Intro to JavaScript

JavaScript is de grootste programmeertaal ter wereld, en zeker bij web development is het niet meer weg te denken. Je kan het voor veel verschillende dingen gebruiken - bijvoorbeeld:

- Verwerken van informatie op je pagina
- Versturen van informatie naar een server, zoals log-in gegevens of chatberichten
- Ontvangen en verwerken van informatie van de server
- En het updaten van je webpagina bij al het bovenstaande

Dit soort functionaliteit wordt ook wel ***client-side*** functionaliteit genoemd. Dat houdt in dat het code is die op de computer van de gebruiker wordt gedraaid.

JavaScript is ontwikkeld volgens een bepaalde standaard - ***ECMAScript***. Deze worden beiden constant geupdate - dit betekent dat er veel manieren zijn om hetzelfde probleem op te lossen. Soms wordt zo'n nieuwe manier veel gebruikt omdat het makkelijker schrijft, maar meestal zijn er wel kleine technische details verschillend.

Elke browser is compatible met een bepaalde versie van ECMAScript. Het kan dus ook zijn dat een nieuwere snellere techniek niet goed bij oudere browsers werkt. Er zijn ook programma's die jouw "nieuwere" code kunnen vertalen naar "oudere" code.

Je linkt JavaScript aan je HTML via een <script> tag - die kan kan je met de src attribute naar je .js file wijzen. Je kan een script tag ook gebruiken om interne JS te coderen.

Deze handout zit vol met voorbeelden. We willen je aanmoedigen om de voorbeelden zelf na te bouwen, zodat je de resultaten in je eigen browser kan controleren. De handout bestaat uit 6 modules met opdrachten en 1 zonder.

Inhoud

Resources

Dit zijn de tutorials en referenties die wij aanraden om JavaScript (en verder) op te pakken:

- ***MDN Web Docs*** (vroeger Mozilla Developer Network)
    - Dit is de beste technische documentatie van alles wat met Web Development te maken heeft. Het is vanwege al die technische details soms wel lastig te begrijpen. Als je iets uit context niet kan begrijpen, vraag het!
    - Je kan over elk webdev-gerelateerd onderwerp wel uitleg vinden als je het googled met MDN erachter.
    - Ze hebben ook een super complete tutorial voor beginners: hier

- ***W3schools***
    - Heeft super complete tutorials met een breed scala aan onderwerpen - van HTML/CSS, beginners EN advanced JavaScript tot Python, SQL en R.
    - Biedt ook certificering

- ***Grasshopper***
    - Gemaakt door Google om mensen te helpen leren coderen. Gefocust op JavaScript, maar veel van de dingen die je leert zijn breder toepasbaar.
    - Heeft een goede mobiele app
    - Bevat ook een goede sectie voor Interview-tips

- ***Web Dev Simplified***
    - Heel goed youtube-kanaal wat ontzettend veel onderwerpen op een goede duidelijke manier toelicht. Biedt beginners-vriendelijke uitleg over belangrijke onderwerpen, intro's voor geavanceerde programmeerconcepten en natuurlijk heel veel video's met tips & tricks

Variables

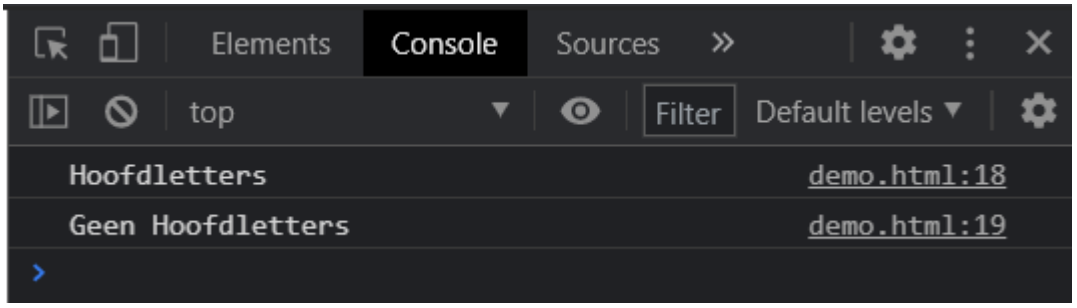
JavaScript gebruikt variables om informatie in op te slaan en te verwerken. Het verzinnen van een passende naam die beschrijft wat voor informatie er wordt opgeslagen is vaak best lastig. Gelukkig kan je deze informatie altijd loggen in de console. Dat ziet er als volgt uit:

In VSCode:

```
var CAPSLOCK = "Hoofdletters";
var capslock = "Geen Hoofdletters";

console.log(CAPSLOCK);
console.log(capslock);
```

In de browser:



Hier kan je gelijk iets heel belangrijks zien: JavaScript is ***case sensitive***!

Bij bovenstaand voorbeeld geeft het keyword ***var*** aan dat er een nieuwe variabele wordt aangemaakt. Dit wordt ook wel een declaratie genoemd. Naast var zijn er nog 2 andere soorten variables: ***let*** en ***const***. Wat de verschillen zijn tussen alledrie ga je bij de opgaven ontdekken.

Elke variabele heeft ook een bepaalde datatype. Weten wat voor datatype er bij een variabele gebruikt wordt is ontzettend belangrijk - gelukkig kan je dat ook met console.log() vinden:

```
let camelCasedVariable = "Text!"
console.log(typeof camelCasedVariable);
```

In de console zal dit string tonen. De naamgeving in dit voorbeeld wordt ook wel camelCase genoemd - dat is bij JavaScript standaard.

Datatypes

Bij JavaScript wordt er onderscheid gemaakt tussen 9 datatypes. 6 daarvan worden ook wel ***primitive datatypes*** genoemd. Dit zijn:

- number — een getal
- string — een stuk tekst
- boolean — is altijd true of false
- undefined — geen waarde toegekend
- bigint — voor hele grote getallen
- symbol — voor unieke waarden

De laatste twee hiervan worden weinig gebruikt. De overige drie datatypes zijn:

- null
- object
- function

Bij JavaScript wordt het datatype van een variabele automatisch toegewezen. Toch is het heel belangrijk om goed te beseffen wat voor datatype er in een variabele zit - als dat iets onverwachts is kan je code dus ook heel rare resultaten tonen.

Al deze informatie in datatypes en variables wil je dus ook kunnen veranderen. Dit doe je met ***operators***. De symbolen hiervoor ben je al lang mee bekend, maar wat ze bij JavaScript doen kan net iets anders zijn dan je verwacht.

Er zijn vrij veel operators en meeste ervan zijn vrij makkelijk te begrijpen, dus we gaan ze hier niet allemaal benoemen. Toch zijn er enkele nieuw of lastig die jullie tegen zullen komen:

- == — gelijke waarde
- != — ongelijke waarde
- === — gelijke waarde en gelijk datatype
- !== — ongelijke waarde OF ongelijk datatype
- ++ — increment
- -- — decrement
- % — modulo: 10 % 3 = 1

Een enkel = teken signaleert dat je een bepaalde waarde ergens aan toewijst, terwijl twee of meer == tekens een vraag aangeven die true of false kan zijn. Het antwoord op zo'n vraag is dus altijd een boolean.

Variables & Datatypes - Opdrachten

Opdracht 1.1:

We onderzoeken het verschil tussen var, let en const. Gegeven zijn een aantal voorbeelden - schrijf eerst je verwachtingen op voordat je ze zelf nabootst. Kijk vervolgens wat er bij jou in de console gelogd wordt - is dat wat je verwachtte? Zo nee, deel het dan aan het einde van de dag met je groep.

a)

```
let letOefening = "Let Tekst"
let letOefening = "Let Tekst Twee!"
console.log(letOefening)
```

b)

```
let letOefening = "Let Tekst"
console.log(letOefening)

letOefening = "Let Tekst Twee!"
console.log(letOefening)
```

c)

```
const constOefening = "Const Tekst";
console.log(constOefening)

constOefening = "Const Tekst Twee!";
console.log(constOefening)
```

d)

```
varOefening = "Var Tekst";
console.log(varOefening)

var varOefening = "Var Tekst Twee!";
console.log(varOefening);

var varOefening = "Var Tekst Drie!";
console.log(varOefening);
```

Opdracht 1.2:

We onderzoeken de verschillende datatypes.

Gegeven de volgende code:

```
let typeQuestion = "Number"
console.log("This is a " + typeof typeQuestion + ", with the value: " + typeQuestion)
```

a) Wat zie je in de console? Was dat wat je verwachtte?
b) Vervang bij het voorbeeld “Number” door de volgenden:
   i) true
   ii) false
   iii) undefined (als je ‘= “Number”’ in zijn geheel weghaalt doet dat hetzelfde)
   iv) 22
   v) 22n
   vi) Symbol()
   vii) null
   viii) {}
   ix) new Object();
   x) function calculateSomething() {}

Opdracht 1.3:

Gegeven de volgende code:

```
let someNumber = 5;
let anotherNumber = "5";
console.log(someNumber + anotherNumber)
```

c) Wat komt hier uit?
d) Wat komt er uit als je van beiden een number maakt?
e) Wat komt er uit als je de + vervangt door:
   i) ==
   ii) ===
   iii) !=
   iv) !==

Opdracht 1.4:

We gaan operators wat nader onderzoeken, en ermee experimenteren.

a) Wat komt er uit het volgende?

```
let increment = 1;
console.log(++increment)
console.log(increment++)
```

b) Wat komt er uit als je de console.log() hierboven omdraait?

c) Wat komt er uit het volgende?

```
let modulo = 24 % 10;
console.log(modulo)

let division = 24 / 10;
console.log(division)
```

d) Wat komt er uit het volgende? (spaties toegevoegd voor helderheid)

```
let someNumber = "Tekst";
console.log( ! someNumber)
```

e) Wat komt er uit bovenstaand als je de waarde van someNumber verandert in:
   i) 5
   ii) Boolean
   iii) false
   iv) “false”
   v) true
   vi) “true”
   vii) null
   viii) undefined

f) Zoek op of experimenteer wat += doet. Werkt deze ook met Strings? Weet je hoe je deze anders kan schrijven?

g) Doe hetzelfde voor -=, *=, /=, %= en **=.

String Methods

Misschien viel het je bij Opdracht 1.2 al op dat het datatype object vaak voorkwam. Dat is geen toeval. Onder de motorkap maakt JavaScript van vanalles en nog wat een Object - zelfs Strings worden automatisch omgezet naar Objects, zodat je er bepaalde functionaliteit mee kan gebruiken. Om die functionaliteit te begrijpen moet je eerst goed snappen wat een JavaScript Object is.

Objects zijn in JavaScript documentatie te herkennen aan het feit dat ze beginnen met een hoofdletter. Datatypes worden juist beschreven met kleine letters.

Een Object bestaat uit properties met bepaalde waarden, zogenoemde ***key-value pairs***. Verder kunnen Objects ook ingebouwde functies hebben - dat zijn functies die het Object op zichzelf kan uitvoeren, om zo een ander resultaat terug te geven.

Ook een String Object heeft dus methods, en 1 (belangrijke) property - lengte. Veel van deze methods gebruiken een index om aan te geven op de hoeveelste plek een bepaald teken staat. ***De index begint bij 0!***

De string “Leuk idee!” heeft dus een lengte van 10

- Op index 0 staat L
- Op index 4 staat een spatie (ook wel whitespace genoemd)
- Op index 9 staat een !

Een lijst met alle String methods kan je hier vinden. De belangrijkste zijn:

- charAt()
- concat()
- includes()
- indexOf() en lastIndexOf()
- replace() en replaceAll()
- split()
- substring() en slice()
- toLowerCase() en toUpperCase()

split() gebruikt een Array om het resultaat te tonen - dat is niets meer dan een lijstje. Later gaan we daar meer op in.

substring() en slice() lijken heel erg op elkaar, maar werken net iets anders. Let op - er is ook een verouderde method substr() die WEER net iets anders werkt - deze wordt nog wel ondersteund maar niet meer gebruikt.

String Methods - Opdrachten

Opdracht 2.1:

Gegeven de volgende String:

```
let someKittens = "De poes van de buurman heeft kittens gehad! Hij vraagt of wij nog kittens willen.";
```

a) Splits de String met een array method op in 2 aparte zinnen. Stop beiden in een aparte variabele.
b) Zo'n geweldig nieuws verdient upper case. Zet someKittens om naar hoofdletters.
c) Geef de eerste en laatste index van het woord "kittens" in de variabele someKittens.
d) Gebruik charAt() op beiden. Welk teken krijg je terug?

Opdracht 2.2:

substring() en slice() lijken erg op elkaar. We onderzoeken de verschillen. Test dit met een eigen string, of gebruik het vorige voorbeeld.

substring() en slice() kunnen 2 waardes meegegeven worden - een index waar de nieuwe waarde begint en een index waar deze moet eindigen.

a) Wat doet substring() als het eerste getal hoger is dan het tweede getal?
b) Wat doet slice() als het eerste getal hoger is dan het tweede getal?

Deze waardes hoeven niet per se positief te zijn - je kan ook negatieve getallen meegeven.

c) Wat doet substring() bij negatieve waardes?
d) Wat doet slice() bij negatieve waardes?

Opdracht 2.3:

Gegeven het volgende voorbeeld:

```
let someKittens = "Die nieuwe kittens zijn zo schattig!";
```

Gebruik split() om alle woorden apart in een Array te zetten.

Conditions

Conditions liggen aan de grondslag van alle logica. Bijvoorbeeld:

- Je wil je site pas tonen als mensen goed ingelogd zijn.
- Je wil de pagina niet herladen als iemand op de navigatie drukt van de pagina waar ze op zijn.
- Je wil dat het formulier niet verzendt als er informatie mist.

Bij al dit soort acties is er dus een bepaalde voorwaarde die voldaan moet zijn. Die acties en voorwaarden kan je op 3 verschillende manieren uitschrijven:

- If, Else & Else If
- Switch
- Ternary Operator.

Een If blok heeft een bepaalde (voorwaarde) en een bepaalde {actie} die met verschillende brackets worden aangegeven. Dat ziet er als volgt uit:

```
let someValue = false;

if (someValue == true) {
    console.log("Zie je dit?");
}
```

Let op de twee == tekens! Zoals het hierboven geschreven staat wordt er niks in de console gelogd - (someValue == true) klopt niet - de waarde is dus false. Als je dit met een enkel = teken zou schrijven, ken je de waarde true toe aan someValue. Zoals je bij Opdracht 1.4 misschien al hebt gemerkt, wordt ook zo'n toekenning door JavaScript gezien als true of false. Zorg dat je goed onderzoekt wanneer je voorwaarde true of false kan zijn!

Je kan zelfs variabelen zonder operators aan een conditie meegeven:

```
let someValue = true;

if (someValue) {
    console.log("Zie je de eerste?");
}

if (!someValue) {
    console.log("Zie je de tweede?")
}
```

Soms wil je meerdere acties ondernemen afhankelijk van dezelfde voorwaarde. Je kan dit natuurlijk doen met meerdere If blokjes, zoals bij het vorige voorbeeld. Als je de voorwaarde omdraait met een ! kunnen ze nooit beiden waar zijn.

Toch is er een nettere manier om dat uit te schrijven - met If/Else:

```
let someValue = true;

if (someValue) {
    console.log("Zie je de eerste?");
} else {
    console.log("Zie je de tweede?")
}
```

Met een If … Else … geef je dus geen optionele code meer aan. Als de conditie niet klopt, wordt de eerste code uitgevoerd, en anders de tweede. Je kan dit zien als een splitsing op een pad - als je verder wil moet je 1 van de 2 kiezen en daarin verder.

Als je dit soort logica in het Nederlands uitschrijft:

- Als (waar) dan (code). Als (onwaar) dan (code). 2 If … blokjes
- Als (waar) dan (code) anders (code) 1 If … Else blokje

Toch is er nog een laatste optie die je met If blokjes kan gebruiken:

- Als (voorwaarde) dan (code) anders ALS (voorwaarde) dan (code)
- Als (voorwaarde) dan (code) anders ALS (voorwaarde) dan (code) anders (code)

Het gaat dus om extra voorwaarden stellen die gevraagd worden ***als er niet aan de eerste voorwaarde voldaan is***. Dit ziet er zo uit:

```
let someValue = 5;

if (someValue < 5) {
    console.log("Kleiner dan 5");
} else if (someValue > 5) {
    console.log("Groter dan 5")
} else {
    console.log("Precies 5")
}
```

Je kan ook meerdere else if (...) {code} achter elkaar hangen. Het is wel heel belangrijk om te beseffen dat de latere condities/code pas worden gecontroleerd/uitgevoerd als de eerdere condities/code niet kloppend/uitgevoerd is.

Je kan alle mogelijke condities met If/Else uitschrijven. Een probleem dat je daarbij wel tegenkomt, is dat het bij complexe logica erg lang en onoverzichtelijk wordt. Ook is If/Else niet heel erg snel. Om dat op te lossen is de Switch bedacht:

```
let condition = 1;

switch (condition) {
    case 0:
    case 1:
    case 2:
        console.log ("The number is 0, 1 or 2")
        break;
    default:
        console.log("The number is not 0, 1 or 2")
}
```

Hierboven zie je een complete Switch. Die kijkt dus 1x naar de conditie, en geeft vervolgens meerdere mogelijkheden. Hierdoor is een Switch niet alleen overzichtelijker maar ook een stuk sneller dan een If/Else blok. Toch zijn er ook een paar punten waar ze verschillen:

- Een If/Else voert altijd 1 code-blok uit, en negeert de rest. Een Switch gaat rustig door naar de volgende case ook als hij een passende heeft gevonden. Een Switch stopt pas bij een “break;”, of als hij bij de laatste case is aangekomen.
- Een case kan geen wiskundige operators gebruiken. Je kan in de conditie wel “someNumber < 5” zetten, maar de case zelf is dan true/false.
- Switch is altijd sneller dan If/Else, al helemaal bij grote If/Else

Een Ternary Operator is eigenlijk hetzelfde principe - het is een andere manier om meerdere regels aan If/Else te schrijven, die sneller en overzichtelijker werkt.

Normale operators werken met 1 of 2 waardes: a++, a + b, a = b etc. De Ternary Operator is de enige die met 3 werkt, en altijd in de volgende vorm:

- “a ? b : c”
- Ook wel: “Voorwaarde ? Waar : Onwaar”

Je zet dus voor een vraagteken een conditie en daarna de code die wordt uitgevoerd als de conditie klopt. Vervolgens een dubbele punt gevolgd door de code die wordt uitgevoerd als de conditie NIET klopt.

Een voorbeeld:

```
let someNumber = 5;

someNumber == 6 ? console.log(someNumber + " is 6") : console.log(someNumber + " is not 6")
```

Conditions - Opdrachten

Opdracht 3.1:

Conditions werken erg vaak met operators. Om goed met conditions te kunnen werken, moet je operators echt goed begrijpen. Daarom onderzoeken we ze hier nogmaals:

a) Wat is het verschil tussen == en ===?
b) Wat is het verschil tussen > en >=? En tussen < en <=?

Soms controleert een conditie meerdere dingen tegelijkertijd. Dit doet met de hulp van ***Logical Operators***:

- && EN
- || OF
- ! NIET

Soms worden deze in combinatie gebruikt. Gegeven volgende code:

```
let x = 4;
let y = 8
if (  ) {
    console.log("Result!")
}
```

c) Schrijf de If conditie zo, dat je alleen “Result!” ziet als x==4 en y==8
d) Schrijf het nu zo dat je alleen “Result!” ziet als x==4 OF y==8
e) Zet nu de console.log(“Results”) in een Else { } na de If. Kan je de conditie omdraaien zodat je alsnog “Results!” te zien krijgt?
f) Kan je het vorige antwoord ook geven zonder || te gebruiken?

Schrijf voor de volgende opdrachten 1 If/Else die 1 variabele waarde controleert, en zorg voor het volgende:

g) Log het in je console als de waarde groter dan 5 EN kleiner dan 10 is.
h) Zo niet, log dan in je console als de waarde 11 of groter EN 20 of kleiner is.
i) Zo niet, log dan in je console als de waarde 21 of 23 is.
j) Zo niet, log dan in je console als de waarde kleiner dan 35 OF tussen de 40 en 45 is.
k) Test of je bij alle uitkomsten kan komen. Verandert er iets als je alle If/Else omzet in aparte If blokken?

Opdracht 3.2:

We oefenen nog wat extra met grotere If/Else blokken, Switch/case en Ternary Operators.

a) Schrijf een If/Else blok die een eigen variabele controleert:
   i) Gelijk aan 3?
   ii) Groter dan 4?
   iii) Groter dan 11?
   iv) Kleiner dan 3?

b) Test je variabele met 2, 3, 4, 5 en 20. Gebeurt er iets onverwachts?

c) Schrijf een If/Else die een variabele op numerieke waarde controleert, en de passende maand in je console logt.
   1) Januari
   2) Februari
   3) Maart
   4) April
   5) Mei
   6) Juni
   7) Juli
   8) Augustus
   9) September
   10) October
   11) November
   12) December

   Andere getallen loggen (“Geen geldige maand!”).

d) Bouw nu opdracht C met een Switch/case.

Gegeven de volgende code:

```
let x = 3;
let results;
if (x<=4 && x>=0) {
    results = 2;
} else {
    results = 5;
}
console.log(results)
```

e) Herschrijf dit als een Ternary Operator.

Functions

Tot nu toe heb je al je code zelf laten draaien zodra de pagina geladen wordt. Maar JavaScript wordt pas echt nuttig, als je functionaliteit af laat hangen van je gebruiker. Zodra die op een knopje drukt, iets invult, of zelfs maar iets met de muis doet kan je die functie oproepen en laten draaien.

Daarvoor moet je dus niet alleen code schrijven die wat doet, maar ook die code koppelen aan een actie van de gebruiker. Dat koppelen kan op 3 manieren gebeuren - de allersimpelste hiervan is via de HTML attribuut onclick of onchange:

```
<button onclick="myFunction()">Functional Buttons!</button>
<br>
<input onchange="myFunction()">
```

Later ga je ook over de andere 2 manieren leren, maar dat laten we nu nog even.

De functie zelf zit natuurlijk gewoon in je JavaScript file (of script tags):

```
let count = 0;
function myFunction() {
    console.log("You have called this function " + count + " times!")
}
```

De naam is vrij zeggend bij onclick en onchange. De eerste is verreweg de belangrijkste, maar daarnaast zijn oninput en onsubmit ook nog best handig om te kennen. Er zijn nog heel veel meer van dit soort ***events*** - later volgt daar meer over.

Functies kunnen ook informatie van een of meer variabelen meekrijgen. Zo'n variabele wordt ook wel een ***parameter*** genoemd. Dat ziet er als volgt uit:

```
function myFunction(parameter) {
    console.log("This function has the parameter: " + parameter)
}
let someVariable = 22;
myFunction(someVariable)
myFunction("All sorts of parameters are allowed")
```

De variabele "parameter" heet dus alleen zo binnen de functie zelf. Natuurlijk kan je een functie ook een variabele van buitenaf laten gebruiken - dat hebben we bij het vorige voorbeeld gezien.

Je parameters hoeven trouwens niet parameter te heten, dat is hier alleen om het uit te leggen. Vaak verwijzen ze naar het doel waar ze voor gebruikt worden, maar helaas noemen sommige luie programmeurs ze a, b, c etc. Hopelijk doen jullie dat beter!

Een functie kan ook meerdere parameters hebben:

```
function myFunction(firstParam, secondParam, thirdParam) {
    console.log("This function has the first parameter: " + firstParam)
    console.log("This function has the second parameter: " + secondParam)
    console.log("This function has the third parameter: " + thirdParam)
}
myFunction("First parameter", 22, true)
```

Soms wil je ook rekening houden met een onbekend aantal parameters. Wil je dat doen, dan moet je Rest Parameters gebruiken:

```
function myFunction(firstParam, ...restParam) {
    console.log("This function has the first parameter: " + firstParam)
    console.log("This function has the rest parameter: " + restParam)
}
myFunction("First parameter", 1, 2, 3, 4, 5, 6)
```

Rest Parameters zijn altijd als laatste in de naam van de functie, en worden aangegeven met de … ervoor. Het resultaat wordt opgeslagen in een Array - later volgt daar meer over.

Met parameters kan je dus zorgen dat een functie wordt opgeroepen en iets meegegeven krijgt. Maar een functie kan ook iets teruggeven! Dat ziet er zo uit:

```
function winBlackjack() {
    return 21;
}
let result = winBlackjack();
console.log(result);
```

Het belangrijkste van functions heb je hiermee gehad. Toch gaan we nog even wat verder met 4 onderdelen die bij functies veel gebruikt worden:

- Functies pauzeren met debugging tools
- Elementen op de pagina via het document selecteren
- het keyword ***this***
- Functies anders schrijven met arrow functions.

Als je een functie halverwege wil pauzeren kan dat door in je IDE ***breakpoints*** te zetten. Hiermee kan je controleren wat voor waardes er in je variabelen zitten. Het keyword ***debugger*** doet hetzelfde als een breakpoint:

```
let timesWon = 0;
function winBlackjack() {
    timesWon++
    debugger;
    console.log("Am I debugged yet?")
}
```

Hiermee stop je de functie voordat de console iets logt. Als je nu met de console open de functie aanroept, krijg je het debugging window te zien. Onder Scope > Strict kan je bij dit voorbeeld de waarde van timesWon inspecteren. De debugger wordt hier niet verder uitgelegd, maar we willen je wel aanmoedigen er verder mee te experimenteren - het is een heel sterk hulpmiddel!

Stel je hebt 2 buttons waarvan je de titel wil gebruiken in je code. Als je net begint kan dit er zo uit zien:

```
<button onclick="pressButtonOne()">Button #1</button>
<button onclick="pressButtonTwo()">Button #2</button>
<script>
    function pressButtonOne() {
        console.log("Button #1")
    }
    function pressButtonTwo() {
        console.log("Button #2")
    }
</script>
```

Met 2 buttons valt dit nog best te doen, maar als je pagina uit tientallen interactieve elementen bestaat is dat toch een stuk lastiger. Daarom ga je met JavaScript elementen selecteren uit de ***DOM***, het ***Document Object Model***.

Als je HTML/CSS een blauwdruk is, dan is de DOM wat je browser ermee bouwt. Het is een kopie van je pagina-layout vol met extra informatie over allemaal onderdelen, die je natuurlijk met JavaScript ook weer aan kan passen.

Via de DOM kunnen we dus rechtstreeks informatie over de buttons in onze code verwerken:

```
<button id="b1" onclick="pressButtonOne()">Button #1</button>
<button id="b2" onclick="pressButtonTwo()">Button #2</button>
<script>
    let b1 = document.getElementById("b1").innerText
    let b2 = document.getElementById("b2").innerText
    function pressButtonOne() {
        console.log(b1)
    }
    function pressButtonTwo() {
        console.log(b2)
    }
</script>
```

Hiermee hebben we onze zelfgeschreven console logs vervangen door iets schaalbaars, maar we moeten nog steeds aparte IDs, variabelen en functies voor elke button schrijven. Kon dat maar makkelijker…

```
<button onclick="pressButton(this)">Button #1</button>
<button onclick="pressButton(this)">Button #2</button>
<script>
    function pressButton(parameter) {
        console.log(parameter.innerText)
    }
</script>
```

Let op - ***this*** kan heel verwarrend werken. Als ik dit zeg en naar wijs, is dat nog wel duidelijk. Maar als ik jou vraag hetzelfde te doen voor mij, weet jij welk dit ik dan met dit bedoel? Daarom is het heel belangrijk dat je bij gebruik van ***this*** heel goed alle mogelijke waardes test, via console.log en/of de debugger.

Tenslotte zijn er ook nog Arrow Functions. Dit zijn functies zonder naam, die sneller en korter geschreven kunnen worden. Technisch is er enig verschil, vooral bij combinatie met ***this***.
Een Arrow Function ziet er als volgt uit:

```
let plusThree = value => value + 3;
console.log(plusThree(2))
```

Hierboven staat praktisch hetzelfde als:

```
function calculate(value) {
    return value + 3;
}
let plusThree = calculate
console.log(plusThree(2))
```

Omdat het korter geschreven is, worden Arrow Functions dus ontzettend vaak gebruikt. Het is nog niet nodig om alle technische details te begrijpen, maar het zal je enorm helpen om de syntax WEL goed te kennen.

Er zijn ontzettend veel luie programmeurs die liever 1 regel typen dan 4. Als je Arrow Functions goed begrijpt zal dat erg veel helpen met hun code ontcijferen!

Meer weten over Arrow Functions? Lees je in op w3schools of via MDN.

Functions - Opdrachten

Opdracht 4.1:

We gaan functies oefenen door bij het begin te beginnen:

a) Maak een button en verbind er een functie aan. Zorg ervoor dat de functie in de console “Hello World” logt.

b) Maak een input element en geef het een bepaald ID. Sla dat ID op in een variabele via document.GetElementByID().

c) Maak een functie met een parameter die deze in de console logt. Geef de ID variabele hier aan mee. Wat zie je in de console?

d) Geef je functie een return waarde, bijvoorbeeld de parameter * 2. Maak vervolgens een nieuwe variabele “result”. Roep je functie op met een getal als parameter en wijs de return waarde toe aan “result”. Log de variabele daarna in je console.

Opdracht 4.2

We gaan een paar veelgebruikte DOM methods onderzoeken. Je bent inmiddels bekend met document.GetElementById(). Veel andere DOM methods geven Arrays terug - je hebt het inmiddels al vaker gelezen maar hoe je die verwerkt gaan we later nog leren :)

a) Onderzoek de volgende DOM methodes en gebruik ze in een voorbeeld:
   i) document.getElementsByClassName
   ii) document.getElementsByTagName
   iii) document.querySelector
   iv) document.querySelectorAll
   v) element.innerHTML
   vi) element.innerText

b) Bij opdracht 4.1c heb je een input als parameter meegegeven. Roep die functie nu aan met een oninput ipv onchange, en console.log() de parameter.value ipv de parameter. Zie je het verschil in beide gevallen?

Opdracht 4.3:

Eerder heb je kennis gemaakt met het String object en de built-in String Methods. Nu gaan we kennismaken met nog zo een - het Math object. Het Math object heeft namelijk ook methods die bij functies erg veel gebruikt worden:

a) Maak een functie randomNumber() en verbindt die aan een button. Zorg dat de functie Math.random() in de console logt. Wat merk je op als je deze meerdere keren gebruikt?

Math.random() geeft dus een getal 0 >= x < 1 terug - het kan wel 0 zijn, maar nooit 1. Als je die laat afronden met Math.round(), zal het vaker afronden naar beneden dan naar boven.

Als je een willekeurig getal wil tonen, moet je Math.floor() gebruiken op Math.random(). Als je dit zo doet, krijg je altijd 0 terug.

b) Verander je functie nu dat hij een parameter x gebruikt, en zorg dat hij het volgende in je console logt: Math.floor(Math.random() * x). Test nu je functie met verschillende waardes voor x. Valt iets je op?

Als je meer wil inlezen over het Math object, kan je dat doen via w3schools of MDN.

c) Er zijn nog 2 belangrijke Math methods die handig zijn om te kennen. Onderzoek wat Math.min() en Math.max() doen. Kan je van beide al een voorbeeld geven?

Opdracht 4.4:

We eindigen deze oefeningen met een lastige:

a) Schrijf een functie waar je een naam aan mee kan geven. Laat de naam vergelijken met de namen van mensen in je groepje. Wanneer je functie de naam wel/niet herkent moet dat als boodschap teruggegeven worden.

b) Schrijf een 2e functie die de eerste functie aanroept, en vervolgens de teruggegeven boodschap in een div element op je pagina toont. Denk aan de DOM methods!

c) Schrijf nu het bovenstaande nogmaals, maar gebruik dit keer Arrow Functions. Als het niet overal lukt is dat niet erg!

Scopes

De *Scope* van een variabele slaat op de plekken waar die herkend wordt. Er bestaan 2 soorten scopes:

- Global Scope (Script Scope valt hier ook onder)
- Local Scope

De scope hangt af van het gebruikte keyword en of het binnen *{ curly brackets }* gedeclareerd wordt. Meestal is dat een functie of if/else blok, maar je kan ze ook zelf creëren:

```
{
    var varScope; // Global Scope

    let letScope; // Local Scope

    const constScope = true; // Local Scope
}

var varScope; // Global Scope

let letScope; // Global/Script Scope

const constScope = true; // Global/Script Scope
```

Var is een oude manier van variabelen aanmaken - tegenwoordig wordt het vermeden ivm veiligheidsrisico's. Als je dan toch een variabele buiten *{ Local Scope }* nodig hebt, houden const/let ze in ***Script Scope***. Dat doet voor je code hetzelfde en is een stuk veiliger.

Als je ***geen*** keyword gebruikt bij het aanmaken van een variabele, maakt JavaScript er automatisch een *var* van. Dit is dus heel gevaarlijk! Declareer veilig, declareer let of const.

```
<div id="result"></div>
<script>
    hiddenVar = "Scope?"
    document.getElementById("result").innerText = hiddenVar;
</script>
```

Arrays

Een Array is een soort datatype. Het is een manier om meerdere variabelen bij elkaar te groeperen - eigenlijk kan je een Array zien als een lijstje. Je herkent een Array aan de vierkante brackets [ ]:

```
let emptyArray = []
let filledArray = [1, 2, 3, 4, 5]
```

Elk item in een Array heeft een bepaalde ***index***. De index is een getal dat aangeeft op welke plek in het Array een bepaald item staat. Als je het hele array in je console logt, krijg je de hele lijst te zien. Als je alleen 1 item uit het array wil tonen, moet je de index gebruiken:

```
console.log(filledArray) // Toont het hele Array
console.log(filledArray[0]) // Toont het eerste item
```

Een JavaScript Array begint te tellen vanaf 0. Dit is voor nieuwe programmeurs vaak erg verwarrend. Als je het eerste item wil tonen, gebruik je dus de index 0!

Je kan ook verschillende soorten datatypes in hetzelfde Array gebruiken. Je kan zelfs Arrays in andere Arrays zetten! Dan wordt het een ***Nested Array*** genoemd:

```
let theMatrix = ["The Matrix", "The Matrix Reloaded", "The Matrix Revolutions"]
let johnWick = ["John Wick", "John Wick 2", "John Wick 3"]
let favorite = "A Scanner Darkly"
let age = 56

let keanuMovies = [theMatrix, johnWick, favorite, keanuRating, age]

console.log(keanuMovies) // Toont het hele array, inclusief subarrays
console.log(keanuMovies[0][2]) // Toont "The Matrix Revolutions"
```

Net zoals Strings zijn Arrays stiekem ook Objects. Dat betekent dat het ook de length property en built-in methods kan gebruiken. Built-in Array methods gaan we later bestuderen.

Loops

Loops zijn een manier om code meerdere keren te laten uitvoeren. Ze worden erg vaak gebruikt in combinatie met Arrays - als je een bepaalde actie op elk item in een Array wil uitvoeren, kan je dus door het Array *loopen*.

JavaScript heeft 5 verschillende loops, waarvan enkele erg veel op elkaar lijken:

- For Loop
- For .. In Loop & For … Of Loop
- While Loop & Do … While Loop

De bekendste is dus de For Loop. Die ziet er als volgt uit:

```
for (let i=0; i < 5; i++) {
    console.log("i is hier " + i)
}
```

Wat je hierboven ziet bestaat uit 4 onderdelen. Je hoeft deze namen niet te onthouden, zo lang je maar begrijpt wat ze doen:

- ***let i=0*** - Een variabele i wordt gedeclareerd en de waarde 0 gegeven de ***initialisatie***
- ***i < 5*** - Die variabele wordt geëvalueerd naar true/false: de ***conditie***
- ***i++*** - Een actie die wordt uitgevoerd aan het einde van elke loop - de ***final expression***
- ***console.log()*** - De actie(s) die de loop moet uitvoeren heten ook wel een ***statement***

Vergeet de ***semicolons (;)*** niet! Je kan de variabele i overigens best een andere naam geven. Zo'n variabele die in een for loop aangemaakt wordt heet ook wel een ***iterator.*** Een enkele uitvoering van de loop wordt ook wel een ***iteratie*** genoemd. Tegenwoordig is dat gewoon afgekort naar de variabele i.

Je hebt ook for ... in en for … of loops. Die zien er zo uit:

```
let array = ["Blink", 182]
for (const index in array) {
    console.log(index)
}
```

```
let array = ["Blink", 182]
for (const item of array) {
    console.log(item)
}
```

Snap je het verschil al? Probeer ze beiden zelf!

Tenslotte zijn er ook nog de while en do … while loops. Een while loop ziet er zo uit:

```
let bugsInCode = true;
while (bugsInCode) {
    fixBugs()
    if(needSleep) break;
}
```

Een while loop heeft alleen een conditie nodig die true/false kan zijn. Zo lang die conditie waar blijft wordt de statement herhaaldelijk uitgevoerd.

Als je conditie in de loop nooit verandert, wordt deze dus oneindig uitgevoerd. Om te voorkomen dat je browser crasht kan je het ***break***. Je hebt deze al bij switch gebruikt zien worden, maar ook in loops kan je je code ermee stopzetten.

Heel vergelijkbaar is ook het keyword ***continue***:

```
let bugsInCode = 1;
while (bugsInCode > 0) {
    console.log(bugsInCode)
    bugsInCode += 2
    if(bugsInCode == 3) continue;
    bugsInCode -= 1
    if(needSleep) break;
}
```

Continue stopt de huidige iteratie en gaat met de volgende verder - break stopt de hele loop.

Een do … while loop is praktisch hetzelfde als een while loop, maar dan voert hij de actie minimaal 1 keer uit - zelfs als de conditie onwaar is:

```
let writesBadCode = false;
do {
    writeBadCode()
} while (writesBadCode);
```

Ook de beste programmeur maakt fouten!

Array Methods

Array Methods kunnen veel verschillende dingen doen - het kan zo simpel zijn als items aan een array toevoegen/verwijderen. Het kan ook heel complex zijn, zoals een functie laten uitvoeren op elk item in een bepaald array en het resultaat als een nieuw array teruggeven.

Simpele Array methods om items aan een Array te verwijderen en toe te voegen:

- shift() en pop() verwijderen het eerste en laatste item
- unshift() en push() voegen een item toe op de eerste/laatste plek

```
let pets = ["Bird", "Dog", "Cat"]
console.log(pets.shift() + " escaped!") // Eerste item uit pets Array ontsnapt!
console.log(pets.pop() + " escaped!") // Laatste item uit pets Array ontsnapt!
console.log(pets) // Toont ["Dog"]
console.log(pets.unshift("Bird")) // Toont 2
console.log(pets.push("Cat")) // Toont 3
console.log(pets) // Toont ["Bird", "Dog", "Cat"]
```

pets.unshift() en pets.push() vangen onze ontsnapte dieren gelukkig, maar de return waarde die getoond wordt is iets heel anders. Wat er wel gereturned wordt is dus de lengte van het nieuwe array. Array methods kunnen dus een return waarde hebben die niet per se overeenkomt met wat ze doen.

Andere belangrijke Array methods.

```
let pets = ["Bird", "Dog", "Cat"]
let outdoorPets = pets.slice(1)
pets.splice(1, 2, "Cat", "Dog")
outdoorPets.sort()
pets.forEach(pet => pet = "My favorite " + pet)
outdoorPets = outdoorPets.map(pet => pet = "My favorite " + pet)
```

- slice() kopieert een array vanaf een bepaalde index waarde (hier 1). slice() kan ook een 2e parameter mee krijgen - kan je bedenken wat dat doet?

- splice() verwijdert items vanaf een bepaalde index (hier 1), voor een bepaalde hoeveelheid items (hier 2) en vervangt deze met de overige items (Cat, Dog). Effectief sorteert dit het array dus alfabetisch. De laatste 3 parameters zijn allemaal optioneel.

- sort() sorteert een array. Je kan deze een functie als parameter meegeven als je op een specifieke manier wil sorteren.

- forEach() en map() doen bijna hetzelfde - ze voeren beiden een functie uit op elk item in een array. Het verschil is dat forEach() het gebruikte array zelf aanpast. map() doet dat niet, en geeft het resultaat in plaats daarvan terug als een nieuw array. In het voorbeeld hebben ze hetzelfde effect - kan je bedenken waarom dat gebeurt?

Arrays & Loops - Opdrachten

Opdracht 5.1:

We oefenen met Arrays.

a) Maak een array met 1 t/m 10 in willekeurige volgorde en log dit.
b) Maak een array met Appel, Aardbei en 3 andere fruitsoorten. Log dit.
c) Gebruik de indexwaarde om Appel en Aardbei in de console te loggen.

Nu gaan we het iets lastiger maken.

d) Gebruik Math.random() en Math.floor() om een random indexwaarde van je array met getallen te selecteren. Gebruik dat getal vervolgens als index om een fruitsoort te selecteren en log dit.

Als je een random getal krijgt met hogere waarde dan de lengte van je fruitsoorten array, zal je code een error geven. Om dit op te lossen kan je de modulo operator gebruiken - het resultaat van X % Y kan nooit groter zijn dan Y!

e) Console log de length property van je fruitsoorten array.
f) Fix de vorige error met modulo en de lengte.

Je kan de index van een item ook gebruiken om de waarde te veranderen.

g) Verander “Appel” naar “Peer”
h) Verwissel “Peer” en “Aardbei” van positie. Kan je dit ook al met/zonder Array method?

Opdracht 5.2:

We oefenen met Loops. Gegeven het volgende array:

```
const dutchSports = ["Voetbal", "Hockey", "Schaatsen"]
```

a) Loop door het array heen en log de items 1 voor 1 in je console.
b) Doe nu hetzelfde met een for...in en een for...of loop
c) Maak een for loop die van de getallen 1 t/m 20 de even getallen logt.

Nu gaan we even oefenen met nested loops. Maak nu een for-loop aan die drie keer draait.

d) Maak daarbinnen een for-loop die 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 logt.
e) Volg die op met een 2e nested loop die 2, 4, 6, 8, 10, 12, 14, 16, 18, 20 logt.
f) Volg die weer op met een 3e nested loop die 3, 6, 9, 12, 15, 18, 21, 24, 27, 30 logt.

Nu maken we het iets lastiger. De Reeks van Fibonacci is een lijst met getallen waarbij elk getal een opsomming is van de vorige 2 getallen.

g) Begin met het array [0,1]. Maak vervolgens een loop die dit array vult met de eerste 50 getallen van de Reeks van Fibonacci. Log dit array.

Stel je hebt het volgende array met random getallen [2,7,5,10,4,9,3,1,8,6] en je wil dit sorteren. Er zijn ontzettend veel verschillende methodes om dingen te sorteren - ze zijn niet allemaal even efficiënt. Een van die methodes heet bubblesort. Een voorbeeld:

h) Bouw een loop die het gegeven array met bubblesort sorteert.

Opdracht 5.3:

We oefenen met Array Methods. Gegeven het volgende array:

```
const dutchSports = ["Voetbal", "Hockey", "Schaatsen"]
```

Gebruik Array methods om het volgende te doen. Elke vraag gebruikt een andere method:

a) Voeg Zeilen en Zwemmen toe aan het einde van het array
b) Voeg Volleybal toe aan het begin van het array
c) Maak een nieuw array ballSports, met de balsporten uit dutchSports
d) Verwijder die items uit dutchSports.
e) Sorteer het dutchSports array.
f) Log elk item in de console.
g) Maak een nieuw array sportsLength, met daarin de lengte van elk item in het dutchSports array.

Opdracht 5.4:

We onderzoeken Array-Like Objects en nog enkele array methods.

Array-Like Objects zijn ook lijsten, maar dan met een ander datatype. Ze worden hetzelfde geschreven. Er is wel een cruciaal technisch verschil - Array-Like Objects kunnen geen Array methods gebruiken. Gegeven de volgende code:

```
<div>Some Text</div>
<div>Some Text</div>
<script>
    let arrayLike = document.querySelectorAll('div')
    console.log(arrayLike) // Toont een nodeList! Dat is een soort DOM Object
</script>
```

a) Gebruik Array.from() om er een nieuw array van te maken.

Onderzoek de volgende array methods en maak voor elke een voorbeeld:

b) filter()
c) find()
d) some()
e) every()
f) includes()

Objects

Een Object is een manier om bepaalde eigenschappen en gedrag te groeperen. Zo kan je per gebruiker alle informatie bij elkaar houden. JavaScript gebruikt aardig wat ingebouwde objecten - met enkele heb je al kennis gemaakt. Maar je kan ook je eigen objecten aanmaken! Dat doe je met een zogenaamde ***Constructor***. De belangrijkste zijn:

- Object Constructor
- Literal Constructor
- Function Constructor
- Singleton Constructor
- Class-based Constructor

De ***Object Constructor***:

```
let objectConstructed = new Object();

objectConstructed.name = "Jens"
objectConstructed.age = "28"
objectConstructed.country = "Netherlands"
objectConstructed.city = "Amsterdam"
```

De ***Literal Constructor***:

```
let literallyConstructed = {
    name : "Jens",
    age : "28",
    country : "Netherlands",
    city: "Amsterdam"
}
```

De ***Function Constructor***:

```
function Person(name, age, country, city) {
    this.name = name
    this.age = age
    this.country = country
    this.city = city
}

let functionConstructed = new Person("Jens", "28", "Netherlands", "Amsterdam")
```

De ***Singleton Constructor***:

```
let singletonConstructed = new function() {
    this.name = "Jens",
    this.age = "28",
    this.country = "Netherlands",
    this.city = "Amsterdam"
}
```

De ***Class-based Constructor:***

```
class Person {
    constructor(name, age, country, city) {
        this.name = "Jens"
        this.age = "28"
        this.country = "Netherlands"
        this.city = "Amsterdam"
    }
    writeBadCode() {
        console.log("Foutje kan iedereen gebeuren!")
    }
}
let classConstructed = new Person("Jens", "28", "Netherlands", "Amsterdam")
```

Elke constructor heeft zijn eigen voordelen en nadelen. Een deel van het verschil zit in het omgaan met de ***Object Prototype.*** Dat is een voorbeeld waar een Object bepaalde properties en gedrag van kunnen overnemen.

Misschien wel het allerbelangrijkste onderwerp wat met Objects te maken heeft is ***JSON***, ***J***ava***S***cript ***O***bject ***N***otation. JSON wordt binnen en buiten JavaScript gebruikt om data over het internet te communiceren. Weten hoe je zo'n file uitleest en verwerkt is dus ontzettend belangrijk! Een goede uitleg over JSON kan je hier vinden.

Objects - Opdrachten

Opdracht 6.1:

We oefenen met het aanmaken van Objects.

a) Maak meerdere objects Pet met een naam en een soort. Maak een ander dier met elk van de 5 soorten constructor uit de voorbeelden en log deze in de console.Zie je een onverwachte property?

De laatste veelgebruikte manier om Objecten aan te maken is met Object.create()

b) Gebruik Object.create() om een Object aan te maken. Doe dit voor elke Pet die je in de vorige opdracht hebt gemaakt, en log ze in de console. Valt er iets op aan de prototype?

c) Maak een array met 10 Person Objects, die een naam en leeftijd hebben. Gebruik vervolgens Array methods om je personen te sorteren op leeftijd.

Opdracht 6.2:

We oefenen met nested Objects.

a) Maak een Club object dat het volgende bijhoudt:
   i) Naam
   ii) Type club
   iii) Aantal :Leden

b) Maak een Contactinformatie object aan en voeg die toe aan elk Club object. De Contactinformatie houdt het volgende bij:
   i) Adres
   ii) Telefoonnummer
   iii) Contactpersoon

c) Maak een array met 5 verschillende Clubs aan. Itereer door het array heen en log voor elke Club de naam, het telefoonnummer en de contactpersoon.