan dit proberen.



## 9.2 styles

Je opent `viervierdegebak.html` in je editor. De pagina verwijst naar `viervierdegebak.js`.  
Je maakt dit bestand.

Een element heeft een `styles` property. Je kan met `styles` elke CSS eigenschap van dat element wijzigen. Je past dit toe. Je wijzigt de achtergrond van het element met de id beschrijving naar roze:  
`"use strict";`

```
document.getElementById("beschrijving").style.backgroundColor="pink";
```

We raden dit af. Het is beter dat je CSS in CSS bestanden schrijft, niet in js bestanden.

Deze techniek is wel interessant om pagina onderdelen tijdelijk te verbergen of opnieuw te tonen.

Je voegt code toe onder `<h2>Ingrediënten</h2>`:

```
<a href="#" data-teverbergenid="ingrediënten">Verbergen</a>
```

Als de gebruiker op de hyperlink klikt, verberg je de ingrediënten.

Het `data-teVerbergenId` attribuut bevat de id van het te verbergen element.

Je voegt een hyperlink toe onder `<h2>Werkwijze</h2>`:

```
<a href="#" data-teverbergenid="werkwijze">Verbergen</a>
```

Als de gebruiker op de hyperlink klikt, verberg je de werkwijze. Je vervangt de vorige code:

```
"use strict";
for (const hyperlink of document.querySelectorAll("a[data-teVerbergenId]")) { ❶
  hyperlink.onclick = function () { ❷
    document.getElementById(this.dataset.teverbergenid).style.display = "none"; ❸
  }
}
```

- (1) De CSS selector `a[data-teVerbergenId]` heeft je de hyperlinks (a elementen) die een attribuut `data-teVerbergenId` hebben. Je itereert over die hyperlinks.
- (2) Je koppelt een anonieme functie aan die hyperlinks.
- (3) Je zoekt het element met de id in het `data-teVerbergenId` attribuut van de hyperlink. Je verbergt dit element.

Je kan dit proberen.

Je breidt de code uit.

Als de gebruiker klikt op Verbergen wijzig je de tekst van de hyperlink naar Tonen.

Als de gebruiker daarna op de hyperlink klikt, toon je het element dat je verborgen had en wijzig je de tekst van de hyperlink terug naar Verbergen. Je wijzigt de code in de anonieme functie:

```
const bijbehorendElement= document.getElementById(this.dataset.teverbergenid);
if (this.innerText === "Verbergen") { ❶
  bijbehorendElement.style.display = "none"; ❷
  this.innerText = "Tonen"; ❸
} else { ❹
  bijbehorendElement.style.display = ""; ❺
  this.innerText = "Verbergen"; ❻
}
```

- (1) Als de tekst van de hyperlink Verbergen is
- (2) verberg je het bijbehorend element
- (3) en wijzig je de tekst van de hyperlink naar Tonen.
- (4) Ander is de tekst van de hyperlink Tonen.
- (5) Je toont dan het bijbehorende element
- (6) en je wijzigt de tekst van de hyperlink naar Verbergen.

Je kan dit proberen.



Standbeeldlijst: zie takenbundel

## 10 PARENT, CHILDREN, SIBLINGS

Je werkt verder in `standbeelden.html` en `standbeelden.js`.

Je gebruikte tot nu methods van `document` om elementen te zoeken:

`getElementById`, `getElementsByTagName`, `querySelectorAll`, `querySelector`.

Je leert hier een andere manier kennen om elementen aan te spreken.

Het gaat over elementen die “*in de buurt zijn*” van een element dat je reeds opzocht met `getElementById`, `getElementsByTagName`, `querySelectorAll` of `querySelector`.

### 10.1 parent

Elk element heeft een parent element.

Het parent element is het element waarvan een element een *rechtstreeks* onderdeel is.

We leggen dit uit aan de hand van volgende regels:

```
<ul id="standbeelden">
  <li>
    <a href="#" data-foto="vrijheidsbeeld" data-plaats="New York">Vrijheidsbeeld</a>
  </li>
```

- (1) Het parent element van dit `li` element is het `ul` element:  
het `li` element is een rechtstreeks onderdeel van het `ul` element.
- (2) Het parent element van dit `a` element is het `li` element:  
het `a` element is een rechtstreeks onderdeel van het `li` element.  
Het parent element van dit `a` element is *niet* het `ul` element:  
het `a` element is geen *rechtstreeks* onderdeel van het `ul` element,  
het `li` element zit tussen het `a` element en het `ul` element.

In je JavaScript code heeft elk element een property `parent`.

`parent` geeft je het parent element van dat element.

Je probeert dit in `standbeelden.js`:

```
"use strict";
const eersteLi = document.querySelector("li");
console.log(eersteLi.parentElement.id);
```

- (1) Je zoekt het eerste `li` element. Dit is het element dat een hyperlink Vrijheidsbeeld bevat.
- (2) Je toont in de console de id van het parent element van dit id element.

Je kan dit proberen. Je krijgt `standbeelden`.

### 10.2 child elementen

Child elementen zijn elementen die een rechtstreeks onderdeel zijn van een element.

We leggen dit uit aan de hand van volgende regels:

```
<ul id="standbeelden">
  <li>
    <a href="#" data-foto="vrijheidsbeeld" data-plaats="New York">Vrijheidsbeeld</a>
  </li>
  <li>
    <a href="#" data-foto="moederLand" data-plaats="Volgograd">Het moederland roept</a>
```

- (1) Dit `li` element is een child element van het `ul` element.
- (2) Dit `a` element is een child element van het `li` element bij ❶.
- (3) Dit `li` element is ook een child element van het `ul` element.  
Het `ul` element heeft dus *meerdere* child elementen.

In je JavaScript code heeft elk element een property `children`.  
`children` geeft je een verzameling met de child elementen van het element.  
 Je kan over die verzameling itereren met `for ... of`:  
 Je probeert dit in `standbeelden.js`:

```
"use strict";
const standbeeldenLijst = document.getElementById("standbeelden");
for (const childElement of standbeeldenLijst.children) {
  console.log(childElement);
}
```

❶  
❷  
❸

- (1) Je zoekt het element met de id `standbeelden`. Dit is het `ul` element uit de bovenstaande uitleg.
- (2) Je itereert over de child elementen van dit element.
- (3) Je toont elk child element in de console.

Je kan dit proberen.

Ook als een element maar één child element bevat, geeft de `children` property je dit als een *verzameling* die één element bevat. Voorbeeld:

```
"use strict";
const eersteLi = document.querySelector("li");
for (const childElement of eersteLi.children) {
  console.log(childElement);
}
```

❶  
❷

- (1) Je zoekt het eerste `li` element. Dit is het element dat een hyperlink Vrijheidsbeeld bevat.
- (2) `children` geeft je de verzameling child elementen van dit element.  
 Die verzameling bevat één element: de hyperlink binnen dat `li` element.

Je kan dit proberen.

### 10.3 siblings

Siblings zijn elementen die hetzelfde parent element hebben als het huidige element.

```
<ul id="standbeelden">
  <li>
    <a href="#" data-foto="vrijheidsbeeld" data-plaats="New York">Vrijheidsbeeld</a>
  </li>
  <li>
    <a href="#" data-foto="moederLand" data-plaats="Volgograd">Het moederland roept</a>
  </li>
  <li>
    <a href="#" data-foto="mannekepis" data-plaats="Brussel">Manneke pis</a>
  </li>
</ul>
```

❶  
❷  
❸  
❹

- (1) De elementen bij ❸ en ❹ zijn siblings van het huidige `li` element.
- (2) Dit element heeft geen siblings.

