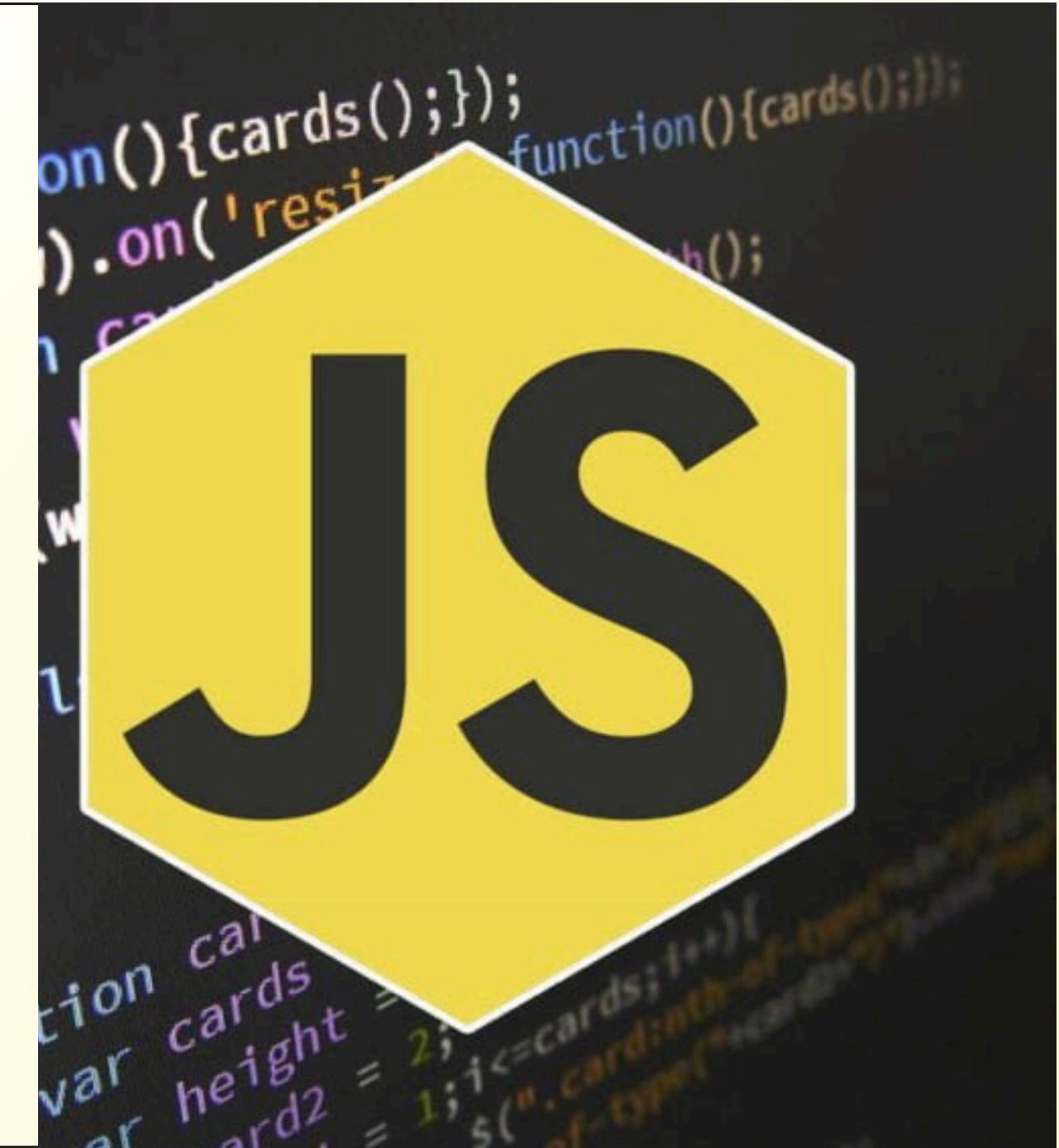


Inleiding tot de JavaScript

- JavaScript maakt webpagina's interactief en "levend". Het is een programmeertaal waarmee je kunt reageren op acties van de gebruiker, zoals klikken, typen en scrollen.



JavaScrpit



JavaScript kan HTML-inhoud wijzigen

- Een van de vele JavaScript HTML-methoden is `getElementById()`.
- Het onderstaande voorbeeld "zoekt" een HTML-element (met `id="demo"`) en verandert de elementinhoud (`innerHTML`) in "Hallo JavaScript":

Voorbeeld:

```
document.getElementById('demo').  
    innerHTML = 'Hello JavaScript';
```

JavaScript kan HTML-eigenschappen wijzigen

- In dit voorbeeld verandert JavaScript de waarde van het `src(source)` attribuut van een ``tag:

Voorbeeld:

```
document.getElementById('demo').  
    setAttribute('src', 'photo.jpg');
```

JavaScript kan CSS-eigenschappen wijzigen

- In dit voorbeeld verandert JavaScript de waarde van de **background-color** attribuut van een **<div>**tag:

Voorbeeld

```
document.getElementById('demo').style  
    .backgroundColor = '#123456';
```

JavaScript kan HTML-inhoud toevoegen

- In dit voorbeeld voegt JavaScript een **<p>**tag toe aan de **<div>**tag met id="demo":

Voorbeeld:

```
var para = document.createElement('p');  
var node = document.createTextNode('This  
is a new paragraph.');//  
para.appendChild(node);  
var element =  
document.getElementById('demo');//  
element.appendChild(para);
```

