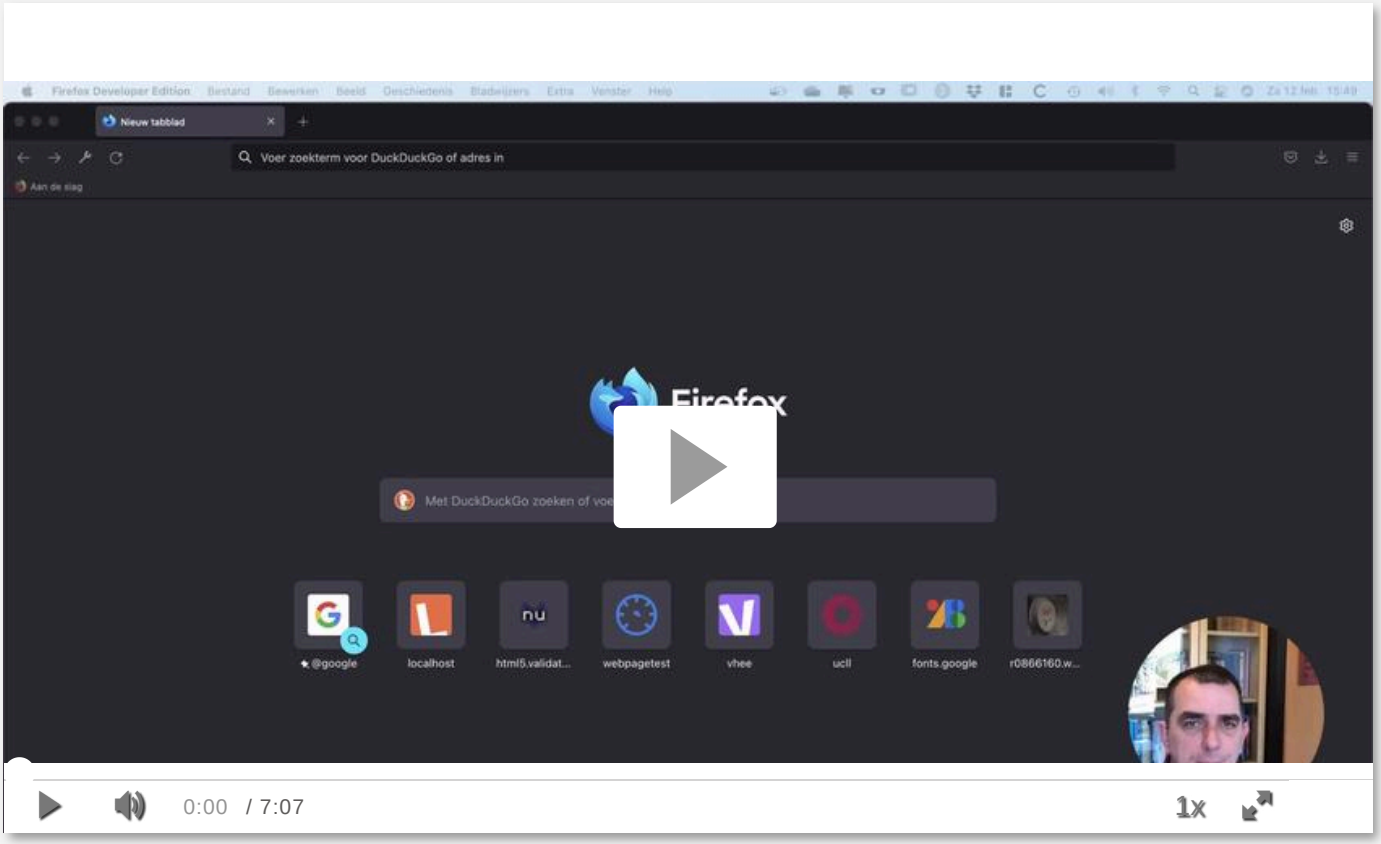
> *JavaScript is the world's most misunderstood programming language.*
> —Douglas Crockford

Een beperkt overzicht over de taal JavaScript

We bekijken in dit deel een heel beperkt aantal JS constructies. We gaan er van uit dat je al de basis van programmeren snapt. Ondanks de beperking die we ons opleggen om *alleen maar de absolute basis te tonen*, wordt dit toch een lang stuk. Hopelijk vind je het wel te verteren door de vele kleine voorbeeldjes en oefeningen. Het overzicht is sterk gebaseerd op https://javascript.info/.