In JavaScript bevat geen method om *alle* siblings van dat element te zoeken.  
 Met wat extra eigen code is dit niet zo moeilijk:

```
"use strict";
const eersteLi = document.querySelector("li");
for (const sibling of eersteLi.parentElement.children) {
  if (sibling !== eersteLi) {
    console.log(sibling);
  }
}
```

❶  
❷  
❸  
❹

- (1) Je zoekt het eerste `li` element. Dit is het element dat een hyperlink Vrijheidsbeeld bevat.
- (2) Je itereert over alle child elementen van het parent element van dit `li` element.  
 De iteratie overloopt alle siblings van dit `li` element en ook het `li` element zelf.
- (3) Je slaat het `li` element zelf over.
- (4) Je toont de andere elementen in de console. Dit zijn de siblings van het `li` element.

Je kan dit proberen.

## 10.4 nextElementSibling

Een element heeft een property `nextElementSibling`.

Die heeft je de sibling die onmiddellijk *na* het element komt (de “volgende” sibling).

Als na het element geen sibling meer komt, heeft de method `null` (“niets”) terug.

Je probeert dit uit: Je toont de siblings die na het tweede `li` element komen.

```
"use strict";  
const tweedeLi = document.querySelector("li:nth-child(2)");  
let volgendeSibling = tweedeLi.nextElementSibling;  
while (volgendeSibling !== null) {  
    console.log(volgendeSibling);  
    volgendeSibling = volgendeSibling.nextElementSibling;  
}
```

**❶**  
**❷**  
**❸**  
**❹**  
**❺**

(1) Je zoekt het tweede `li` element. Dit is het element dat een hyperlink Het moederland ... bevat.

(2) Je vraagt de volgende sibling van dit `li` element.

(3) Je controleert of er een volgende sibling was.

(4) Je toont de sibling.

(5) Je vraagt de volgende sibling van die sibling.

Je kan dit proberen.

## 10.5 previousElementSibling

Een element heeft een property `previousElementSibling`.

Die heeft je de eerste sibling die onmiddellijk *voor* het element komt (de “vorige sibling”).

Als voor het element geen sibling meer komt, heeft de method `null` (“niets”) terug.

Je probeert dit uit: Je toont de siblings die voor het voorlaatste `li` element komen.

```
"use strict";  
const voorlaatsteLi = document.querySelector("li:nth-last-child(2)");  
let vorigeSibling = voorlaatsteLi.previousElementSibling;  
while (vorigeSibling !== null) {  
    console.log(vorigeSibling);  
    vorigeSibling = vorigeSibling.previousElementSibling;  
}
```

**❶**  
**❷**  
**❸**  
**❹**  
**❺**

(1) Je zoekt het voorlaatste `li` element. Dit is het element dat een hyperlink Manneke pis bevat.

(2) Je vraagt de vorige sibling van dit `li` element.

(3) Je controleert of er een vorige sibling was.

(4) Je toont de sibling.

(5) Je vraagt de vorige sibling van die sibling.

Je kan dit proberen.



Marsepein: zie takenbundel

## 11 BEPERKING

Je opent beperking.html in je editor. De pagina verwijst naar beperking.js.  
Je maakt dit bestand met daarin volgende code:

```
"use strict"
for (const element of document.getElementsByTagName("button")) {
  element.onclick = function () {
    console.log(this.innerText);
  };
}
```

❶  
❷  
❸

- (1) Je itereert over de button elementen.
- (2) Je koppelt een anonieme functie aan het click event van zo'n button.
- (3) Je toont de tekst tussen de begintag en de eindtag van de button.

Sommige mensen met een beperking kunnen de muis niet gebruiken.

Ze gebruiken dan het toetsenbord om de computer te bedienen.

Als je wil dat ze ook je website kunnen gebruiken, koppel je JavaScript code aan het click event van buttons en hyperlinks, *niet* aan het click event van andere elementen (bvb. <div> elementen).

### 11.1 Button

De gebruikers gebruiken de TAB toets om te navigeren van button naar button in de pagina.

Als ze op de button komen die ze willen bedienen, drukken ze op de ENTER toets.

JavaScript voert dan het click event van die button uit.

Je kan dit proberen.

### 11.2 Hyperlink

De gebruikers gebruiken de TAB toets ook om te navigeren van hyperlink naar hyperlink.

Als ze op de hyperlink komen die ze willen bedienen, drukken ze op de ENTER toets.

JavaScript voert dan het click event van die hyperlink uit.

Je vervangt in beperking.html de buttons door hyperlinks:

```
<a href="#">Eerste</a>
<a href="#">Tweede</a>
<a href="#">Derde</a>
```

Je vervangt in beperking.js "button" door "a"

Je kan dit proberen.

Als je JavaScript code koppelt aan het click event van een ander element (bvb. een <div> element), kan de de gebruiker je code niet activeren met het toetsenbord.

Hij kan met de TAB toets niet navigeren naar het element.

Je vervangt in beperking.html de hyperlinks door div's:

```
<div>Eerste</div>
<div>Tweede</div>
<div>Derde</div>
```

Je vervangt in beperking.js "a" door "div"

Je kan dit proberen. Je kan niet naar de <div> elementen navigeren met je toetsenbord.

## 12 ELEMENT TOEVOEGEN

### 12.1 createElement en appendChild

Je opent voornamen.html in je editor. De pagina verwijst naar voornamen.js. Je maakt dit bestand.

Je voegt een element toe in 2 stappen:

1. Je maakt het element in het RAM geheugen met de createElement method van document. Het element behoort daarna nog niet tot de DOM. De gebruiker ziet het element dus niet.
2. Je voegt het nieuwe element toe als laatste child van een bestaand element. Je doet dit met de appendChild method van dat bestaand element. Je geeft het nieuwe element mee als parameter van de method appendChild. Het nieuwe element wordt het laatste child element van het bestaand element waarop je de method appendChild uitvoerde. De gebruiker ziet het nieuwe element.

Je begint met een eenvoudig voorbeeld: Je voegt aan het body element een child element toe.

Dit is een span element met de tekst Copyright VDAB.

```
"use strict";
const spanCopyright = document.createElement("span");           ❶
spanCopyright.innerText = "Copyright © VDAB";                   ❷
const body = document.querySelector("body");                    ❸
body.appendChild(spanCopyright);                                 ❹
```

- (1) Je maakt in het RAM geheugen een span element.
- (2) Je maakt tekst tussen de <span> en </span> tags van dit element.
- (3) Je zoekt het body element.
- (4) Je voegt het nieuwe span element toe als laatste child element van het body element.

Je kan dit proberen.

### 12.2 createTextNode

Je wil soms geen element, maar tekst toevoegen aan een element.

De tekst kan letters bevatten, maar kan ook een getal of een datum.

Je doet dit in 2 stappen:

1. Je maakt een text node met de createTextNode method van document. Een text node stelt een stuk tekst voor. Je geeft de tekst die je wil opnemen in die text node mee als parameter: document.createTextNode("VDAB"); De text node behoort daarna nog niet tot de DOM. De gebruiker ziet de tekst dus niet.
2. Je voegt de text node toe aan een bestaand element. Je doet dit met de appendChild method van een bestaand element.

Je voegt extra code toe: Je voegt na Copyright © VDAB het jaar van de systeemdatum toe:

```
const jaar = new Date().getFullYear();           ❶
const textNode = document.createTextNode(jaar);  ❷
spanCopyright.appendChild(textNode);             ❸
```

- (1) Je maakt een variabele met het jaar van de systeemdatum.
- (2) Je maakt in het RAM geheugen een text node met als tekst het jaar.
- (3) Je voegt de text node toe als laatste onderdeel van het span element.

