VDAB

samen sterk voor werk

Van

naar

{ JavaScript; }

Inhoudsopgave

1 INLEIDING

1.1 Vereiste voorkennis

Blockly

1.2 Inleiding

Een HTML-pagina kan JavaScriptcode bevatten.
De browser voert de code uit.

Je leert in de cursus de basis van JavaScript.
Je ziet code in Blockly. Je ziet daaronder dezelfde code in JavaScript.
Je hoeft de Blocklycode niet te maken, de JavaScriptcode wel.

Je ziet rechts in de JavaScriptcode soms ❶, ❷, ❸, ...
Je tikt die tekens niet in je code.
Ze dienen enkel als verwijzing naar uitleg onder de code.

1.3 Editor

Je kan JavaScriptcode tikken met veel soorten editors.

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

1.4 Browser

Je kan de programma's in de cursus uitvoeren in elke moderne browser
(Chrome, Firefox, Edge, Safari, ...). Veel programma's zullen niet werken in Internet Explorer.
Dit is een verouderde browser. Hij ondersteunt de moderne mogelijkheden van JavaScript niet.

1.5 HTML pagina

Je maakt met je favoriete HTML-editor een bestand `index.html` met volgende inhoud:

```
<!doctype html>
<html lang="nl">
  <head>
    <meta charset="UTF-8">
    <title>Van Blockly naar JavaScript</title>
    <link rel="icon" href="javascript.ico" type="image/x-icon">          ❶
    <script src="index.js"></script>                                     ❷
  </head>
  <body>
  </body>
</html>
```

(1) Je vindt `javascript.ico` in het materiaal bij de cursus.
Je plaatst `javascript.ico` in dezelfde directory als `index.html`.
(2) Je verwijst naar het bestand `index.js`. Dit bestand zal je JavaScriptcode bevatten.
`js` is de standaard extensie voor JavaScripbestanden.

Je maakt in dezelfde directory een leeg bestand `index.js`.

2 EERSTE PROGRAMMA

2.1 Tekst tonen

Je toont volgende tekst: `Programmeren`.

Blockly:

JavaScript (je tikt de code in `index.js`):

```
console.log("Programmeren");
```

(1)

(1) Je toont tekst met de opdracht `console.log`. Je tikt de tekst tussen `("en ")`.
Je eindigt elke opdracht in JavaScript met `;`

Je slaat het bestand op.

Je bekijkt het resultaat in de browser. Je ziet dit resultaat als je in de browser **F12** drukt, in het tabblad `Console` onderaan in de browser.

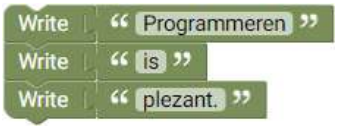
Opmerking: Je gebruikt in de cursus de console als een eenvoudig middel om tekst te tonen. De gebruiker kijkt niet in de console, maar ziet het resultaat van de HTML in je pagina Je bewerkt in een volgende cursus die HTML met JavaScriptcode.

2.2 Meerdere teksten tonen

Je toont volgende teksten:

```
Programmeren
is
plezant.
```

Blockly:



JavaScript:

```
console.log("Programmeren");
console.log("is");
console.log("plezant.");
```

2.3 Hoofdlettergevoelig

JavaScript is hoofdlettergevoelig.
Je mag **c**`onsole.log("Programmeren");` dus niet schrijven als **C**`onsole.log("Programmeren");`

2.4 Taak

Toon volgende tekst:

```
Ik
word
programmeur
```

Bewaar in `programmeur.js`.

Je vindt achter in de cursus een voorbeeldoplossing.

3 COMMENTAAR

Je kan aan je code commentaar toevoegen.

Deze commentaar is bedoeld als uitleg over de code voor jezelf en je collega programmeurs.

De computer negeert de commentaar tijdens het uitvoeren van je programma.

Er bestaan twee soorten commentaar.

3.1 Commentaar die begint met //

Als een regel `//` bevat, wordt alles vanaf `//` tot het einde van de regel aanzien als commentaar.

Voorbeeld:

```
console.log("Programmeren");  // Ik toon Programmeren
console.log("is");  // Ik toon is
console.log("plezant.");  // Ik toon plezant.
```

3.2 Commentaar tussen /* en */

Als een regel begint met `/*` wordt de rest van die regel en de daaropvolgende regels aanzien als commentaar tot er een regel is die begint met `*/`.

Voorbeeld:

```
/*
 Dit programma toont:
 Programmeren
 is
 plezant.
 */
console.log("Programmeren");
console.log("is");
console.log("plezant.");
```

4 GETALLEN

Je toont het getal `3.14`.

Blockly:

JavaScript:

```
console.log(3.14);
```

Je kan dit proberen.

4.1 Rekenen

JavaScript heeft volgende bewerkingen om te rekenen:

`+`	optellen
`-`	aftrekken
`*`	vermenigvuldigen
`/`	delen
`%`	restbepaling na deling (de rest van 11 gedeeld door 3 is 2, dus 11 % 3 is 2).

Blockly:

JavaScript:

```
console.log(5 + 2);
```

Je kan dit uitproberen.

4.2 Taak

Toon hoeveel 3 maal 4 is.

Bewaar in `vermenigvuldigen.js`.

Je vindt achter in de cursus een voorbeeldoplossing.

5 VERGELIJKEN

5.1 Getallen vergelijken

JavaScript heeft volgende operatoren (bewerkingen) om te vergelijken:

`===`	zijn twee getallen aan elkaar gelijk?
`!==`	zijn twee getallen verschillend?
`<`	is een getal kleiner dan een ander getal?
`<=`	is een getal kleiner of gelijk aan een ander getal?
`>`	is een getal groter dan een ander getal?
`>=`	is een getal groter of gelijk aan een ander getal?

Het resultaat van elke vergelijking is `true` of `false`.

Opmerking: je kan ook vergelijken of waarden aan elkaar gelijk zijn met `==` (in plaats van met `===`).
De regels wanneer waarden aan elkaar gelijk zijn, zijn bij `==` complex.
Deze complexe regels leiden tot bizarre resultaten. Voorbeelden:
een array met één element met de waarde `0` is bijvoorbeeld gelijk aan de waarde `false`,
een array met één element met de waarde `1` is gelijk aan de waarde `true`., ...

Bij `===` zijn de regels eenvoudig en leiden niet tot bizarre resultaten.
JavaScript experts raden daarom aan `===` te gebruiken.

Blockly:

JavaScript:

```
console.log(5 > 3);
```

Je kan dit proberen.

5.2 Teksten vergelijken

Blockly:

JavaScript:

```
console.log("limonade" < "bier");
```

5.3 Taak

Controleer of `4.6` kleiner is of gelijk aan `5`.

Bewaar in `kleinerOfGelijkAan.js`.

Je vindt achter in de cursus een voorbeeldoplossing.

6 VARIABELEN

Blockly:

JavaScript:

Een keyword is een woord in JavaScript dat een vooraf gedefinieerde betekenis heeft.
JavaScript bevat drie keywords om een variabele te maken:

- **`const`**
  Je maakt hiermee een variabele die je bij de aanmaak een beginwaarde geeft.
  Je wijzigt daarna die variabele niet meer.
  Als je dit toch probeert, krijg je een fout.
  JavaScript experts raden aan om alle variabelen die je met const kan maken met const te maken, en niet met let (zie hier onder). Const lijkt beperkt omdat je de waarde van de variabele na de aanmaak niet meer kan wijzigen. Dit heeft echter het positieve gevolg dat je de variabele ook niet meer *per ongeluk* kan wijzigen.
- **`let`**
  Je maakt hiermee een variabele waarvan je de waarde na de aanmaak nog kan wijzigen.
- **`var`**
  `var` is een verouderd keyword met nadelen. We gebruiken het niet.

```
const voornaam = "Alexandra";                                    ❶
const kinderen = 3;
const gehuwd = true;
console.log(voornaam, kinderen, gehuwd);                         ❷
```

(1) Je maakt een variabele met **`const`**, de naam van de variabele, **=** en de inhoud van de variabele.
(2) Je kan meerdere waarden *naast mekaar* tonen in de console.
Je geeft de waarden, gescheiden door een komma, mee aan `console.log();`

Je kan dit proberen.

Je kan een variabele maar één keer maken. De volgende code geeft dus een fout:

```
const voornaam = "Alexandra";
const voornaam = "Alexandra";
```

De tekst van de foutmelding is in elk merk van browser lichtjes anders.
Bij Chrome is de tekst bijvoorbeeld Identifier 'voornaam' has already been declared

6.1 Een variabele wijzigen

Je wijzig de code. Je plaatst de variabele `kinderen` eerst op 3. Je wijzigt ze daarna naar `4`:

```
let kinderen = 3;                                                ❶
kinderen = 4;                                                    ❷
console.log(kinderen);
```

(1) Je maakt de variabele met **`let`**. Je kan de variabele dan bij ❷ wijzigen.

Je kan dit proberen.

6.2 "use strict";

Je kan bij het wijzigen van een variabele een tikfout maken in de naam van de variabele:

```
let kinderen = 3;                ❶
linderen = 4;                    ❷
console.log(kinderen);           ❸
```

(1) Je maakt een variabele `kinderen` en initialiseert die op `3`.
(2) Je wil de variabele kinderen wijzigen, maar je maakt een tikfout: `linderen`.
Nu gebeurt iets vervelends: je krijgt geen foutmelding !
JavaScript maakt een *nieuwe* variabele `linderen` en initialiseert die op `4`.
(3) Je verwacht `4` te zien, maar je ziet `3`, door de tikfout bij ❷.
In een *klein* programma zal je de tikfout snel vinden.
In een *groot* programma kan het lang duren voor je de tikfout vindt.