1 Variabelen

Meer info over variabelen: https://javascript.info/variables

JavaScript is een *case-sensitive* taal. Dat betekent dat `getal` en `Getal` twee verschillende variabelen zijn. Het is gebruikelijk om variabelen te noteren in *camelCase*. Dat betekent dat je de eerste letter van de variabele in kleine letters schrijft en de eerste letter van elk volgend woord in hoofdletters. Dus `maxGetal` in plaats van `maxgetal` of `MaxGetal`.

Je kan een variabele bekijken als een doos met een unieke naam erop. In deze doos kan je data bewaren. Je *declareert* een variabele met `let` of `const`. Je ziet ook soms nog `var`, maar is in de meeste gevallen verouderd. Als je weet dat een variabele na de eerste toekenning niet meer mag of kan veranderen, dan declareer je de variabele liefst met `const`. Als je later in je code dan per ongeluk toch deze variabele probeert een andere waarde te geven, krijg je een foutmelding.

```
// dit is commentaar van 1 regel
"use strict"; // zet JS in strict mode = veiliger
let x = 10; // x heeft waarde 10
let dubbel = 2 * x; // de variabele y bevat nu het getal 20
x = 11 // x bevat nu het getal 11, geen let meer gebruiken!
let getal; // de doos getal is klaar, maar bevat nog niets
const maxGetal = 100; // maxGetal mag je hierna geen nieuwe waarde geven
```

2 Datatypes

Meer info over datatypes: https://javascript.info/types

JS is *zwak getypeerd*, in tegenstelling tot talen zoals Java of C#, die sterk getypeerd zijn. De voornaamste datatypes die we in ons beperkt overzicht bekijken zijn:

- number: getallen, zowel geheel als kommagetallen
- string: 0 of meerdere karakters tussen enkele of dubbele aanhalingstekens
- boolean: true / false
- null: voor onbekende waarden
- undefined als er nog geen waarde toegekend is
- object: voor meer complexe datastructuren

De operator `typeof` kan nuttig zijn om het type van een variabele op te vragen

```
// datatypes
let start = 2;
typeof start; // geeft number als resultaat
let boodschap = 'De waarde is:';
let stap = "1"; // stap is een string
let klaar = false;
```

3 Typeconversie

Meer info over typeconversie: https://javascript.info/type-conversions

Een zwak getypeerde taal laat heel wat bewerkingen toe die in een sterk getypeerde taal fouten zouden opleveren. Een voorbeeld dat gebruik maakt van de declaraties in het vorige stuk: een number (start) optellen bij een string (stap). In C# zou je hier expliciete casting op los moeten laten. JS maakt er geen probleem van en plakt beide variabelen aan elkaar:

```
let res = start + stap // res bevat de string '21'
```

4 Enkele operatoren

Meer info over operatoren: https://javascript.info/operators

4.1 Wiskunde

Eerst een paar *begrippen* kort verduidelijken. In de JS-code `4 + 8` noemen we `+` een *operator*. De getallen 4 en 8 zijn dan de linker- en rechteroperand (men gebruikt soms ook de benaming 'argumenten'). De operator `+` is een *binaire* operator omdat er *twee* operanden zijn.

De verwachte wiskundige operatoren zijn beschikbaar: `+ - * /`. We kunnen ook de rest bij een gehele deling (`%`) en machtsverheffing (`**`) gebruiken.

```
3 + 8.1; // geeft 11.1
1 / 3; // 0.3333333333333333
14 % 3; // geeft 2, de rest bij gehele deling van 14 door 3
2 ** 8 // 256 = 2 tot de 8ste macht
```

Voor moeilijker wiskundige bewerkingen is er het `Math` object, dat heel wat eigenschappen en methoden heeft. Denk hierbij aan alle toetsen op je rekenmachine die niet de gewone bewerkingen vormen, zoals sin, cos, log, afronden enz.

4.2 Strings aan elkaar plakken

Twee strings plak je (zoals in zoveel andere programmeertalen) met de binaire operator `+`. In het Engels heet deze bewerking 'string concatenation'.

```
let leeftijd = 12;
console.log('Ik ben ' + leeftijd + ' jaar oud.');
```

Een krachtiger manier om met strings om te gaan zijn zogenaamde *backticks*. Die laten immers toe om *elke uitdrukking in te voegen in een string* door gebruik te maken van ${...}. Bovenstaand voorbeeld kan hiermee korter geschreven worden:

```
console.log(`Ik ben ${leeftijd} jaar oud.`); // krachtiger en korter dan werken met +
```

4.3 Toekenning

De toekenning `=` is ook een operator. Eentje met een lage prioriteit, bovendien. Dat is de reden waarom in `let x = 3 * 2` eerst de operator `*` uitgevoerd wordt. Het resultaat van deze berekening wordt dan toegekend aan de variabele `x` door de operator `=`.

4.4 Modify-in-place

Het komt vaak voor dat we een operator moeten toepassen op een variabele en het nieuwe resultaat daarvan in diezelfde variabele moeten opslaan. Denk bvb. aan het verhogen of verlagen van een teller. Dat gaat als volgt:

```
let teller = 0;
teller = teller + 5; // teller heeft nu waarde 5
teller += 2 // teller is nu 7, korte versie van teller = teller + 2
teller *= 10 // teller is nu 70, korte versie van teller = teller * 10
```

4.5 Verhoog / verlaag

Een van de meest gebruikte wiskundige bewerkingen is een variabele met 1 verhogen of verlagen. Dat doe je met de operatoren `++` en `--`.

```
let x = 3;
console.log(x++) // print 4 in console
```

Er is een klein maar belangrijk verschil tussen `i++` en `++i`. Beiden doen ongeveer hetzelfde. Zoek zelf het verschil even op tussen beide, bvb op de MDN-pagina over uitdrukkingen en operatoren bij de rubriek 'Arithmetic operators'. Los dan volgende (moeilijke!) oefening op.

Bekijk volgende code en voorspel wat er na het uitvoeren van deze regels in de variabelen `s1`, `s2` en `som` staat.

```
let som = 6;
let s1 = som++;
let s2 = ++som;
```

Toon / verberg oplossing

5 Vergelijken

Meer info over vergelijken: https://javascript.info/comparison

5.1 Resultaat van vergelijking is een boolean

Je kent heel wat vergelijkingsmogelijkheden uit wiskunde: groter dan (`>`), groter dan of gelijk aan (`>=`), kleiner dan (`<`), kleiner dan of gelijk aan (`<=`), gelijk aan (`==` of `===`) en niet gelijk aan (`!=`). Merk op dat een *enkel* gelijkheidsteken een *toekenning* en geen vergelijking is.

Het resultaat van een vergelijking is altijd een boolean. Enkele voorbeelden:

```
let getal = 4.6;
getal < 5; // true
7 <= getal; // false
getal == 4; // false
getal = -2 // geen vergelijking maar een toekenning aan de variabele!
```

5.2 Strings vergelijken

Je kan ook strings vergelijken. Details vind je op javascript.info. Een paar kleine voorbeelden als illustratie:

```
'appel' < 'banaan'; // true volgens alfabetische volgorde
'appel' < 'appelmoes'; // true want eerste 5 letters gelijk en tweede woord is langer
'appel' == 'peer'; // false
'peer' != 'banaan'; // true
```

5.3 Variabelen met een verschillend type vergelijken

Als je twee variabelen met een verschillend type vergelijkt zal JS die eerst naar getallen converteren. De boolean `true` wordt gelijk aan 1, de boolean `false` aan 0.

```
3 >= '1'; // true
false < 1; // true want 0 < 1
'05' == 5 // true, de string wordt eerst naar het getal 5 geconverteerd
```

5.4 Stricte gelijkheid ===

Meestal is het veiliger om te testen op *strikte gelijkheid*. Hiervoor bestaat `===`. Als de operanden (de waarden aan beide kanten van de drie gelijkheidstekens) een verschillend teken hebben, geeft de vergelijking sowieso `false`.

```
3.1 === 3.1; // true
3.1 == '3.1'; // true want string wordt een getal
3.1 === '3.1'; // false want verschillend type
```

> ⚠ Als het kan, kies altijd voor de strikte gelijkheid `===`!

5.5 'truthy' en 'falsy'

Voor de volledigheid willen we toch even ook deze begrippen vermelden, want dit is toch wel belangrijk (en veelgebruikt!) concept in JS. Alles wat niet 'falsy' ('valsachtig') is, is 'truthy' ('waarachtig'), zie de pagina op MDN over 'falsy'. Bij het testen van voorwaarden wordt er niet gekeken naar true / false, maar wel naar truthy / falsy. **Falsy is: false, 0, lege string, null, undefined, NaN. Al de rest is truthy**.

6 Voorwaarden: if, else

Meer info over voorwaarden: https://javascript.info/ifelse

6.1 if ()

Soms wil je verschillende acties uitvoeren, afhankelijk van een bepaalde voorwaarde. Hiervoor bestaan in JS `if` en de 'vraagtekenoperator' `?`.

Stel dat je naar de console de boodschap "getal ... is even" wilt schrijven als de waarde van een bepaalde variabele even is, dan zou dit eenvoudig voorbeeld er als volgt kunnen uitzien:

```
let test = 5;
if (test %2 === 0) {
  console.log('Het getal ' + test + ' is even');
}
```

In bovenstaand voorbeeld is de bewerking `%2` de 'modulo 2' bewerking, of de rest bij deling door 2. Als een getal bij deling door 2 een rest gelijk aan 0 heeft, is het een even getal. Merk ook op dat we de strikte gelijkheid `===` gebruiken. Er zal niets verschijnen naar de console, want de test tussen ronde haakjes () geeft `false`. Je had de accolades ook kunnen weglaten na de `if` en alles op één regel schrijven, maar voor de duidelijkheid zou ik voorstellen om het toch altijd zo met accolades {} en inspringen te noteren.

Vanaf hier wordt het verhaal over truthy / falsy belangrijk. De code `if (5) console.log("gelukt")` zal wel degelijk de string "gelukt" uitschrijven in de console. Het getal 5 in de `if` is namelijk niet 0 en dus is deze voorwaarde truthy.

6.2 if () else

We breiden dit voorbeeld uit tot:

```
let test = 5;
if (test %2 === 0) {
  console.log('Het getal ' + test + ' is even');
} else {
  console.log('Het getal ' + test + ' is oneven');
}
// in de console verschijnt nu de string 'Het getal 5 is oneven'
```

6.3 ?

De vraagtekenoperator `?` wordt ook wel 'ternaire operator' genoemd omdat dit de enige operator is die *drie* operanden heeft. Het voorbeeld hierboven kan korter geschreven worden als:

```
let test = 5;
console.log('Het getal ' + test + ' is ' + (test %2 === 0 ? 'even' : 'oneven'));
// in de console verschijnt nu de string 'Het getal 5 is oneven'
```

Nog een voorbeeld: `let magAlcoholKopen = (leeftijd > 18) ? true : false;` kent aan de variabele `magAlcoholKopen` de waarde `true` toe als de leeftijdstest slaagt en `false` in het andere geval.

7 Logische operatoren: of, en, niet

Meer info over logische operatoren: https://javascript.info/logical-operators

Je bent ongetwijfeld uit één van je programmeerOPO's al vertrouwd met de logische operatoren 'en' (`&&`), 'of' (`||`) en 'niet' (`!`).

De 'of' werkt zoals je verwacht: als één van beide operanden waar ('truthy') is, is de volledige uitdrukking `true`. JS kent echter – zoals vele talen – het principe van 'kortsluiting'. Van zodra in een 'of' de waarde `true` (eigenlijk 'truthy') bereikt wordt, stopt de evaluatie. Deze techniek wordt o.a. gebruikt om voor de 'of' iets te doen dat nodig is voor het stukje erna.

```
let x = 3;
x === 3 || console.log('dit wordt niet geschreven');
// geeft true, de zin wordt niet naar de console geschreven

x === 2 || console.log('dit wordt wel geschreven');
// het resultaat is undefined, zin wordt wel geschreven
```

Ook 'en' werkt gelijkaardig: alleen als beide operanden waar (eigenlijk 'truthy') zijn is de volledige uitdrukking `true`. Ook hier geldt het principe van kortsluiting, maar dan wel voor 'falsy': van zodra in een 'en' er één 'falsy' waarde optreedt, stopt de evaluatie meteen en komen de andere delen niet meer aan bod.

8 Lussen: while en for

Meer info over lussen: https://javascript.info/while-for

Je bent ondertussen al in contact gekomen met lussen in het OPO 'programmeren basis' en kent verschillende soorten lussen. We geven een voorbeeld van een `while` en een `for` lus. Hetzelfde voorbeeld wordt twee keer uitgewerkt: *bereken de som van alle vijfvouden met drie getallen (geen nullen vooraan) en schrijf deze getallen naar de console* .

8.1 While

De while-lus heeft volgende structuur:

```
while (voorwaarde) {
  // code
}
```

De getallen met drie cijfers gaan van 100 tot 1000 (niet inbegrepen). Om alle veelvouden van 5 te bereiken volstaat het om van 100 te vertrekken en dit getal telkens met 5 te verhogen. In de lus houdt een variabele `som` de som van alle getallen bij. Zorg dat je bij een `while` lus niet vergeet om op het einde een aanpassing (in dit geval verhoging met 5) te doen aan een lusvariabele, want anders krijg je een *oneindige lus*.

```
let getal = 100; // declareer variabele en initialiseer op 100
let som = 0;
while (getal < 1000) {
  console.log(getal);
  som += getal // kort voor som = som + getal
  getal += 5 // verhoog getal met 5
}
```

Zolang de waarde tussen haakjes (de uitdrukking `getal < 1000)` 'truthy' is, wordt de code tussen de accolades uitgevoerd.

Op het moment dat `getal` van 995 naar 1000 gaat, wordt de waarde van de vergelijking 'falsy' en stopt de lus. De waarde van de variabele `getal` is dan 1000, in `som` zit de waarde 98550 (= 100 + 105 + ... + 995).

Er bestaat ook een `do … while` lus. Het enige verschil is dat deze lus minstens één keer zal uitgevoerd worden, want de test gebeurt pas na de eerste uitvoer van de

code tussen accolades.

8.2 For

We werken hetzelfde voorbeeld uit als bij de `while`. De structuur van een for-lus is:

```
for (begin; voorwaarde; stap) {
  // code
}
```

De alternatieve versie van het voorbeeld met deze lus kan er dan als volgt uitzien:

```
let som = 0;
for (let i = 100; i < 1000; i += 5) {
  console.log(i);
  som += i;
}
```

Het is gebruikelijk om als lusvariabele een korte naam te kiezen, bvb. `i`.

8.3 Een lus onderbreken

Je kan een lus stoppen met `break` of een stap onderbreken met `continue`. Sorry, dat kan ik hier niet uitleggen … mijn vingers blokkeren op het klavier. Ik heb namelijk leren programmeren in een tijd waarin dit absoluut verboden was om te doen, want het produceerde ‘spaghetticode’. Mijn proffen hebben dit er zo ingestampt (en beloofden ons ‘hel en verdoemenis’) dat ik niet in staat ben deze techniek te gebruiken. Ik zal altijd het onderbreken in de lusvoorwaarde zelf programmeren.

Er bestaan nog andere soorten for-lussen: `for … of` en `for … await … of`. De `for … in` wordt afgeraden om te gebruiken. Meer over deze ‘speciale’ lussen later.

9 Functies

9.1 Klassieke functienotatie

Meer info over functies: https://javascript.info/function-basics

Als je op verschillende plaatsen in je script een gelijkaardige actie wil doen, is het altijd handig om een functie te maken. Functies zijn de hoofdbouwblokken van een programma.

Een *functiedeclaratie* ziet er zo uit:

```
function naam(parameter1, parameter2, …) {
  // code
}
```

Als een functie iets moet berekenen en dat resultaat teruggeven, gebruik je een `return`. Die zorgt er voor dat de waarde wordt teruggegeven en de verdere uitvoer van de functie gestopt wordt.

We komen even terug op het voorbeeld van bij de lussen, waar we de som 100 + 105 + … 995 berekenden. Stel dat je ook sommen zoals 3 + 5 + 7 + … +31 en 10 + 11 + 12 + … 19 wilt uitrekenen. Je merkt dat er hier een patroon is: een beginwaarde, stapgrootte en een eindwaarde (in wiskunde noemen we dit een *rekenkundige rij*). We schrijven hiervoor een functie en beperken ons even tot een positieve stapgrootte:

```
function berekenSom(begin, eind, stap) {
  // berekent de som van alle getallen van begin
  // tot eind, met stapgrootte stap
  let som = 0; // initialiseer som
  for (let i = begin; i <= eind; i += stap) {
    som += i;
  }
  return som
}
```

Met deze functie kan ik nu het totaal aantal ogen op een dobbelsteen berekenen: `berekenSom(1, 6, 1)`. Je bekomt 21 als resultaat van de functie-oproep.

Het is belangrijk om te beseffen dat variabelen die je declareert binnen de functiedefinitie alleen tijdens het uitvoeren van de functie bestaan. Als de functie gedaan is, kan je bvb de waarde van de variabele `som` niet opvragen.

> Als ik de functie oproep met `berekenSom(6, 1, -1)` kom ik als waarde 0 uit. Kan je dat verklaren? Toch zou ik graag de som 6 + 5 + 4 + 3 + 2 + 1 kunnen uitrekenen. Pas de code van de functie aan zodat je ook sommen met negatieve stap kan uitrekenen. Er zijn meerdere oplossingen mogelijk.

Toon / verberg oplossing

9.2 Vette pijl notatie, functie-uitdrukking

Meer info over arrow functions: https://javascript.info/arrow-functions-basics

Er is een modernere – kortere – manier om functies te noteren, de zogenaamde 'pijlnotatie'. Je zal die veel tegenkomen in moderne JS-scripts. We tonen het even met een voorbeeld. Volgende functie (gewone notatie) berekent het gemiddelde van twee getallen. Merk trouwens op in het voorbeeld dat we de functie als een waarde kunnen *toekennen aan een variabele* . Men spreekt dan over een 'functie-uitdrukking'.

```
let gemiddelde = function(a, b) {
  return (a + b) / 2;
}
```

In 'arrow notation' wordt dat:

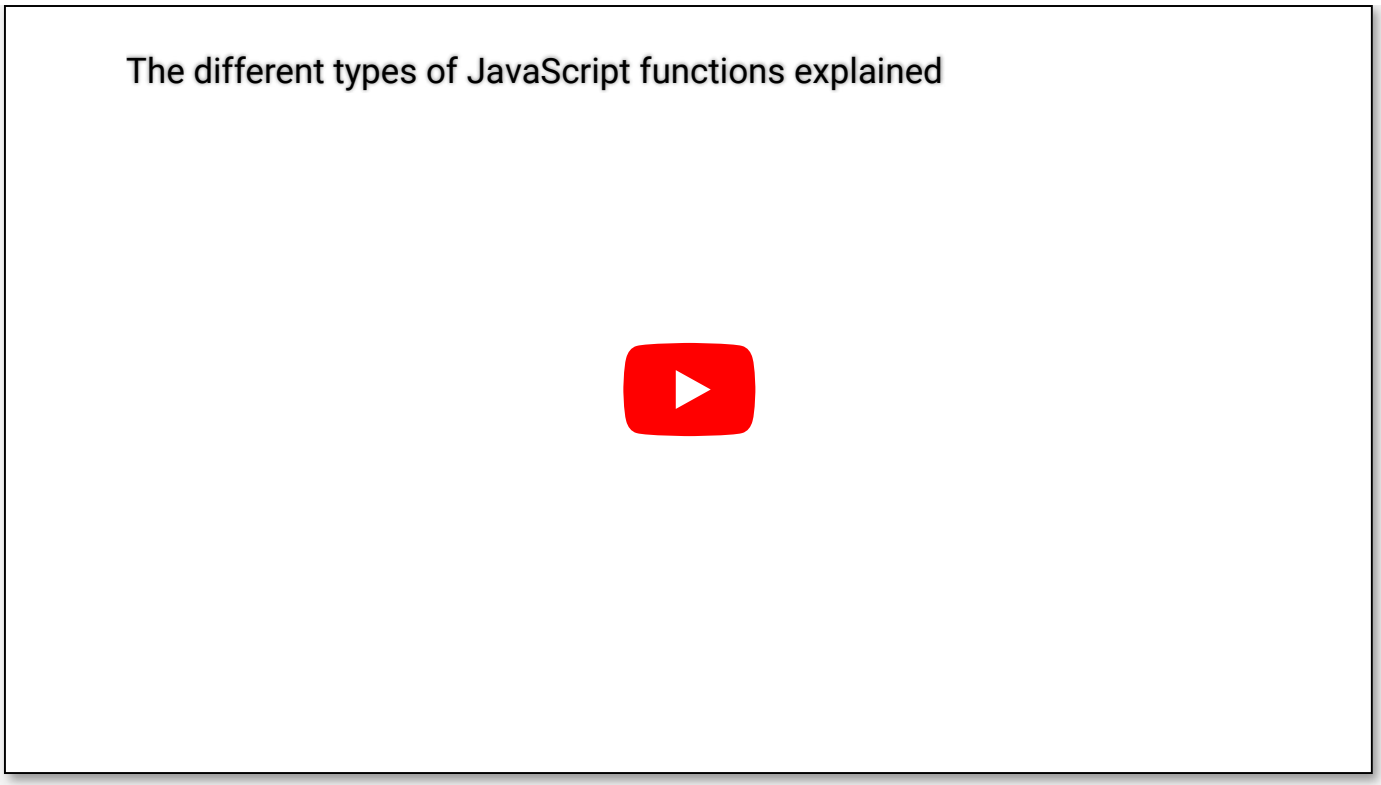
```
let gemiddelde = (a, b) => {
  return (a + b) / 2;
}
```

Aangezien er maar één uitdrukking in de `return` staat, kan je dit korter noteren als volgt:

```
let gemiddelde = (a, b) => (a + b) / 2;
```

9.3 Hoe kiezen tussen verschillende functie-notaties?

Kevin Powell geeft (met de hulp van Chris Ferdinani) een kort en duidelijk overzicht van de verschillende manieren om met functies te werken in JS. Op het einde van het filmpje komt de problematiek rond het 'this' sleutelwoord aan bod. We zien in dit inleidend OPO echter geen klassen, dus dat laatste stukje kan je overslaan (al is het wel nuttig voor het vervolgOPO 'frontend gevorderd').



↑ Naar top