Je kan dit proberen. Je ziet dat VDAB en het jaar aan mekaar kleven.

Je kan dit oplossen door de code bij ❷ te wijzigen:

```
const textNode = document.createTextNode(`${jaar}`);  ❶
```

- (1) Je maakt een string met een spatie en de inhoud van de variabele jaar.

Je kan dit proberen.

### 12.3 Groter voorbeeld

Je maakt de pagina praktisch. Nadat de gebruiker een voornaam tikt en op de knop klikt voeg je die voornaam toe aan de lijst (het ul element):

```
"use strict";
document.getElementById("toevoegen").onclick = function () {
    const voornaamInput = document.getElementById("voornaam");
    const voornaam = voornaamInput.value;
    const li = document.createElement("li");
    li.innerText = voornaam;
    document.getElementById("voornamen").appendChild(li);
    voornaamInput.value = "";
    voornaamInput.focus();
};
```

❶  
❷  
❸  
❹  
❺  
❻  
❼  
❽

- (1) Je koppelt een anonieme functie aan het click event van de knop.
- (2) Je zoekt het invoervak waarin de gebruiker een voornaam tikte.
- (3) Je onthoudt de voornaam die de gebruiker in dit invoervak tikte.
- (4) Je maak in het RAM geheugen een li element.
- (5) De tekst tussen de <li> en </li> tags van dit element is de ingetikte voornaam.
- (6) Je voegt dit element toe als laatste child element van het ul element.
- (7) Je maakt de inhoud van het invoervak terug leeg.
- (8) en je plaatst de cursor in het invoervak zodat de gebruiker een volgende voornaam kan tikken.

Je kan dit proberen.

### 12.4 Rijen toevoegen aan een table

Je opent personen.html in je editor. De pagina verwijst naar personen.js. Je maakt dit bestand.

De gebruiker tikt een voornaam, een familienaam en drukt op de knop toevoegen.

Jij voegt dan een rij toe aan de tbody van de tabel in het HTML document.

Je zou dit kunnen doen met veel createElement en appendChild opdrachten.

Je kan dit met enkele andere methods wat korter:

```
"use strict";
document.getElementById("toevoegen").onclick = function () {
    const tbody = document.querySelector("tbody");
    const tr = tbody.insertRow();
    const voornaamTd = tr.insertCell();
    const voornaamInput = document.getElementById("voornaam");
    voornaamTd.innerText = voornaamInput.value;
    const familienaamTd = tr.insertCell();
    const familienaamInput = document.getElementById("familienaam");
    familienaamTd.innerText = familienaamInput.value;
    voornaamInput.value = "";
    familienaamInput.value = "";
    voornaamInput.focus();
};
```

❶  
❷  
❸

- (1) De method insertRow voegt een rij (tr element) toe. De rij behoort direct tot de DOM.
- (2) De method insertCell voeg een kolom (td element) toe. De kolom behoort direct tot de DOM.
- (3) De tekst in de kolom is de ingetikte voornaam.

Je kan dit proberen.



Planning: zie takenbundel

## 13 ELEMENT VERWIJDEREN

Je werkt verder in `personen.html` en `personen.js`.

Je verwijdert een element door de `remove` method uit te voeren op dat element.

Je voegt aan de tabel een extra kolom met een hyperlink X toe.

Als de gebruiker hierop klikt, verwijder je de bijbehorende rij uit de tabel.

Je voegt in `personen.html` een `th` element toe:

```
<th>Verwijderen</th>
```

Je voegt in `personen.js` opdrachten toe, voor `voornaamInput.value = ""`;

```
const verwijderTd = tr.insertCell();  
const verwijderHyperlink = document.createElement("a");  
verwijderHyperlink.innerText = "X";  
verwijderHyperlink.href="#";  
verwijderTd.appendChild(verwijderHyperlink);  
verwijderHyperlink.onclick = function () {  
    const tr = this.parentElement.parentElement;  
    tr.remove();  
};
```

❶  
❷  
❸  
❹  
❺  
❻  
❼  
❽

(1) Je voegt een kolom (`td` element) toe aan de rij.

(2) Je maakt een hyperlink (`a` element) in het RAM geheugen.

(3) Je plaatst de tekst van de hyperlink op `X`.

(4) Je vult het `href` attribuut met `#`. De hyperlink dient niet om naar een andere pagina te navigeren. De hyperlink dient enkel om JavaScript code uit te voeren als de gebruiker op klikt. Als je het `href` attribuut niet invult, kan de gebruiker niet klikken op de hyperlink.

(5) Je voegt de hyperlink toe aan de nieuwe kolom.

(6) Je koppelt een anonieme functie aan het click event van de hyperlink.

(7) Je gaat van de hyperlink naar zijn parent `td` element.

Je gaat van dit `td` element naar zijn parent `tr` element.

(8) Je verwijdert dit `tr` element.

Je kan dit proberen.



## 14 KEUZES

Je opent `fastfood.html` in je editor. De pagina verwijst naar `fastfood.js`. Je maakt dit bestand.

De keuze voor de hamburger bevat een bijbehorende legende met de tekst `Hamburger` ....

Je toont in de legende het aantal hamburgers:

```
"use strict";
const aantalHamburgers = document.getElementById("hamburgers").length; ❶
document.getElementById("aantalHamburgers").innerText = aantalHamburgers;
```

(1) Je zoekt het select element. Dit is het element met de id `hamburgers`.

De `length` property geeft je het aantal option elementen in dat select element.

Je toont het aantal friet keuzes in de legende bij de keuze van de friet. Je voegt code toe:

```
const aantalFieten = document.getElementsByName("friet").length; ❶
document.getElementById("aantalFieten").innerText = aantalFieten;
```

(1) De method `getElementsByName` is *nog* een manier om elementen te zoeken.

De method geeft je de verzameling elementen waarvan het *name* attribuut gelijk is aan `friet`.

De `length` property van die verzameling geeft je het *aantal* elementen.

Je toont het aantal dessert keuzes in de legende bij de keuze van het dessert. Je voegt code toe:

```
const aantalDesserts = document.getElementsByName("dessert").length;
document.getElementById("aantalDesserts").innerText = aantalDesserts;
```

Je kan dit proberen.

Als de gebruiker een hamburger kiest, toon je die hamburger onder in de pagina. Je voegt code toe:

```
document.getElementById("hamburgers").onchange = function () { ❶
    document.getElementById("keuzeBurger").innerText = this.value; ❷
};
```

(1) Telkens de gebruiker in een select een andere option kiest, treedt het `change` event op.

(2) Je vindt de tekst van de gekozen option in de `value` property van de select.

Als de gebruiker een friet kiest, toon je de gekozen friet onder in de pagina. Je voegt code toe:

```
for (const inputFriet of document.getElementsByName("friet")) { ❶
    inputFriet.onchange = function () { ❷
        document.getElementById("keuzeFriet").innerText = this.value; ❸
    };
}
```

(1) Je itereert over alle elementen met `friet` als `name` attribuut.

(2) Als de gebruiker zo'n friet kiest, treedt het `change` event op.

Je koppelt een anonieme functie aan dit event.

(3) Je toont onder in de pagina het `value` attribuut van de gekozen friet.

De gebruiker kan één of meerdere desserts kiezen. Je toont die desserts onder in de pagina.

Je voegt code toe:

```
for (const inputDessert of document.getElementsByName("dessert")) { ❶
    inputDessert.onchange = function () { ❷
        let keuzes = "";
        for (const eenInputDessert of document.getElementsByName("dessert")) { ❸
            if (eenInputDessert.checked) { ❹
                keuzes += `${eenInputDessert.parentElement.innerText},`; ❺
            }
        }
        document.getElementById("keuzeDessert").innerText = keuzes.slice(0, -1);❻
    };
}
```

(1) Je itereert over alle "dessert keuzes": de elementen met `dessert` als `name` attribuut.

(2) Als de gebruiker zo'n dessert aanvinkt of uitvinkt, treedt het `change` event op.

- (3) Je itereert dan terug over alle “dessert keuzes”: de elementen met dessert als name attriboot.
- (4) Als een dessert aangevinkt is, bevat zijn checked property de waarde true.
- (5) Je gaat van het checkbox naar zijn parent element: het label element dat de checkbox bevat.  
Je vraagt de tekst binnen dat label element. je voegt die tekst en een komma toe aan keuzes.
- (6) Na de iteratie bevat keuzes achteraan een overbodige komma. Je neemt daarom alle tekens uit keuzes, behalve het laatste teken. Je toont die string onder in de pagina.

Je kan dit proberen.

Het gebeurt regelmatig dat je aan de gebruiker *namen* van producten (bvb. hamburgers) toont, maar dat je als programmeur ook het *nummer* van het gekozen product wil weten.

Je geeft elke hamburger een nummer:

```
<option value="1234">Hamburger</option>
<option value="5678">Cheeseburger</option>
<option value="9012">Fishburger</option>
<option value="3456">Chickenburger</option>
<option value="7890">Veggieburger</option>
<option value="1357">Bickyburger</option>
<option value="2456">Mexicanburger</option>
<option value="8024">Brazucaburger</option>
<option value="1470">Blackangusburger</option>
<option value="2581">Lamsburger</option>
<option value="9876">Camenbertburger</option>
```

Je geeft elke friet een nummer:

```
<label><input type="radio" name="friet" value="5">Small</label>
<label><input type="radio" name="friet" value="3">Medium</label>
<label><input type="radio" name="friet" value="1">Large</label>
```

Je geeft elke dessert een nummer:

```
<label><input type="checkbox" name="dessert" value="6">Fruit</label>
<label><input type="checkbox" name="dessert" value="4">Ijsje</label>
<label><input type="checkbox" name="dessert" value="2">Muffin</label>
```

Als de gebruiker een hamburger kiest toon je onder in de pagina het nummer en de naam van de gekozen hamburger. Je wijzigt de opdracht

```
document.getElementById("keuzeBurger").innerText = this.value;
```

naar

```
const gekozenNummer = this.value;
const gekozenNaam = this.options[this.selectedIndex].innerText;
document.getElementById("keuzeBurger").innerText =
  `${gekozenNummer}:${gekozenNaam}`;
```

❶

- (1) this.selectedIndex geeft je het *volgnummer* van de gekozen option in de select.  
this.options geeft je alle options van de select als een array.  
this.options[this.selectedIndex] geeft je dus de gekozen option.

Als de gebruiker een friet kiest toon je onder in de pagina het nummer en de naam van die friet.

Je wijzigt de opdracht

```
document.getElementById("keuzeFriet").innerText = this.value;
```

naar

```
const gekozenNummer = this.value;
const gekozenNaam = this.parentElement.innerText;
document.getElementById("keuzeFriet").innerText =
  `${gekozenNummer}:${gekozenNaam}`;
```

❶

- (1) this.parentElement geeft je het label element dat de gekozen friet bevat.

Als de gebruiker desserts kiest toon je onder in de pagina de nummers en namen van die desserts.

Je wijzigt de opdrachten

```
keuzes += `${eenInputDessert.parentElement.innerText},`;
```

naar

```
keuzes += `${eenInputDessert.value}:${eenInputDessert.parentElement.innerText},`;
```

Je kan dit proberen.



Proeven: zie takenbundel



Voorkeurburgers: zie takenbundel

## 15 VALIDEREN

Je opent mens.html in je editor. De pagina verwijst naar mens.js. Je maakt dit bestand.

Er zijn veel pagina's waar de gebruiker gegevens intikt. Vooraleer je de gegevens opslaat of berekeningen doet met de gegevens controleer je (valideer je) of de gegevens correct zijn.

Er bestaan twee methodes om gegevens te valideren

1. De validatie helemaal coderen in JavaScript
2. Samenwerken met validatie attributen van HTML elementen.

### 15.1 Methode 1: de validatie helemaal coderen in JavaScript

```
"use strict";
document.getElementById("toevoegen").onclick = function () {
    verbergAlleFoutMeldingen();
    let allesOK = true;
    const voornaam = document.getElementById("voornaam").value;
    if (voornaam === "") {
        document.getElementById("voornaamFout").style.display= "inline";
        allesOK = false;
    }
    const kinderen = document.getElementById("kinderen").value;
    const kinderenFout = document.getElementById("kinderenFout");
    if (kinderen=== "" || isNaN(kinderen) || kinderen < 0) {
        document.getElementById("kinderenFout").style.display= "inline";
        allesOK = false;
    }
    const geslacht = document.getElementById("geslacht").value;
    if (geslacht === "") {
        document.getElementById("geslachtFout").style.display= "inline";
        allesOK = false;
    }
    document.getElementById("toegevoegd").style.display =
        allesOK ? "inline" : "";
};
function verbergAlleFoutMeldingen() {
    for (const melding of document.querySelectorAll("span[class=fout]")) {
        melding.style.display = "";
    }
}
```

- (1) Je verbergt alle foutmeldingen.  
Ze kunnen zichtbaar zijn na een vorige klik op de knop, waarbij de gebruiker velden leeg liet.
- (2) Je zal deze variabele op false plaatsen zodra een veld verkeerd is.
- (3) Deze variabele bevat de ingetikte voornaam.
- (4) Je controleert of de voornaam leeg is.
- (5) Als dit zo is, toon je de foutmelding die hoort bij de voornaam.
- (6) Minstens één veld is verkeerd.
- (7) Je controleert of het vak kinderen ingevuld is. Je controleert ook of het een getal bevat, met de ingebakken functie `isNaN`. `isNaN` geeft `true` terug als de de parameter *geen* getal is.  
Je controleert ook of kinderen niet negatief is.
- (8) Als er geen enkele validatiefout was toon je de tekst Toegevoegd. Anders verberg je de tekst.
- (9) Je itereert over alle span elementen die een attribuut `class` hebben gelijk aan `fout`.

Je kan dit proberen.

Het nadeel van deze method is dat je *veel* JavaScript code moet schrijven, zeker als je *veel* velden moet valideren.

## 15.2 Methode 2: samenwerken met validatie attributen van HTML elementen.

Je wijzigt het invoervak voor de voornaam:

```
<input id="voornaam" autofocus required>
```

❶

(1) required geeft aan dat dit invoervak verplicht in te vullen is.

Je wijzigt het invoervak voor de kinderen:

```
<input id="kinderen" required type="number" min="0">
```

❶

(1) required geeft aan dat dit invoervak verplicht in te vullen is.

type="number" geeft aan dat dit invoervak een getal moet bevatten.

min="0" geeft aan dat dit getal minstens 0 moet zijn.

Je wijzigt het keuze voor het geslacht:

```
<select id="geslacht" required>
```

❶

(1) required geeft aan dat een keuze verplicht is.

Als de gebruiker (Maak een keuze) kiest, is de bijbehorende value een lege string.

De gebruiker heeft dan geen geslacht gekozen.