Je lost dit op. Je tikt in je code altijd als eerste opdracht `"use strict";`

```
"use strict";
let kinderen = 3;
linderen = 4;                    ❶
console.log(kinderen);
```

(1) Je verwijst naar een variabele `linderen`. Je maakte die niet met `const`, `let` of `var`.
JavaScript stopt je code en toont in de console een foutmelding.
De tekst van de foutmelding is in elk merk van browser lichtjes anders.
Bij Chrome is de tekst bijvoorbeeld Uncaught ReferenceError: linderen is not defined

6.3 Inhoud van een variabele gebruiken in een berekening

Alexandra krijgt € `300` kindergeld per kind. Je toont het totale kindergeld.

JavaScript:

```
"use strict";
const kinderen = 4;
console.log("Totaal kindergeld:", kinderen * 300);
```

Je kan dit proberen.

6.4 Inhoud van een variabele opvragen aan de gebruiker

Blockly:

JavaScript:

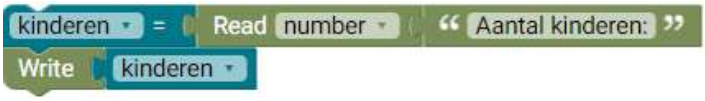
```
"use strict";
const voornaam = prompt("Voornaam:");    ❶
console.log(voornaam);
```

(1) JavaScript bevat een functie `prompt`. De gebruiker kan dan in een pop-upvenster een tekst tikken. Je geeft de tekst `Voornaam:` mee als parameter van de functie `prompt`.
Het pop-upvenster toont die tekst als uitleg bij wat de gebruiker moet tikken.
De functie geeft de tekst die gebruiker tikte terug als returnwaarde.

Je kan dit proberen.

6.5 Getal opvragen aan de gebruiker

Blockly:



JavaScript:

```
"use strict";
const kinderen = Number(prompt("Aantal kinderen:"));
console.log(kinderen);
```

(1) JavaScript bevat een functie `Number`. Je geeft als parameter een tekst die cijfers bevat. `Number` geeft de getalwaarde terug die de cijfers voorstellen.

Je kan dit proberen.

6.6 Variabele met een waarde verhogen of verlagen

Als een variabele een getal bevat, kan je die variabele met volgende opdracht verhogen:

```
kinderen = kinderen + 2;
```

Je kan dit ook korter schrijven:

```
kinderen += 2;
```

Als een variabele een getal bevat, kan je die variabele met volgende opdracht verlagen:

```
teBetalen = teBetalen - 10;
```

Je kan dit ook korter schrijven:

```
teBetalen -= 10;
```

6.7 Variabele met 1 verhogen of verlagen

Als een variabele een getal bevat, kan je die variabele met volgende opdracht met 1 verhogen:

```
kinderen = kinderen + 1;
```

Je kan dit ook korter schrijven:

```
kinderen++;
```

Als een variabele een getal bevat, kan je die variabele met volgende opdracht met 1 verlagen:

```
teBetalen = teBetalen - 1;
```

Je kan dit ook korter schrijven:

```
teBetalen--;
```

6.8 Hoofdlettergevoelig

Ook namen van variabelen zijn hoofdlettergevoelig.
De variabele **k**inderen is een andere variabele dan de variabele **K**inderen.

6.9 Taak

Vraag aan de gebruiker de lengte van een rechthoek.

Vraag daarna de breedte van de rechthoek.

Toon daarna de oppervlakte van de rechthoek (lengte x breedte).

Bewaar in `rechthoek.js`.

Je vindt achter in de cursus een voorbeeldoplossing.

7 IF

De gebruiker tikt een temperatuur.
Enkel als de temperatuur kleiner is of gelijk aan `0` toon je de tekst `Het vriest.`
en de tekst `Je doet beter een dikke jas aan.`

Blockly:

JavaScript:

```
"use strict";
const temperatuur = Number(prompt("Tik de temperatuur:"));
if (temperatuur <= 0) {                                          ❶
  console.log("Het vriest.");
  console.log("Je doet beter een dikke jas aan.");
}                                                                ❷
```

(1) Je tikt een if met **`if`** en een voorwaarde tussen **`()`**. Je tikt de opdrachten, die JavaScript enkel moet uitvoeren als de voorwaarde waar is, tussen **`{`** en **`}`**.
(2) Hier eindigt de if.

Je kan dit proberen.

7.1 Else

Als het niet vriest, toon je de tekst `Een lichte jas volstaat.`

Blockly:

JavaScript:

```
"use strict";
const temperatuur = Number(prompt("Tik de temperatuur:"));
if (temperatuur <= 0) {
  console.log("Het vriest.");
  console.log("Je doet beter een dikke jas aan.");
} else {                                                         ❶
  console.log("Een lichte jas volstaat.");
}                                                                ❷
```

(1) Je tikt een else met **`else`**. Je tikt de opdrachten, die tot de else behoren, tussen **`{`** en **`}`**.
(2) Hier eindigt de else.

Je kan dit proberen.

Opmerking: als een if of een else (of verder in de cursus een while of for) maar één opdracht bevat, moet je de de opdracht niet omsluiten met { en }.

JavaScript-experts raden echter aan *altijd* { en } te gebruiken, ook als er maar één opdracht is.

7.2 Taak

De prijs van een limonade is € 3.

Vraag de gebruiker hoeveel limonades hij koopt.

Toon daarna hoeveel hij in totaal moet betalen.
Als hij meer dan 10 limonades koopt, krijgt hij 5 % korting.
Voorbeeld 1: hij koopt 4 limonades, hij betaalt € 12.
Voorbeeld 2: hij koopt 15 limonades, hij betaalt € 42.75 (15 x 3 en daarop 5 % korting).

Bewaar in `limonades.js`.

Je vindt achter in de cursus een voorbeeldoplossing.

8 SWITCH

De gebruiker tikt de hoeveelste hij is in een loopwedstrijd.

- Als hij `1` tikt, toon je de tekst `Gouden medaille.`
- Als hij `2` tikt, toon je de tekst `Zilveren medaille.`
- Als hij `3` tikt, toon je de tekst `Bronzen medaille.`
- Anders toon je de tekst `Geen medaille.`

Blockly:

JavaScript:

```
"use strict";
const hoeveelste = Number(prompt("De hoeveelste ben je:"));
switch (hoeveelste) {                                          ❶
  case 1:                                                      ❷
    console.log("Gouden medaille.");                           ❸
    break;                                                     ❹
  case 2:
    console.log("Zilveren medaille.");
    break;
  case 3:
    console.log("Bronzen medaille.");
    break;
  default:                                                     ❺
    console.log("Geen medaille.");
}                                                              ❻
```

(1) Je tikt een switch met **`switch`**, de variabele die je wil controleren tussen `()` en `{`.
(2) Je tikt **`case`**, een eerste waarde die de variabele bij ❶ kan hebben en `:`
(3) Je tikt de opdrachten die JavaScript enkel moet uitvoeren als de variabele bij ❶ de waarde bij ❷ heeft.
(4) Je sluit die opdrachten af met **`break`**.
(5) JavaScript voert de opdrachten na **`default`** enkel uit als de variabele bij ❶ verschilt van de waarden vermeld bij alle **`case`** opdrachten.
(6) Hier eindigt de switch.

Als je na de opdrachten van een case `break` weglaat, voert JavaScript na die opdrachten ook de opdrachten uit van de volgende case. Je ziet dit als je de eerste `break` opdracht weglaat. Als je `1` intikt, zie je de teksten `Gouden medaille.` en `Zilveren medaille.`

8.1 Taak

Vraag een landcode aan de gebruiker.
Als hij `B` tikt toon je `België`. Als hij `NL` tikt toon je `Nederland`. Als hij `FR` tikt toon je `Frankrijk`. Anders toon je `Onbekend`.

Bewaar in `landcodes.js`.

Je vindt achter in de cursus een voorbeeldoplossing.

9 WHILE

Je houdt bij hoeveel geld de mensen storten voor een goed doel.
Telkens iemand een bedrag stort, tikt de gebruiker dit bedrag in en toon je de tussenopbrengst.
Als de actie voorbij is, mag je programma ook stoppen. We spreken af dat de gebruiker `0` tikt (waar hij normaal een gestort bedrag tikt) om aan te geven dat de actie voorbij is.

Blockly:

JavaScript:

```
"use strict";
let totaal = 0;
let gestort = Number(prompt("Tik een gestort bedrag of tik 0 om te stoppen"));
while (gestort !== 0) {                                                          ❶
  totaal += gestort;
  console.log(totaal);
  gestort = Number(prompt("Tik een gestort bedrag of tik 0 om te stoppen"));
}                                                                                ❷
console.log("Eindopbrengst:", totaal);
```

(1) Je tikt een while met **`while`** en een voorwaarde tussen **`()`**. Je tikt de opdrachten, die JavaScript moet uitvoeren zolang de voorwaarde waar is, tussen **`{`** en **`}`**.
(2) Hier eindigt de de while.

Je kan dit proberen.

9.1 Taak

De prijs van een limonade is € 3.

Vraag de gebruiker hoeveel limonades hij koopt.

Zolang hij een getal intikt dat kleiner is of gelijk aan 0,
laat je de gebruiker opnieuw intikken hoeveel limonades hij koopt.

Toon daarna hoeveel hij in totaal moet betalen.

Bewaar in `positief.js`.

Je vindt achter in de cursus een voorbeeldoplossing.

10 FOR

Je toont de getallen van `1` tot en met `10`.

Blockly:

JavaScript:

```
"use strict";
for (let teller = 1; teller <= 10; teller++) {                    ❶
  console.log(teller);
}                                                                 ❷
```

(1) Je tikt een for met **`for`**, drie stukken code tussen **`()`** en **`{`**. Je scheidt de stukken met **`;`**.

a. JavaScript voert het eerste stuk code één keer uit als de iteratie start:
`let teller = 1;`
Dit maakt een variabele teller en initialiseert die op `1`.

b. JavaScript voert het tweede stuk code uit vóór elke iteratie: `teller <= 10;`
Als de voorwaarde in dit stuk code waar is, voert JavaScript de iteratie nog eens uit.

c. JavaScript voert het derde stuk code uit op het einde van elke iteratie: `teller++;`

(2) Hier eindigt de for.

10.1 Taak

De gebruiker tikt een getal.
Jij toont de tafel van vermenigvuldiging van dit getal: `1` x dit getal, `2` x dit getal, ... `10` x dit getal.

Bewaar in `tafel.js`.

Je vindt achter in de cursus een voorbeeldoplossing.

11 NESTEN

11.1 Voorbeeld 1

De gebruiker tikt een temperatuur.
Als de temperatuur kleiner is of gelijk aan `0` zijn er nog twee mogelijkheden:
als de temperatuur kleiner is dan `-40` , toon je de tekst `Blijf thuis`.
Anders toon je de tekst `Doe een dikke jas aan`.

Blockly:

JavaScript:

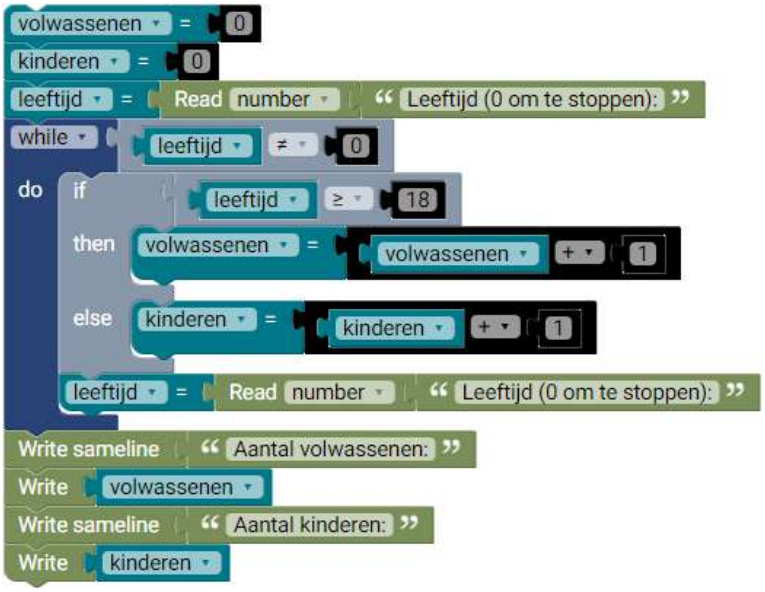
```
"use strict";
const temperatuur = Number(prompt("Tik de temperatuur:"));
if (temperatuur <= 0) {
  if (temperatuur < -40) {
    console.log("Blijf thuis.");
  } else {
    console.log("Doe een dikke jas aan.");
  }
}
```

Je kan dit uitproberen.

11.2 Voorbeeld 2

Een pretpark wil op het einde van een dag weten hoeveel volwassenen het park bezochten
en hoeveel kinderen het park bezochten. Je vraagt aan de gebruiker de leeftijd van elke bezoeker.
Als de leeftijd groter is of gelijk aan 18 verhoog je de variabele met het aantal volwassenen met `1` .
Anders verhoog je de variabele met het aantal kinderen met 1 .
Op het einde van de dag tikt de gebruiker `0` als leeftijd om het intikken te stoppen.
Je toont dan het aantal volwassenen en het aantal kinderen.

Blockly:



JavaScript:

```
"use strict";
let volwassenen = 0;
let kinderen = 0;
let leeftijd = Number(prompt("Leeftijd (0 om te stoppen):"));
while (leeftijd !== 0) {
  if (leeftijd >= 18) {
    volwassenen++;
  } else {
    kinderen++;
  }
  leeftijd = Number(prompt("Leeftijd (0 om te stoppen):"));
}
console.log("Aantal volwassenen:", volwassenen);
console.log("Aantal kinderen:", kinderen);
```

Je kan dit proberen.

11.3 Taak

Vraag aan de gebruiker of hij werk heeft.
Als hij `ja` antwoordt toon je de tekst `Veel werkplezier`.
Anders vraag je de gebruiker of hij een opleiding wil volgen.
Als hij `ja` antwoordt, toon je `Je vindt opleidingen op www.vdab.be/opleidingen.`
Anders toon je `Je vindt vacatures op www.vdab.be/jobs`.

Bewaar in `vdab.js`.

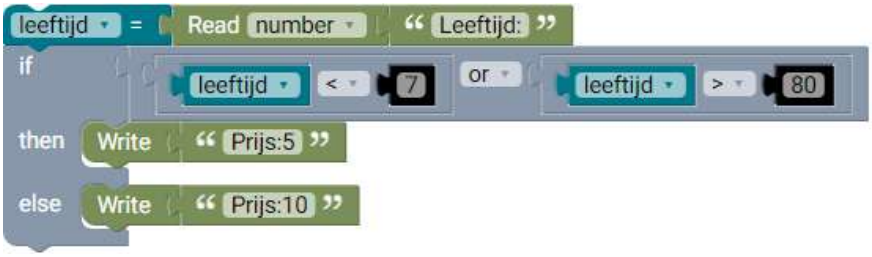
Je vindt achter in de cursus een voorbeeldoplossing.

12 OR EN AND

12.1 Or

De toegangsprijs in een pretpark is € 5 als je leeftijd kleiner is dan 7 ***of*** groter is dan 80.
Anders betaal je € 10.

Blockly:



JavaScript:

```
"use strict";
const leeftijd = Number(prompt("Leeftijd:"));
if (leeftijd < 7 || leeftijd > 80) {                    ❶
  console.log("Prijs:5");
} else {
  console.log("Prijs:10");
}
```

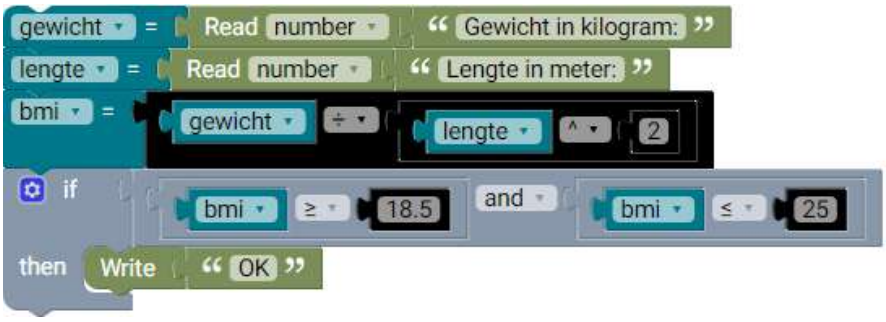
(1) **||** betekent or.

Je kan dit proberen.

12.2 And

Je maakt een programma dat je BMI (body mass index) berekent.
BMI is een berekening: gewicht gedeeld door kwadraat van de lengte.
Als je BMI ligt tussen 18.5 ***en*** 25 heb je een normaal gewicht.

Blockly:



JavaScript:

```
"use strict";
const gewicht = Number(prompt("Gewicht in kilogram:"));
const lengte = Number(prompt("Lengte in meter"));
const bmi = gewicht / (lengte * lengte);
if (bmi >= 18.5 && bmi <= 25) {                         ❶
  console.log("OK");
}
} else {
  console.log("Niet OK");
}
```

(1) **&&** betekent and.

Je kan dit proberen.

12.3 Taak

Maak een programma voor een pretpark.

De gebruiker van het programma tikt per bezoeker de leeftijd van die bezoeker.
De gebruiker tikt op het einde van de dag `0` als leeftijd om het intikken te stoppen.

Je toont dan het aantal bezoekers dat korting kreeg.
Een bezoeker krijgt korting als zijn leeftijd kleiner is dan `7` of groter is dan `80`.

Bewaar in `korting.js`.

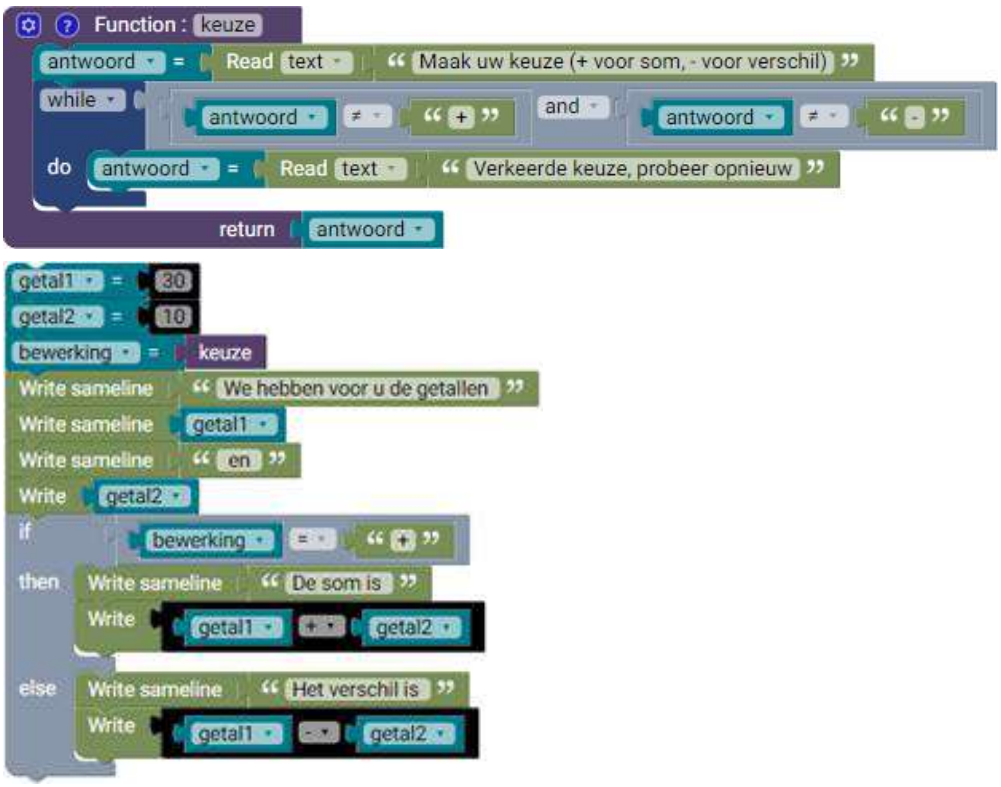
Je vindt achter in de cursus een voorbeeldoplossing.