JavaScript kan HTML-inhoud verwijderen

- In dit voorbeeld verwijdert JavaScript een **<p>**tag uit de **<div>**tag met id="demo":

Voorbeeld:

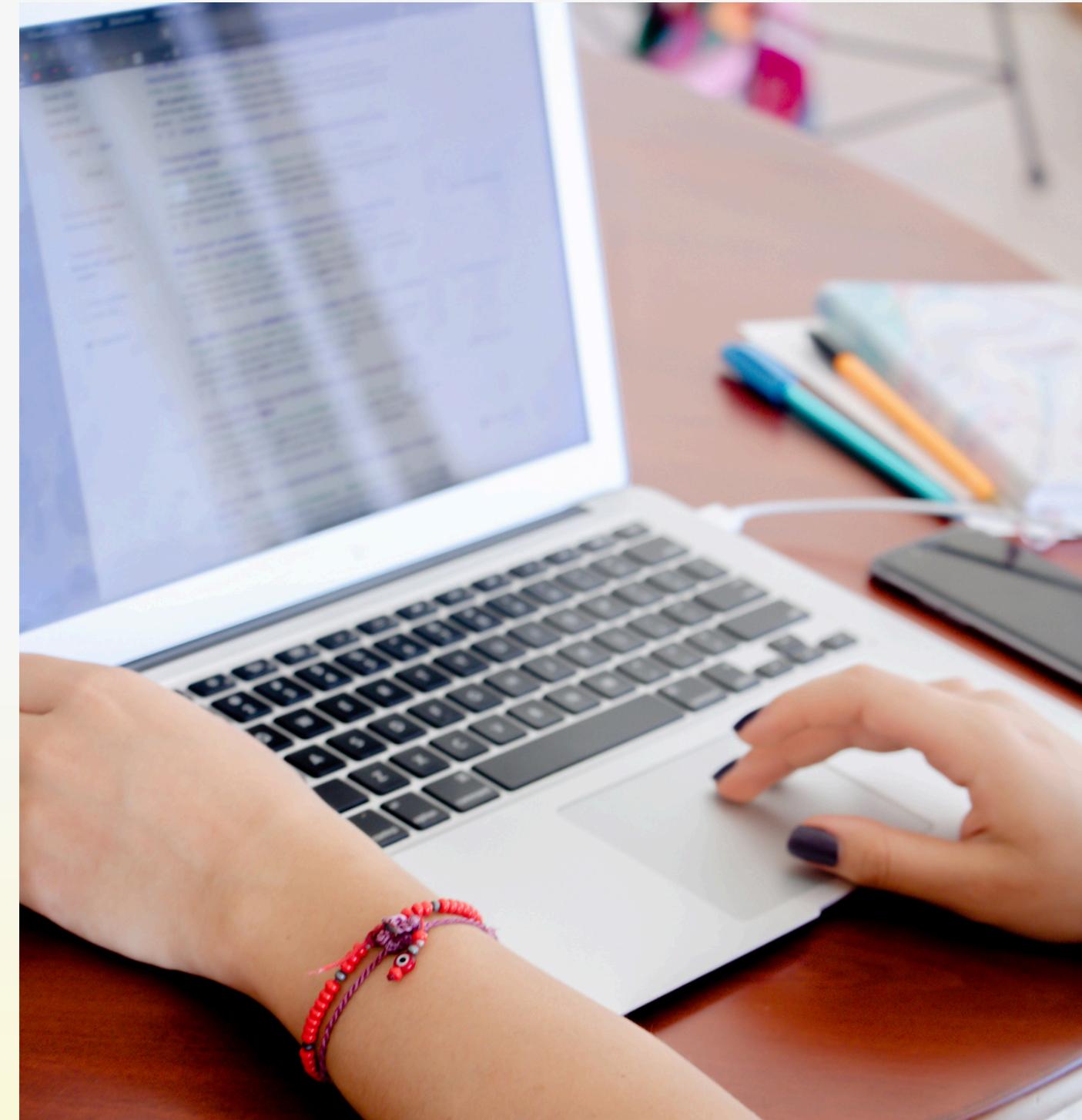
```
var element =  
document.getElementById('demo');  
var child =  
document.getElementsByTagName('p')[0];  
element.removeChild(child);
```

JavaScript basis

- De **<script>** tag kan gebruikt worden om JavaScript te plaatsen in een HTML-bestand.
- in HTML wordt JavaScript-code ingevoegd tussen **<script>** en **</script>** tags.

Voorbeeld:

```
<script>  
document.getElementById('demo')  
.innerHTML = 'Hello JavaScript';  
</script>
```



JAVASCRIPT FUNCTIONS





JavaScript functions

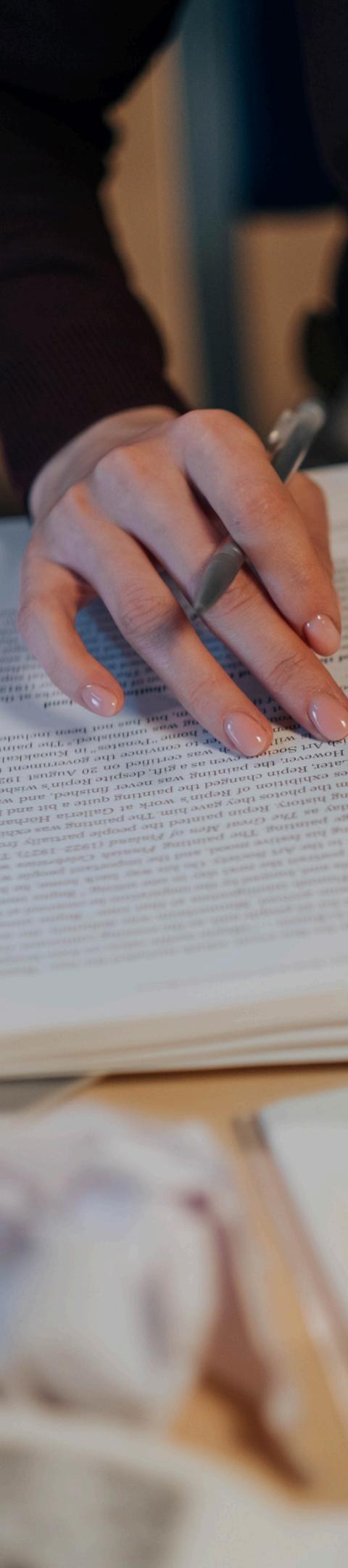
- Een JavaScript functie is een blok code die uitgevoerd wordt wanneer de functie wordt aangeroepen.

JavaScript in <head> of <body>

- U kunt JavaScript in de <head> of <body> plaatsen.

Voorbeeld:

```
<head>
<script>
  document.getElementById('demo').innerHTML = 'Hello JavaScript';
</script>
</head>
```



Extern javascript bestand

- Een extern javascript bestand is een bestand dat JavaScript kan gebruiken.

Externe javascript voordelen

- Het scheidt HTML en code
- Het maakt HTML en JavaScript gemakkelijker te lezen en te onderhouden
- JavaScript-bestanden in cache kunnen het laden van pagina's versnellen

Voorbeeld:

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

Javascript uitvoeren

- JavaScript kan gegevens op verschillende manieren "weergeven":
 - console.log()
 - innerHTML
 - document.write()
 - window.alert()

Voorbeeld innerHTML gebruiken:

```
<script>
  document.getElementById('demo').innerHTML = 'Hello JavaScript';
</script>
```

Voorbeeld console.log gebruiken:

```
<script>
  console.log('Hello JavaScript');
</script>
```

Voorbeeld document.write gebruiken:

```
<script>
  document.write('Hello JavaScript');
</script>
```

Voorbeeld window.alert gebruiken:

```
<script>
  window.alert('Hello JavaScript');
```

Uitvoeren na content loaded

Wanneer onze website wordt ingeladen dan zal onze javascript code onmiddelijk worden uitgevoerd, om dit probleem kunnen we oplossen op verschillende manieren.

- defer attribuut
- window.onload
- DOMContentLoaded eventlistener

defer attribuut

Het defer attribuut wordt gebruikt om bij het inladen van externe javascript bestanden de uitvoering uit te stellen tot wanneer het volledige HTML document is geladen.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">

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

<title>Document</title>
</head>
...
</html>
```

Naast het defer attribuut zijn er nog twee andere mogelijkheden.

window.onload

Het **window.onload**-event treedt op wanneer het volledige HTML-document, inclusief afbeeldingen, stylesheets en externe bronnen, is geladen. Je kunt een eventlistener toevoegen aan dit event om ervoor te zorgen dat je JavaScript-code pas wordt uitgevoerd wanneer alles is geladen.

```
window.onload = function() {  
    // JavaScript-code hier  
};
```

DOMContentLoaded eventlistener

Het **DOMContentLoaded**-event treedt op wanneer het initiële HTML-document is geladen en geparsed, zonder te wachten op afbeeldingen en externe bronnen. Dit event kan eerder plaatsvinden dan **window.onload**, omdat het niet wacht op de volledige pagina.

```
document.addEventListener('DOMContentLoaded', function() {  
    // JavaScript-code hier  
});
```

Javascript statements

- Javascript statements zijn statements die uitgevoerd worden door de JavaScript engine.

Voorbeeld:

```
<script>
    var x = 5;
    var y = 6;
    var z = x + y;
    document.getElementById('demo').innerHTML = z;
</script>
```

- Semi-colons (;) zijn nodig om de statements af te sluiten.

Voorbeeld:

```
<script>
    var naam = 'John Doe';
    document.getElementById('demo').innerHTML = naam;
</script>
```

Javascript comments

- Javascript comments zijn teksten die niet uitgevoerd worden door de JavaScript engine.

Voorbeeld:

```
<script>
    // Dit is een commentaar
    var x = 5;
    var y = 6;
    var z = x + y;
    document.getElementById('demo').innerHTML = z;
</script>
```

Javascript variables

- Javascript variables zijn variabelen die een waarde kunnen bevatten.

Voorbeeld:

```
<script>  
    var x = 5;  
    var y = 6;  
    var z = x + y;  
    document.getElementById('demo').innerHTML = z;  
</script>
```

Javascript operators

- Javascript operators zijn operatoren die een waarde kunnen bevatten.
- Javascript gebruikt **rekenkundige operatoren**, **logische operatoren**, **string operatoren**, **array operatoren** en de **assignment operator (=)**.

Voorbeeld:

```
<script>
    var x = 5;
    var y = 6;
    var z = x + y;
    document.getElementById('demo').innerHTML = z;
</script>
```

Javascript variabelen

Drie manieren om een JavaScript-variabele te declareren.

- var variabele = waarde;
- let variabele = waarde;
- const variabele = waarde;

Wanneer javascript var wordt gebruikt?

JavaScript var is een globale variabele. Als u wilt dat uw code in een oudere browser wordt uitgevoerd, moet u var gebruiken als kernwoord.
Voorbeeld:

Voorbeeld:

```
var a = 1;  
let b = 2;  
const c = 3;
```

Note: var is een globale variabele.

Note: let is een lokale variabele.

Note: const is een constante variabele.

Javascript operators

Toewijzingsoperators zijn de operatoren die gebruikt worden om een waarde te kopieren naar een andere variabele.

Voorbeeld:

```
var a = 1;  
var b = a;  
b = 2;  
console.log(a); // 1  
console.log(b); // 2
```

Javascript operators

Rekenkundige operatoren zijn de operatoren die gebruikt worden om een berekening uit te voeren.

Voorbeeld:

```
var a = 1;  
var b = 2;  
var c = a + b;  
console.log(c); // 3
```

Javascript operators

Tekenreeksoperators zijn de operatoren die gebruikt worden om tekst te maken.

Voorbeeld:

```
var a = "1";
var b = "2";
var c = a + b;
console.log(c); // 12
```

Javascript operators

Relational operators zijn de operatoren die gebruikt worden om een waarde te bepalen of een bepaalde voorwaarde is.

Voorbeeld:

```
var a = 1;  
var b = 2;  
var c = a > b;  
console.log(c); // false
```

Javascript operators

Vergelijgingsoperators zijn de operatoren die gebruikt worden om een waarde te bepalen of een bepaalde voorwaarde is.

Voorbeeld:

```
var a = 1;  
var b = 2;  
var c = a == b;  
console.log(c); // false
```

Javascript operators

Logische operatoren zijn de operatoren die gebruikt worden om een waarde te bepalen of een bepaalde voorwaarde is.

Voorbeeld:

```
var a = 1;
var b = 2;
var c = a == b;
var d = a != b;
console.log(c); // false
console.log(d); // true
```

Javascript operators

Typeof operator is de operator die gebruikt wordt om het type van een variabele te bepalen.

Voorbeeld:

```
var a = 1;
var b = "2";
var c = a == b;
console.log(typeof a); // number
console.log(typeof b); // string
console.log(typeof c); // boolean
```

Javascript data types

Javascript variabelen kunnen een waarde hebben die een waarde heeft van een bepaalde data type.

- Number
- String
- Boolean
- Object
- Array
- Function
- Undefined
- Null
- NaN
- Infinity

Number datatype voorbeeld:

```
var age = 30;  
console.log(age); // 30
```

String datatype voorbeeld:

```
var name = "John";  
console.log(name); // John
```

Boolean datatype voorbeeld:

```
var isMarried = false;  
console.log(isMarried); // false
```

Javascript data types

Object datatype voorbeeld:

```
var person = {  
    firstName: "John",  
    lastName: "Doe",  
    age: 30  
};  
console.log(person); // {firstName: "John", lastName: "Doe", age: 30}
```

Array datatype voorbeeld:

```
var cars = ["Saab", "Volvo", "BMW"];  
console.log(cars); // ["Saab", "Volvo", "BMW"]
```

Javascript data types

Function datatype voorbeeld:

```
var sayHello = function() {  
    console.log("Hello");  
}  
sayHello(); // Hello
```

Undefined datatype voorbeeld:

```
var x;  
console.log(x); // undefined
```

Javascript data types

Null datatype voorbeeld:

```
var x = null;  
console.log(x); // null
```

Nan datatype voorbeeld:

```
var x = NaN;  
console.log(x); // NaN
```

Infinity datatype voorbeeld:

```
var x = Infinity;  
console.log(x); // Infinity
```

Javascript functies

een functie is een codeblok dat uitgevoerd wordt wanneer u een bepaalde voorwaarde heeft.

Voorbeeld:

```
function hello() {  
    console.log("Hello World!");  
}
```

Javascript functie syntax

Een JavaScript-functie wordt gedefinieerd met het volgende formaat:

```
- functionName(parameter1, parameter2, ...) {  
    // code  
}
```

functionName is de naam van de functie.

Function return

Een functie kan een waarde teruggeven met het return-statement.

```
function add(a, b) {  
    return a + b;  
}
```

De functie add() geeft de som van de twee parameters terug.

Waagom functies?

Functies zijn een belangrijk onderdeel van JavaScript omdat ze het mogelijk maken om code te hergebruiken.

- Een functie kan worden uitgevoerd wanneer u het nodig hebt.
- U kunt dezelfde functie meerdere keren uitvoeren met verschillende argumenten, om verschillende resultaten te krijgen.

Functies gebruikt als variabelen

U kunt een functie toewijzen aan een variabele.

```
var x = function (a, b) {return a * b};
```

- De variabele x is nu een functie.
- De functie kan worden uitgevoerd door de variabele te gebruiken als een functie.

```
var x = function (a, b) {return a * b};  
var z = x(4, 3);
```

De variabele `z` zal nu de waarde 12 bevatten.

Local variabelen

Variabelen die binnen een functie worden gemaakt, kunnen enkel gebruikt worden binnen de functie.

```
function myFunction() {  
    var carName = "Volvo";  
}
```

- De variabele carName is lokaal voor de functie myFunction().
- Andere functies kunnen geen toegang krijgen tot de variabele carName.

Global variabelen

Variabelen die buiten een functie worden gemaakt, zijn globaal voor het hele script.