Je wijzigt mens.js:

```
"use strict";
```

```
document.getElementById("toevoegen").onclick = function () {
  verbergAlleFoutMeldingen();
```

```
  let allesOK = true;
```

```
  if (! document.getElementById("voornaam").checkValidity()) {
    document.getElementById("voornaamFout").style.display= "inline";
    allesOK = false;
```

❶

```
  }
  if (! document.getElementById("kinderen").checkValidity()) {
    document.getElementById("kinderenFout").style.display= "inline";
    allesOK = false;
```

```
  }
  if (! document.getElementById("geslacht").checkValidity()) {
    document.getElementById("geslachtFout").style.display = "inline";
    allesOK = false;
```

```
  }
  document.getElementById("toegevoegd").style.display =
    allesOK ? "inline" : "";
};
```

```
function verbergAlleFoutMeldingen() {
  for (const melding of document.querySelectorAll("span[class=fout]")) {
    melding.style.display = "";
```

❹

```
  }
}
```

(1) De method checkValidity geeft true terug als het invoervak correct is ten opzichte van de validatie attributen die je in HTML aan dit invoervak toevoegde. Als het invoervak fouten bevat de method false terug.

Je kan dit proberen.

De code is een beetje korter dan bij de eerste methode.

De code kan nog korter:

```
"use strict";
```

```
document.getElementById("toevoegen").onclick = function () {
```

```
  const verkeerdeElementen=
```

```
    document.querySelectorAll("input:invalid,select:invalid");
```

❶

```
  for (const element of verkeerdeElementen) {
    document.getElementById(`${element.id}Fout`).style.display = "inline";
```

❷

```
  }
}
```

```
const correcteElementen =  
  document.querySelectorAll("input:valid,select:valid");  
for (const element of correcteElementen) {  
  document.getElementById(`${element.id}Fout`).style.display = "";  
}  
document.getElementById("toegevoegd").style.display =  
  verkeerdeElementen.length === 0 ? "inline" : "";  
};
```

- (1) De CSS selector `input:invalid` geeft je de verzameling inputs met verkeerde waarden. De CSS selector `select:invalid` geeft je de verzameling selects met verkeerde waarden. `verkeerdeElementen` is dus de verzameling met verkeerde inputs én verkeerde selects.
- (2) Je zoekt de foutmelding die hoort bij de verkeerde input of select. De id van de foutmelding is gelijk aan de id van het verkeerde element plus "Fout". De id van de foutmelding bij de input met de id voornaam is bijvoorbeeld `voornaamFout`. Je toont de foutmelding.
- (3) Je zoekt de inputs en selects die *wel* correct (valid) zijn.
- (4) Je verbergt de bijbehorende foutmeldingen.
- (5) Als de verzameling verkeerde elementen leeg is, zijn alle gegevens correct. Je toont dan de tekst Toegevoegd. Zoniet verberg je de tekst.

Je kan dit proberen.

Ook als je extra gegevens opvraagt (een verplicht in te vullen familienaam, een verplicht in te vullen gemeentenaam, ...) moet je deze code niet uitbreiden om de extra gegevens te valideren.



Delen: zie takenbundel

## 16 LOCAL STORAGE, SESSION STORAGE

Je onthoudt tot nu gegevens in variabelen in het RAM geheugen.  
Zodra de gebruiker de pagina verlaat, gaan deze gegevens verloren.

Je kan gegevens langer bewaren. Browsers bevatten daartoe verschillende technologieën:

- **Permanente cookies**  
De browser bewaart de gegevens die je hierin plaatst op de harde schijf.
- **Tijdelijke cookies**  
De browser bewaart de gegevens die je hierin plaatst in het RAM geheugen, tot je de browser sluit. Een website kan meerdere pagina's bevatten. Als je in de pagina `index.html` iets onthoudt in een tijdelijke cookie, navigeert naar een andere pagina van de website en terug navigeert naar `index.html` is het gegeven nog beschikbaar in de cookie.
- **Local storage**  
De browser bewaart de gegevens die je hierin plaatst op de harde schijf.
- **Session storage**  
De browser onthoudt de gegevens die je hierin plaatst in het RAM geheugen, tot je de browser sluit. Een website kan meerdere pagina's bevatten. Als je in de pagina `index.html` iets onthoudt in session storage, navigeert naar een andere pagina van de website en terug navigeert naar `index.html` is het gegeven nog beschikbaar.
- **IndexedDB**  
De browser bewaart de gegevens die je hierin plaatst op de harde schijf. IndexedDB is geavanceerd en valt buiten het bereik van deze basiscursus.
- **Web SQL**  
De browser bewaart de gegevens die je hierin plaatst op de harde schijf. Sommige browsers (zoals Firefox) ondersteunen Web SQL *niet*.

Cookies bestaan al lang, hebben beperkingen en zijn moeilijker te programmeren.

Je leert twee technologieën kennen: local storage en session storage.

### 16.1 Local storage

Je opent `kleuren.html` in je editor. De pagina verwijst naar `kleuren.js`. Je maakt dit bestand.

- Als de gebruiker op de 1° knop klikt, onthoud je in local storage dat zijn favoriete achtergrondkleur roze is.
- Als de gebruiker op de 2° knop klikt, onthoud je in local storage dat zijn favoriete achtergrondkleur lichtblauw is.

```
"use strict";
for (const button of document.querySelectorAll("button[data-achtergrondkleur]")) { ❶
  button.onclick = function () {
    const kleur = this.dataset.achtergrondkleur; ❷
    localStorage.setItem("achtergrondkleur", kleur); ❸
    document.querySelector("body").style.backgroundColor = kleur; ❹
  };
}
```

- (1) Je itereert over de buttons met die een attribuut `data-achtergrondkleur` hebben.
- (2) Je leest de inhoud van het attribuut `data-achtergrondkleur`. Dit is de gekozen kleur.
- (3) Je bewaart de kleur in local storage. Het object `localStorage` stelt de local storage voor. Je bewaart een gegeven in local storage met de method `setItem`. Je geeft 2 parameters mee. De 1° parameter is de *naam* onder de welke je het gegeven wil bewaren. Je kan meerdere gegevens bewaren. Je geeft elk gegeven een eigen naam. De 2° parameter is het gegeven zelf.
- (4) Je gebruikt de kleur als achtergrondkleur van de pagina.

Je kan dit proberen.

Je ziet met volgende stappen in de Chrome browser dat de kleur is bewaard in local storage:

1. Je drukt F12.
2. Je kiest onder in de browser het tabblad Application.
3. Je klikt links op het pijltje voor Local Storage.
4. Je klikt daarbinnen op `http://127.0.0.1:5000` (als je de Live Server extension van Visual Studio code gebruikt) of `file://` (als je de pagina rechtstreeks opende vanaf de harde schijf).
5. Je ziet een kolom Key met daarin achtergrondkleur en een kolom Value met daarin de gekozen kleur.

De gebruiker kiest met de 3<sup>e</sup> en 4<sup>e</sup> knop zijn favoriete tekstkleur. Je voegt code toe:

```
for (const button of document.querySelectorAll("button[data-tekstkleur]")) {
  button.onclick = function () {
    const kleur = this.dataset.tekstkleur;
    localStorage.setItem("tekstkleur", kleur);
    document.querySelector("body").style.color = kleur;
  };
}
```

Je kan dit proberen.

Als de gebruiker de pagina opent, zoek je in de local storage of de gebruiker bij een vorig bezoek kleuren gekozen heeft. Als dit zo is, pas je de kleuren toe op de pagina. Je voegt code toe:

```
const achtergrondkleur = localStorage.getItem("achtergrondkleur");           ❶
if (achtergrondkleur !== null) {                                           ❷
  document.querySelector("body").style.backgroundColor = achtergrondkleur; ❸
}
const tekstkleur = localStorage.getItem("tekstkleur");
if (tekstkleur !== null) {
  document.querySelector("body").style.color = tekstkleur;
}
```

- (1) Je leest een gegeven uit de local storage met de method `getItem`.  
Je geeft als parameter de naam van het gegeven mee dat je wil lezen.  
De method geeft je het gegeven als de local storage dit gegeven bevat.  
De method geeft je `null` (niets) als de local storage dit gegeven niet bevat.
- (2) Je controleert of je de achtergrondkleur kon lezen in de local storage. Als de gebruiker nog nooit op de 1<sup>e</sup> of 2<sup>e</sup> knop klikte, bevat de local storage nog geen achtergrondkleur.
- (3) Je past de achtergrondkleur toe op de pagina.

Je kan dit proberen.

Als de gebruiker op de 5<sup>e</sup> knop klikt, verwijder je de 2 gegevens uit local storage. Je voegt code toe:

```
document.getElementById("vergeet").onclick = function () {
  localStorage.removeItem("achtergrondkleur");           ❶
  localStorage.removeItem("tekstkleur");
  const body = document.querySelector("body");
  body.style.backgroundColor = "";                       ❷
  body.style.color = "";
};
```

- (1) Je verwijdert een gegeven met de method `removeItem`.  
Je geeft als parameter de naam van het te verwijderen gegeven mee.
- (2) Je past de gekozen kleur niet meer toe in de pagina.

Je kan dit proberen.

Opmerking: Je kan *alle* gegevens verwijderen uit local storage met de method `clear`.

Elke website heeft zijn eigen gegevens in local storage. De method `clear` verwijdert dus enkel de gegevens van die website uit local storage, niet de gegevens van andere websites.



## 16.2 Session storage

Session storage werkt zoals local storage, maar de browser onthoudt de gegevens van session storage in zijn RAM geheugen, tot je de browser sluit.

Session storage is interessant om gegevens te onthouden over *meerdere* pagina's heen (dus niet enkel in de huidige pagina).

Je opent wiebenje.html in je editor. De pagina verwijst naar wiebenje.js. Je maakt dit bestand.

Als de gebruiker op de knop klikt, onthoud je zijn ingetikte naam in session storage:

```
"use strict";
document.getElementById("onthoudMe").onclick = function () {
    sessionStorage.setItem("voornaam", document.getElementById("voornaam").value);
};
```

Je kan dit proberen.

Je ziet met volgende stappen in Chrome dat de voornaam is bewaard in session storage:

1. Je drukt F12.
2. Je kiest onder in de browser het tabblad Application.
3. Je klikt links op het pijltje voor Session Storage.
4. Je klikt daarbinnen op `http://127.0.0.1:5000` (als je de Live Server extension van Visual Studio code gebruikt) of `file://` (als je de pagina rechtstreeks opende vanaf de harde schijf).
5. Je ziet links een kolom Key met daarin voornaam een een kolom Value met de voornaam.  
Opmerking: als je de Live Server extension van Visual Studio code gebruikt is er ook een gegeven met de naam `IsThisFirstTime...`

Als de gebruiker op de hyperlink Hallo klikt, ziet hij de pagina `hallo.html`.

Je toont daarin de voornaam die je onthouden hebt in session storage.

Je tikt code in `hallo.js`:

```
"use strict";
const voornaam = sessionStorage.getItem("voornaam");
if (voornaam !== null) {
    document.getElementById("voornaam").innerText = voornaam;
}
```

Je kan dit proberen.



Bezocht: zie takenbundel

## 17 JSON

Je leerde reeds een object literal kennen.

Je vult in het volgende voorbeeld de variabele `persoon` met een object literal:

```
const persoon = { voornaam: "Hans", kinderen: 3, gehuwd: true };
```

Je moet soms een object literal voorstellen als een tekst.

Voorbeeld: je wil een object literal opslaan in local storage.

Je kan echter enkel een tekst opslaan in local storage, geen object literal.

Oplossing: je maakt een tekstvoorstelling van de object literal en je slaat die tekstvoorstelling op.

De tekstvoorstelling van een object literal heet JSON (JavaScript Object Notation).

Je ziet hier onder de tekstvoorstelling van de object literal boven in deze pagina:

```
{ "voornaam": "Hans", "kinderen": 3, "gehuwd": true }
```

❶

(1) De namen van de properties (bvb. `voornaam`) zijn bij JSON strings tussen dubbele quotes.

JavaScript bevat ook een *object* met de naam `JSON`.

- `JSON` bevat een method `stringify`.  
Je zet met `stringify` een object literal om naar zijn JSON formaat.
- `JSON` bevat een method `parse`.  
Je zet met `parse` een string in JSON formaat om naar een object literal.

Je opent `tijdstippen.html` in je editor. De pagina verwijst naar `tijdstippen.js`.

Je maakt dit bestand.

Je probeert het object `JSON` uit:

```
"use strict";
```

```
const persoon = { voornaam: "Hans", kinderen: 3, gehuwd: true };
```

```
const persoonInJSONFormaat = JSON.stringify(persoon);
```

```
console.log(persoonInJSONFormaat);
```

```
const hierIsDePersoonTerug = JSON.parse(persoonInJSONFormaat);
```

```
console.log(hierIsDePersoonTerug.voornaam);
```

❶

❷

❸

❹

(1) Je roept de method `stringify` op. Je geeft een object literal mee als parameter.  
`stringify` geeft je een string terug met de JSON voorstelling van die object literal.

(2) Je toont deze string in de console.

(3) Je roept de method `parse` op. Je geeft een string in JSON formaat mee als parameter.  
`parse` geeft je een nieuwe object literal terug die een voorstelling is van de string.

(4) De object literal heeft ook property `voornaam`. De inhoud van de property is `Hans`.

Je kan dit proberen.

Je werkt de pagina uit: Je onthoudt de 10 laatste momenten waarop de gebruiker de pagina bezoekt. Je onthoudt die momenten als elementen van een array.

Je onthoudt een JSON voorstelling van de array in local storage.

```
"use strict";
```

```
let bezoeken = JSON.parse(localStorage.getItem("bezoeken"));
```

```
if (bezoeken === null) {
```

```
    bezoeken = [];
```

```
} else {
```

```
    if (bezoeken.length === 10) {
```

```
        bezoeken.shift();
```

```
    }
```

```
}
```

```
bezoeken.push(new Date().toLocaleString());
```

```
localStorage.setItem("bezoeken", JSON.stringify(bezoeken));
```

❶

❷

❸

❹

❺

❻

❼

(1) Je leest uit de local storage het gegeven met de naam `bezoeken`  
en je converteert dit gegeven (een string) naar een object literal.

- (2) Als de object literal `null` is betekent dit dat het gegeven niet voorkomt in local storage.  
De gebruiker bezoekt de pagina dan voor de eerste keer.  
Als de object literal niet `null` is bevat hij de array met tijdstippen van de vorige bezoeken.
- (3) Je maakt bij het eerste bezoek een lege array.
- (4) Als die array 10 elementen bevat
- (5) verwijder je het 1<sup>o</sup> element met de `shift` method.
- (6) Je voegt een nieuw element met de systeemtijd toe aan de array.
- (7) Je zet de array om naar een string in JSON formaat en je bewaart die string in local storage.

Je kan dit proberen. Naarmate je de pagina “refresh” zie je in de local storage de array uitrekken tot hij maximaal 10 elementen bevat. Vanaf dan zie je bij elke “refresh” het 1<sup>o</sup> element verdwijnen en een nieuw element achteraan in de array.

Je voegt code toe. De gebruiker ziet zo de tijdstippen ook in de pagina zelf:

```
const ul = document.getElementById("bezoeken");
for (const bezoek of bezoeken) {
  const li = document.createElement("li");
  li.innerText = bezoek;
  ul.appendChild(li);
}
```

Je kan dit proberen.



Mandje: zie takenbundel

## 18 FETCH

De browser vraagt een website verschillende soorten data: HTML, CSS, afbeeldingen, ...

Sommige websites bieden ook data aan in JSON formaat.

De data kan koersen van munten bevatten, koersen van beursaandelen, artikel prijzen, ...

Soms is de data te betalen.

Je leert hier hoe je in je JavaScript code deze data vraagt aan zo'n website en het antwoord met die data gebruikt in je eigen website.

Je leest als voorbeeld JSON data van de website <https://reqres.in/>.

Voor je de data leest in je JavaScript code is het interessant de data op te vragen met je browser. Je ziet zo de *structuur* van de data.

Je surft naar <https://reqres.in/api/users/1>. Je krijgt de data over een persoon met het nummer 1 in JSON formaat. Je ziet de data hier in een mooiere opmaak:

```
{
  "data": {
    "id": 1,
    "email": "george.bluth@reqres.in",
    "first_name": "George",
    "last_name": "Bluth",
    "avatar": "https://s3.amazonaws.com/uifaces/faces/twitter/calebogden/128.jpg"
  }
}
```

❶  
❷  
❸

(1) De JSON data is (zoals altijd) een voorstelling van een object literal.

(2) Deze object literal bevat een property met als naam data.

Deze property bevat op zijn beurt een object literal.

(3) Deze object literal heeft enkele properties. De 1<sup>o</sup> property heeft als naam id en als inhoud 1.

Als je de JSON data zou omzetten naar een object literal die je onthoudt

in een variabele met de naam user, toon je met volgende code de voornaam van de persoon:

```
console.log(user.data.first_name);
```

Als je surft naar <https://reqres.in/api/users/2> zie je data over de persoon met nummer 2.

Je geeft dus met het getal achter in de URL aan over welke persoon je data wenst.

Je leert nu stap per stap hoe je vanuit je JavaScript code dezelfde JSON data opvraagt.

Je opent user.html in je editor. De pagina verwijst naar user.js. Je maakt dit bestand.

```
"use strict";
const promise = fetch("https://reqres.in/api/users/1");
promise.then(verwerkResponse);
function verwerkResponse(response) {
  console.log(response);
}
```

❶  
❷  
❸  
❹

(1) Je vraagt data van een website met de ingebakken functie fetch.

Je geeft als parameter de URL mee waarop de website de data aanbiedt.

fetch geeft je een "promise" object terug.

Een promise (belofte) is iets dat niet nu, maar in de toekomst zal gebeuren.

Je vraagt de data *nu* aan de website, maar je krijgt die data maar in in de toekomst.

De internetverbinding naar de website kan traag zijn. Het duurt dan enkele seconden

voor je de data krijgt. Je kan *nu* (het moment dat je de data vraagt) nog niet in die data kijken.

(2) Als je het antwoord met de data uiteindelijk van de website krijgt, wil jij iets met die data doen.

In ons voorbeeld wil je de functie verwerkResponse (bij ❸) uitvoeren.

Je geeft dit aan met de then method van de promise. Je geeft de naam van je functie mee

als parameter. Als de data (bvb. na 2 seconden) binnenkomt, voert de promise je functie uit.

- (3) Als de data binnenkomt, roept de promise je functie op. De promise biedt je functie informatie aan over het antwoord van de website, als een parameter van je functie.  
 Je mag die parameter een naam geven naar eigen keuze. Hier is die naam `response`.
- (4) Je toont het antwoord van de website voorlopig in de console.

Je probeert dit. Je ziet dat de `response` een object is. Je ziet de JSON data die zich in de `response` bevindt nog niet. Je zal daarvoor straks nog code toevoegen. Je ziet wel dat de `response` een property `status` heeft met als inhoud `200`. De menselijke betekenis van `200` is OK. Dit betekent dat de website je vraag (naar data) correct kon beantwoorden.

Je wijzigt de code: je vraagt data van een onbestaande user:

```
const promise = fetch("https://reqres.in/api/users/1000000")
```

Je probeert dit. De `status` property van de `response` bevat nu `404`.

De menselijke betekenis van `404` is Not Found: de website vond niet wat jij gevraagd hebt.

De opdrachten

```
const promise = fetch("https://reqres.in/api/users/1000000");
promise.then(verwerkResponse);
```

worden meestal verkort tot 1 opdracht:

```
fetch("https://reqres.in/api/users/1000000").then(verwerkResponse)
```

Je wijzigt de volledige code:

```
"use strict";
fetch("https://reqres.in/api/users/1").then(verwerkResponse);
function verwerkResponse(response) {
  const nietGevondenDiv = document.getElementById("nietGevonden");
  if (response.ok) {
    nietGevondenDiv.style.display = "";
  } else {
    nietGevondenDiv.style.display = "block";
  }
}
```

❶  
❷  
❸

- (1) De `ok` property bevat `true` als de `status` property gelijk is aan `200`.  
 Je zou deze regel ook kunnen tikken als `if (response.status === 200) {`
- (2) Als de website de data heeft gevonden, verberg je de foutmelding "Niet gevonden".
- (3) Anders toon je de foutmelding.

Je probeert dit. Je ziet de foutmelding niet. Je wijzigt in de URL `1` naar `1000000`.

Je ziet de foutmelding. Je wijzigt in de URL `1000000` terug naar `1`.

Als de status ok is, laat je de JSON data omzetten naar een object literal. Je voegt na

`nietGevondenDiv.style.display = ""`; code toe:

```
const promise = response.json();
promise.then(verwerkUser);
```

❶  
❷

- (1) Je vraag met de method `json` de JSON data in de `response` om te zetten naar een object literal. Dit omzetten kan enkele milliseconden duren, afhankelijk van de omvang van de JSON data. Ook `json` geeft je een promise terug. Je geeft met de method `then` van die promise aan dat als de JSON data uiteindelijk helemaal is omgezet naar een object literal, je functie `verwerkUser` (zie verder) moet uitgevoerd worden.

De opdrachten bij ❶ en ❷ worden meestal verkort tot één opdracht:

```
response.json().then(verwerkUser);
```

Je voegt ook onder in de source de functie `verwerkUser` toe:

```
function verwerkUser(user) {
  console.log(user);
}
```

❶

- (1) Nadat de JSON data is omgezet naar een object literal roept de promise je functie op. De promise biedt je functie de object literal, als een parameter van je functie.

Je probeert dit. Je ziet in de console een object literal met een property data.

Deze property is terug een object literal met de properties id, email, ...

Je wijzigt de code in de functie verwerkUser: je toont de data in de pagina (niet meer in de console):

```
document.getElementById("nummer").innerText = user.data.id;
document.getElementById("voornaam").innerText = user.data.first_name;
document.getElementById("familienaam").innerText = user.data.last_name;
document.getElementById("emailAdres").innerText = user.data.email;
document.getElementById("avatar").src = user.data.avatar;
```

Je kan dit proberen.

Je breidt de pagina uit. Je laat de gebruiker een nummer intikken en je toont de data van de user met dat nummer (in plaats van de user met het nummer 1).

Je voegt in user.html regels toe na <body>:

```
<label>Nummer:</label>
<input id="zoekNummer" autofocus required type="number" min="1">
<button id="zoeken">Zoeken</button>
```

Je verbergt initieel het <dl> element:

```
<dl id="user" class="verborgen">
```

Je voegt een regel toe voor </body>:

```
<div id="nummerFout" class="fout">Verkeerd nummer.</div>
```

Je vervangt in user.js de opdracht

```
fetch("https://reqres.in/api/users/1").then(verwerkResponse);
```

door

```
document.getElementById("zoeken").onclick = function () {
  const nummerInput = document.getElementById("zoekNummer");
  const nummerFout = document.getElementById("nummerFout");
  if (nummerInput.checkValidity()) {
    nummerFout.style.display = "";
    fetch(`https://reqres.in/api/users/${nummerInput.value}`).then(verwerkResponse);
  } else {
    nummerFout.style.display = "block";
  }
};
```

- (1) Je controleert of de gebruiker een correct nummer tikte.
- (2) De gebruiker tikte een correct nummer. Je verbergt de foutmelding over het nummer.
- (3) Je haalt de JSON data op van de user met het ingetikte nummer.
- (4) De gebruiker tikte een verkeerd nummer. Je toont de foutmelding over het nummer.

Je breidt de code in de functie verwerkResponse uit:

```
const nietGevondenDiv = document.getElementById("nietGevonden");
const userDl = document.getElementById("user");
if (response.ok) {
  nietGevondenDiv.style.display = "";
  userDl.style.display = "block";
  response.json().then(verwerkUser);
} else {
  nietGevondenDiv.style.display = "block";
  userDl.style.display = "";
}
```

- (1) Je hebt de user gevonden. Je toont het <dl> element waarin je de user data afbeeldt.
- (2) Je hebt de user *niet* gevonden. Je verbergt het <dl> element waarin je de user data afbeeldt.



Users: zie takenbundel

## 19 GET, POST, PUT, DELETE

### 19.1 HTTP

Als je de functie `fetch` oproept stuur je een vraag naar een website en krijg je een antwoord terug.

- De vraag heet in technische termen een request.
- Het antwoord heet een response.

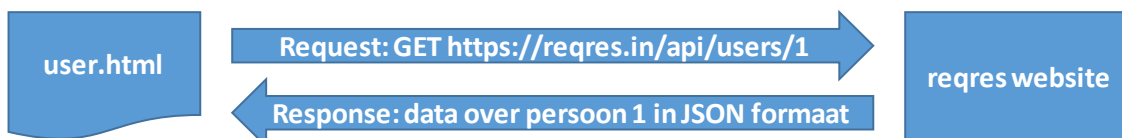
De request en de response bevatten bytes. De grove opbouw van de bytes is gedefinieerd in het HTTP protocol. HTTP staat voor Hypertext Transfer Protocol.

Opmerking: ook als een browser een HTML pagina, een afbeelding, een CSS bestand of een js bestand vraagt aan een website, gebruikt de browser het HTTP protocol.

Een HTTP request bevat minstens twee onderdelen:

- de URL waar je de request naar stuurt (bvb. `https://reqres.in/api/users/1`)
- een woord die de request method aangeeft. De belangrijkste request methods:
  - GET Je geeft hiermee aan dat je data *vraagt* aan de website.  
De requests die je tot nu stuurde met de functie `fetch` waren GET requests.
  - DELETE Je vraagt de website iets te *verwijderen*.  
Je vraagt met een DELETE request naar `https://reqres.in/api/users/1` om de gebruiker 1 te verwijderen.
  - POST Je vraagt de website iets *toe te voegen*.  
Je vraagt met een POST request naar `https://reqres.in/api/users` om een gebruiker toe te voegen.  
Je request moet dan ook de data van die nieuwe gebruiker bevatten.
  - PUT Je vraagt de website iets te *wijzigen*.  
Je vraagt met een PUT request naar `https://reqres.in/api/users/1` om gebruiker 1 te wijzigen.  
Je request moet dan ook de te wijzigen data van gebruiker 1 bevatten.

Het voorbeeld van het vorige hoofdstuk:



### 19.2 POST

Je verstuurde in het vorige hoofdstuk GET requests met de functie `fetch`.

Je ziet hier een voorbeeld hoe je een POST request stuurt.

Je opent `toevoegen.html` in je editor. De pagina verwijst naar `toevoegen.js`.

Je maakt dit bestand.

Als de gebruiker op de knop klikt, vraag je aan de website `reqres.in` om een user toe te voegen met de voornaam en de familienaam die de gebruiker intikte.

In de praktijk valideer je eerst de ingetikte voornaam en familienaam.

We doen het hier niet, om de code kort te houden.

```
"use strict";
```

```
document.getElementById("toevoegen").onclick = function () {
  const user = {
    first_name: document.getElementById("voornaam").value,
    last_name: document.getElementById("familienaam").value
  };
};
```

❶  
❷  
❸

```

fetch("https://reqres.in/api/users",
  {
    method: "POST",
    body: JSON.stringify(user)
  }
).then(verwerkResponse);
};
function verwerkResponse(response) {
  const foutDiv = document.getElementById("fout");
  if (response.ok) {
    foutDiv.style.display = "none";
    response.json().then(verwerkDataInResponse);
  } else {
    foutDiv.style.display = "";
  }
}
function verwerkDataInResponse(data) {
  document.getElementById("nummer").innerText = data.id;
  document.getElementById("toegevoegd").style.display = "block";
}

```

- (1) Je maakt een object literal.  
Deze stelt de nieuwe user voor die je zal vragen toe te voegen met een POST request.
- (2) Je maakt een property `first_name` met als waarde de ingetikte voornaam.
- (3) Je maakt een property `last_name` met als waarde de ingetikte familienaam.
- (4) Je verstuurt de request met de functie `fetch`.  
De eerste parameter is de URL waar je de request naar stuurt.
- (5) De tweede parameter is een object literal met extra informatie over de request.
- (6) Je vult de property `method` met `POST`. Je geeft zo aan een POST request te versturen.
- (7) Je vult de property `body` met de JSON voorstelling van het user object dat je maakte bij ❶.  
`fetch` stuurt de JSON voorstelling mee in de binnenkant (`body`) van de request.
- (8) Als de response op de request terug binnenkomt (dit kan even duren)  
vraag je de functie `verwerkResponse` uit te voeren.
- (9) De website heeft de user met succes toegevoegd.
- (10) Je verbergt dan de foutmelding.
- (11) Je vraagt de data in de response van JSON formaat om te zetten in een object literal  
en daarna de functie `verwerkDataInResponse` uit te voeren.
- (12) De website heeft je request niet kunnen verwerken. Je toont een foutmelding.
- (13) De data in de response bevat twee properties:  
een property `id` met het nummer die de website toekende aan de nieuwe user  
en een property `createdAt` met het tijdstip waarop de website de nieuwe user toevoegde.  
Je toont het nummer dat de website toekende aan de nieuwe user.

Je kan dit proberen.

De website doet slechts *alsof* hij de nieuwe user permanent toevoegt aan zijn database.  
Als hij dit *wel* zou doen en het nummer van de nieuwe user 777 is, zou je na het toevoegen een GET request kunnen sturen naar `https://reqres.in/api/users/777` om de user op te vragen.



## 20 HERHALINGSOEFENINGEN



Friet: zie takenbundel



Saus: zie takenbundel



Albums: zie takenbundel



Letters: zie takenbundel



Hoogspringen: zie takenbundel



Verspringen: zie takenbundel

## 21 COLOFON

<b>Domeinexpertisemanager:</b>	Jean Smits
<b>Moduleverantwoordelijke:</b>	Hans Desmet
<b>Medewerkers:</b>	Hans Desmet
<b>Versie:</b>	25/10/2019
<b>Nummer dotatielijst:</b>	