13 FUNCTIES

JavaScript heeft geen procedures, enkel functies.
Als je een procedure wil maken, maak je een functie die geen waarde teruggeeft.

13.1 Voorbeeld 1

Blockly:



JavaScript:

```
"use strict";
function keuze() {                                                    ❶
  let antwoord = prompt("Maak uw keuze (+ voor som, - voor verschil)");  ❷
  while (antwoord !== "+" && antwoord !== "-") {
    antwoord = prompt("Verkeerde keuze, probeer opnieuw");
  }
  return antwoord;                                                    ❸
}                                                                     ❹
const getal1 = 30;
const getal2 = 10;
const bewerking = keuze();                                            ❺
console.log("We hebben voor u de getallen ", getal1, " en ", getal2);
if (bewerking === "+") {
  console.log("De som is ", getal1 + getal2);
} else {
  console.log("Het verschil is ", getal1 - getal2);
}
```

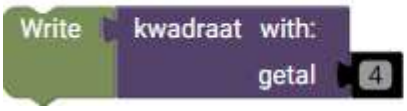
(1) Je maakt een functie met **`function`**, de naam van de functie, **`()`** en **`{`**.
(2) Je maakt de variabele antwoord in een functie. De variabele is enkel *in die functie* gekend.
Men noemt dit een *lokale* variabele.
JavaScript experts raden aan variabele zo lokaal mogelijk te maken:
   a. Als je een variabele enkel in een functie nodig hebt, maak je de variabele in de functie.
   b. Als je een variabele enkel in de **`{}`** van een **`if`**, **`while`** of **`for`** nodig hebt, maak je de variabele in de **{}**

(3) Je geeft een waarde van de functie terug met **`return`** en de waarde die je wil teruggeven.
(4) Hier eindigt de functie.
(5) Je roept de functie op door de naam van de functie te tikken en daaarna `()`.

Je kan dit proberen.

13.2 Voorbeeld 2 (functie met parameter):

Blockly:



JavaScript:

```
"use strict";
function kwadraat(getal) {                                    ❶
  return getal * getal;
}
console.log(kwadraat(4));                                     ❷
```

(1) Je vermeldt de naam van een parameter tussen de `()` van de functie.
Als je *meerdere* parameters nodig hebt, scheid je ze met een komma.
(2) Je geeft een waarde mee voor de parameter bij de oproep van de functie.

Je kan dit proberen.

13.3 Taak

Je maakt een functie met de naam `even`. De functie heeft een parameter met een getal.
De functie geeft `true` terug als dit getal even is. Anders geeft de functie `false` terug.

14 ARRAY

14.1 Voorbeeld 1

Je maakt een programma dat:

1. Een array-variabele `voornamen` maakt met `3` elementen: `Aeneas`, `Alissia` en `Anaïs`.
2. Het element met volgnummer `0` toont.
3. De inhoud van het element `0` op de waarde `Alexandra` plaatst.
4. Het element met volgnummer `0` terug toont.

Blockly:

JavaScript:

```
"use strict";
const voornamen = [ "Aeneas", "Alissia", "Anaïs" ];                ❶
console.log(voornamen[0]);                                         ❷
voornamen[0] = "Alexandra";                                        ❸
console.log(voornamen[0]);
```

(1) Je maakt een array met `[`, de elementen van de array en `]`.
Je scheidt de elementen met een komma.
(2) Je verwijst naar een element met de array naam, `[`, het volgnummer van het element en `]`.
(3) Je maakte de variabele `voornamen` met `const`. Je kan aan de variabele geen andere waarde toekennen. Je kan wel een element *in de array* wijzigen.

Je kan dit proberen.

Je kan alle elementen van de array overlopen met volgende code:

```
"use strict";
const voornamen = [ "Aeneas", "Alissia", "Anaïs" ];
for (let index = 0; index !== 3; index++) {
  console.log(voornamen[index]);
}
```

Je kan dit proberen.

De expressie `voornamen.length` geeft je het aantal elementen in de array:

```
"use strict";
const voornamen = [ "Aeneas", "Alissia", "Anaïs" ];
console.log("Aantal voornamen:", voornamen.length)
for (let index = 0; index !== voornamen.length; index++) {
  console.log(voornamen[index]);
}
```

Je kan dit proberen.

Je interpreteert `voornamen.length` als
de `length`-property (eigenschap) van de array in de variabele `voornamen`.

Je kan met **`for of`** elegant itereren over alle elementen van een array:

```
"use strict";
const voornamen = [ "Aeneas", "Alissia", "Anaïs" ];
for (const voornaam of voornamen) {                                ❶
  console.log(voornaam);
}
```

(1) Je tikt een variabelenaam (`voornaam`), **`of`** en de array waarover je wil itereren.
`voornaam` bevat bij elke iteratie het volgend element in de array.
De iteratie stopt automatisch als alle elementen doorlopen zijn.

Je kan **`for ..of`** niet gebruiken:

- als je van achter naar voor itereert over de elementen van een array.
- als je niet itereert over *alle* elementen,
  maar bijvoorbeeld enkel over de elementen met een even volgnummer.

Je kan dit proberen.

14.2 Voorbeeld 2

Je hebt in voorbeeld 1 een array gemaakt
en daarbij *onmiddellijk* de elementen van die array meegegeven:

```
const voornamen = [ "Aeneas", "Alissia", "Anaïs" ];
```

Je maakt in volgende voorbeeld een lege array. Je voegt *daarna* elementen toe:

```
"use strict";
const voornamen = [];                                    ❶
voornamen.push(prompt("Voornaam:"));                     ❷
voornamen.push(prompt("Voornaam:"));
for (const voornaam of voornamen) {
  console.log(voornaam);
}
```

(1) Je maakt een lege array.
(2) De gebruiker tikt een voornaam. Je voegt die voornaam toe aan de array
door op de array de `push`-method uit te voeren. Je tikt dit als
`naamVanDeArray.push(waardeDieJeWilAanDeArrayToevoegen);`
Een method is een opdracht (bewerking) die je op een array uitvoert.
De array bevat nu één element.

Je maakt bij ❶ de variabele `voornamen` met `const`.
Je kan daarna aan de variabele geen andere waarde meer toekennen (`voornamen=...`).
Je kan wel aan de array elementen toevoegen (zoals bij ❷), elementen wijzigen, ...

Je kan dit proberen.

14.3 Taak

Maak een programma.

De gebruiker tikt temperaturen tot hij `777` tikt.

Toon daarna enkel de vriestemperaturen (temperaturen <= `0`).

Bewaar in `vriestemperaturen.js`.

Je vindt achter in de cursus een voorbeeldoplossing.

15 OBJECT LITERAL

15.1 Algemeen

Je beschrijft met een object literal de eigenschappen (properties) van een ding of van een persoon.
Je beschrijft bijvoorbeeld de eigenschappen van een rechthoek.
De rechthoek heeft een lengte van `3`, een breedte van `2` en een `rode` kleur:

```
"use strict";
const rechthoek = { lengte : 3, breedte : 2, kleur : "rood"};          ❶
console.log(rechthoek.lengte);                                         ❷
rechthoek.lengte = 4;                                                  ❸
console.log(rechthoek.lengte);
```

(1) Je maakt een object literal als een verzameling properties tussen `{` en `}`.
Je vermeldt per property de naam van de property (bijvoorbeeld `lengte`), `:`
en de waarde van de property (bijvoorbeeld `3`). Je scheidt de properties met een komma.
(2) Je leest een property als
naamVanDeVariabeleDieEenObjectLiteralBevat.naamVandeOpGevraagdeProperty
(3) Je wijzigt een property als
naamVanDeVariabeleDieEenObjectLiteralBevat.naamVandeTeWijzigenProperty,
`=` en de nieuwe waarde voor die property.

Je kan dit proberen.

Je maakt bij ❶ de variabele `rechthoek` met `const`.
Je kan daarna aan de variabele geen andere waarde meer toekennen (`rechthoek=...`).
Je kan wel propertiesvan de object literal wijzigen, zoals bij❸.

Sommige programmeurs vinden de regel bij ❶ leesbaarder als:

```
const rechthoek = {
  lengte : 3,
  breedte : 2,
  kleur : "rood"
};                                                                     ❶
```

Je kan de waarden voor de properties ook opvragen aan de gebruiker:

```
"use strict";
const rechthoek = {
  lengte : Number(prompt("Lengte:")),
  breedte : Number(prompt("Breedte:")),
  kleur : prompt("Kleur:")
};
console.log(rechthoek.lengte, rechthoek.breedte, rechthoek.kleur);
```

Je kan dit proberen.

15.2 Array van object literals

Je maakt een een verzameling van 2 rechthoeken.
Je doet dit door een array te maken die 2 object literals bevat:

```
"use strict";
const rechthoeken = [                                                  ❶
  { lengte: 3, breedte: 2, kleur: "rood" },                            ❷
  { lengte: 7, breedte: 4, kleur: "groen" }                            ❸
];                                                                     ❹
for (const rechthoek of rechthoeken) {                                 ❺
  console.log(rechthoek.lengte, rechthoek.breedte, rechthoek.kleur);   ❻
}
```

(1) Je begint de array met `[`.
(2) Het eerste element is een object literal.
(3) Het tweede element is een object literal.