```
var carName = "Volvo";  
function myFunction() {  
    // code here can use carName  
}
```

- De variabele carName is globaal voor het script.
- Alle functies in het script kunnen toegang krijgen tot de variabele carName.

Javascript objecten

- een object is een verzameling van gegevens en functies die samen een logisch geheel vormen.
- een object is een variabele die meerdere waarden kan bevatten.
- een object kan een functie bevatten die een actie uitvoert wanneer het object wordt aangeroepen.

Voorbeeld:

```
var person = {
    firstName: "John",
    lastName : "Doe",
    id      : 5566,
    fullName : function() {
        return this.firstName + " " + this.lastName;
    }
};
```

Javascript object syntax

Hieronder zie je een voorbeeld hoe we een javascript object definiëren:

```
var objectName = {  
    property1: value1,  
    property2: value2,  
    ...  
};
```

- De naam van het object is de naam van het object.
- De naam van de eigenschap is de naam van de eigenschap.
- De waarde van de eigenschap is de waarde van de eigenschap.

Een instantie van een javascript object kunnen we op volgende manier gaan aanmaken:

```
var objectName = new Object();
```

Object properties

U kunt de eigenschappen van een object ophalen met de naam van het object gevolgd door een punt en de naam van de eigenschap.

```
var person = {  
    firstName: "John",  
    lastName : "Doe",  
    id      : 5566  
};
```

- De naam van het object is person.
- De naam van de eigenschap is firstName.
- De waarde van de eigenschap is John.

U kunt de waarde van de eigenschap ophalen met de volgende code:

```
person.firstName;
```

Object methods

U kunt een functie in een object definiëren als een methode.

```
var person = {  
    firstName: "John",  
    lastName : "Doe",  
    id      : 5566,  
    fullName : function() {  
        return this.firstName + " " + this.lastName;  
    }  
};
```

- De naam van de functie is `fullName`.

De functie kan worden uitgevoerd door de naam van het object gevolgd door een punt en de naam van de functie te gebruiken.

```
person.fullName();
```

this keyword

In JavaScript verwijst het trefwoord `this` naar het object waarop het wordt gebruikt.

In een methode, `this` verwijst naar het eigen object.

```
var person = {
    firstName: "John",
    lastName : "Doe",
    id      : 5566,
    fullName : function() {
        return this.firstName + " " + this.lastName;
    }
};
```

- In dit voorbeeld, `this` verwijst naar het eigen object `person`.
- In een objectmethode `this` verwijst naar het eigen object.

HTML:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Event propagation</title>
    <link rel="stylesheet" href="style.css">
  </head>
  <body>
    <form>
      <button type="submit">Submit</button>
    </form>
    <script src="script.js"></script>
  </body>
</html>
```

String methoden

In JavaScript hebben strings een aantal methoden. Deze methoden kunnen we gebruiken om strings te manipuleren. In deze les gaan we kijken naar de volgende methoden:

- **toUpperCase()**
- **toLowerCase()**
- **trim()**
- **split()**
- **replace()**
- **charAt()**
- **indexOf()**
- **lastIndexOf()**
- **substring()**
- **slice()**
- **startsWith()**
- **endsWith()**
- **includes()**

toUpperCase()

De `toUpperCase()` methode zet alle letters in een string om naar hoofdletters. De methode heeft geen parameters en geeft een nieuwe string terug.

```
let str = "Hello World!";
let res = str.toUpperCase();
console.log(res); // "HELLO WORLD!"
```

toLowerCase()

De `toLowerCase()` methode zet alle letters in een string om naar kleine letters. De methode heeft geen parameters en geeft een nieuwe string terug.

```
let str = "Hello World!";
let res = str.toLowerCase();
console.log(res); // "hello world!"
```

trim()

De trim() methode verwijdert spaties aan het begin en het einde van een string. De methode heeft geen parameters en geeft een nieuwe string terug.

```
let str = "Hello World!";
let res = str.toUpperCase();
console.log(res); // "HELLO WORLD!"
```

split()

De split() methode splitst een string op in een array van substrings. De methode heeft één parameter: een string die wordt gebruikt om de string op te splitsen. De methode geeft een nieuwe array terug.

```
let str = "Hello World!";
let res = str.split(" ");
console.log(res); // ["Hello", "World!"]
```

replace()

De replace() methode vervangt een deel van een string met een andere string. De methode heeft twee parameters: een string die wordt vervangen en een string die wordt gebruikt om de string te vervangen. De methode geeft een nieuwe string terug.

```
let str = "Hello World!";
let res = str.replace("World", "Dolly");
console.log(res); // "Hello Dolly!"
```

charAt()

De charAt() methode geeft het karakter op een bepaalde positie in een string terug. De methode heeft één parameter: een nummer dat de positie van het karakter aangeeft. De methode geeft een nieuwe string terug.

```
let str = "Hello World!";
let res = str.charAt(0);
console.log(res); // "H"
```

indexOf()

De indexOf() methode geeft de positie van de eerste keer dat een substring voorkomt in een string terug. De methode heeft één parameter: een string die wordt gezocht. De methode geeft een nummer terug.

```
let str = "Hello World!";
let res = str.indexOf("World");
console.log(res); // 6
```

lastIndexOf()

De lastIndexOf() methode geeft de positie van de laatste keer dat een substring voorkomt in een string terug. De methode heeft één parameter: een string die wordt gezocht. De methode geeft een nummer terug.

```
let str = "Hello World!";
let res = str.lastIndexOf("World");
console.log(res); // 6
```

substring()

De substring() methode geeft een deel van een string terug. De methode heeft twee parameters: een nummer dat de positie van het eerste karakter aangeeft en een nummer dat de positie van het laatste karakter aangeeft. De methode geeft een nieuwe string terug.

```
let str = "Hello World!";
let res = str.substring(0, 5);
console.log(res); // "Hello"
```

slice()

De slice() methode geeft een deel van een string terug. De methode heeft twee parameters: een nummer dat de positie van het eerste karakter aangeeft en een nummer dat de positie van het laatste karakter aangeeft. De methode geeft een nieuwe string terug.

```
let str = "Hello World!";
let res = str.slice(0, 5);
console.log(res); // "Hello"
```

startsWith()

De startsWith() methode geeft true terug als een string begint met een bepaalde substring. De methode heeft één parameter: een string die wordt gezocht. De methode geeft een boolean terug.

```
let str = "Hello World!";
let res = str.startsWith("Hello");
console.log(res); // true
```

endsWith()

De endsWith() methode geeft true terug als een string eindigt met een bepaalde substring. De methode heeft één parameter: een string die wordt gezocht. De methode geeft een boolean terug.

```
let str = "Hello World!";
let res = str.endsWith("World");
console.log(res); // true
```

includes()

De includes() methode geeft true terug als een string een bepaalde substring bevat. De methode heeft één parameter: een string die wordt gezocht. De methode geeft een boolean terug.

```
let str = "Hello World!";
let res = str.includes("World");
console.log(res); // true
```

NUMBER METHODEN



Number methoden

In JavaScript hebben numbers een aantal methoden. Deze methoden kunnen we gebruiken om numbers te manipuleren. In deze les gaan we kijken naar de volgende methoden:

- `toFixed()`
- `toPrecision()`
- `toString()`
- `toExponential()`
- `valueOf()`
- `parseInt()`

Number methoden

toFixed()

De `toFixed()` methode formateert een `number` met een specifiek aantal decimalen. De methode heeft één parameter: het aantal decimalen. De methode geeft een nieuwe string terug.

```
let num = 5.56789;
let n = num.toFixed(2);
console.log(n); // "5.57"
```

toPrecision()

De `toPrecision()` methode formateert een `number` met een specifiek aantal cijfers. De methode heeft één parameter: het aantal cijfers. De methode geeft een nieuwe string terug.

```
let num = 5.56789;
let n = num.toPrecision(2);
console.log(n); // "5.6"
```

Number methoden

toString()

De `toString()` methode zet een `number` om naar een `string`. De methode heeft één parameter: de basis waarin de `number` wordt omgezet. De methode geeft een nieuwe `string` terug.

```
let num = 10;
let n = num.toString();
console.log(n); // "10"
```

toExponential()

De `toExponential()` methode formateert een `number` als een exponent. De methode heeft één parameter: het aantal decimalen. De methode geeft een nieuwe `string` terug.

```
let num = 5.56789;
let n = num.toExponential(2);
console.log(n); // "5.57e+0"
```

Number methoden

valueOf()

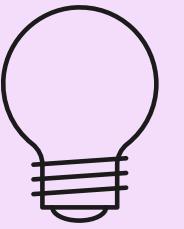
De valueOf() methode geeft de waarde van een number terug. De methode heeft geen parameters en geeft een number terug.

```
let num = 10;
let n = num.valueOf();
console.log(n); // 10
```

parseInt()

De parseInt() methode zet een string om naar een number. De methode heeft één parameter: de string die wordt omgezet. De methode geeft een number terug.

```
let str = "10";
let n = parseInt(str);
console.log(n); // 10
```



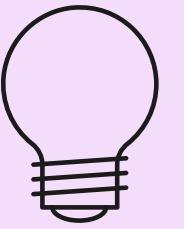
Wat zijn arrays?

Arrays zijn een soort variabele die meerdere waarden kan bevatten. Arrays worden vaak gebruikt om een lijst van waarden op te slaan. In JavaScript worden arrays aangegeven met vierkante haken. De waarden in een array worden gescheiden door een komma.

```
let colors = ["red", "green", "blue"];
```

Arrays kunnen ook leeg zijn. In dat geval worden de vierkante haken leeg gelaten.

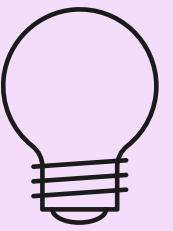
```
let colors = [];
```



Array methoden

Arrays hebben een aantal methoden. Deze methoden kunnen we gebruiken om arrays te manipuleren. In deze les gaan we kijken naar de volgende methoden:

- push()
- pop()
- shift()
- unshift()
- indexOf()
- lastIndexOf()
- slice()
- splice()
- join()
- sort()
- reverse()
- concat()
- includes()
- find()
- findIndex()
- forEach()
- map()
- filter()
- reduce()
- split()



Array methoden

push()

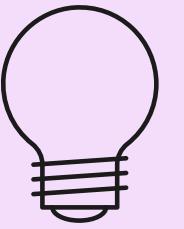
De push() methode voegt een of meerdere waarden aan het einde van een array toe. De methode heeft één of meerdere parameters: de waarden die aan de array worden toegevoegd. De methode geeft de nieuwe lengte van de array terug.

```
let colors = ["red", "green", "blue"];
let newLength = colors.push("yellow");
console.log(newLength); // 4
console.log(colors); // ["red", "green", "blue", "yellow"]
```

pop()

De pop() methode verwijdert de laatste waarde van een array. De methode heeft geen parameters en geeft de verwijderde waarde terug.

```
let colors = ["red", "green", "blue"];
let lastColor = colors.pop();
console.log(lastColor); // "blue"
console.log(colors); // ["red", "green"]
```



Array methoden

shift()

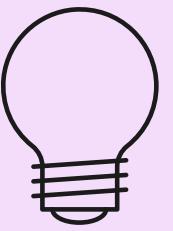
De shift() methode verwijdert de eerste waarde van een array. De methode heeft geen parameters en geeft de verwijderde waarde terug.

```
let colors = ["red", "green", "blue"];
let firstColor = colors.shift();
console.log(firstColor); // "red"
console.log(colors); // ["green", "blue"]
```

unshift()

De unshift() methode voegt een of meerdere waarden aan het begin van een array toe. De methode heeft één of meerdere parameters: de waarden die aan de array worden toegevoegd. De methode geeft de nieuwe lengte van de array terug.

```
let colors = ["red", "green", "blue"];
let newLength = colors.unshift("yellow");
console.log(newLength); // 4
console.log(colors); // ["yellow", "red", "green", "blue"]
```



Array methoden

indexOf()

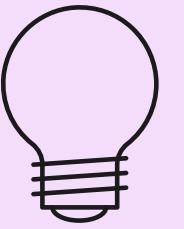
De indexOf() methode zoekt een waarde in een array en geeft de index van de eerste overeenkomst terug. De methode heeft één parameter: de waarde die wordt gezocht. De methode geeft -1 terug als de waarde niet wordt gevonden.

```
let colors = ["red", "green", "blue"];
let index = colors.indexOf("green");
console.log(index); // 1
```

lastIndexOf()

De lastIndexOf() methode zoekt een waarde in een array en geeft de index van de laatste overeenkomst terug. De methode heeft één parameter: de waarde die wordt gezocht. De methode geeft -1 terug als de waarde niet wordt gevonden.

```
let colors = ["red", "green", "blue", "green"];
let index = colors.lastIndexOf("green");
console.log(index); // 3
```



Array methoden

slice()

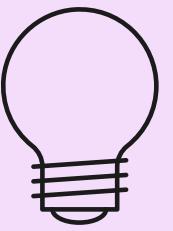
De slice() methode maakt een kopie van een deel van een array. De methode heeft twee parameters: de index van het eerste element dat wordt gekopieerd en de index van het laatste element dat wordt gekopieerd. De methode geeft een nieuwe array terug.

```
let colors = ["red", "green", "blue", "yellow", "purple"];
let newColors = colors.slice(1, 3);
console.log(newColors); // ["green", "blue"]
```

splice()

De splice() methode verwijdert een deel van een array en voegt nieuwe waarden toe. De methode heeft drie parameters: de index van het eerste element dat wordt verwijderd, het aantal elementen dat wordt verwijderd en de waarden die worden toegevoegd. De methode geeft een nieuwe array terug.

```
let colors = ["red", "green", "blue", "yellow", "purple"];
let newColors = colors.splice(1, 2, "orange", "white");
console.log(newColors); // ["green", "blue"]
console.log(colors); // ["red", "orange", "white", "yellow", "purple"]
```



Array methoden

join()

De join() methode maakt een string van de waarden in een array. De methode heeft één parameter: het teken dat wordt gebruikt om de waarden te scheiden. De methode geeft een string terug.

```
let colors = ["red", "green", "blue"];
let string = colors.join(", ");
console.log(string); // "red, green, blue"
```

sort()

De sort() methode sorteert de waarden in een array. De methode heeft één parameter: een functie die wordt gebruikt om de waarden te vergelijken. De methode geeft de gesorteerde array terug.

```
let numbers = [4, 2, 5, 1, 3];
numbers.sort();
console.log(numbers); // [1, 2, 3, 4, 5]
```



Array methoden

reverse()

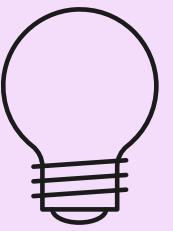
De reverse() methode keert de volgorde van de waarden in een array om. De methode heeft geen parameters en geeft de omgekeerde array terug.

```
let colors = ["red", "green", "blue"];
colors.reverse();
console.log(colors); // ["blue", "green", "red"]
```

forEach()

De forEach() methode voert een functie uit voor elk element in een array. De methode heeft één parameter: de functie die wordt uitgevoerd. De functie heeft drie parameters: de waarde van het element, de index van het element en de array zelf.

```
let colors = ["red", "green", "blue"];
colors.forEach(function(color, index, array) {
  console.log(color, index, array);
});
```



Array methoden

map()

De map() methode voert een functie uit voor elk element in een array en maakt een nieuwe array met de resultaten. De methode heeft één parameter: de functie die wordt uitgevoerd. De functie heeft drie parameters: de waarde van het element, de index van het element en de array zelf. De methode geeft een nieuwe array terug.

```
let numbers = [1, 4, 9];
let roots = numbers.map(function(num) {
  return Math.sqrt(num);
});

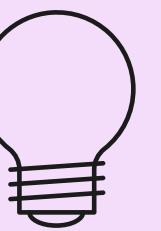
console.log(roots); // [1, 2, 3]
```

filter()

De filter() methode voert een functie uit voor elk element in een array en maakt een nieuwe array met de elementen waarvoor de functie true teruggeeft. De methode heeft één parameter: de functie die wordt uitgevoerd. De functie heeft drie parameters: de waarde van het element, de index van het element en de array zelf. De methode geeft een nieuwe array terug.

```
let numbers = [1, 4, 9];
let even = numbers.filter(function(num) {
  return num % 2 === 0;
});

console.log(even); // [4]
```



Array methoden

every()

De `every()` methode voert een functie uit voor elk element in een array en geeft `true` terug als de functie `true` teruggeeft voor elk element. De methode heeft één parameter: de functie die wordt uitgevoerd. De functie heeft drie parameters: de waarde van het element, de index van het element en de array zelf. De methode geeft `true` of `false` terug.

```
let numbers = [1, 4, 9];
let allEven = numbers.every(function(num) {
  return num % 2 === 0;
});

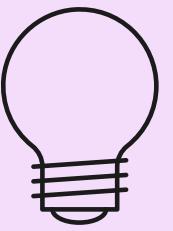
console.log(allEven); // false
```

some()

De `some()` methode voert een functie uit voor elk element in een array en geeft `true` terug als de functie `true` teruggeeft voor minstens één element. De methode heeft één parameter: de functie die wordt uitgevoerd. De functie heeft drie parameters: de waarde van het element, de index van het element en de array zelf. De methode geeft `true` of `false` terug.

```
let numbers = [1, 4, 9];
let someEven = numbers.some(function(num) {
  return num % 2 === 0;
});

console.log(someEven); // true
```



Array methoden

reduce()

De reduce() methode voert een functie uit voor elk element in een array en geeft een enkele waarde terug. De methode heeft twee parameters: de functie die wordt uitgevoerd en de initiële waarde. De functie heeft vier parameters: de accumulatiewaarde, de waarde van het element, de index van het element en de array zelf. De methode geeft een waarde terug.

```
let numbers = [1, 4, 9];
let sum = numbers.reduce(function(accumulator, num) {
  return accumulator + num;
}, 0);

console.log(sum); // 14
```

split()

De split() methode maakt een array van een string. De methode heeft één parameter: het teken dat wordt gebruikt om de string te splitsen. De methode geeft een array terug.

```
let string = "red, green, blue";
let colors = string.split(", ");
console.log(colors); // ["red", "green", "blue"]
```

Dates & Math

Wat zijn dates?

Dates zijn een soort variabele die een datum en tijd opslaat. In JavaScript worden dates aangegeven met de Date constructor. De constructor heeft één parameter: de datum en tijd die opgeslagen moet worden. Als er geen parameter wordt meegegeven, wordt de huidige datum en tijd opgeslagen.

```
let date = new Date();
console.log(date); // 2020-03-11T13:00:00.000Z
```

Date methoden

Dates hebben een aantal methoden. Deze methoden kunnen we gebruiken om de datum en tijd te manipuleren. In deze les gaan we kijken naar de volgende methoden:

- getDate()
- getDay()
- getFullYear()
- getHours()
- getMilliseconds()
- getMinutes()
- getMonth()
- getSeconds()
- getTime()
- setDate()
- setFullYear()
- setHours()

Dates & Math

getDate()

De getDate() methode geeft de dag van de maand terug. De methode heeft geen parameters.

```
let date = new Date();
let day = date.getDate();
console.log(day); // 11
```

getDay()

De getDay() methode geeft de dag van de week terug. De methode heeft geen parameters.

```
let date = new Date();
let day = date.getDay();
console.log(day); // 2
```

Dates & Math

getFullYear()

De `getFullYear()` methode geeft het jaar terug. De methode heeft geen parameters.

```
let date = new Date();
let year = date.getFullYear();
console.log(year); // 2020
```

getHours()

De `getHours()` methode geeft het uur terug. De methode heeft geen parameters.

```
let date = new Date();
let hours = date.getHours();
console.log(hours); // 13
```

Dates & Math

getMilliseconds()

De getMilliseconds() methode geeft het aantal milliseconden terug. De methode heeft geen parameters.

```
let date = new Date();
let milliseconds = date.getMilliseconds();
console.log(milliseconds); // 0
```

getMinutes()

De getMinutes() methode geeft het aantal minuten terug. De methode heeft geen parameters.

```
let date = new Date();
let minutes = date.getMinutes();
console.log(minutes); // 0
```

Dates & Math

getMonth()

De getMonth() methode geeft het maand terug. De methode heeft geen parameters.

```
let date = new Date();
let month = date.getMonth();
console.log(month); // 2
```

getSeconds()

De getSeconds() methode geeft het aantal seconden terug. De methode heeft geen parameters.

```
let date = new Date();
let seconds = date.getSeconds();
console.log(seconds); // 0
```

Dates & Math

getTime()

De `getTime()` methode geeft het aantal milliseconden sinds 1 januari 1970 terug. De methode heeft geen parameters.

```
let date = new Date();
let time = date.getTime();
console.log(time); // 1583948000000
```

setDate()

De `setDate()` methode verandert de dag van de maand. De methode heeft één parameter: de dag van de maand.

```
let date = new Date();
date.setDate(12);
console.log(date); // 2020-03-12T13:00:00.000Z
```

Dates & Math

setFullYear()

De `setFullYear()` methode verandert het jaar. De methode heeft één parameter: het jaar.

```
let date = new Date();
date.setFullYear(2021);
console.log(date); // 2021-03-11T13:00:00.000Z
```

setHours()

De `setHours()` methode verandert het uur. De methode heeft één parameter: het uur.

```
let date = new Date();
date.setHours(14);
console.log(date); // 2020-03-11T14:00:00.000Z
```

Math

Wat is Math?

Math is een object dat een aantal methoden heeft die je kunt gebruiken om wiskundige berekeningen te maken. In deze les gaan we kijken naar de volgende methoden:

- Math.abs()
- Math.ceil()
- