

# JAVASCRIPT BASIS

## 1. Wat is JAVASCRIPT

**JavaScript is NIET Java.** Hoewel de namen op elkaar lijken, is Java een volledig andere, objectgeoriënteerde programmeertaal. De keuze voor de naam "JavaScript" was destijds een marketingbeslissing en heeft niets te maken met Java.

**JavaScript** is een veelzijdige programmeertaal die voornamelijk wordt gebruikt voor webontwikkeling. Het stelt ontwikkelaars in staat om interactiviteit op websites mogelijk te maken, zoals het reageren op gebruikersacties zoals klikken, slepen, swipen, en typen. JavaScript wordt met name gebruikt in **front-end development**, dat wil zeggen op de client-side (in de browser). Dankzij de opkomst van frameworks zoals **Node.js** kan JavaScript ook op de server-side worden gebruikt, waardoor het een volledige full-stack programmeertaal is geworden.

### Belangrijke kenmerken van JavaScript:

- **Platformonafhankelijk:** JavaScript werkt in elke moderne webbrowser en is daarmee een van de meest toegankelijke programmeertalen.
- **Interactiviteit:** Het voegt dynamische en interactieve elementen toe aan websites, zoals formuliervalidatie, animaties, en het dynamisch aanpassen van inhoud.
- **Case-sensitive:** JavaScript maakt onderscheid tussen hoofdletters en kleine letters. myVariable en myvariable zijn twee verschillende dingen.
- **Client-side en Server-side:** Naast client-side gebruik met frameworks als React, Vue, of Angular, is JavaScript dankzij Node.js ook populair geworden voor server-side development.

## 2. HISTORIEK

In het begin van het internet had elke browser zijn eigen scripttaal voor het web:

- **Netscape:** LiveScript
- **Internet Explorer:** JScript

JavaScript evolueerde echter snel naar een universele standaard die breed werd geaccepteerd. Tegenwoordig is JavaScript de enige front-end taal die samen met HTML en CSS wordt gebruikt voor het maken van interactieve webpagina's.

Sinds 2009 wordt de ontwikkeling van JavaScript beheerd door **ECMA**

**International**, en de officiële naam van de standaard is **ECMAScript** (afgekort ES). ECMAScript 5 (ES5) is jarenlang de norm geweest, maar sinds 2015 zijn we overgestapt naar **ES6** (ook wel ECMAScript 2015 genoemd).

### Versies en nieuwe standaarden:

- **ES6 (ECMAScript 2015):** Introduceerde veel nieuwe functies zoals let, const, pijlfuncties (=>), en classes. Dit was een grote sprong voorwaarts voor de taal en wordt nog steeds breed ondersteund door moderne browsers.
- **ES7 t/m ES9:** Jaarlijks worden er nieuwe functies toegevoegd. Bijvoorbeeld, **ES7 (2016)** voegde de Array.prototype.includes() methode toe, en **ES8 (2017)** introduceerde **asynchrone functies** (async/await).
- **ES9 en verder:** Dit zijn de meest recente versies met kleinere updates, zoals **object rest/spread** en verbeteringen aan asynchrone iteratie. Sinds **ES6** volgen de releases een jaarlijks schema, waarbij telkens kleine verbeteringen worden doorgevoerd.

### Backward compatibility:

JavaScript is ontworpen om **achterwaarts compatibel** te zijn. Dit betekent dat code geschreven in oudere versies zoals ES5 nog steeds werkt in moderne browsers die ES6 en hoger ondersteunen. Veel oudere browsers ondersteunen echter mogelijk niet alle nieuwe functies van ES6 en verder, waardoor het soms nodig is om zogenaamde **polyfills** te gebruiken of code te "transpilen" naar oudere standaarden (bijvoorbeeld met behulp van tools zoals Babel).

### Opmerking:

Hoewel de meeste moderne browsers ES6 en latere versies ondersteunen, zijn er oudere browsers waarin sommige nieuwe functies niet werken. Daarom is het belangrijk om te testen of je code functioneert in de browsers die je doelgroep gebruikt, vooral als je ook oudere systemen wilt ondersteunen.

### 3. WANNEER GEBRUIK JE JAVASCRIPT

Hoewel JavaScript een krachtige tool is voor het toevoegen van interactiviteit en dynamiek aan websites, zijn er bepaalde taken waarvoor het gebruik van JavaScript tegenwoordig niet meer nodig is dankzij de verbeteringen in CSS3 en HTML5. Door deze evoluties kun je veel functionaliteiten realiseren zonder afhankelijk te zijn van JavaScript, wat vaak resulteert in snellere en betere prestaties.

Hier zijn enkele gevallen waarin je geen JavaScript hoeft te gebruiken:

- CSS neemt nu het volgende over:
- Image swaps (rollover): Voor het wisselen van afbeeldingen bij hover of klik kun je puur CSS gebruiken, zonder dat er JavaScript nodig is. Dit gebeurt meestal met behulp van de :hover pseudoklasse.
- Rollover menus: Interactieve navigatiemenu's die uitklappen wanneer een gebruiker eroverheen hovert, kunnen volledig met CSS worden gemaakt door gebruik te maken van de :hover pseudoklasse en display of visibility eigenschappen.
- Tooltips: Eenvoudige tooltips kunnen ook met CSS worden gerealiseerd door gebruik te maken van :hover of :focus en ::before of ::after om de tooltip te tonen bij interactie.
- CSS3 Transities en Animaties:
- CSS3 biedt krachtige tools voor het maken van animaties en overgangen (transitions). Waar voorheen JavaScript nodig was voor animaties, kun je nu vloeiende overgangen en complexere animaties maken met CSS. Dit omvat onder andere fades, slides, en andere visuele effecten zonder een enkele regel JavaScript.
- HTML5 Form Controls:
- Met HTML5 zijn er nieuwe, ingebouwde formulierelementen en validatiemogelijkheden geïntroduceerd. Elementen zoals datumprikkers, kleurenkiezers, en invoervalidatie (zoals vereiste velden en patroonvalidatie) kunnen rechtstreeks met HTML worden uitgevoerd, zonder dat JavaScript-scripts nodig zijn om basisvalidatie of invoermethoden te implementeren.

#### Belangrijkste voordelen van het gebruik van CSS3 en HTML5 in plaats van JavaScript:

- **Prestatieverbeteringen:** Omdat CSS en HTML natively door de browser worden ondersteund en uitgevoerd, zijn ze vaak sneller dan JavaScript-oplossingen, die extra rekenkracht en scripts vereisen.
- **Toegankelijkheid en eenvoud:** Voor eenvoudige visuele effecten en validatie zijn CSS3 en HTML5 eenvoudiger te implementeren, begrijpen, en onderhouden.
- **Responsiviteit:** CSS kan dynamisch reageren op verschillende schermformaten en apparaten met media queries, zonder dat je JavaScript nodig hebt om de lay-out aan te passen.

## **Wanneer je wel JavaScript moet gebruiken:**

Ondanks de voordelen van CSS3 en HTML5 zijn er nog veel gevallen waarin JavaScript essentieel blijft, zoals:

- **Complexe interacties:** Wanneer je geavanceerde functionaliteiten zoals drag-and-drop, dynamische gegevensverwerking of gebruikersinterfacecomponenten zoals modals of carrousels wilt implementeren.
- **Asynchrone interactie met servers:** Voor taken zoals het ophalen van gegevens zonder de pagina te vernieuwen (bijv. via AJAX of Fetch API).
- **Geavanceerde gebruikerservaring:** Denk aan single-page applicaties (SPA's) of frameworks zoals React, Vue, of Angular, die intensief gebruik maken van JavaScript om dynamische webapplicaties te bouwen.

## 4. JAVASCRIPT JARGON

### Javascript JARGON

- Data types
  - Numeriek
  - Strings
  - Booleans
  - Arrays
  - Objects
    - built-in: window, document, ... zijn allemaal objecten met hun eigen properties (eigenschappen).
- Variabelen: var, let, const
- Operatoren
  - Logische en wiskundige operatoren
- Functies
  - zijn collecties (verzamelingen) van statements die een waarde retourneren.
- Controle structuren
  - If, else, switch
- Loops
  - For, while, do while,...
- De DOM (DOCUMENT OBJECT MODEL) voor javascript
  - Veel methodes
    - Bijv. document.getElementById,...
  - Veel properties
    - value, checked, className, id,...
  - Event listeners
  - Hoisting
  - Event handling
  - Scope en closures

## 5. CONSOLE (Google Chrome - Developer Tools)

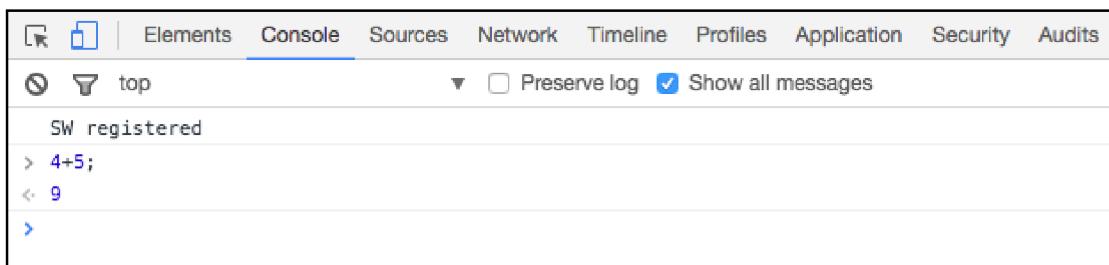
De Console in Google Chrome en andere moderne browsers is een krachtig hulpmiddel waarmee je direct JavaScript kunt uitvoeren en interactie kunt hebben met je code. Het biedt de mogelijkheid om snel te testen, fouten op te sporen en zelfs je code te optimaliseren door gebruik te maken van verschillende ingebouwde functies.

### Wat is de Console?

De Console is een onderdeel van de ontwikkelaarstools van een browser en stelt je in staat om JavaScript direct uit te voeren, variabelen en functies te evalueren, en interactie te hebben met de DOM (Document Object Model). Het is een van de meest waardevolle tools voor zowel beginners als ervaren ontwikkelaars bij het debuggen en analyseren van JavaScript-code.

In de console kun je JavaScript opdrachten typen en direct het resultaat zien.

Bijvoorbeeld:



Om dit te openen in google chrome:

Druk F12 en klik op het tabblad CONSOLE

of

CTRL + SHIFT + J

Om dit te openen in firefox:

Druk F12 en klik op het tabblad WEB CONSOLE

of

CTRL + SHIFT + K

Dit maakt de console ideaal voor snel testen en debuggen van codefragmenten.

Functies die we later zullen gebruiken binnen onze editor zijn console.log, console.time, console.table, console.group,...

## 5.1. VARIABELEN

### VAR

In JavaScript gebruiken we het sleutelwoord var om variabelen te declareren. Een variabele is een soort "doosje" waarin we gegevens kunnen opslaan, zoals getallen of tekst. We gebruiken variabelen om deze gegevens later opnieuw te gebruiken in ons script.

Het gebruik van var was lange tijd de standaard manier om variabelen aan te maken, maar tegenwoordig gebruiken veel programmeurs liever let of const, omdat var enkele eigenschappen heeft die voor beginners verwarrend kunnen zijn. Desondanks is het belangrijk om te begrijpen hoe var werkt, omdat je het vaak tegenkomt in oudere code.

Een belangrijk kenmerk van var is dat variabelen die ermee worden gedeclareerd, **function-scope** of **globale scope** hebben. Dit betekent dat als je een variabele met var aanmaakt binnen een functie, die alleen binnen die functie beschikbaar is. Maar als je var buiten een functie gebruikt, wordt de variabele globaal, wat betekent dat het overal in je script toegankelijk is.

#### **Handeling: Hoe declareer en gebruik je een variabele met var?**

Het declareren van een variabele met var is eenvoudig. Volg deze stappen:

```
var mijnGetal = 10;  
console.log(mijnGetal);
```

### LET

let is een modernere manier om variabelen te declareren in JavaScript, en het heeft een paar belangrijke verbeteringen ten opzichte van var. Waar var function-scope heeft, heeft let block-scope. Dit betekent dat variabelen die met let worden gedeclareerd, alleen binnen het blok code waarin ze zijn gedefinieerd, beschikbaar zijn. Een blok code kan bijvoorbeeld een set haakjes {} zijn, zoals in een if-statement of een for-loop.

```
var mijnGetal = 10;  
console.log(mijnGetal);
```

Je ziet dus momenteel geen wezenlijk verschil.

## VERSCHIL TUSSEN VAR EN LET

```
if (true) {  
    var varVariable = "Ik ben een var!";  
    let letVariable = "Ik ben een let!";  
}  
  
console.log(varVariable); // Dit werkt, varVariable is toegankelijk buiten het if-blok.  
console.log(letVariable); // Dit geeft een foutmelding, letVariable is niet toegankelijk buiten het  
if-blok.
```

### Uitleg:

1. **var werkt buiten het blok:** De variabele varVariable is met var gedeclareerd binnen het if-blok, maar is ook toegankelijk buiten dat blok. Daarom toont console.log(varVariable) de waarde "Ik ben een var!".
2. **let blijft binnen het blok:** De variabele letVariable is met let gedeclareerd binnen hetzelfde if-blok, maar zodra we buiten het if-blok proberen om de variabele te benaderen, krijgen we een foutmelding. Dit komt omdat letVariable alleen binnen het blok {} toegankelijk is.

## CONST

const is een manier om een variabele te declareren in JavaScript die **niet opnieuw kan worden toegewezen**. Dit betekent dat zodra je een waarde toekent aan een variabele die met const is gedeclareerd, je die waarde niet kunt veranderen. Echter, als de variabele een object of array is, kun je **de inhoud van het object of de array wel aanpassen**, maar niet het object zelf vervangen. Net als let heeft const **block-scope**, wat betekent dat een variabele gedeclareerd met const alleen toegankelijk is binnen het blok waarin het is gedefinieerd.

### Belangrijkste eigenschappen van const:

1. **Niet her-toewijgbaar:** De waarde van een const-variabele kan niet opnieuw worden toegewezen.
2. **Block-scope:** Net zoals let, heeft const een bereik dat beperkt is tot het blok waarin het is gedeclareerd.

```
const mijnGetal = 10;  
console.log(mijnGetal);
```

Dit declareert een constante variabele genaamd mijnGetal en kent de waarde 10 toe. Je kunt deze variabele gebruiken, maar je kunt er geen nieuwe waarde aan toekennen. Als je probeert de waarde van mijnGetal te veranderen, zal je een foutmelding krijgen:

```
mijnGetal = 20; // Dit zal een fout veroorzaken, omdat je geen nieuwe waarde kunt toekennen  
aan een const.
```

## 5.2. PLACEHOLDERS

Placeholders in JavaScript worden gebruikt binnen de `console.log()`-functie om specifieke waarden dynamisch in een string weer te geven. Deze placeholders gebruiken het %-symbool gevuld door een letter die het type van de waarde bepaalt die moet worden ingevoegd. Het zorgt ervoor dat het formaat van de output wordt gecontroleerd en kan worden aangepast. Dit is handig voor debugging en het overzichtelijk weergeven van data in de console.

### Basis Placeholder Syntax:

De basisvorm voor placeholders in JavaScript is:

```
console.log("%placeholder", waarde1, waarde2, ...);
```

Waarbij de %placeholder verwijst naar een specifieke formattering, en de waarden die je wilt invoegen achter de string worden geplaatst. Bijvoorbeeld:

```
var begroeting = "Hallo";
var naam = "Tom";
console.log("Begroeting: %s, Naam: %s", begroeting, naam);
// Output: "Begroeting: Hallo, Naam: Tom"
```

### Veelgebruikte Placeholders:

- **%s:** Voor strings.
- **%d of %i:** Voor gehele getallen (integers).
- **%f:** Voor drijvende kommagetallen (floating-point numbers).
- **%c:** Voor het toepassen van CSS-stijlen in de console-uitvoer.

```
let begroeting = "Goedemorgen";
let naam = "Alice";
console.log("%s, %s!", begroeting, naam);
// Output: "Goedemorgen, Alice!"
```

```
let leeftijd = 30;
console.log("Leeftijd: %d", leeftijd);
// Output: "Leeftijd: 30"
```

```
let prijs = 19.99;
console.log("De prijs is %f euro", prijs);
// Output: "De prijs is 19.990000 euro" (standaard met 6 decimalen)
```

```
console.log("De prijs is %.2f euro", prijs);
// Output: "De prijs is 19.99 euro"
```

```
console.log("%cDit is een gestileerde tekst", "color: blue; font-size: 16px; font-weight: bold;");
// Output: "Dit is een gestileerde tekst" in blauw, vetgedrukt, en met lettergrootte 16px
```

```
let product = "Laptop";
let prijs = 899.99;
let voorraad = 25;
console.log("Product: %s, Prijs: %.2f, Voorraad: %d stuks", product, prijs, voorraad);
// Output: "Product: Laptop, Prijs: 899.99, Voorraad: 25 stuks"
```

### 5.3. EDITOR

We zullen de console nog veel gebruiken, maar vanaf dit punt coderen we in een editor. Er zijn verschillende editors die zich lenen tot het coderen van javascript.

Er zijn gratis editors zoals: Sublime Text, Notepad++, ...

Maar als professioneel developer kijk je best naar professionele editors zoals: vscode of phpstorm/webstorm van Jetbrains.

Wij zullen het beste pakket hiervoor gebruiken nl. phpstorm/webstorm die een professioneel pakket is en door zeer veel softwarehuizen wordt gebruikt.

Voor developers die voornamelijk met frontend bezig zijn kan webstorm gebruikt worden, maar voor mensen die Full Stack coderen gebruiken we beter PHPStorm die speciaal voor PHP developers werd ontwikkeld. Wij zullen PHPSTORM installeren. Dit pakket heeft een gratis geldigheid van 1 jaar met een studielicentie.

## 5.4. INLINE JAVASCRIPT VS EXTERNE JAVASCRIPT

**Inline JavaScript:** Dit betekent dat je JavaScript direct in het HTML-document plaatst, binnen een `<script>`-tag. Dit kan handig zijn voor kleine stukjes code die slechts op één specifieke pagina moeten worden uitgevoerd, maar het heeft enkele nadelen.

Voorbeeld van inline JavaScript:

```
<html>
  <head>
    <title>Inline JavaScript</title>
  </head>
  <body>
    <p>Welkom op mijn website!</p>
    <script>
      document.body.style.backgroundColor = "lightblue";
    </script>
  </body>
</html>
```

**Nadelen van inline JavaScript:**

- Slechte schaalbaarheid: Als je veel JavaScript hebt, kan het moeilijk worden om dit te beheren binnen één HTML-bestand.
- Verminderde herbruikbaarheid: Je kunt dezelfde code niet eenvoudig hergebruiken op andere pagina's.
- Veiligheidsproblemen: Inline JavaScript kan vatbaar zijn voor Cross-Site Scripting (XSS)-aanvallen.
- 

**Externe JavaScript:** Hier verwijst een `<script>`-tag naar een extern .js-bestand. Dit is de aanbevolen manier, vooral voor grote of herbruikbare code.

```
<html>
  <head>
    <title>Externe JavaScript</title>
    <script src="script.js"></script>
  </head>
  <body>
    <p>Welkom op mijn website!</p>
  </body>
</html>
```

**script.js:**

```
document.body.style.backgroundColor = "lightgreen";
```

**Voordelen van externe JavaScript:**

- Herbruikbaarheid: Je kunt hetzelfde scriptbestand op meerdere pagina's gebruiken.
- Onderhoudbaarheid: Het is eenvoudiger om je code georganiseerd en gescheiden van je HTML te houden.
- Betere caching: Browsers kunnen externe JavaScript-bestanden cachen, wat de laadtijd van pagina's verkort.

## JavaScript in de <head> vs. de <body>

Het plaatsen van JavaScript in het document kan ook een impact hebben op de laadtijd en de werking van je pagina. Het transcript legt uit dat JavaScript in de <head> of onderaan de <body> kan worden geplaatst, maar dat dit verschillende resultaten oplevert.

- **JavaScript in de <head>:** Als je een script in de <head> van je document plaatst, wordt het uitgevoerd voordat de rest van de pagina is geladen. Dit kan problemen veroorzaken omdat het script de laadtijd van de pagina vertraagt en mogelijk nog niet-bestand elementen probeert te manipuleren.
- **Voorbeeld van probleem met script in <head>:**

```
<html>
<head>
<script>
  document.body.style.backgroundColor = "lightyellow"; // Body is nog niet geladen
</script>
</head>
<body>
  <p>Dit is een testpagina</p>
</body>
</html>
```

Oplossing: Plaats JavaScript onderaan de <body>, zodat het pas wordt uitgevoerd wanneer de rest van de pagina is geladen.

**JavaScript onderaan de <body>:** Dit zorgt ervoor dat de hele pagina eerst wordt geladen en weergegeven, waarna JavaScript wordt uitgevoerd. Dit is een klassieke oplossing om problemen met het laden van DOM-elementen te voorkomen.

Voorbeeld van correct gebruik:

```
<html>
<head>
  <title>JavaScript onderaan</title>
</head>
<body>
  <p>Dit is een testpagina</p>
  <script>
    document.body.style.backgroundColor = "lightyellow"; // Body is nu geladen
  </script>
</body>
</html>
```

## 5.5. GEBRUIK VAN ASYNC EN DEFER

In moderne webontwikkeling gebruiken we vaak de **async** en **defer** attributen in de <script>-tag om te controleren hoe en wanneer JavaScript-bestanden worden geladen. Deze methoden zijn efficiënter dan het plaatsen van scripts onderaan de pagina en kunnen laadtijden verkorten zonder dat je scripts moet splitsen.

- **async:** Het async-attribuut laadt JavaScript-bestanden asynchroon. Dit betekent dat de browser het HTML-document blijft parseren terwijl het het JavaScript-bestand downloadt. Zodra het script is gedownload, wordt het uitgevoerd. Dit kan leiden tot een klein "renderblokkerend" effect omdat het script wordt uitgevoerd zodra het klaar is, zelfs als het document nog niet volledig is geladen.
- Voorbeeld met async:

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

Gebruik: Handig voor scripts die niet afhankelijk zijn van de volledige DOM, zoals analytische scripts of externe services.

**defer:** Het defer-attribuut zorgt ervoor dat het JavaScript-bestand wordt gedownload terwijl de browser het HTML-document blijft parseren, maar het script wordt pas uitgevoerd wanneer het hele document is geladen. Dit voorkomt het renderblokkerende probleem en zorgt ervoor dat het script pas wordt uitgevoerd wanneer alle DOM-elementen beschikbaar zijn.

Voorbeeld met defer:

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

Gebruik: Dit is de aanbevolen manier om JavaScript-bestanden te laden die afhankelijk zijn van DOM-elementen, zoals interactiviteit op de pagina.

### Samenvatting van verschillen:

Eigenschap	Inline JavaScript	Externe JavaScript	async	defer
<b>Locatie</b>	In het HTML-document, binnen <script>	In een extern .js bestand	Extern .js bestand, uitgevoerd asynchroon	Extern .js bestand, uitgevoerd na DOM-load
<b>Gebruik</b>	Kleinere scripts, éénmalig gebruik	Grote scripts, herbruikbaar	Externe scripts die snel moeten worden geladen	Scripts die afhankelijk zijn van de DOM
<b>Problemen</b>	Niet schaalbaar, kan traag zijn	Efficiënter en schaalbaarder	Kan renderblokkering veroorzaken	Geen renderblokkering, aanbevolen voor DOM
<b>Timing</b>	Direct uitgevoerd op volgorde	Afhankelijk van locatie <script>	Uitgevoerd zodra het script is geladen	Uitgevoerd na het laden van de hele pagina

## 5.6. JAVAScriptOEFENINGEN - JS1

### Bestandenstructuur:

1. **Maak een nieuwe map aan:** genaamd JAVAScriptOEFENINGEN
2. **Maak een submap aan:** genaamd js1
3. **In de map js1 maak je een bestand aan genaamd:** index.html
4. **Maak een apart JavaScript-bestand aan:** genaamd script.js in dezelfde map.

### HTML-bestand (index.html):

```
<!DOCTYPE html>
<html lang="nl">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Javascript Basis</title>
</head>
<body>
  <h1>Javascript Basis</h1>
  <!-- Script met defer geladen -->
  <script src="script.js" defer></script>
</body>
</html>
```

### Uitleg over het gebruik van defer:

- **Waarom defer?**
- Het defer-attribuut zorgt ervoor dat het JavaScript-bestand wordt gedownload terwijl de HTML wordt geparsed, maar de uitvoering van het script wordt uitgesteld tot nadat de gehele DOM is geladen. Dit voorkomt het renderen van de pagina te blokkeren en zorgt ervoor dat het JavaScript pas wordt uitgevoerd wanneer de volledige structuur van de pagina beschikbaar is.
- **Voordelen van defer:**
- Het script kan veilig elementen van de DOM manipuleren, omdat deze al volledig zijn gerenderd wanneer het script wordt uitgevoerd.
- De prestaties van de pagina verbeteren doordat de browser niet hoeft te stoppen met het parsen van HTML om het JavaScript uit te voeren.

### **JavaScript-bestand (script.js):**

```
// Dit is een commentaarregel in JavaScript  
// De onderstaande regel print "Hallo console!" in de browserconsole.  
console.log('Hallo console!');  
  
// Hier kun je ook toekomstige scripts en functies toevoegen die interactiviteit aan de pagina  
toevoegen
```

### **COMMENTAARLIJNEN IN JAVASCRIPT**

Commentaar wordt gebruikt om notities in je code toe te voegen. Dit helpt bij het uitleggen van wat je code doet, zowel voor jezelf als voor anderen die de code later misschien moeten lezen. Commentaar wordt door de browser genegeerd en heeft dus geen invloed op de werking van het programma.

Er zijn twee soorten commentaar in JavaScript:

#### **1. Enkele regel commentaar (Single-line comment)**

Dit wordt aangegeven met een dubbele slash //. Alles wat na de dubbele slash komt, wordt door JavaScript genegeerd tot het einde van die regel.

```
// Dit is een enkele regel commentaar  
let x = 10; // Je kunt ook commentaar na een statement plaatsen
```

In dit voorbeeld worden zowel de eerste regel als alles na de dubbele slash in de tweede regel door de browser genegeerd. Het is handig voor korte notities of uitleg bij specifieke regels code.

#### **2. Meerdere regels commentaar (Multi-line comment)**

Voor langere stukken commentaar gebruik je /\* om het begin van het commentaar aan te geven en \*/ om het einde van het commentaar aan te geven. Dit type commentaar kan meerdere regels omvatten.

#### **Voorbeeld:**

```
/*  
Dit is een commentaar over meerdere regels.  
Het is handig als je een uitleg wilt geven die meer ruimte nodig heeft,  
of als je delen van je code tijdelijk wilt uitschakelen.  
*/  
let y = 20;
```

## 6. VARIABELEN EN DATATYPES

We weten reeds dat Javascript case-sensitive (hoofdlettergevoelig) werkt. Daarnaast dienen we ook rekening te houden met de regels voor opmaak voor variabelen. Zorg ervoor dat je variabelen een verstaanbare inhoud hebben. Bijvoorbeeld: `a+b` vervang je beter door `getal1 + getal2`

Variabelen worden voorafgegaan door het gereserveerde woord **var** in ES5.

Een var is undefined als datatype tot er een waarde aan wordt gegeven.

EEN VAR IS PER DEFINITIE EEN STRING WANNEER ER EEN SAMENVOEGING (CONCATENATIE) IS VAN VERSCHILLENDÉ DATATYPES. DIT OMDAT HET VERSCHIL NIET GEKEND IS OP DAT OGENBLIK TUSSEN BIJVOORBEELD EEN NUMBER EN EEN STRING.

Bijvoorbeeld: `var mijnNaam`

Hoe mogen variabelen NIET geschreven worden:

- GEEN gereserveerde worden gebruiken! (`var, break, ...`)
- NIET BEGINNEN met een cijfer
- GEEN gebruik van het koppelteken

GEEN gebruik van ongeldige tekens (`&, !, ....`)

De datatypes en primitives die we kunnen gebruiken zijn:

- Number: Gebruik voor numerieke waarden.
- String: Gebruik voor tekstgegevens.
- Boolean: Gebruik voor waar/onwaar waarden.
- *Undefined: Gebruik wanneer een variabele geen waarde heeft* (standaard).
- Null: Gebruik om aan te geven dat een variabele bewust leeg is.
- Object: Gebruik voor complexe gegevensstructuren met meerdere waarden.
- Array: Gebruik voor lijsten van waarden.

**null** en **undefined** zijn speciale primitive waarden in JavaScript, elk met hun eigen betekenis en gebruik.

**unsigned** is geen datatype in JavaScript; het is een concept dat voorkomt in andere programmeertalen maar niet in JavaScript bestaat.

Wanneer je gebruik wenst te maken van integers of floating-point getallen dan dien je steeds de variabelen om te zetten van een string naar deze data types.

Dit doe je met de respectievelijke **functies** `parseInt(variabele)` of `parseFloat(variabele)`.

Wanneer er niet naar een kommagetal (Float) omgezet kan worden dan krijg je een Nan foutmelding.

Nan = Not a number.

## PARSEINT VOORBEELD

Gebruik: Wanneer je een string wilt omzetten naar een geheel getal, zonder rekening te houden met eventuele decimale punten. Deze functie stopt met parsen zodra hij een niet-numeriek teken tegenkomt. Als de string met een letter begint, krijg je NaN terug.

**Let op:** Het negeert decimale punten en stopt bij niet-numerieke tekens, wat niet altijd gewenst is als je met volledige getallen werkt.

```
let strNum1 = "123";      // Geldige integer string
let strNum2 = "123.45";   // Decimale waarde, maar parseInt negeert de decimalen
let strNum3 = "123abc";   // Stopt bij het eerste niet-numerieke teken
let strNum4 = "abc123";   // Begint met letters, geeft NaN terug

let num1 = parseInt(strNum1); // Geeft 123
let num2 = parseInt(strNum2); // Geeft 123 (decimale waarde wordt genegeerd)
let num3 = parseInt(strNum3); // Geeft 123 (stopt bij de eerste niet-numerieke waarde)
let num4 = parseInt(strNum4); // Geeft NaN (start met een niet-numeriek teken)

console.log(num1); // Output: 123
console.log(num2); // Output: 123
console.log(num3); // Output: 123
console.log(num4); // Output: NaN
```

## PARSEFLOAT VOORBEELD

Gebruik: Wanneer je een string wilt omzetten naar een floating-point getal (met decimalen). Het stopt met parsen zodra het een niet-numeriek teken tegenkomt, maar behoudt de decimalen tot het eerste niet-numerieke teken.

```
let strNum1 = "123.45"; // Geldige floating-point string
let strNum2 = "123abc"; // Stopt bij het eerste niet-numerieke teken
let strNum3 = "abc123"; // Begint met letters, geeft NaN terug

let num1 = parseFloat(strNum1); // Geeft 123.45 (behoudt decimalen)
let num2 = parseFloat(strNum2); // Geeft 123 (stopt bij niet-numerieke tekens)
let num3 = parseFloat(strNum3); // Geeft NaN (start met een niet-numeriek teken)

console.log(num1); // Output: 123.45
console.log(num2); // Output: 123
console.log(num3); // Output: NaN
```

## NUMBER VOORBEELD

Gebruik: Wanneer je een string wilt omzetten naar een getal, en het maakt niet uit of het een geheel getal of een floating-point getal is. In tegenstelling tot parseInt() en parseFloat(), houdt Number() rekening met het volledige getal en negeert niet-numerieke tekens niet. Als de string niet volledig kan worden omgezet naar een getal, geeft het NaN terug.

```
let strNum1 = "123"; // Geldig integer
let strNum2 = "123.45"; // Geldig floating-point getal
let strNum3 = "123abc"; // Bevat niet-numerieke tekens, dus geeft NaN
let strNum4 = "abc123"; // Start met niet-numerieke tekens, geeft NaN

let num1 = Number(strNum1); // Geeft 123
let num2 = Number(strNum2); // Geeft 123.45 (houdt floating-point getallen)
let num3 = Number(strNum3); // Geeft NaN (bevat niet-numerieke tekens)
let num4 = Number(strNum4); // Geeft NaN (start met niet-numerieke tekens)

console.log(num1); // Output: 123
console.log(num2); // Output: 123.45
console.log(num3); // Output: NaN
console.log(num4); // Output: NaN
```

## Samenvatting van gebruik:

Functie	Omschrijving	Gebruiksscenario's
parseInt()	Converteert een string naar een <b>integer</b> (geheel getal) en negeert decimalen of niet-numerieke tekens	Gebruik wanneer je een string hebt en alleen een geheel getal wilt, ongeacht decimalen
parseFloat()	Converteert een string naar een <b>floating-point</b> getal (inclusief decimalen)	Gebruik wanneer je een string hebt en de gehele waarde, inclusief decimalen, wilt behouden
Number()	Converteert een string naar een getal (integer of floating-point), maar faalt volledig bij niet-numerieke tekens	Gebruik wanneer je zeker weet dat de string alleen een geldig nummer bevat zonder niet-numerieke tekens

Open terug **script.js** onder de **map js1** en pas als volgt aan:

### Number

Gebruik: Wanneer je werkt met getallen, zowel gehele getallen als decimale getallen, zoals prijzen, leeftijden, lengtes, etc.

*Je kan ook nagaan welk datatype wordt weergegeven. We gebruiken hiervoor de functie **typeof**.*

```
let age = 25; // Geheel getal  
let price = 99.99; // Decimaal  
console.log(typeof age); // Output: "number"  
console.log(typeof price); // Output: "number"
```

### String

Gebruik: Voor tekst of tekenreeksen, zoals namen, berichten, of elke tekst die je wilt weergeven of opslaan.

```
let greeting = "Hallo wereld!";  
let name = 'John';  
console.log(typeof greeting); // Output: "string"  
console.log(typeof name); // Output: "string"
```

### Boolean

Gebruik: Wanneer je een binaire waarde (true of false) nodig hebt, bijvoorbeeld voor statuswaarden zoals ingelogd of niet-ingelogd, ja of nee vragen, of conditiechecks.

```
let isLoggedIn = true;  
let hasAccess = false;  
console.log(typeof isLoggedIn); // Output: "boolean"  
console.log(typeof hasAccess); // Output: "boolean"
```

### Undefined

Gebruik: Wanneer je een variabele declareert, maar er nog geen waarde aan hebt toegekend. Dit betekent dat de variabele nog geen specifieke waarde heeft.

```
let undefinedVariable;  
console.log(typeof undefinedVariable); // Output: "undefined"
```

### Null

Gebruik: Wanneer je expliciet wilt aangeven dat een variabele geen waarde heeft of dat iets niet bestaat. Dit is handig om een variabele te resetten of een "lege" waarde te geven.

```
let emptyValue = null;  
console.log(typeof emptyValue); // Output: "object" (dit is een bekend eigenaardigheid in JavaScript)
```

### Undefined assigned

Gebruik: Wanneer je een variabele expliciet wilt toewijzen aan undefined, hoewel dit zelden nodig is omdat JavaScript dit automatisch doet als je geen waarde toewijst.

```
let assignedUndefined = undefined;  
console.log(typeof assignedUndefined); // Output: "undefined"
```

## Object

Gebruik: Wanneer je meerdere eigenschappen of waarden aan één entiteit wilt toewijzen. Objecten zijn ideaal voor complexe gegevens zoals een gebruiker, product, of configuraties.

```
let person = {  
    name: "Jane",  
    age: 30,  
    isStudent: false  
};  
console.log(typeof person); // Output: "object"
```

## Array

Gebruik: Voor een gestructureerde lijst van gegevens, zoals een reeks items, nummers, of objecten. Arrays worden gebruikt wanneer je een collectie van gegevens wilt opslaan die je later door kunt lopen of bewerken.

```
let fruits = ["Apple", "Banana", "Cherry"];  
console.log(typeof fruits); // Output: "object" (arrays zijn een subtype van objecten in  
JavaScript)  
console.log(Array.isArray(fruits)); // Output: true
```

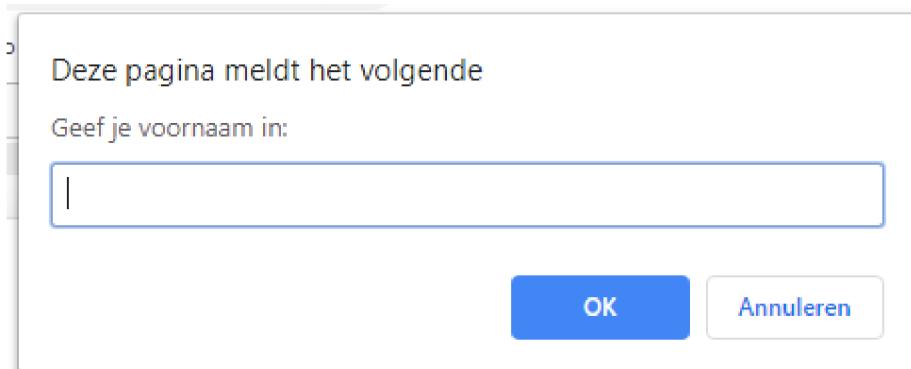
We kunnen variabelen ook laten inlezen door de gebruiker d.m.v. de **prompt** functie.

Voeg nog een laatste lijn toe en probeer maar uit in de browser:

```
var voorNaam = prompt('Geef je voornaam in:');  
console.log(voorNaam);
```

Een andere optie is de **alert** functie, die enkel een melding op het scherm in een popup venster weer zal geven.

## RESULTAAT



Een andere mogelijkheid om een vraag te stellen aan de gebruiken is **window.confirm** ipv **window.prompt**. Aangezien confirm en prompt globale variabelen zijn dienen we window. niet te laten voorafgaan aan de methode. Een confirm heeft in tegenstelling tot de prompt als return een boolean, nl. true or false.

Voorbeeld:

```
if(window.confirm("Ben je zeker dat je dit wil verwijderen?")){
    deleteIets(id);
}
```



Een alert is de derde mogelijkheid. Hiermee kan je enkel een bericht (message) weergeven aan de gebruiker.



## CODE

```
console.log(typeof voorNaam);
```

### **NaN = NOT a number.**

NaN staat voor **Not-a-Number** en komt voor wanneer een berekening of operatie in JavaScript geen geldig nummer als resultaat kan opleveren. Dit gebeurt vaak bij mislukte wiskundige operaties of bij het proberen om niet-numerike waarden om te zetten naar nummers.

### **Voorbeeld 1: Wanneer krijg je NaN?**

Stel dat je probeert een getal te vermenigvuldigen met een niet-numerike waarde, zoals een string of een niet-definieerbare variabele.

```
let result = 5 * "abc"; // Proberen een getal met een string te vermenigvuldigen  
console.log(result); // Output: NaN
```

### **Voorbeeld 2: isNaN() gebruiken om te controleren**

De **isNaN()** functie wordt gebruikt om te controleren of een waarde niet een geldig nummer is. Als de waarde geen nummer is (zoals NaN), zal deze functie true retourneren.

```
let value1 = 5 * "abc";  
let value2 = 5 * 2;  
  
console.log(isNaN(value1)); // Output: true, want de waarde is NaN  
console.log(isNaN(value2)); // Output: false, want de waarde is een geldig nummer (10)
```

In het eerste voorbeeld is de uitkomst true omdat `5 * "abc"` geen geldig getal oplevert. In het tweede voorbeeld is de uitkomst false omdat `5 * 2` een geldig getal oplevert.

## 7. OPERATOREN

### 7.1. REKENKUNDIGE OPERATOREN

De basis rekenkundige operatoren in JavaScript zijn:

1. Optelling (+): Wordt gebruikt om twee of meer waarden bij elkaar op te tellen.
2. Aftrekking (-): Wordt gebruikt om een waarde van een andere waarde af te trekken.
3. Vermenigvuldiging (\*): Wordt gebruikt om twee of meer waarden met elkaar te vermenigvuldigen.
4. Deling (/): Wordt gebruikt om een waarde door een andere waarde te delen.
5. Modulus (%): Geeft de rest van een deling terug.

Naast deze basisoperatoren zijn er ook enkele geavanceerde rekenkundige operatoren:

1. Exponentiatie (\*\*): Wordt gebruikt om een getal tot de macht van een ander getal te verheffen.
2. Incrementeer (++): Verhoogt de waarde van een variabele met 1.
3. Decrementeer (--) : Verlaagt de waarde van een variabele met 1.
4. Negatie (-): Verandert het teken van een getal (maakt positief negatief en omgekeerd).

We voegen volgende code toe aan ons JS1 bestand.

#### Optelling (+):

```
let a = 10;  
let b = 5;  
let som = a + b;  
console.log("Optelling: " + som); // Output: 15
```

#### Aftrekking (-):

```
let verschil = a - b;  
console.log("Aftrekking: " + verschil); // Output: 5
```

#### Vermenigvuldiging (\*):

```
let product = a * b;  
console.log("Vermenigvuldiging: " + product); // Output: 50
```

#### Deling (/):

```
let quotient = a / b;  
console.log("Deling: " + quotient); // Output: 2
```

### **Modulus (%):**

```
let rest = a % b;  
console.log("Modulus: " + rest); // Output: 0 (want 10 gedeeld door 5 geeft geen rest)  
  
let result = 10.5 % 3;  
console.log(result); // Output: 1.5
```

### **Exponent (\*\*):**

```
let macht = a ** 2; // Verheft a tot de macht van 2  
console.log("Exponentiatie: " + macht); // Output: 100
```

### **Increment (++):**

```
let c = 5;  
c++;  
console.log("Incrementeer: " + c); // Output: 6
```

### **Decrement (- -):**

```
let d = 5;  
d--;  
console.log("Decrementeer: " + d); // Output: 4
```

### **Negatie (-):**

```
let negatief = -a; // Verandert het teken van a  
console.log("Negatie: " + negatief); // Output: -10
```

Volledige code:

```
let getal1 = 10;  
let getal2 = 4;  
  
let som = getal1 + getal2;  
let verschil = getal1 - getal2;  
let product = getal1 * getal2;  
let quotient = getal1 / getal2;  
let rest = getal1 % getal2;  
let macht = getal1 ** 2;  
  
console.log("Optelling: " + som); // Output: 14  
console.log("Aftrekking: " + verschil); // Output: 6  
console.log("Vermenigvuldiging: " + product); // Output: 40  
console.log("Deling: " + quotient); // Output: 2.5  
console.log("Modulus: " + rest); // Output: 2  
console.log("Exponentiatie: " + macht); // Output: 100  
  
let waarde = 5;  
waarde++;  
console.log("Incrementeer: " + waarde); // Output: 6  
waarde--;  
console.log("Decrementeer: " + waarde); // Output: 5  
  
let negatiefGetal = -getal1;  
console.log("Negatie: " + negatiefGetal); // Output: -10
```

### 7.1.1. VOORRANGSREGELS

De wiskundige voorrangsregels in JavaScript volgen dezelfde principes als in de wiskunde, bekend als de BIDMAS-regel (in het Engels soms als PEMDAS):

- B: Haakjes (Brackets / Parentheses)
- I: Machtverheffingen (Indices / Exponentiation)
- D: Deling (Division)
- M: Vermenigvuldiging (Multiplication)
- A: Optelling (Addition)
- S: Aftrekking (Subtraction)

JavaScript volgt deze volgorde wanneer het berekeningen uitvoert. Als meerdere operatoren in één expressie voorkomen, wordt eerst de operatie met de hoogste prioriteit uitgevoerd. Haakjes kunnen worden gebruikt om delen van de berekening te groeperen en krijgen altijd voorrang boven andere operatoren.

Samenvatting van de volgorde:

- 1 Haakjes (())
- 2 Machtverheffing (\*\*)
- 3 Vermenigvuldiging, deling en modulus (\*, /, %)
- 4 Optelling en aftrekking (+, -)
- 5 Incrementeer (++), Decrementeer (--) en negatie (-)
- 6 Toewijzingsoperatoren (=, +=, -=)

```
let getal1 = 8;  
let getal2 = 4;  
  
console.log(getal1 + getal2 * getal1); // Output: 40  
console.log((getal1 + getal2) * getal1); // Output: 96
```

### 7.1.2. SHORTHAND NOTATIES

Shorthand notaties worden gebruikt om code korter en eenvoudiger te maken. Ze zijn in feite een combinatie van een toewijzingsoperator (`=`) en een bewerkingsoperator (zoals `+`, `-`, `*`, etc.). Dit bespaart tijd en maakt je code overzichtelijker, vooral bij veel voorkomende bewerkingen.

De algemene vorm van shorthand notaties is:

`x <operator>= y;`  
of  
`x += y;`

Dit betekent:

`x = x <operator> y;`

of

`x = x + y;`

```
let x = 5;  
let y = 6;  
x += y; // x wordt 5 + 6 = 11  
console.log(x); // Output: 11
```

## 7.2. LOGISCHE OPERATOREN

Logische operatoren worden gebruikt om meerdere voorwaarden te evalueren. Ze geven een boolean resultaat terug, wat betekent dat het antwoord altijd true of false is. Logische operatoren worden vaak gebruikt in vergelijking met waarden en bij het controleren van meerdere condities.

Hier is een overzicht van de belangrijkste logische operatoren in JavaScript:

Operator	Beschrijving
&& (AND)	Retourneert true als <b>beide</b> voorwaarden waar zijn. Als een van beide onwaar is, retourneert het false.
`	Gebruikt voor template literals. Hiermee kun je strings met variabelen interpoleren of multi-line strings schrijven. Voorbeeld: let name = 'John'; let greeting = `Hallo, \${name};`
! (NOT)	Keert de waarheid van een voorwaarde om: true wordt false en andersom.
==	Vergelijkt twee waarden. Retourneert true als de waarden gelijk zijn, maar maakt geen onderscheid in datatype.
===	Vergelijkt zowel de waarde als het datatype. Retourneert true als <b>zowel de waarde als het datatype</b> gelijk zijn.
!=	Controleert of twee waarden verschillend zijn. Retourneert true als de waarden ongelijk zijn.
!==	Controleert of twee waarden <b>of het datatype</b> verschillend zijn. Retourneert true als de waarden of het datatype ongelijk zijn.
>	Groter dan: retourneert true als de linkerkant groter is dan de rechterkant.
<	Kleiner dan: retourneert true als de linkerkant kleiner is dan de rechterkant.
>=	Groter dan of gelijk aan: retourneert true als de linkerkant groter of gelijk is aan de rechterkant.
<=	Kleiner dan of gelijk aan: retourneert true als de linkerkant kleiner of gelijk is aan de rechterkant.

### && (AND) Operator:

```
let a = true;  
let b = false;  
console.log(a && b); // Output: false (omdat niet beide waarden waar zijn)
```

### ``` (Backtick) Operator:

```
let naam = "John";  
let begroeting = `Hallo, ${naam}! Hoe gaat het?`;  
console.log(begroeting); // Output: "Hallo, John! Hoe gaat het?"
```

### || (OR) Operator:

```
let a = true;  
let b = false;  
console.log(a || b); // Output: true (omdat ten minste één van de waarden waar is)
```

### **! (NOT) Operator:**

```
let a = true;  
console.log(!a); // Output: false (omdat de waarheid van `a` omgedraaid wordt)
```

### **== en === Vergelijking:**

```
let x = 5;  
let y = '5';  
  
console.log(x == y); // Output: true (vergelijking van de waarde, datatype wordt genegeerd)  
console.log(x === y); // Output: false (zowel waarde als datatype moeten gelijk zijn)
```

### **!= en !== Vergelijking:**

```
let x = 5;  
let y = '5';  
  
console.log(x != y); // Output: false (waarde is hetzelfde)  
console.log(x !== y); // Output: true (verschillend datatype)
```

### **Vergelijkingen (>, <, >=, <=):**

```
let num1 = 10;  
let num2 = 20;  
  
console.log(num1 > num2); // Output: false (10 is niet groter dan 20)  
console.log(num1 <= num2); // Output: true (10 is kleiner dan of gelijk aan 20)
```

## 8. CONTROLESTRUCTUREN

### 8.1. IF - ELSE IF - ELSE STATEMENT

De if-else-structuur is een van de meest gebruikte controlestructuren in JavaScript. Hiermee kun je beslissingen nemen in je code en verschillende acties uitvoeren op basis van voorwaarden. De basisvorm van een if-else-structuur ziet er als volgt uit:

#### Basis Syntax:

```
if (conditie) {  
    // Code die wordt uitgevoerd als de conditie waar (true) is  
} else {  
    // Code die wordt uitgevoerd als de conditie onwaar (false) is  
}
```

#### IF STATEMENT

Het if-statement voert code uit als de gegeven conditie waar is.

```
let getal1 = 10;  
let getal2 = 5;
```

```
if (getal1 > getal2) {  
    console.log(getal1 + ' is groter dan ' + getal2);  
}
```

Uitleg: In dit voorbeeld wordt gecontroleerd of getal1 groter is dan getal2. Als dat zo is (wat waar is, aangezien 10 groter is dan 5), wordt het bericht in de console weergegeven.

#### ELSE STATEMENT

Het else-statement wordt uitgevoerd wanneer de conditie in de if-statement onwaar is.

```
let getal1 = 10;  
let getal2 = 20;
```

```
if (getal1 > getal2) {  
    console.log(getal1 + ' is groter dan ' + getal2);  
} else {  
    console.log(getal1 + ' is kleiner of gelijk aan ' + getal2);  
}
```

## ELSE IF STATEMENT

Met het else if-statement kun je meerdere voorwaarden testen. Als de eerste if-conditie onwaar is, test de else if een andere conditie.

```
let getal1 = 10;  
let getal2 = 10;
```

```
if (getal1 > getal2) {  
    console.log(getal1 + ' is groter dan ' + getal2);  
} else if (getal1 == getal2) {  
    console.log(getal1 + ' is gelijk aan ' + getal2);  
} else {  
    console.log(getal1 + ' is kleiner dan ' + getal2);  
}
```

### of met literals (backticks)

```
if (getal1 > getal2) {  
    console.log(`${getal1} is groter dan ${getal2}`);  
} else if (getal1 == getal2) {  
    console.log(`${getal1} is gelijk aan ${getal2}`);  
} else {  
    console.log(`${getal1} is kleiner dan ${getal2}`);  
}
```

#### Voorbeeld 1: Leeftijdscontrole

In dit voorbeeld wordt gecontroleerd of de gebruiker een volwassene, tiener, of kind is op basis van hun ingevoerde leeftijd.

```
let leeftijd = prompt('Voer je leeftijd in:');
```

```
if (leeftijd >= 18) {  
    console.log('Je bent een volwassene.');//  
} else if (leeftijd < 18 && leeftijd >= 12) {  
    console.log('Je bent een tiener.');//  
} else {  
    console.log('Je bent een kind.');//  
}
```

#### Voorbeeld 2: Beroep op basis van gebruikersinvoer

- De gebruiker wordt gevraagd hun naam en beroep in te voeren.  
Afhankelijk van de invoer toont het programma een aangepast bericht.
- Dit is een voorbeeld van een if - else if - else structuur waarin meerdere condities worden gecontroleerd

```
let naam = prompt('Geef uw naam in:');
```

```
let beroep = prompt('Geef uw beroep in, maak een keuze: bediende, arbeider, werkzoekend');
```

```
if (beroep === 'bediende') {  
    console.log(`Het beroep van ${naam} is ${beroep}`);  
} else if (beroep === 'arbeider') {  
    console.log(`Het beroep van ${naam} is ${beroep}`);  
} else if (beroep === 'werkzoekend') {  
    console.log(`${naam} is momenteel op zoek naar werk.`);  
} else {  
    console.log('Beroep onbekend.');//  
}
```

### 8.1.1. SHORTHAND NOTATIE (TERNARY OPERATOR)

Het if-else if - else statement kan ook korter worden geschreven, dit noemen we de shorthandnotatie.

Shorthandnotatie of voorwaardelijke notatie:

**(conditie) ? waar : onwaar;**

#### Zonder shorthand

```
let leeftijd = prompt('Voer je leeftijd in:');

if (leeftijd >= 18) {
    console.log('Je bent een volwassene.');
} else if (leeftijd < 18 && leeftijd >= 12) {
    console.log('Je bent een tiener.');
} else {
    console.log('Je bent een kind.');
}
```

#### Met shorthand

```
let leeftijd = prompt('Voer je leeftijd in:');

console.log(
getal1 > getal2
? `${getal1} is groter dan ${getal2}`
: getal1 == getal2
? `${getal1} is gelijk aan ${getal2}`
: `${getal1} is kleiner dan ${getal2}`
);
```

## 8.2. SWITCH

Een **switch statement** wordt gebruikt wanneer je een variabele wilt controleren tegen meerdere mogelijke waarden. Het is vaak gemakkelijker en leesbaarder in gebruik dan een lange reeks **if-else if-statements** wanneer je met meer dan drie opties werkt.

### Basis Syntax:

```
switch (expressie) {  
    case waarde1:  
        // code om uit te voeren als expressie overeenkomt met waarde1  
        break;  
    case waarde2:  
        // code om uit te voeren als expressie overeenkomt met waarde2  
        break;  
    // meer cases indien nodig  
    default:  
        // code die wordt uitgevoerd als geen enkele case overeenkomt  
}
```

- **switch:** Het begint met een switch-woord, gevolgd door de expressie die je wilt evalueren (vaak een variabele).
- **case:** Elke case vergelijkt de expressie met een mogelijke waarde. Als de expressie overeenkomt met de waarde in de case, wordt de code in die case uitgevoerd.
- **break:** De break-instructie zorgt ervoor dat de code stopt met uitvoeren nadat de overeenkomstige case is uitgevoerd. Als je geen break gebruikt, zal de code doorlopen naar de volgende case, zelfs als die niet overeenkomt.
- **default:** De default-sectie is optioneel en wordt uitgevoerd als geen van de case-waarden overeenkomt met de expressie.

Voorbeeld van een switch-statement:

```
let onderwijs = 'vdab';  
  
switch (onderwijs) {  
    case 'vdab':  
        console.log('Gegeven door de vdab');  
        break;  
    case 'syntra':  
        console.log('Gegeven door syntra');  
        break;  
    default:  
        console.log('Gegeven door een andere instelling');  
}
```

Voorbeeld: Keuze van een dag

```
let dag = 3;

switch (dag) {
  case 1:
    console.log('Maandag');
    break;
  case 2:
    console.log('Dinsdag');
    break;
  case 3:
    console.log('Woensdag');
    break;
  case 4:
    console.log('Donderdag');
    break;
  case 5:
    console.log('Vrijdag');
    break;
  default:
    console.log('Onbekende dag');
}
```

### Vergelijking: Switch vs. If-Else If-Else

Kenmerk	Switch	If-Else If-Else
Gebruik	Voor het controleren van een enkele waarde tegen meerdere mogelijke vaste waarden	Voor het controleren van meerdere condities die niet beperkt zijn tot vaste waarden
Leesbaarheid	Beter leesbaar bij meer dan drie opties	Wordt snel onleesbaar met veel opties
Flexibiliteit	Alleen geschikt voor gelijkheidscontroles (==)	Kan elke voorwaarde controleren (>, <, ==, !=, etc.)
Uitvoersnelheid	Efficiënt bij veel vaste waarden	Minder efficiënt bij veel voorwaarden

Een switch statement is handig wanneer je een variabele wilt vergelijken met meerdere vaste waarden en eenvoudig een actie wilt uitvoeren op basis van die waarde. Het biedt betere leesbaarheid en prestaties bij eenvoudige gelijkheidscontroles met veel mogelijke opties. Gebruik echter een if-else if-structuur wanneer je complexere logica moet implementeren of wanneer je niet beperkt bent tot exacte gelijkheidscontroles.

## 9. ITERATIES (LOOPS OF LUSSEN)

In JavaScript worden loops gebruikt om een codeblok herhaaldelijk uit te voeren, totdat een bepaalde voorwaarde niet meer voldoet. Dit is nuttig wanneer je bijvoorbeeld een actie meerdere keren wilt herhalen, zoals het itereren door een lijst met items of het uitvoeren van een berekening een specifiek aantal keren.

JavaScript biedt verschillende soorten loops, waaronder:

- for loop
- while loop
- do-while loop
- for-in en for-of loops (voor itereren door objecten en arrays)

### 9.1.1. FOR LOOP

De for-loop is een van de meest gebruikte en meest veelzijdige loops in JavaScript. Het geeft je volledige controle over de startwaarde, eindconditie, en de wijziging (increment/decrement) die bij elke iteratie moet plaatsvinden.

#### Wanneer gebruik je een for-loop?

- **Bekende herhalingen:** Gebruik een for-loop wanneer je vooraf weet hoe vaak een stuk code moet worden herhaald. Bijvoorbeeld, als je wilt itereren over een array, of als je een exact aantal herhalingen moet uitvoeren.
- **Standaard iteraties:** Een for-loop is handig wanneer je een reeks doorloopt met een vaste regelmaat, zoals het verhogen of verlagen van een teller.

#### Basis Syntax:

```
for (initialisatie; conditie; update) {  
    // Code om uit te voeren tijdens elke iteratie  
}
```

- **Initialisatie:** Dit is de startwaarde van de teller. Meestal wordt hier een variabele zoals `i` geïnitialiseerd, bijvoorbeeld `let i = 0`.
- **Conditie:** Dit is de voorwaarde die controleert of de loop moet blijven draaien. Zolang deze conditie waar is (`true`), blijft de loop doorlopen. Zodra het onwaar is (`false`), stopt de loop.
- **Update:** Dit is de wijziging die bij elke iteratie plaatsvindt. Meestal is dit een increment (`i++`), maar het kan ook een andere operatie zijn, zoals `i += 2` of `i--` (decrement).

Eenvoudig Voorbeeld:

In dit voorbeeld start de loop met `i = 1` en blijft draaien zolang `i` kleiner of gelijk is aan 5. Bij elke iteratie wordt `i` met 1 verhoogd (`i++`), en de string wordt in de console afdrukkt.

```
for (let i = 1; i <= 5; i++) {  
    console.log(`Dit is iteratie nummer ${i}`);  
}
```

Output:

Dit is iteratie nummer 1  
Dit is iteratie nummer 2  
Dit is iteratie nummer 3  
Dit is iteratie nummer 4  
Dit is iteratie nummer 5

### Voorbeeld 1:Standaard oplopende for-loop

Deze loop begint bij `i = 1` en blijft lopen totdat `i` groter is dan 10. Elke keer wordt `i` met 1 verhoogd.

```
for (let i = 1; i <= 10; i++) {  
    console.log(i);  
}
```

Output:

1  
2  
3  
4  
5  
6  
7  
8  
9  
10

## Voorbeeld 2: For-loop met een stapgrootte van 2

Hier wordt i telkens met 2 verhoogd. Dit zorgt ervoor dat alleen even getallen tussen 0 en 10 worden afgedrukt.

```
for (let i = 0; i <= 10; i += 2) {  
    console.log(i);  
}
```

Uitvoer:

```
0  
2  
4  
6  
8  
10
```

## Voorbeeld 3: Aflopende for-loop

Deze loop begint bij 10 en loopt naar beneden (decrement) tot 1. Elke keer wordt i met 1 verlaagd.

```
for (let i = 10; i > 0; i--) {  
    console.log(i);  
}
```

Output:

```
10  
9  
8  
7  
6  
5  
4  
3  
2  
1
```

#### Voorbeeld 4: Nested Loops (Geneste lussen)

Hier hebben we een loop binnen een andere loop. De buitenste loop voert 3 iteraties uit en elke keer dat deze draait, voert de binnennste loop 2 iteraties uit.

```
for (let i = 1; i <= 3; i++) {  
    console.log(`Buitenste loop: iteratie ${i}`);  
  
    for (let j = 1; j <= 2; j++) {  
        console.log(` Binnenste loop: iteratie ${j}`);  
    }  
}
```

Output:

```
Buitenste loop: iteratie 1  
    Binnenste loop: iteratie 1  
    Binnenste loop: iteratie 2  
Buitenste loop: iteratie 2  
    Binnenste loop: iteratie 1  
    Binnenste loop: iteratie 2  
Buitenste loop: iteratie 3  
    Binnenste loop: iteratie 1  
    Binnenste loop: iteratie 2
```

#### 9.1.2. WHILE LOOP

Gebruik een while-loop als je de iteraties wilt laten doorgaan totdat aan een bepaalde voorwaarde is voldaan. Dit kan handig zijn als je niet vooraf weet hoeveel keer je wilt herhalen.

#### Basis Syntax:

```
while(conditie){  
    code  
}
```

```
let i = 1;  
while (i <= 5) {  
    console.log(i);  
    i++;  
}
```

### 9.1.3. DO WHILE LOOP

Deze loop lijkt op de while-loop, maar voert altijd ten minste één keer de code uit, ongeacht de voorwaarde.

```
let i = 1;  
do {  
    console.log(i);  
    i++;  
} while (i <= 5);
```

### 9.1.4. FOR OF LOOP

Gebruik de for-of loop om eenvoudig door **arrays** te itereren. Arrays komen in één van de volgende hoofdstukken nog aan bod. Maar om toch een voorbeeld te geven hier reeds een klein stukje code.

```
let fruits = ['appel', 'banaan', 'peer'];  
for (let fruit of fruits) {  
    console.log(fruit);  
}
```

### 9.1.5. FOR IN LOOP

Gebruik de for-in loop om door **objecten** te itereren (gebruikt voor het itereren door de eigenschappen van een object).

```
let persoon = {naam: "John", leeftijd: 30};  
for (let eigenschap in persoon) {  
    console.log(`{$eigenschap}: ${persoon[eigenschap]}`);  
}
```

## 10. ARRAYS

Een **array** in JavaScript is een datastructuur waarmee je meerdere elementen van dezelfde of verschillende datatypes kunt opslaan binnen één variabele. Dit is handig omdat je met een array niet meerdere variabelen hoeft te creëren voor elk individueel element, en je de data eenvoudig kunt ordenen en opvragen via een index of sleutel.

### Voordelen van Arrays:

- **Efficiëntie:** Je kunt een grote hoeveelheid data opslaan in een enkele variabele.
- **Toegang via index:** Je kunt eenvoudig gegevens opvragen en manipuleren op basis van hun positie (index).
- **Flexibiliteit:** Arrays in JavaScript kunnen verschillende datatypes bevatten, zoals strings, getallen, booleans, en zelfs andere arrays.

### Basisconcept van Arrays:

Een array heeft een **index** die altijd begint bij **0**. Dus het eerste element van een array zit op index 0, het tweede element op index 1, enzovoort.

### Voorbeeld van een eenvoudige array:

```
let studenten = ['Tom', 'Pieter', 'Els', 'Bart'];
console.log(studenten[0]); // Toont: 'Tom'
console.log(studenten[3]); // Toont: 'Bart'
```

In dit voorbeeld bevat de array vier namen. De namen worden opgeslagen op hun respectieve indexnummers. Tom staat op index 0, Pieter op index 1, enzovoort.

### Soorten Arrays in JavaScript:

**Genummerde (Indexed) Arrays:** Een array waarin de elementen zijn gerangschikt op numerieke indexen, beginnend vanaf 0.

#### Voorbeeld:

```
let nummers = [10, 20, 30, 40];
console.log(nummers[1]); // Toont: 20
```

**Associatieve Arrays:** Dit concept wordt niet direct ondersteund in JavaScript. In plaats daarvan gebruik je objecten om sleutels toe te wijzen aan waarden, zoals dit:

#### Voorbeeld (gebruik object in plaats van array):

```
let persoon = {
  naam: 'Tom',
  leeftijd: 25,
  beroep: 'programmeur'
};
console.log(persoon['naam']); // Toont: Tom
```

**Multidimensionale Arrays:** Dit zijn arrays binnen arrays. Elke index van de buitenste array bevat weer een nieuwe array.

**Voorbeeld:**

```
let matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
];
console.log(matrix[1][2]); // Toont: 6
```

**Voorbeeld:**

```
let cursisten = ['Tom', 'Pieter', 'Els', 'Bart'];
let cursusJaar = new Array(2017, 2018, 2019);

console.log(cursisten);      // ['Tom', 'Pieter', 'Els', 'Bart']
console.log(cursusJaar);     // [2017, 2018, 2019]

// Wijzigen van array-elementen
cursisten[0] = 'Pieter';
console.log(cursisten);      // ['Pieter', 'Pieter', 'Els', 'Bart']

// Toevoegen aan einde van array
cursisten.push('Marieke');
console.log(cursisten);      // ['Pieter', 'Pieter', 'Els', 'Bart', 'Marieke']

// Verwijderen van het einde van array
cursisten.pop();
console.log(cursisten);      // ['Pieter', 'Pieter', 'Els', 'Bart']

// Toevoegen aan het begin van array
cursisten.unshift('Thomas');
console.log(cursisten);      // ['Thomas', 'Pieter', 'Pieter', 'Els', 'Bart']

// Verwijderen van het begin van array
cursisten.shift();
console.log(cursisten);      // ['Pieter', 'Pieter', 'Els', 'Bart']

// Ophalen van indexnummer
console.log(cursisten.indexOf('Bart')); // Toont: 3
```

[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array)

Voorbeelden van deze functies zijn:

**join()**: Converteert alle elementen in een array naar een string, gescheiden door een bepaalde delimiter (zoals een komma of een spatie).

```
let mijnArray = [10, 20, 30, 40];
console.log(mijnArray.join());      // Output: "10,20,30,40"
console.log(mijnArray.join(' - ')); // Output: "10 - 20 - 30 - 40"
```

**reverse()**: Keert de volgorde van de elementen in een array om, zodat het laatste element het eerste wordt.

```
let mijnArray = [10, 20, 30, 40];
mijnArray.reverse();
console.log(mijnArray); // Output: [40, 30, 20, 10]
```

**sort()**: Sorteert de elementen van een array alfabetisch of numeriek.

```
let kleuren = ['rood', 'geel', 'blauw'];
kleuren.sort();
console.log(kleuren); // Output: ['blauw', 'geel', 'rood']
```

**concat()**: Voegt twee of meer arrays samen en retourneert een nieuwe array.

```
let array1 = [1, 2, 3];
let array2 = [4, 5, 6];
let nieuweArray = array1.concat(array2);
console.log(nieuweArray); // Output: [1, 2, 3, 4, 5, 6]
```

**slice()**: Retourneert een deel van een array zonder de originele array te wijzigen. Het neemt twee parameters: de startindex en de eindindex (de eindindex wordt niet inbegrepen).

```
let mijnArray = [10, 20, 30, 40, 50];
let subArray = mijnArray.slice(1, 3);
console.log(subArray); // Output: [20, 30]
```

**splice()**: Voegt elementen toe aan een array of verwijdert ze op een bepaalde positie. Het neemt ten minste twee parameters: de index om te starten, en het aantal te verwijderen elementen.

```
let mijnArray = [10, 20, 40];
mijnArray.splice(2, 0, 30); // Voegt 30 toe op index 2
console.log(mijnArray); // Output: [10, 20, 30, 40]
```

**toString()**: Converteert een array naar een string waarbij de elementen gescheiden worden door een komma.

```
let mijnArray = [10, 20, 30, 40];
console.log(mijnArray.toString()); // Output: "10,20,30,40"
```

## 10.1. ARRAYS EN FOR-LOOP

De for-loop is een handige manier om door een array te itereren. Hiermee kun je elk element in een array een voor een doorlopen, waarbij je via de index toegang krijgt tot elk element.

Hieronder zie je dat het conditie gedeelte wordt bepaald d.m.v. de `length` methode. Herinner je dat deze functie het aantal elementen van een array kan weergeven. Ieder afzonderlijk element uit de array wordt vervolgens weergegeven door de teller `[i]` te gebruiken (**cursisten[i]**).

html:

```
<h1>Arrays</h1>
<ul id="cursisten"></ul>

<script>
let cursisten = ['Tom', 'Tim', 'Bart', 'Els'];

for (let i = 0; i < cursisten.length; i++) { // Correct begin op index 0
    document.querySelector('#cursisten').innerHTML += `<li>${cursisten[i]}</li>`;
}
</script>
```

### Toevoeging: Array manipulatie

Om de oefening verder uit te breiden en te leren hoe je arrays kunt manipuleren, voegen we methoden toe zoals `push()` en `splice()` om de array dynamisch te veranderen tijdens de iteratie.

```
let cursisten = ['Tom', 'Tim', 'Bart', 'Els'];
cursisten.push('Sara'); // Voegt een nieuw element toe
cursisten.splice(2, 1); // Verwijdt 1 element vanaf index 2 (Bart)

for (let i = 0; i < cursisten.length; i++) {
    document.querySelector('#cursisten').innerHTML += `<li>${cursisten[i]}</li>`;
}
```

## 10.2. ARRAYS EN WHILE-LOOP

Een while-loop werkt goed wanneer je door een array wilt itereren, maar je niet vooraf weet hoeveel iteraties er precies nodig zijn of als de loop op basis van een bepaalde voorwaarde moet stoppen.

html:

```
<h1>Arrays</h1>
<ul id="cursisten"></ul>

<script>
let cursisten = ['Tom', 'Tim', 'Bart', 'Els'];
let i = 0;

while (i < cursisten.length) {
    document.querySelector('#cursisten').innerHTML += `<li>${cursisten[i]}</li>`;
    i++;
}
</script>
```

Uitleg:

i = 0: De teller i begint bij 0 en blijft itereren zolang i kleiner is dan de lengte van de array.

while (i < cursisten.length): Dit zorgt ervoor dat de loop stopt wanneer alle elementen zijn doorlopen.

### Toevoeging: Controleer voor lege arrays

We kunnen de while-loop uitbreiden om een controle toe te voegen, zodat het systeem waarschuwt als de array leeg is.

```
if (cursisten.length === 0) {
    document.querySelector('#cursisten').innerHTML = `<li>Geen cursisten beschikbaar</li>`;
} else {
    let i = 0;
    while (i < cursisten.length) {
        document.querySelector('#cursisten').innerHTML += `<li>${cursisten[i]}</li>`;
        i++;
    }
}
```

Controle voor lege arrays: Als de array leeg is, wordt een bericht weergegeven in plaats van een lege lijst.

### 10.3. ARRAYS EN FOREACH-LOOP

De forEach-loop is een handige manier om door een array te itereren zonder expliciet een teller (i) te hoeven gebruiken. Het biedt een meer leesbare manier om door alle elementen van een array te lopen, en het stelt je in staat zowel de index als het item zelf te gebruiken in de loop.

html:

```
<h1>Arrays</h1>
<ul id="cursisten"></ul>

<script>
let cursisten = ['Tom', 'Tim', 'Bart', 'Els'];

cursisten.forEach(function(item, index) {
    document.querySelector('#cursisten').innerHTML += `<li>${index}: ${item}</li>`;
});
</script>
```

#### Uitleg:

- **forEach()**: Deze methode voert een functie uit voor elk element in de array, waarbij zowel het element (item) als de index beschikbaar zijn.
- **item en index**: item is het daadwerkelijke element in de array en index is de positie van dat element.

### Toevoeging: Meer geavanceerde manipulaties met forEach()

We kunnen de forEach() uitbreiden om meer complexe taken uit te voeren, zoals het conditioneel aanpassen van elementen.

```
let cursisten = ['Tom', 'Tim', 'Bart', 'Els'];

cursisten.forEach(function(item, index) {
    if (index % 2 === 0) { // Markeer elk even indexelement als 'belangrijk'
        document.querySelector('#cursisten').innerHTML += `<li><strong>${index}: ${item}</strong></li>`;
    } else {
        document.querySelector('#cursisten').innerHTML += `<li>${index}: ${item}</li>`;
    }
});
```

## 10.4. ARRAYS EN FOR IN LOOP

De for...in-loop wordt doorgaans gebruikt om door de eigenschappen van objecten te itereren, maar kan ook worden gebruikt om door de indexen van een array te lopen. Dit betekent dat in het geval van een array de for...in-loop de indexen van de array zal doorlopen in plaats van de waarden zelf. Dit is iets waar vaak verwarring over bestaat.

html:

```
<h1>Arrays</h1>
<ul id="cursisten"></ul>

<script>
let cursisten = ['Tom', 'Tim', 'Bart', 'Els'];

for (let index in cursisten) {
    document.querySelector('#cursisten').innerHTML += `<li>${cursisten[index]}</li>`;
}
</script>
```

### Uitleg:

- **for...in:** In de context van arrays geeft de for...in-loop de **indexen** van de array terug, niet de waarden zelf. Daarom moeten we binnen de loop de array-elementen opvragen via cursisten[index] om de waarde te verkrijgen.
- **Gebruik met arrays:** Hoewel de for...in-loop kan werken voor arrays, is het meestal beter om **for...of** of **forEach()** te gebruiken voor het itereren door de waarden van een array, omdat for...in oorspronkelijk bedoeld is voor objecten.

### Toevoegingen en uitbreidingen:

**Itereren door objecten (echte toepassing van for...in):** Om te laten zien waar de for...in-loop echt voor bedoeld is, voegen we een voorbeeld toe waarbij we door de eigenschappen van een object itereren.

Code met objecten en for...in:

```
let student = {
    naam: 'Tom',
    leeftijd: 25,
    cursus: 'JavaScript'
};

for (let eigenschap in student) {
    console.log(`{$eigenschap}: ${student[eigenschap]}`);
}
```

**for...in met objecten:** Hier geeft for...in de eigenschappen van het object (naam, leeftijd, cursus) terug en we kunnen de bijbehorende waarden (student[eigenschap]) opvragen.

## 10.5. BUILT-IN FUNCTIES

JavaScript biedt een breed scala aan ingebouwde functies die je kunt gebruiken om veel voorkomende taken efficiënter uit te voeren. Deze functies zijn onderdeel van het JavaScript-programmeerframework en vereisen geen extra installatie of setup. Hier volgt een overzicht van enkele veelgebruikte string- en number-functies, samen met voorbeelden van hun gebruik.

## 10.6. STRING FUNCTIES

String functies in JavaScript helpen bij het manipuleren van tekst. Laten we een overzicht geven van enkele van de belangrijkste functies.

String functies in JavaScript helpen bij het manipuleren van tekst. Laten we een overzicht geven van enkele van de belangrijkste functies.

Hoe?	Functie	Gebruik
Hoofdletters	toUpperCase()	mijnString.toUpperCase()
Kleine letters	toLowerCase()	mijnString.toLowerCase()
Strings samenvoegen	concat()	mijnString1.concat(mijnString2)
Karakter op halen	charAt(index)	mijnString.charAt(0) haalt het eerste karakter op.
ASCII waarde	charCodeAt(index)	mijnString.charCodeAt(0) geeft ASCII waarde van eerste karakter
String splitsen	split(delimiter)	mijnString.split(",") verdeelt de string op basis van een delimiter
Substring ophalen	substring(start,end)	mijnString.substring(0,3) haalt de eerste drie karakters op
Eerste positie	indexOf()	mijnString.indexOf("woord") geeft positie van het woord terug
Laatste positie	lastIndexOf()	mijnString.lastIndexOf("woord") zoekt vanaf het einde
Vervangen	replace()	mijnString.replace('a', 'b') vervangt alle 'a' door 'b'

### Voorbeeld:

```
let tekst = "Hallo, JavaScript!";
console.log(tekst.toUpperCase()); // Output: "HALLO, JAVASCRIPT!"
console.log(tekst.charAt(1)); // Output: "a"
console.log(tekst.split(" ")); // Output: ["Hallo,", "JavaScript!"]
console.log(tekst.replace("Hallo", "Hi")); // Output: "Hi, JavaScript!"
```

De volledige lijst kan je online vinden op de volgende link. Aan de linkerkant van de pagina staan alle string functies opgelist:

[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/String](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String)

## 10.7. NUMBER FUNCTIES

De ingebouwde functies voor getallen in JavaScript helpen bij het omgaan met verschillende numerieke taken zoals afrondingen, exponenten, en het omzetten van strings naar nummers.

Hoe?	Functie	Gebruik
Getal naar String	toString()	mijnGetal.toString()
Afronden naar exponent	toExponential()	mijnGetal.toExponential(2)
Afronden op vaste decimalen	toFixed()	mijnGetal.toFixed(2)
Converteren naar nummer	Number()	Number(mijnString)
Parseer integer	parseInt()	parseInt("50")
Parseer float	parseFloat()	parseFloat("50.99")
Pi waarde	Math.PI	Math.PI
Afronden	Math.round()	Math.round(4.6) => 5
Absoluut getal	Math.abs()	Math.abs(-7) => 7
Machtverheffing	Math.pow()	Math.pow(2, 3) => 8
Vierkantswortel	Math.sqrt()	Math.sqrt(16) => 4
Maximaal getal in array	Math.max()	Math.max(1, 2, 3) => 3
Minimaal getal in array	Math.min()	Math.min(1, 2, 3) => 1
Willekeurig getal	Math.random()	Geeft een willekeurig getal tussen 0 en 1

Voorbeeld:

```
let getal = 123.456;
console.log(getal.toFixed(2));      // Output: 123.46
console.log(parseInt("100 jaar")); // Output: 100
console.log(Math.pow(2, 3));       // Output: 8
console.log(Math.random());        // Geeft een willekeurig getal tussen 0 en 1
```

Math.floor(): Rondt een getal naar beneden af

```
console.log(Math.floor(4.7)); // Output: 4
```

Voorbeelden van built-in functions:

Voorbeeld 1:

```
console.time('response in');
alert('Click to continue');
console.timeEnd('response in');
alert('One more time');
console.timeEnd('response in');
```

Bekijk de output.

Voorbeeld 2:

```
var elements = document.getElementsByTagName('*'); //selecteer alle
elementen op de pagina
console.time('Loop time');
for (var i = 0; i < 5000; i++) {
  for (var j = 0, length = elements.length; j < length; j++) {
    // nothing to do ...
  }
}
console.timeEnd('Loop time');
```

Bekijk de output.

[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Number](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number)

[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Math](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math)

## 10.8. LOOP: CONTINUE AND BREAK

### 10.8.1. CONTINUE

Wanneer aan een conditie wordt voldaan, wordt de continue gebruikt om deze NIET uit te voeren, maar verder te gaan met het volgende item in bijvoorbeeld een array. Zie voorbeeld hieronder

#### CODE

```
/**LOOP:CONTINUE*/
var data = ['Tim', 'Tom', 1980, 1973, 'designer', 'developer'];
for(var i = 0; i < data.length; i++){
    if(typeof data[i] === 'string') continue; //wanneer gelijk aan string DOE NIJS!
    console.log(data[i]); // wanneer verschillend van string, druk af!
}
```

#### RESULTAAT

1980
1973

### 10.8.2. BREAK

Een break is een HARDE onderbreking van de code wanneer aan een conditie wordt voldaan.

#### CODE

Hieronder onderbreken we het loopen van de array wanneer hij een item tegenkomt die GEEN STRING is. In dit geval is dit het jaartal 1980. De code wordt onderbroken en de andere items worden niet meer gecontroleerd!

```
/**LOOP:BREAK*/
var data = ['Tim', 'Tom', 1980, 1973, 'designer', 'developer'];
for(var i = 0; i < data.length; i++){
    if(typeof data[i] !== 'string') break; //springt uit de array wanneer verschillend van een string
    console.log(data[i]); // wanneer verschillend van string, druk af!
}
```

#### RESULTAAT

Tim
Tom

## 11. DEBUGGEN

Debuggen is een essentieel onderdeel van het ontwikkelen van software. Het helpt je om fouten in je code op te sporen en op te lossen door te observeren hoe de code wordt uitgevoerd. In JavaScript kun je dit eenvoudig doen met behulp van de ingebouwde browserontwikkelaarstools (zoals Chrome DevTools of Firefox Developer Tools).

Laten we dieper ingaan op het gebruik van breakpoints, watchers, en andere belangrijke debugtechnieken.

### Wat is debuggen?

Debuggen is het proces van het identificeren en oplossen van fouten (bugs) in je code. In de context van webontwikkeling kun je gebruikmaken van de **Inspect Element**-tools in browsers zoals Google Chrome om je JavaScript-code stap voor stap te analyseren en fouten op te sporen.

### Breakpoints: Stap voor stap door je code

**Breakpoints** stellen je in staat om de uitvoering van je code op een specifiek punt te pauzeren, zodat je variabelen, logica en andere aspecten van je code in detail kunt onderzoeken. Wanneer je een breakpoint toewoogt, wordt de code onderbroken op dat punt, en kun je de waarden van variabelen bekijken en analyseren wat er op dat moment in je code gebeurt.

- **Hoe zet je een breakpoint?**
- Open de ontwikkelaarstools (in Chrome: rechtersklik op de pagina → Inspecteren).
- Ga naar de tab **Sources** (Bronnen).
- Zoek het JavaScript-bestand dat je wilt debuggen.
- Klik op het nummer van de regel waar je de code wilt pauzeren om een breakpoint toe te voegen (er verschijnt een blauwe balk aan de linkerzijde).

Voorbeeld:

```
let x = 5;  
let y = 10;  
let z = x + y; // Hier voeg je een breakpoint toe  
console.log(z);
```

Wanneer je de pagina opnieuw laadt, wordt de uitvoering gepauzeerd op de regel waar het breakpoint is geplaatst, zodat je kunt zien wat de waarden van x, y en z zijn op dat moment.

## Watchers: Volg specifieke variabelen

Met **watchers** kun je specifieke variabelen volgen zonder dat je de code uitvoert. Dit is vooral handig wanneer je wilt zien hoe de waarde van een variabele in de loop van de tijd verandert, zonder dat je voortdurend `console.log()` in je code hoeft te plaatsen.

- **Hoe voeg je een watcher toe?**
- Terwijl je in de **Sources**-tab zit, vind je het gedeelte **Watch** aan de rechterkant.
- Klik op het plus-icoon en voeg de naam van de variabele in die je wilt volgen.

**Voorbeeld:** Als je een watcher toevoegt voor de variabele `z`, kun je stap voor stap volgen hoe de waarde van `z` verandert.

## Console Logging versus Breakpoints

Veel beginnende ontwikkelaars vertrouwen op `console.log()` om hun code te debuggen. Hoewel dit nuttig is voor eenvoudige gevallen, biedt het gebruik van breakpoints en watchers een veel krachtigere manier om je code te analyseren zonder de uitvoer van je code onnodig te vervuilen.

```
function bereken(x, y) {  
    let resultaat = x + y;  
    console.log(resultaat); // Dit kan vervangen worden door een breakpoint  
    return resultaat;  
}  
  
bereken(5, 10);
```

In plaats van telkens `console.log()` te gebruiken, kun je een breakpoint toevoegen op de regel `let resultaat = x + y;` en de waarde van `resultaat` direct in de debugger volgen.