(4) Je sluit de array met `]`.
(5) Je itereert met de variabele `rechthoek` over de object literals in de array `rechthoeken`.
(6) Je toont de eigenschappen van de rechthoek.

Je kan dit proberen.

De gebruiker tikt in volgende code rechthoeken, tot hij als lengte `0` tikt.
Je onthoudt de rechthoeken in een array.

```
"use strict";
const rechthoeken = [];
let lengte = Number(prompt("Lengte:"));
while (lengte !== 0) {
rechthoeken.push({
  lengte: lengte,
  breedte: Number(prompt("Breedte:")),
  kleur: prompt("Kleur:") }),
  lengte = Number(prompt("Lengte:"))
}
for (const rechthoek of rechthoeken) {
  console.log(rechthoek.lengte, rechthoek.breedte, rechthoek.kleur);
}
```

Je kan dit proberen.

15.3 Property met object literal

Een property van een object literal kan zelf een object literal zijn:

```
"use strict";
const adres = {
  straat:"Keizerslaan",
  huisnummer:11,
  gemeente: {                          ❶
    postcode:1000,                     ❷
    naam:"Brussel"
  }
};
console.log(adres.gemeente.naam);      ❸
```

(1) De property `gemeente` is zelf een object literal.
(2) `postcode` is een property van `gemeente`.
(3) Je toont `naam` die een property is van `gemeente` die een property is van `adres`.

Je kan dit proberen.

15.4 Property met array

Een property van een object literal kan een array zijn:

```
"use strict";
const persoon = {
  familienaam: "Desmet",
  voornamen: ["Hans", "Cyriel"]
};
for (const voornaam of persoon.voornamen) {
  console.log(voornaam);
}
```

Je kan dit proberen.

15.5 Taak

Vraag van personen de naam, de lengte en het gewicht tot de gebruiker als naam `stop` tikt.

Toon daarna van de personen de naam en het BMI.
Het BMI is het gewicht gedeeld door kwadraat van de lengte.

Bewaar in `bmi.js`.

Je vindt achter in de cursus een voorbeeldoplossing.

16 STRING

String is het technische woord voor tekst.

Je leert hier interessante handelingen die je op een String kan uitvoeren.

16.1 Lengte

Een String is een object. De length property geeft je het aantal tekens in de String:

```
"use strict";
const naam = prompt("Naam:");
console.log("Je naam bevat", naam.length, "letters.");
```

16.2 Letter uit een String lezen

Een String lijkt op een array. Elk element in de array is één teken van de String.
Het eerste teken heeft het volgnummer `0`. Je leest met de syntax `[]` een letter uit een String:

```
"use strict";
const naam = prompt("Naam:");
console.log("De eerste letter in je naam is", naam[0]);                ❶
```

(1) Je leest de eerste letter (met het volgnummer `0`).

Je toont dus met volgende code één per één de letters in een String:

```
"use strict";
const naam = prompt("Naam:");
for (let volgnummer = 0; volgnummer !== naam.length; volgnummer++) {   ❶
  console.log(naam[volgnummer]);
}
```

(1) De variabele `volgnummer` itereert over de volgnummers van de tekens.
Je initialiseert `volgnummer` op het volgnummer van het eerste teken: 0.
Je verhoogt `volgnummer` bij elke iteratie.
De iteratie gaat door zolang `volgnummer` verschilt van het aantal tekens.
Als `naam` `Jan` bevat, is de `length` gelijk aan `3` en zijn de volgnummer `0`, `1` en `2`.

Zoals bij array's is de for of syntax gemakkelijker en leesbaarder om één per één de letters te lezen:

```
"use strict";
const naam = prompt("Naam:");
for (const letter of naam) {                                           ❶
  console.log(letter);
}
```

(1) Je tikt een vrij te kiezen variabelenaam (`letter`). Je tikt daarna **`of`**.
Je tikt daarna de String waarover je letter per letter wil itereren.
Tijdens de for-lus verwijst `letter` bij elke iteratie naar het volgende teken in de String.
De iteratie stopt automatisch als alle tekens doorlopen zijn.

16.3 Concateneren

Concateneren betekent String samenvoegen, "aan mekaar kleven". Je doet dit met het **+** teken:

```
"use strict";
const voornaam = prompt("Voornaam:");
const familienaam = prompt("Familienaam:");
const naam = voornaam + " " + familienaam;                             ❶
console.log(naam);
```

(2) Je kleeft de voornaam, een spatie en de familienaam aan mekaar tot een nieuwe String.

16.4 Template literal

Code om te concateneren is zelden leesbaar. Een alternatief is een template literal.
Dit is een string waarin je kan verwijzen naar variabelen tussen ${ en }.

```
"use strict";
const voornaam = prompt("Voornaam:");
const familienaam = prompt("Familienaam:");
const naam = `${voornaam} ${familienaam}`;                    ❶
console.log(naam);
```

(1) Je omsluit een template literal met ` en `, niet met " en ".
Programmeurs noemen het teken ` ook BackTick.
Je maakt dit teken op een Azerty toetsenbord met de combinatie van `AltGr` en de toets


.

(2) Je verwijst in de template literal naar de variabele `voornaam` met `${voornaam}`.
JavaScript vervangt die verwijzing met de *inhoud* van de variabele `voornaam`.
Als voornaam `Hans` bevat en familienaam `Desmet`, bevat naam dus `Hans Desmet`.

Je kan tussen ${ en } ook een berekening schrijven:

```
"use strict";
const getal1 = 7;
const getal2 = 3;
console.log(`De som van ${getal1} en ${getal2} is ${getal1 + getal2}`);
```

16.5 Methods om een String te onderzoeken

Strings hebben veel methods (bewerkingen die je op Strings uitvoert).

Je leert hier methods kennen waarmee je onderzoekt of een String aan een voorwaarde voldoet.
De methods geven `true` terug als de String aan de voorwaarde voldoet.
Anders geven de methods `false` terug.

```
"use strict";
const voornaam = "Alexandra";
console.log(voornaam.includes("dr"));                         ❶
console.log(voornaam.endsWith("dra"));                        ❷
console.log(voornaam.startsWith("An"));                       ❸
```

(1) Dit geeft `true`: `Alexandra` bevat (to include) `dr`.
(2) Dit geeft `true`: `Alexandra` eindigt met (ends with) `dra`.
(3) Dit geeft `false`: `Alexandra` begint niet met (start with) `An`.

String bevatten ook methods waarmee je onderzoekt
op de *hoeveelste* plaats een String voorkomt in een andere String:

```
"use strict";
const spreuk = "De juiste tijd kiezen is tijd besparen";
console.log(spreuk.indexOf("tijd"));                          ❶
console.log(spreuk.lastIndexOf("tijd"));                      ❷
```

(1) Dit geeft `10` want `tijd` komt voor het eerst op de `10`° positie voor in de spreuk.
(2) Dit geeft `25` want `tijd` komt voor het laatst op de `25`° positie voor in de spreuk.

16.6 Methods die nieuwe Strings teruggeven

Sommige methods geven een *nieuwe* String terug, gebaseerd op de String waarop je de method uitvoert. De String waarop je de method uitvoert blijft ongewijzigd.

```
"use strict";
const voornaam = "Alexandra";
console.log(voornaam.toUpperCase());                    ❶
console.log(voornaam.toLowerCase());                    ❷
console.log(voornaam.slice(2));                         ❸
console.log(voornaam.slice(2, 6));                      ❹
console.log(voornaam.slice(-2));                        ❺
console.log(voornaam.replace("x", "ks"));               ❻
console.log(voornaam);                                  ❼
const metSpaties = "   Alexandra   ";
console.log(`>${metSpaties.trim()}<`);                  ❽
console.log(`>${metSpaties.trimStart()}<`);             ❾
console.log(`>${metSpaties.trimEnd()}<`);               ❿
```

(1) Dit geeft `ALEXANDRA`: `Alexandra` in hoofdletters.
(2) Dit geeft `alexandra`: `Alexandra` in kleine letters.
(3) Dit heeft `exandra`: alle letters vanaf de positie met volgnummer `2`.
(4) Dit heeft `exan`: de letters vanaf de positie `2` tot voor de positie `6`.
(5) Dit heeft `ra`: de laatste `2` letters.
(6) Dit geeft `aleksandra`: `Alexandra` met de `x` vervangen door `ks`.
(7) Alle methods gaven een *nieuwe* String terug. De oorspronkelijke String is ongewijzigd.
(8) Dit geeft `>Alexandra<`: de spaties vooraan én achteraan zijn verwijderd.
(9) Dit geeft `>Alexandra   <`: de spaties vooraan zijn verwijderd.
(10) Dit geeft `>   Alexandra<`: de spaties achteraan zijn verwijderd.

16.7 Taak

De gebruiker tikt een woord.
De gebruiker tikt daarna een letter.
Je toont hoeveel keer de letter in het woord voorkomt.
Bij het woord `programmeur` en de letter `r` toont je: `r komt 3 keer voor in programmeur.`

Bewaar in `letterInWoord.js`.

Je vindt achter in de cursus een voorbeeldoplossing.

16.8 Taak

De gebruiker tikt een woord. Je zegt of dit woord een palindroom is.
Een palindroom is een woord dat van achter naar voor gelezen hetzelfde is als van voor naar achter. Een voorbeeld van een palindroom: `lepel`.

Bewaar in `palindroom.js`.

Je vindt achter in de cursus een voorbeeldoplossing.

17 TOFIXED

Als je rekent met getallen met decimalen (cijfers na de komma), kan je afrondingsfoutjes krijgen.

Je ziet dit met volgende code:

```
"use strict";
const prijsVanEenAppel = 0.34;
const aantalAppels = 7;
const teBetalen = prijsVanEenAppel * aantalAppels;
console.log(teBetalen);
```

Je probeert dit. Je krijgt `2.3800000000000003`.

Je lost dit op. Je wijzigt de laatste opdracht:

```
console.log(teBetalen.toFixed(2));                                    ❶
```

(1) Je kan op een getal de method `toFixed` uitvoeren. `toFixed` geeft een String terug.
De String bevat een afgeronde versie van het getal.
Je bepaalt met de parameter van `toFixed` (hier `2`) het aantal decimalen waarop je afrondt.

Als je met het resultaat van `toFixed` wil verder rekenen,
moet je de String die `toFixed` teruggeeft omzetten naar een getal.

```
"use strict";
const prijsVanEenAppel = 0.34;
const aantalAppels = 7;
let teBetalen = prijsVanEenAppel * aantalAppels;
teBetalen = Number(teBetalen.toFixed(2));                             ❶
console.log(teBetalen);
const prijsVanEenPeer = 0.32;
const aantalPeren = 9;
teBetalen += prijsVanEenPeer * aantalPeren;                           ❷
console.log(teBetalen.toFixed(2));
```

(1) Je zet met de `Number` functie de String met de afgeronde waarde om naar een getal.
(2) Je kan met dit getal verder rekenen.

18 LAMBDA

Een lambda is een compacte manier om een functie te schrijven:

```
"use strict";
const kwadraat = getal => getal * getal;                          ❶
console.log(kwadraat(4));                                         ❷
const som = (getal1, getal2) => getal1 + getal2;                  ❸
console.log(som(4, 2));
const verschil = (getal1, getal2) =>
  { const hetVerschil = getal1 - getal2; return hetVerschil; };   ❹
console.log(verschil(5, 2));
```

(1) Het deel na **=** is de eigenlijke lambda. We kijken eerst naar die eigenlijke lambda.
De voorbeeld lambda heeft 1 parameter: `getal`. Je tikt de parameter voor **=>**.
Je tikt na **=>** hetgeen de lambda teruggeeft. Het keyword `return` is overbodig.
Je onthoud met `kwadraat =` de lambda in een variabele kwadraat.
Je doet dit om de lambda te kunnen oproepen. Je doet de oproep bij ❷.

(2) De variabele `kwadraat` bevat een lambda. Je roept de lambda op.
Je geeft 4 mee als parameter. Je toont de waarde die de lambda teruggeeft (16).

(3) De lambda heeft *meerdere* parameters. Je tikt dan de parameternamen tussen `(` en `)` voor `=>`.

(4) Als de lambda meerdere opdrachten bevat, tik je die opdrachten tussen `{` en `}`.
Als zo'n lambda een waarde teruggeeft doe je dit met het `return` keyword.

Je kan dit proberen.

Een synoniem voor een lambda is een arrow function (omdat in een lambda `=>` voorkomt).

18.1 forEach

Je leerde al dat je met de syntax `for .. of` kan itereren over de elementen van een array.

Je leert hier een andere manier: de forEach method:

```
"use strict";
const getallen = [3, 7];
getallen.forEach(getal => console.log(getal));                    ❶
```

(1) Een array heeft een method `forEach`. Je geeft aan `forEach` een lambda mee.
`forEach` roept die lambda op per element uit de array.
`forEach` geeft bij elke oproep het element aan de lambda als de parameter van de lambda.

Je kan dit proberen.

18.2 filter

Een array heeft een method `filter`. `filter` geeft een nieuwe array terug.
De nieuwe array bevat elementen uit de array waarop je `filter` opriep.
Elementen komen enkel in de nieuwe array als ze voldoen aan een voorwaarde.
Jij definieert die voorwaarde als een lambda die je meegeeft aan `filter`.

Voorbeeld: je maakt een array met de positieve getallen uit een oorspronkelijke array:

```
"use strict";
const getallen = [-1, 2, 3, 4, 0];
const positieveGetallen = getallen.filter(getal => getal > 0);    ❶
positieveGetallen.forEach(getal => console.log(getal));
```

(1) Je geeft aan `filter` een lambda mee. `filter` roept die lambda op per element uit de array.
Als de lambda true teruggeeft, neemt `filter` het element op in een nieuwe array.
Nadat `filter` alle elementen uit de oorspronkelijke array heeft doorlopen,
geeft `filter` de nieuwe array terug.

Je kan dit proberen.

Als je doel enkel is de positieve getallen te tonen, kan de code korter:

```
"use strict";
[-1, 2, 3, 4, 0].filter(getal => getal > 0)
  .forEach(getal => console.log(getal));
```

Je kan dit proberen.

`filter` werkt natuurlijk ook op een array met object literals.
De volgende code toont enkel de personen die `2` kinderen hebben:

```
"use strict";
[ { naam: "Hans", kinderen: 3 },
  { naam: "Hendrik", kinderen: 2 },
  { naam: "Gerard", kinderen: 2 },
  { naam: "José", kinderen: 2 }
].filter(persoon => persoon.kinderen === 2)
 .forEach(persoon => console.log(`${persoon.naam}:${persoon.kinderen}`));
```

Je kan dit proberen.

18.3 map

Een array heeft een method `map`. `map` geeft een nieuwe array terug.
De nieuwe array bevat *evenveel* elementen als de array waarop je `map` opriep. Elk element in de nieuwe array is het resultaat van een bewerking die je doet op het bijbehorende element uit de oorspronkelijke array. Jij definieert die bewerking als een lambda die je meegeeft aan `map`.

Voorbeeld: je maakt een array met getallen die de kwadraten zijn
van getallen uit een oorspronkelijke array:

```
"use strict";
[2, 4, 5].map(getal => getal * getal)                    ❶
  .forEach(kwadraat => console.log(kwadraat));           ❷
```

(1) Je geeft aan `map` een lambda mee. `map` roept die lambda op per array element.
`map` voegt de return waarde van de lambda toe aan een nieuwe array.
Nadat `map` alle elementen uit de oorspronkelijke array heeft doorlopen,
geeft `map` de nieuwe array terug. Die bevat de waarden `4`, `16` en `25`.
(2) Je itereert over de nieuwe array en je toont elk element.

Je kan dit proberen.

Voorbeeld: Je hebt een array met rechthoeken.
Je maakt op basis van die array een nieuwe array met de oppervlakten van die rechthoeken:

```
"use strict";
[
  { lengte: 3, breedte: 2 },
  { lengte: 5, breedte: 4 }
] .map(rechthoek => rechthoek.lengte * rechthoek.breedte)   ❶
  .forEach(oppervlakte => console.log(oppervlakte));
```

(1) map geeft een array terug met de waarden `6` en `20`
(de oppervlakten van de rechthoeken uit de oorspronkelijke array).

Je kan dit proberen.

18.4 reduce

Een array heeft een method `reduce`. `reduce` geeft één waarde terug.
Die waarde is een “samenvatting” (som, minimum, ...) van de elementen in de array.
Jij definieert hoe je tot die samenvatting komt als een lambda die je meegeeft aan `reduce`.

Voorbeeld: je maakt de som van de getallen in een array:

```
"use strict";
console.log([2, 5, 8, 11].reduce((som, getal) => som + getal));   ❶
```

(1) `reduce` roept de lambda op en geeft het 1° en 2° element van de array mee als parameters.
Jij geeft de som van de twee getallen (`7`) terug.
`reduce` roept de lambda terug op en geeft de return waarde van de vorige oproep (`7`)
en het 3° element (`8`) mee als parameters. Jij geeft de som (`15`) terug.
`reduce` roept de lambda terug op en geeft de return waarde van de vorige oproep (`15`)
en het 4° element (`11`) mee als parameters. Jij geeft de som (`26`) terug.
Dit (`26`) is uiteindelijk de som van alle getallen.

Je kan dit proberen.

Voorbeeld: je bepaalt het kleinste getal in een array met getallen:

```
"use strict";
console.log([5, 8, 2, 11].reduce(
(kleinste, getal)=> {if (kleinste < getal) return kleinste; return getal;})); ❶
```

(1) `reduce` roept de lambda op en geeft het 1° en 2° element van de array mee als parameters.
Jij geeft de kleinste van de twee (`5`) terug.
`reduce` roept de lambda terug op en geeft de return waarde van de vorige oproep (`5`)
en het 3° element (`2`) mee als parameters. Jij geeft de kleinste (`2`) terug.
`reduce` roept de lambda terug op en geeft de return waarde van de vorige oproep (`2`)
en het 4° element (`11`) mee als parameters. Jij geeft de kleinste (`2`) terug.
Dit (`2`) is uiteindelijk het kleinste getal.

Je kan dit proberen.

Voorbeeld van een *combinatie* van map en reduce.
Je maakt de som van de oppervlakten van rechthoeken:

```
console.log(
  [
    { lengte: 3, breedte: 2 },
    { lengte: 5, breedte: 4 }
  ].map(rechthoek => rechthoek.lengte * rechthoek.breedte)
  .reduce((som, oppervlakte) => som+ oppervlakte));
```

Je kan dit proberen.

18.5 Vergelijking

Je kan filter, map en reduce vergelijken met handelingen die je doet op aardappelen:

- Filter: Je houdt van aardappelen enkel de gezonde aardappelen over.
- Map: Je schilt elke aardappel.
  Je transformeert daarbij elke niet-geschilde aardappel tot een geschilde aardappel.
- Reduce: Je maakt van een verzameling aardappelen één ding: aardappelpuree.

Je kan op een verzameling aardappels die je kocht in de winkel *alle 3* de handelingen doen.
Dit is een voorbeeld van de combinatie van filter, map en reduce.

18.6 Taak

Je maakt een array met personen. Elke persoon heeft een naam en spaargeld.
Je houdt enkel de personen over met meer dan `€ 1000` spaargeld.
Je toont de namen en het spaargeld van de personen met meer dan `€ 1000` spaargeld.

Bewaar in `spaargeld.js`.

Je vindt achter in de cursus een voorbeeldoplossing.

19 CLASS

19.1 Algemeen

Je leerde een object (iets of iemand uit de werkelijkheid) voor te stellen met een object literal.

Je leert hier een andere manier kennen om een object voor te stellen: een class.
Een class is geen *concreet* object, maar een *sjabloon* van een object.
De class `Rechthoek` is niet de concrete rechthoek met lengte `5` en breedte `3`
en is niet de concrete rechthoek met lengte `7` en breedte `4`.
De class `Rechthoek` is het sjabloon van om het even welke rechthoek.
De class `Rechthoek` definieert dat een rechthoek een lengte en een breedte heeft,
maar definieert niet de concrete waarde (bvb. `7`) voor de lengte of voor de breedte.

Je geeft de concrete waarden mee als je een object maakt op basis van de class (zie verder).

19.2 Class maken, objecten maken

Je maakt een class `Rechthoek`.
Je definieert nog niet dat die de properties `lengte` en `breedte` heeft. Je doet dat later.

```
"use strict";
class Rechthoek {                                          ❶
}
const rechthoek = new Rechthoek();                         ❷
const nogEenRechthoek = new Rechthoek();                   ❸
```

(1) Je maakt een class met het keyword `class` en de naam van de class.
Je omsluit de code van de class (dit komt later) met accolades.
(2) Je maakt een object op basis van een class met het keyword `new` en de naam van de class.
Je onthoudt dit object in de variabele `rechthoek`.
(3) Je maakt een tweede object op basis van de class `Rechthoek`.

19.3 Constructor

Een constructor is een onderdeel van een class. JavaScript roept de constructor op
telkens je een object van de class maakt met het keyword `new`. Je ziet dit als volgt:

```
"use strict";
class Rechthoek {
  constructor() {                                          ❶
    console.log("Een Rechthoek object wordt gemaakt");
  }
}
const rechthoek = new Rechthoek();                         ❷
const nogEenRechthoek = new Rechthoek();                   ❸
```

(1) Een constructor heeft de syntax van een functie.
De naam van deze “functie” moet het keyword `constructor` zijn.
(2) Je maakt een object van de class `Rechthoek`. JavaScript roept daarbij de constructor op.
(3) Je maakt nog een object van de class `Rechthoek`. JavaScript roept daarbij de constructor op.

Je voert het programma uit. Je ziet twee keer `Een Rechthoek object wordt gemaakt.`

Volgende stap: We geven elke rechthoek twee properties: `lengte` en `breedte`.
Voorlopig is de `lengte` van elke rechthoek initieel `5` en is de `breedte` van elke rechthoek initieel `3`.

```
"use strict";
class Rechthoek {
  constructor() {
    this.lengte = 5;                                       ❶
    this.breedte = 3;
  }
}
```

```
const rechthoek = new Rechthoek();                                    ❷
console.log(rechthoek.lengte);                                        ❸
console.log(rechthoek.breedte);
const nogEenRechthoek = new Rechthoek();
console.log(nogEenRechthoek.lengte);
console.log(nogEenRechthoek.breedte);
nogEenRechthoek.lengte = 7;                                           ❹
console.log(nogEenRechthoek.lengte);                                  ❺
console.log(rechthoek.lengte);                                        ❻
```

(1) Het keyword `this` staat voor het "huidig" object. In een constructor betekent dit: het object dat je aan het maken bent. Je maakt een property in dit object met het keyword `this`, een punt, de naam van de property en een beginwaarde voor de property.
(2) Je maakt een `Rechthoek` object. JavaScript roept daarbij de constructor op. Deze geeft het object een property `lengte` met de waarde `5` en een property `breedte` met de waarde `3`.
(3) Je vraagt de `lengte` property. Je krijgt `5`.
(4) Je wijzigt de `lengte` property naar `7`.
(5) Je vraagt de `lengte` property. Je krijgt `7`.
(6) De `lengte` property van het andere `Rechthoek` object bevat nog `5`.

19.4 Constructor met parameters

Het is onhandig dat elke rechthoek initieel een `lengte` van `5` en een `breedte` van `3` heeft.
Je wijzigt de constructor. Je geeft bij het aanmaken van een rechthoek de lengte en de breedte van die rechthoek mee. Je maakt bijvoorbeeld een rechthoek met een lengte van `7` en een breedte van `4` als `new Rechthoek(7, 4)`

```
"use strict";
class Rechthoek {
  constructor(lengte, breedte) {                                      ❶
    this.lengte = lengte;                                             ❷
    this.breedte = breedte;                                           ❸
  }
}
const rechthoek = new Rechthoek(5, 3);                                ❹
console.log(rechthoek.lengte);
console.log(rechthoek.breedte);
const nogEenRechthoek = new Rechthoek(7, 4);
console.log(nogEenRechthoek.lengte);
console.log(nogEenRechthoek.breedte);
```

(1) Je maakt bij ❹ een voorbeeld object. Je ziet dat je een waarde `5` meegeeft voor de `lengte` en een waarde `3` voor de `breedte`. Je voegt aan de constructor bijbehorende parameters toe. JavaScript roept bij ❹ de constructor op.
De parameter `lengte` bevat `5`. De parameter `breedte` bevat `3`.
(2) Je brengt de inhoud van de parameter `lengte` over naar de property `lengte`.
(3) Je brengt de inhoud van de parameter `breedte` over naar de property `breedte`.

19.5 Method

Je kan aan een class ook methods toevoegen. Dit zijn handelingen die objecten kunnen uitvoeren.
Je voegt een method verdubbel toe. Die verdubbelt de lengte en de breedte van een rechthoek.
Je voegt de method toe na de constructor:

```
verdubbel() {                                                         ❶
  this.lengte *= 2;                                                   ❷
  this.breedte *= 2;
}
```

(1) Een method heeft de syntax van een functie.
(2) Je verdubbelt de inhoud van de property `lengte` van het huidig object (`this`).

Je kan een `Rechthoek` object maken en die rechthoek verdubbelen:

```
const rechthoek = new Rechthoek(5, 3);
rechthoek.verdubbel();                                    ❶
console.log(rechthoek.lengte);
console.log(rechthoek.breedte);
```

(1) Je voert de method uit op het `Rechthoek` object in de variabele `rechthoek`.

19.6 Method met parameters

Een method kan parameters hebben.
Je voegt een method `rekUit` toe. Die heeft een parameter `factor`. Je rekt de lengte en de breedte van de rechthoek uit met die factor. Je voegt de method toe na de method `verdubbel`:

```
rekUit(factor) {
  this.lengte *= factor;
  this.breedte*= factor;
}
```

Je kan een `Rechthoek` object maken en die rechthoek uitrekken:

```
const rechthoek = new Rechthoek(5, 3);
rechthoek.rekUit(3);
console.log(rechthoek.lengte);
console.log(rechthoek.breedte);
```

19.7 Method met return waarde

Een method kan bij zijn oproep (uitvoering) een waarde teruggeven.
Je voegt een method `oppervlakte` toe. Deze geeft de oppervlakte van de rechthoek terug.
Je voegt de method toe na de method `rekUit`.

```
oppervlakte() {
  return this.lengte * this.breedte;
}
```

Je kan een `Rechthoek` object maken en die oppervlakte van die rechthoek vragen:

```
const rechthoek = new Rechthoek(5, 3);
console.log(rechthoek.oppervlakte());
```

19.8 Ander voorbeeld

```
"use strict";
class GoedDoel {                                          ❶
  constructor(naam) {                                     ❷
    this.naam = naam;
    this.gestorteBedragen = [];                           ❸
  }
  stort(bedrag) {                                         ❹
    this.gestorteBedragen.push(bedrag);                   ❺
  }
  opbrengst() {                                           ❻
    if (this.gestorteBedragen.length === 0) {             ❼
      return 0;                                           ❽
    }
    return this.gestorteBedragen.reduce(                  ❾
      (totaal, bedrag) => totaal + bedrag);
  }
}
const doel = new GoedDoel("Kom op tegen kanker");
doel.stort(10);
doel.stort(20);
doel.stort(50);
doel.gestorteBedragen.forEach(bedrag => console.log(bedrag));
console.log(doel.opbrengst());
```

(1) Deze class stelt een goed doel voor, zoals “Kom op tegen kanker” of “Cliniclowns”.
(2) Als je een goed doel maakt, geef je de naam (bvb. Cliniclowns) van dat goed doel mee.
(3) Je houdt in een array de bedragen bij die gestort worden voor het goed doel.
(4) Je stort met deze method een bedrag voor het goed doel.
(5) Je voegt het bedrag toe aan de array met alle gestorte bedragen.
(6) Je vraagt met deze method de totale opbrengst van het goed doel.
(7) Als er nog geen bedrag gestort is
(8) is de opbrengst nul.
(9) Anders bereken je het totaal van alle gestorte bedragen.

19.9 Taak

Je maakt een class `Artikel`.
Als je een artikel maakt, geef je de naam mee, de prijs exclusief BTW en het BTW %.
De class bevat een method `prijsInclusiefBTW`. Deze geeft de prijs inclusief BTW terug.

Je maakt van deze class een object.
Je toont de prijs inclusief BTW van dit object.

Bewaar in `artikel.js`.

Je vindt achter in de cursus een voorbeeldoplossing.

19.10 Taak

Je maakt een class `Persoon`.
Als je een persoon maakt, geef je de familienaam mee.
De class bevat een method `voornaam` met een voornaam als parameter.
Je voegt daarmee een voornaam toe (een persoon kan meerdere voornamen hebben).
De class bevat een method `naam`. Deze geeft de volledige naam terug: alle voornamen aan mekaar gekleefd, gescheiden door een spatie en daarna de familienaam.

Je maakt van deze class een object.
Je voegt enkele voornamen toe.
Je toont de volledige naam

Bewaar in `persoon.js`.

Je vindt achter in de cursus een voorbeeldoplossing.

20 EINDTAKEN

20.1 Landen

De gebruiker tikt per land de naam en de oppervlakte.
Hij herhaalt dit tot hij als naam `stop` tikt.

Je toont daarna de gemiddelde oppervlakte van de landen.

Je toont daarna de landen met een oppervlakte onder dit gemiddelde.
Je toont per land de naam en de oppervlakte.

Je toont daarna de landen met een oppervlakte vanaf dit gemiddelde.
Je toont per land de naam en de oppervlakte.

Bewaar in `landen.js`.

Je vindt achter in de cursus een voorbeeldoplossing.

20.2 Personen

De gebruiker tikt per persoon de naam en de lengte.
Hij herhaalt dit tot hij als naam `stop` tikt.

Je toont daarna de kleinste lengte van de personen.

Je toont daarna de personen met een lengte gelijk aan de kleinste lengte.
Je toont per persoon de naam en de lengte.

Je toont daarna de grootste lengte van de personen.

Je toont daarna de personen met een lengte gelijk aan de grootste lengte.
Je toont per persoon de naam en de lengte.

Bewaar in `personen.js`.

Je vindt achter in de cursus een voorbeeldoplossing.

21 VOORBEELDOPLOSSINGEN

21.1 programmeur

```
console.log("Ik");
console.log("word");
console.log("programmeur.");
```

21.2 vermenigvuldigen

```
console.log(3 x 4);
```

21.3 kleinerOfGelijkAan

```
console.log(4.6 <= 5);
```

21.4 rechthoek

```
"use strict";
const lengte = Number(prompt("Lengte:"));
const breedte = Number(prompt("Breedte:"));
console.log(lengte * breedte);
```

21.5 limonades

```
"use strict";
const aantal = Number(prompt("Aantal limonades:"));
let totaal = 3 * aantal;
if (aantal > 10) {
  totaal *= 0.95;
}
console.log(totaal);
```

21.6 landcodes

```
"use strict";
const code = prompt("Landcode:");
switch (code) {
  case "B":
    console.log("België");
    break;
  case "NL":
    console.log("Nederland");
    break;
  case "FR":
    console.log("Frankrijk");
    break;
  default:
    console.log("Onbekend")
}
```

21.7 positief

```
"use strict";
let aantal = Number(prompt("Aantal limonades:"));
while (aantal <= 0) {
  aantal = Number(prompt("Aantal limonades:"));
}
console.log(aantal * 3);
```

21.8 tafel

```
"use strict";
const getal = Number(prompt("Getal:"));
for (let getal2 = 1; getal2 <= 10; getal2++) {
  console.log(getal * getal2);
}
```

21.9 vdab

```
"use strict";
const werk = prompt("Heb je werk (ja/nee):");
if (werk === "ja") {
  console.log("Veel werkplezier.");
  } else {
  const opleiding = prompt("Wil je een opleiding volgen (ja/nee):");
  if (opleiding === "ja") {
    console.log("Je vindt opleidingen op www.vdab.be/opleidingen.");
  } else {
    console.log("Je vindt vacatures op www.vdab.be/jobs.");
  }
}
```

21.10 korting

```
"use strict";
let korting = 0;
let leeftijd = Number(prompt("Leeftijd (0 om te stoppen):"));
while (leeftijd !== 0) {
  if (leeftijd < 7 || leeftijd > 80) {
    korting++;
  }
  leeftijd = Number(prompt("Leeftijd (0 om te stoppen):"));
}
console.log("Aantal bezoekers met korting:", korting);
```

21.11 even

```
"use strict";
function even(getal) {
  return getal % 2 === 0;
}
console.log(even(4), even(5));
```

21.12 vriestemperaturen

```
"use strict";
const vriesTemperaturen = [];
let temperatuur = Number(prompt("Temperatuur:"));
while (temperatuur !== 777) {
  if (temperatuur <= 0) {
    vriesTemperaturen.push(temperatuur);
  }
  temperatuur = Number(prompt("Temperatuur:"));
}
for (const temperatuur of vriesTemperaturen) {
    console.log(temperatuur);
}
```

21.13 BMI

```
"use strict";
const personen = [];
let naam = prompt("naam:");
while (naam !== "stop") {
  personen.push({
    naam: naam,
    gewicht: Number(prompt("gewicht:")),
    lengte: Number(prompt("lengte:"))
  })
  naam = prompt("naam:");
}
```

```
for (const persoon of personen) {
  console.log(persoon.naam, persoon.gewicht/(persoon.lengte*persoon.lengte));
}
```

21.14 letterInWoord

```
"use strict";
const woord = prompt("Woord:");
const zoekLetter = prompt("Letter:");
let aantal = 0;
for (const letter of woord) {
  if (letter === zoekLetter) {
    aantal++;
  }
}
console.log(`${zoekLetter} komt ${aantal} keer voor in ${woord}`);
```

21.15 palindroom

```
"use strict";
const woord = prompt("Woord:");
let palindroom = true;
for (let volgnummer=0;volgnummer<woord.length / 2 && palindroom;volgnummer++) {
  if (woord[volgnummer] !== woord[woord.length - volgnummer - 1]) {
    palindroom = false;
  }
}
if (palindroom) {
  console.log(`${woord} is een palindroom.`)
} else {
  console.log(`${woord} is geen palindroom.`)
}
```

21.16 spaargeld

```
"use strict";
[
  { naam: "Georges", spaargeld: 1500 },
  { naam: "Jules", spaargeld: 700 },
  { naam: "Jacques", spaargeld: 2000 }
].filter(persoon => persoon.spaargeld > 1000)
 .forEach(persoon => console.log(persoon.naam, persoon.spaargeld));
```

21.17 artikel

```
"use strict";
class Artikel {
  constructor(naam, prijsExclusiefBTW, BTWPercentage) {
    this.naam = naam;
    this.prijsExclusiefBTW = prijsExclusiefBTW;
    this.BTWPercentage = BTWPercentage;
  }
  prijsInclusiefBTW() {
    return this.prijsExclusiefBTW * (1 + this.BTWPercentage / 100);
  }
}
const smartphone = new Artikel("Smartphone", 200, 19);
console.log(smartphone.prijsInclusiefBTW());
```

21.18 persoon

```
"use strict";
class Persoon {
  constructor(familienaam) {
    this.familienaam = familienaam;
    this.voornamen = [];
  }
  voornaam(voornaam) {
    this.voornamen.push(voornaam);
  }
  naam() {
    let naam = "";
    this.voornamen.forEach(voornaam => naam += voornaam + " ");
    return naam + this.familienaam;
  }
}
const ik = new Persoon("Desmet");
ik.voornaam("Hans");
ik.voornaam("Cyriel");
console.log(ik.naam());
```

21.19 landen

```
"use strict";
const landen = tikLanden();
if (landen.length !== 0) {
  const gemiddelde = berekenGemiddelde(landen);
  console.log("Gemiddelde:", gemiddelde);
  toonLandenOnderGemiddelde(landen, gemiddelde);
  toonLandenVanafGemiddelde(landen, gemiddelde);
}
function tikLanden() {
  const landen = [];
  let naam = prompt("Naam:");
  while (naam !== "stop") {
    landen.push({ naam: naam, oppervlakte: Number(prompt("Oppervlakte:")) });
    naam = prompt("Naam:");
  }
  return landen;
}
function berekenGemiddelde(landen) {
  return landen.map(land => land.oppervlakte)
   .reduce((totaal, oppervlakte) => totaal + oppervlakte) / landen.length;
}
function toonLandenOnderGemiddelde(landen, gemiddelde) {
  console.log("Landen onder gemiddelde:");
  landen.filter(land => land.oppervlakte < gemiddelde)
    .forEach(land => console.log(land.naam, land.oppervlakte));
}
function toonLandenVanafGemiddelde(landen, gemiddelde) {
  console.log("Landen vanaf gemiddelde:");
  landen.filter(land => land.oppervlakte >= gemiddelde)
    .forEach(land => console.log(land.naam, land.oppervlakte));
}
```

21.20 personen

```
"use strict";
const personen = tikPersonen();
if (personen.length !== 0) {
  const kleinsteLengte = berekenKleinsteLengte(personen);
  console.log("Kleinste lengte:", kleinsteLengte);
  toonPersonenMetLengte(personen, kleinsteLengte);
  const grootsteLengte = berekenGrootsteLengte(personen);
  console.log("Grootste lengte:", grootsteLengte);
  toonPersonenMetLengte(personen, grootsteLengte);
}
function tikPersonen() {
  const personen = [];
  let naam = prompt("Naam:");
  while (naam !== "stop") {
    personen.push({ naam: naam, lengte: Number(prompt("Lengte:")) });
    naam = prompt("Naam:");
  }
  return personen;
}
function berekenKleinsteLengte(personen) {
  return personen.map(persoon => persoon.lengte)
    .reduce((kleinste, lengte) => {
      if (kleinste < lengte) {
        return kleinste;
      }
      return lengte;
      });
}
function berekenGrootsteLengte(personen) {
  return personen.map(persoon => persoon.lengte)
    reduce((grootste, lengte) => {
      if (grootste > lengte) {
        return grootste;
      }
      return lengte;
      });
}
function toonPersonenMetLengte(personen, lengte) {
  personen.filter(persoon => persoon.lengte === lengte)
    .forEach(persoon => console.log(persoon.naam, persoon.lengte));
}
```

22 COLOFON

Domeinexpertisemanager: Jean Smits

Moduleverantwoordelijke: Hans Desmet

Medewerkers: Hans Desmet

Versie: 16/10/2019

Nummer dotatielijst: