# Javascript de basis

### Voorbeeld

### Te gebruiken bij:

-

### Eigenschappen:

### Info

* JavaScript is een programmeertaal voor webbrowsers. Het zorgt ervoor dat webpagina’s niet alleen statische informatie laten zien, maar ook dynamische interactie mogelijk maken.
* JavaScript is ooit ontwikkeld als een programmeertaal die je in de browser uitvoert om interactie op een webpagina mogelijk te maken: *client-side-scripting.*

Dit script zorgt ervoor dat de webpagina er, op basis van de acties van de gebruiker, anders uit komt te zien dan de originele content.

1. Inline event handlers: Je kunt JavaScript-code rechtstreeks in HTML-elementen plaatsen met behulp van attributen zoals onclick, onload, onkeyup, enz. Bijvoorbeeld:

html

<button onclick="myFunction()">Klik hier</button>

1. Event listeners: Een event listener is een functie die wordt uitgevoerd wanneer een specifiek event optreedt. Je kunt event listeners toevoegen aan elementen met behulp van JavaScript-methoden zoals addEventListener(). Bijvoorbeeld:

javascript

const button = document.querySelector('button');

button.addEventListener('click', myFunction);

Event object: Bij het verwerken van een event wordt er vaak een event object doorgegeven aan de event handler-functie. Dit object bevat informatie over het event en kan worden gebruikt om specifieke details op te halen, zoals de geklikte coördinaten, het ingevoerde tekstvak, enzovoort.

1. Event bubbling en capturing: Events in JavaScript volgen een bubbelings- en vangmechanisme, waarbij een event van het binnenste naar het buitenste element (bubbling) of van het buitenste naar het binnenste element (capturing) kan stromen. Dit kan nuttig zijn bij het bepalen van de volgorde waarin event handlers worden uitgevoerd wanneer er meerdere elementen genest zijn.
2. Preventing default behavior: Sommige events hebben een standaardgedrag dat automatisch wordt uitgevoerd, zoals het volgen van een link bij een klik. Met behulp van de preventDefault()-methode van het event object kun je voorkomen dat het standaardgedrag optreedt.

JavaScript zorgt ervoor dat we content kunnen aanspreken en aanpassen terwijl de gebruiker de webpagina al bekijkt in de browser. Dit zorgt ervoor dat de webpagina zich interactief gedraagt, omdat hij reageert op wat de gebruiker doet.

#### Run javascript

Console.log();

Icm: nodemon naam.js in de terminal draait deze als het is verandert

Daanrna kan nodemon stoppe door ctrl C.

#### Overzicht

1. Toegang tot content: met behulp van JavaScript kun je onderdelen van HTML-elementen selecteren, zoals bijvoorbeeld de waarde van een <input>-element uit een formulier.
2. Content aanpassen: JavaScript kan HTML-elementen, attributen of tekst op een pagina verwijderen of toevoegen. Denk hierbij aan het toevoegen van de melding: ‘Dit wachtwoord is onjuist’ onder een <input>-element.
3. Regels bepalen (script schrijven): zoals je een recept zou volgen, kun je ook stappen en acties bepalen die de browser moet volgen. Als je bijvoorbeeld een rekentool hebt gemaakt die jouw belastingschaal uitrekent, kun je met JavaScript vaststellen dat de ingevulde waardes verzameld moeten worden, dat daar een berekening mee wordt gemaakt en de uitkomst dan weer teruggegeven wordt aan de gebruiker.
4. Op gebeurtenissen reageren: Je kunt bepalen dat een bepaald script gevolgd moet worden wanneer een specifieke gebeurtenis plaatsvindt, bijvoorbeeld wanneer de gebruiker op een knop heeft geklikt, een webpagina klaar is met laden of gegevens zijn ingevuld.

#### Var, const let

In JavaScript worden var, const en let gebruikt om variabelen te declareren. Ze hebben enkele belangrijke verschillen in termen van scope en toewijzing van waarden. Hier is een overzicht van de verschillen tussen var, const en let in JavaScript:

##### var:

Variabelen gedeclareerd met var hebben function scope of global scope, wat betekent dat ze zichtbaar zijn binnen de hele functie waarin ze zijn gedeclareerd of op het globale niveau als ze buiten een functie worden gedeclareerd.

Ze kunnen meerdere keren op dezelfde scope worden gedeclareerd, en de meest recente declaratie overschrijft de vorige waarde.

Ze kunnen ook gebruikt worden voordat ze zijn gedeclareerd (hoisting genoemd), maar hun waarde is dan undefined.

var-variabelen kunnen opnieuw worden toegewezen en worden geüpdatet.

##### let:

Variabelen gedeclareerd met let hebben block scope, wat betekent dat ze alleen zichtbaar zijn binnen het blok waarin ze zijn gedeclareerd (bijvoorbeeld binnen een if-statement, loop, of functie).

Ze kunnen slechts één keer per scope worden gedeclareerd en kunnen niet opnieuw worden gedeclareerd in hetzelfde blok.

Ze kunnen ook gebruikt worden voordat ze zijn gedeclareerd, maar in tegenstelling tot var, veroorzaakt het refereren naar een let-variabele voordat deze is gedeclareerd een "temporal dead zone" error.

let-variabelen kunnen opnieuw worden toegewezen en worden geüpdatet binnen dezelfde scope.

##### const:

Variabelen gedeclareerd met const hebben ook block scope, vergelijkbaar met let.

Ze moeten eenmaal worden toegewezen bij de declaratie en kunnen niet opnieuw worden toegewezen. Ze zijn constant.

Ze moeten worden geïnitialiseerd met een waarde, en die waarde kan niet worden gewijzigd.

const-variabelen mogen niet worden opnieuw toegewezen of bijgewerkt binnen dezelfde scope.

##### Andere variablen

Primitieve variabelen:

1. Number: Voor het opslaan van numerieke waarden, zoals let age = 25;.
2. String: Voor het opslaan van tekstwaarden tussen aanhalingstekens, zoals let name = "John";.
3. Boolean: Voor het opslaan van een logische waarde (true of false), zoals let isLogged = true;.

Samengestelde variabelen:

1. Array: Voor het opslaan van meerdere waarden in een geordende lijst, zoals let numbers = [1, 2, 3, 4];.
2. Object: Voor het opslaan van complexe gegevensstructuren met eigenschappen, zoals let person = { name: "John", age: 25 };.

Speciale variabelen:

1. null: Voor het expliciet aangeven van een lege waarde, zoals let value = null;.
2. undefined: Voor het aangeven van een variabele die is gedeclareerd maar nog geen waarde heeft, zoals let variable;.

Het gebruik van const en let wordt aanbevolen boven var, omdat ze betere controle bieden over de scope en toewijzing van waarden. Het gebruik van const wordt vooral aanbevolen voor variabelen waarvan de waarde niet verandert, terwijl let wordt gebruikt voor variabelen waarvan de waarde kan veranderen.

let counter = 1;

counter = 2; // ---> er gebeurt iets waardoor we de counter verhogen

counter = 3; // ---> er gebeurt iets waardoor we de counter verhogen

console.log(counter); // geeft 3

const counter = 1;

counter = 2; // <--- overschrijven mag dus niet bij een constante!

console.log(counter); // geeft “TypeError: Assignment to constant variable.”

Wanneer je gaat rekenen met numerieke waardes kun je de volgende rekenkundige operatoren gebruiken:

Afbeelding met tekst, schermopname, Lettertype, nummer

Automatisch gegenereerde beschrijving

1. Groepering: ( )

Hiermee kun je expressies groeperen en de uitvoeringsvolgorde bepalen.

Unary operators (unaire operators): +, -, !, typeof, ++, --, etc.

1. Deze operators werken op één operand en voeren een specifieke actie uit, zoals het wijzigen van het teken van een getal, het logisch omkeren van een booleanwaarde, het verhogen/verlagen van een waarde met 1, enzovoort.
2. Multiplication/Division operators (vermenigvuldigings-/delingsoperators): \*, /, %

Deze operators voeren vermenigvuldiging, deling en modulo (restwaarde na deling) uit.

1. Addition/Subtraction operators (optel-/aftrekoperators): +, -

Deze operators voeren optelling en aftrekking uit.

1. Relational operators (relationele operators): <, >, <=, >=, instanceof, in

Deze operators vergelijken waarden en geven een booleanwaarde terug die aangeeft of de relatie waar is.

1. Equality operators (gelijkheidsoperators): ==, ===, !=, !==

Deze operators vergelijken waarden op gelijkheid en geven een booleanwaarde terug die aangeeft of de vergelijking waar is.

1. Logical operators (logische operators): &&, ||

Deze operators voeren logische operaties uit op booleanwaarden en geven een booleanwaarde terug op basis van de logische voorwaarden.

1. Assignment operators (toekenningsoperators): =, +=, -=, \*=, /=, %=, etc.

Deze operators wijzen waarden toe aan variabelen met verschillende toekenningstechnieken, zoals eenvoudige toekenning, toevoegende toekenning, aftrekkende toekenning, enzovoort.

Op het zelfde niveau wordt er van links naar rechts rekend

De + operator is de enige operator die je kunt toepassen op tekstuele waardes (strings)

const houseNumber = 12;

const street = "Prinsenlaan";

const address = street + houseNumber; // geeft “Prinsenlaan12” (zonder spatie)

const score1 = "zeven";

const score2 = "vier";

const total = score1 \* score2; // geeft NaN (Not A Number) want je kunt “zeven” en “vier” niet converteren naar een getal

JavaScript is namelijk een *loosely typed* taal: haar variabelen zijn niet direct geassocieerd met een bepaald datatype. Dat is lekker makkelijk, maar soms ook een beetje vreemd. Het betekent namelijk dat we variabelen telkens opnieuw een ander datatype kunnen toewijzen:

let banaan = 42; // eerst een getal

banaan = "geel"; // nu wordt het een string

banaan = true; // ... en nu een boolean

Bovendien mogen we met behulp van onze operatoren ook nog datatypes samenvoegen, in tegenstelling tot backend talen zoals Java. Dit betekent dat we, technisch gezien, string waardes mogen optellen bij getallen en vice versa.

2 + 2 // geeft 4

"2" + 2 // geeft 22

"2" + "2" - 6 // geeft 16

#### beslisingsvormen

1. Een **conditie** waarop we differentiëren, in dit geval: is de score groter of gelijk aan 5.5?
2. **Conditionele statements**: afgebakende blokken code waarin we kunnen specificeren wat er moet gebeuren als de conditie waar is (print “Je hebt het gehaald!”) of niet waar is (print “Je hebt het niet gehaald..”).

if (grade >= 5.5) {

// het eerste blok wordt uitgevoerd als de conditie true is

console.log("Je hebt het gehaald!");

} else {

// het tweede blok wordt uitgevoerd als de conditie false is

console.log("Je hebt het niet gehaald..");

}

// code gaat hier weer verder

#### Vergelijkingsoperatoren

We zetten onze conditie altijd binnen de () van de if-statement. In bovenstaand voorbeeld vergelijken we het cijfer dat in de variabele grade staat opgeslagen met de numerieke waarde 5.5. We hebben de vergelijkingsoperator "groter of gelijk aan" gebruikt, maar er zijn er nog veel meer.

if (recipe !== undefined) { // als recipe al toegewezen is..

// print dan de naam van het recept

console.log("Vandaag maken we het recept: " + recipe);

} else {

recipe = "Spinazie lasagne"; // en anders… wijs recipe toe

}

Bij gebruik van een vergelijkingsoperator zal de return value altijd ofwel true of false zijn. Je hoeft hem daarom niet altijd direct binnen een conditie te zetten: je kunt de vergelijking ook opslaan in een variabele:

const grade = 4;

const isPassed = grade >= 5.5; // geeft false

console.log("Student passed: " + isPassed) // geeft "Student passed: false"

Afbeelding met tekst, schermopname, nummer, Lettertype

Automatisch gegenereerde beschrijving

We maken in JavaScript gebruik van =, == en ===. Ze beteken alledrie iets anders.

Een klein geheugensteuntje met betrekking tot bovenstaande operatoren:

* = wordt gebruikt om een waarde aan een variabele toe te wijzen (géén vergelijking)
* == is een vergelijking op waarde
* === is een vergelijking op waarde én type

#### Logica operatoren

Maar wat als we een vergelijking willen maken op basis van twee condities of zelfs drie? Stel dat we willen checken of onze ingrediënten array al toegewezen is, maar we willen ook weten of er minimaal drie items in staan voor we boodschappen kunnen doen… Daarvoor hebben we logica operatoren!

Afbeelding met tekst, schermopname, Lettertype, diagram

Automatisch gegenereerde beschrijving

Dit kunnen we toepassen op onze ingrediënten array array:

if ((ingredients !== undefined) && (ingredients.length > 3)) {

console.log("We kunnen boodschappen doen!");

}

Er bestaan drie soorten logica operatoren. En, of, en niet:

Ze evalueren van links naar rechts en stoppen er ook zo snel mogelijk mee (short-circuit).

Afbeelding met tekst, handschrift, Lettertype

Automatisch gegenereerde beschrijving

#### Switch Statements

Het is daarmee een hele efficiënte manier om tussen scenario's te *switchen*. Onze compiler kan ook sneller over de vergelijkingen in onze statement springen, omdat de waardes altijd van hetzelfde type zijn. Bovendien maakt het onze code overzichtelijker. Bekijk het onderstaande voorbeeld eens.

const fruit = "Appels";

switch (fruit) {

case 'Bananen':

console.log('Het zijn bananen!');

break;

case 'Appels':

console.log('Het zijn appels!');

break;

case 'Citroenen':

console.log('Het zijn citroenen!');

break;

default:

console.log('Er is geen vrucht gekozen');

}

break zien staan. Deze is ook erg belangrijk. Dit zorgt ervoor dat we, nadat we de code van de gematchte case hebben uitgevoerd, ook daadwerkelijk uit de switch *"breaken"*

In sommige gevallen kan het zo zijn dat we meerdere cases hebben met dezelfde uitkomst. Kijk eens naar het volgende voorbeeld:

const browser = 'Chrome';

switch (browser) {

case 'Edge':

console.log('Gelukkig gebruik je geen Internet Explorer meer!');

break;

case 'Chrome':

console.log('Deze browser wordt ondersteund!');

break;

case 'Firefox':

console.log('Deze browser wordt ondersteund!');

break;

case 'Safari':

console.log('Deze browser wordt ondersteund!');

break;

case 'Opera':

console.log('Deze browser wordt ondersteund!');

break;

default:

console.log('Hmmm... Hopelijk ziet de pagina er nog ok uit.');

}

Of zo:

const browser = 'Chrome';

switch (browser) {

case 'Edge':

console.log('Gelukkig gebruik je geen Internet Explorer meer!');

break;

case 'Chrome':

case 'Firefox':

case 'Safari':

case 'Opera':

console.log('Deze browser wordt ondersteund!');

break;

default:

console.log('Hmmm... Hopelijk ziet de pagina er nog ok uit.');

}

#### structurele datatypes

##### Arrays

een lijst waardes die iets met elkaar gemeen hebben. Arrays zijn vooral handig als je niet weet hoeveel items je precies nodig gaat hebben, omdat je dit niet hoeft te specificeren.

const lasagneIngredients = [“Geraspte kaas”, “Spinazie”, “Knoflook”, “Olijfolie”, “Lasagne bladen”, “Ui”];

Je hebt toegang tot waardes in een array alsof ze in een genummerde lijst zouden staan. Let wel: de nummering van arrays begint altijd bij 0.

Afbeelding met tekst, schermopname, Lettertype, nummer

Automatisch gegenereerde beschrijvingAfbeelding met tekst, schermopname, Lettertype, nummer

Automatisch gegenereerde beschrijving

We kunnen individuele items aanspreken op basis van hun index-nummer.

lasagneIngredients[2]; // geeft "Knoflook"

lasagneIngredients[0]; // geeft "Geraspte kaas"

Elke array heeft een length property die bijhoudt hoeveel items er in de

array staan. lasagneIngredients.length; // geeft 6

Als we ineens tot de conclusie komen dat we geen olijfolie, maar kokosolie voor ons recept willen gebruiken moeten we dit kunnen aanpassen.

lasagneIngredients[3] = "Kokosolie";

###### voorbeeld:

const movies = [

{

title: "The Shawshank Redemption",

releaseDate: 1994,

rating: 9.2,

actors: ["Tim Robbins", "Morgan Freeman", "Bob Gunton"]

},

{

title: "The Godfather",

releaseDate: 1972,

rating: 9.2,

actors: ["Marlon Brando", "Al Pacino", "James Caan"]

},

{

title: "The Dark Knight",

releaseDate: 2008,

rating: 8.5,

actors: ["Christian Bale", "Heath Ledger", "Gary Oldman"]

},

{

title: "Pulp Fiction",

releaseDate: 1994,

rating: 7.8,

actors: ["John Travolta", "Samuel L. Jackson", "Uma Thurman"]

},

{

title: "The Lord of the Rings: The Return of the King",

releaseDate: 2003,

rating: 8.9,

actors: ["Elijah Wood", "Ian McKellen", "Viggo Mortensen"]

},

{

title: "12 Angry Men",

releaseDate: 1957,

rating: 6.2,

actors: ["Henry Fonda", "Lee J. Cobb", "Martin Balsam"]

}

];

// // Opdracht 1a: Log het eerste object in de 'movies' array

// console.log(movies[0]);

//

// // Opdracht 1b: Log de titel van het eerste object in de 'movies' array

// console.log(movies[0].title);

//

// // Opdracht 3a: Log de lijst van acteurs uit van de film "Pulp Fiction" uit de 'movies' array

// console.log(movies[3].actors);

//

// // Opdracht 3b: Log de tweede acteur van de film "The Shawshank Redemption"

// console.log(movies[0].actors[1]);

//

// // Opdracht 3c: Log de laatste acteur van de laatste film in 'movies' array

// const lastMovie = movies[movies.length - 1];

// console.log(lastMovie.actors[lastMovie.actors.length - 1]);

// ------------------------------------------------------------------------------ //

// Opdracht 1a: Log het eerste object in de 'movies' array

console.log(movies[0]);

// Verwacht antwoord:

// {

// title: "The Shawshank Redemption",

// releaseDate: 1994,

// rating: 9.2,

// actors: [ "Tim Robbins", "Morgan Freeman", "Bob Gunton" ]

// }

// Opdracht 1b: Log de titel van het eerste object in de 'movies' array

console.log(movies[0].title);

// Verwacht antwoord:

// The Shawshank Redemption

// ------------------------------------------------------------------------------ //

// Opdracht 2a: Log de hoeveel films er in de 'movies' array staan (de lengte van de array). Let op: wanneer de array langer of korter wordt, moet dit nog steeds werken. Je mag dus niet zelf tellen!

console.log(movies.length);

// Verwacht antwoord:

// 6

// Opdracht 2b: Log het indexnummer van de laatste film in de 'movies' array. Let op: wanneer de array langer of korter wordt, moet dit nog steeds werken!

console.log(movies.length -1);

// Verwacht antwoord:

// 5

// Opdracht 2c: Log het laatste object in de 'movies' array. Let op: wanneer de array langer of korter wordt, moet dit nog steeds werken!

// Tip: maak gebruik van de code die je in de vorige opdracht hebt gemaakt.

console.log(movies[movies.length -1]);

// Verwacht antwoord:

// {

// title: '12 Angry Men',

// releaseDate: 1957,

// rating: 6.2,

// actors: [ 'Henry Fonda', 'Lee J. Cobb', 'Martin Balsam' ]

// }

// ------------------------------------------------------------------------------ //

// Opdracht 3a: Log de lijst van acteurs uit van de film "Pulp Fiction" uit de 'movies' array.

console.log(movies[3].actors);

// Verwachte antwoord:

// ["John Travolta", "Samuel L. Jackson", "Uma Thurman"]

// Opdracht 3b: Log de tweede acteur van de film "The Shawshank Redemption". Tip: je spreekt dus eerst de array, daarna het object en vervolgens weer een array aan.

console.log(movies[0].actors[1]);

// Verwacht antwoord:

// Morgan Freeman

// Opdracht 3c: Log de laatste acteur van de laatste film in 'movies' array. Let op: wanneer de array langer of korter wordt, of wanneer er meer of minder acteurs in dit rijtje staan, moet dit nog steeds werken!

// Tip: gebruik hiervoor o.a. jouw code uit 2b voor.

console.log(movies[movies.length - 1].actors[movies[movies.length - 1].actors.length - 1]);

// Verwacht antwoord:

// Martin Balsam

// ------------------------------------------------------------------------------ //

// Opdracht 4a: Maak een if-else-statement die het woord 'Uitstekende film!!' logt,

// wanneer een film een hoger cijfer krijgt dan een 8 en het woord 'Goede film!' logt, wanneer deze lager is.

// Zorg dat je de onderstaande variabele gebruikt en pas eventueel het entry-nummer aan,

// om te kijken of je algoritme ook werkt wanneer je een andere film selecteert!

const selectedMovie = movies[3];

if (selectedMovie.rating > 8) {

console.log("Uitstekende film!!");

} else {

console.log("Goede film!");

}

// Verwacht antwoord bij 'const selectedMovie = movies[0]':

// Uitstekende film!!

// Verwacht antwoord bij 'const selectedMovie = movies[3]':

// Goede film!

// Opdracht 4b: Maak een if-else-if-else-statement die het woord 'Topper!' logt,

// wanneer een film een hoger cijfer krijgt dan een 8 en het woord 'Kijktip!' logt, wanneer deze een 8 of lager is,

// maar wel hoger dan een 6 en een 'Matige film' logt bij een lager cijfer dan 6.

// Gebruik weer de variabele uit de vorige opdracht.

// Tip: gebruik de && selector als het getal binnen twee condities moet vallen

const selectedMovie = movies[0];

if (selectedMovie.rating > 8) {

console.log("Topper!");

} else if (selectedMovie.rating <= 8 && selectedMovie.rating > 6) {

console.log("Kijktip!");

} else {

console.log("Matige film");

}

// Verwacht antwoord bij 'const selectedMovie = movies[0]':

// Uitstekende film!!

// Verwacht antwoord bij 'const selectedMovie = movies[3]':

// Goede film!

// Verwacht antwoord bij 'const selectedMovie = movies[5]':

// Matige film

##### String

#### Objecten

Een **object** groepeert en set variabelen bij elkaar die iets met elkaar gemeen hebben.

In tegenstelling tot een array, die alleen een lijst waardes bevat,

kun je met een object ook een stukje context toevoegen.

##### Variabelen(properties) vs functies (methodes)

Je kunt zowel variabelen als functies in een object plaatsen.

verschil is:

* variabelen in een object properties genoemd worden
* functies in een object methodes genoemd worden

###### Voorbeeld 2

// === Opdracht 1 ===

// Maak een object met daarin jouw voornaam (string), achternaam (string), leeftijd (getal) en geboortedatum (date object). De datum hoeft niet geformat te worden.

// Log dit in de terminal.

const birthDate = new Date("1988-12-25");

const year = birthDate.getFullYear(); // Jaar (bijv. 1988)

const month = birthDate.getMonth() + 1; // Maand (bijv. 12, waarbij januari wordt weergegeven als 1)

const day = birthDate.getDate(); // Dag (bijv. 25)

const currentDate = new Date();

let differenceInYears = currentDate.getFullYear() - year;

const isBeforeBirthday =

currentDate.getMonth() < birthDate.getMonth() ||

(currentDate.getMonth() === birthDate.getMonth() && currentDate.getDate() < birthDate.getDate());

if (isBeforeBirthday) {

differenceInYears -= 1;

}

const person = {

voornaam: "Matthijs",

achternaam: "van der Maas",

leeftijd: differenceInYears,

geboortedatum: {

jaar: year,

maand: month,

dag: day

}

};

console.log(person);

// === Opdracht 2 ===

// Zorg ervoor dat de volgende string in de terminal wordt gelogd:

// "Hoi! Ik ben <voornaam>, ik ben <leeftijd> jaar oud."

console.log("Hoi! Ik ben " + person.voornaam + ", ik ben " + person.leeftijd + " jaar oud.");

// === Opdracht 3: bekijk het volgende object ===

const product = {

productName: "Smeg Waterkoker",

series: "Smeg jaren 50 retro lijn",

material: ["roestvrijstaal", "kunststof"],

color: "creme",

wireless: false,

"b-h-d": "17 x 24,6 x 22,3 cm",

};

// 3a. Log de naam van de serie in de terminal.

console.log(product.productName);

// 3b. Overschrijf de oude waarde van de property color ("creme") naar "roze". Log het object in de terminal om te checken of het gelukt is.

product.color = "roze";

console.log(product.color);

// 3c. Log de tweede materiaalsoort ("kunststof") in de terminal

console.log(product.material[product.material.length - 1]);

// 1d. Log de afmetingen (b-h-d) in de terminal

console.log(product["b-h-d"])

// === Opdracht 4 ===

// Declareer een variabele met daarin een string die het product beschrijft. Gebruik hiervoor de properties productName en material uit het object.

// ---- Verwachte uitkomst:

// "De Smeg Waterkoker is gemaakt van roestvrijstaal en kunststof"

console.log("De " + product.productName + " is gemaakt van " + product.material[0] + " en "+ product.material[1]+".");

een object van maken:

Afbeelding met tekst, diagram, Lettertype, lijn

Automatisch gegenereerde beschrijving

De **keys** beschrijven altijd waar de data over gaat. De **values** zijn de data zelf. Een object kan nooit twee dezelfde keys bevatten, omdat we de key gebruiken om de corresponderende data aan te spreken.

const recipe = {

name: 'Spinazie lasagne',

cookingTime: 45,

isGlutenFree: false,

ingredients: ['Lasagne bladen', 'Spinazie', 'Kaas'],

printInfo: () => {

console.log('Dit recept is niet glutenvrij.');

}

};

De waarde van een property kan dus een string, getal, boolean, array of zelfs weer een nieuw object zijn. De waarde van een methode is altijd een functie

##### Waardes in een object aanspreken

hoe je vervolgens weer toegang krijgt tot die data

Dit kan door middel van de **punt notatie .** of met de **blokhaak syntax []** :

Afbeelding met tekst, Lettertype, schermopname

Automatisch gegenereerde beschrijving

We gebruiken eigenlijk bijna altijd de punt notatie, tenzij er één van de volgende zaken met de keys aan de hand zijn:

* De key van de property bevat speciale tekens zoals een streepje (-)
* De key van de property is een getal (toegestaan, maar vermijd dit zo veel mogelijk!)
* De key van de property is een variabele (wordt gebruikt in complexere situaties)
* const key = "name";
* const person = {
* name: "Piet",
* }
* // dit lijkt nu misschien niet erg logisch, maar in sommige situaties komt dit goed van pas!
* console.log(person[key]); // geeft Piet

##### Waardes in een object aanpassen

Het aanpassen van property-waardes in een object lijkt erg op het opnieuw toewijzen van een variabele. We gebruiken hier alleen de punt-notatie of blokhaak-syntax:

Afbeelding met tekst, Lettertype, schermopname, diagram

Automatisch gegenereerde beschrijving

<https://vimeo.com/530855407/462c49e220>

Script runnen

Als je de code wil runnen kun je dit doen door het volgende in de terminal in te voeren:

nodemon arrays.js

Als je wisselt van bestand moet je nodemon eerst stoppen (ctrl + c voor zowel Windows als Mac) en dan opnieuw starten met de nieuwe bestandsnaam, zoals bijvoorbeeld:

nodemon objects.js

#### Gecombineerde collecties

Je hebt inmiddels geleerd dat we twee soorten structurele datatypes hebben: arrays voor lijsten en objecten voor dataverzamelingen.

const soccerPlayer = {

name: 'Davy Klaassen',

position: 'middenvelder',

club: 'Ajax',

};

Echter, als we de rest van het Nederlands Elftal ook willen opslaan gaan we het met een object niet redden.

const team = [{}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}];

const team = [

{

name: 'Davy Klaassen',

position: 'middenvelder',

club: 'Ajax',

},

{

name: 'Memphis Depay',

position: 'aanvaller',

club: 'FC Barcelona',

},

{

name: 'Daley Blind',

position: 'verdediger',

club: 'Ajax',

},

];

Hierbij is het gebruikelijk dat ieder object in de array dezelfde keys bevat (zoals name, position en club)

Stel dat we ons object nu willen aanvullen met de datums waarop deze spelers doelpunten hebben gescoort voor het Nederlands Elftal.  
Hierbij is het geen probleem dat alle goals-arrays een andere lengte hebben.

const team = [

{

name: 'Davy Klaassen',

position: 'middenvelder',

club: 'Ajax',

goals: ['18-04-2022', 28-02-2018', '04-01-2018'],

},

{

name: 'Memphis Depay',

position: 'aanvaller',

club: 'FC Barcelona',

goals: ['23-04-2022', '18-04-2022', '12-04-2022', 28-02-2018', '04-01-2018'],

},

{

name: 'Daley Blind',

position: 'verdediger',

club: 'Ajax',

goals: ['23-04-2022', '28-02-2018', '04-01-2018'],

},

];

ook informatie opslaan over de momenten dat spelers gele kaarten kregen: namelijk de datum en de partij waar ze op dat moment tegen speelden. Het gaat om twee stukjes informatie (object) die meerdere keren voor kunnen komen (array)

const team = [

{

name: 'Daley Blind',

position: 'verdediger',

club: 'Ajax',

goals: ['23-04-2022', '28-02-2018', '04-01-2018'],

yellowCards: [

{

competitor: 'FR',

date: '23-04-2022',

},

{

competitor: 'IT',

date: '23-04-2022',

},

{

competitor: 'EN',

date: '23-04-2022',

},

],

},

// andere voetballers staan hier nog tussen...

];

#### Datastructuren en databases

Zelf onze datastructuren opbouwen is één ding, maar vaak zul je informatie ontvangen van een externe bron, zoals een database,

Er zijn twee belangrijke pijlers waaraan je kunt zien of iets een array of een object is:

1. Als het een array is, zal de collectie tussen blokhaken staan. Bijvoorbeeld: ['kaas', 'spinazie', 'eieren'].
2. Als het een object is, zullen er meerdere properties zijn die tussen accolades staan. Elk property wordt gescheiden door een komma en er wordt altijd een : gebruikt om de property-naam van de waarde te scheiden. Bijvoorbeeld: {name: 'Spinazie lasagne', cookingTime: 12}.

// Voorbeeld 1

const students = [

{

name: 'Corrie Claploper',

education: 'HBO Software Development',

},

{

name: 'Robin Peertjes',

education: 'Full Stack Developer Bootcamp',

},

];

In het eerste voorbeeld, students, kun je zien dat dit een array is omdat het buitenste element een blokhaak [] is. Daarna zie je de {}, wat laat zien dat dat er twee objecten in deze array staan. In het geval dat we de opleiding van Robin willen printen, doen we dat als volgt:

console.log(students[1].education); // geeft HBO Software Development

// Voorbeeld 2

const soccerInfo = {

players: [

{

name: 'Daley Blind',

age: 23

},

{

name: 'Memphis Depay',

age: 28

},

],

};

In het tweede voorbeeld, soccerInfo, is het duidelijk dat het een **object** is, omdat het buitenste element een accolade {} is. Daarin zie je een property players staan. De inhoud van die property begint weer met een [] dus dat is een array gevuld met... objecten! Wanneer we *Daley Blind* willen printen, doen we dat als volgt:

console.log(soccerInfo.players[0].name); // geeft Daley Blind

// Voorbeeld 3

const player = {

name: 'Memphis Depay',

age: 28,

teams: ['Atletico Madrid', 'FC Barcelona']

};

In het derde voorbeeld, player, is het duidelijk dat het een **object** is, omdat alle properties in accolades staan. De property teams bevat weer een array - dit zie je aan de []. Wanneer we *FC Barcelona* willen printen, doen we dit als volgt:

console.log(player.teams[1]); // geeft 'FC Barcelona'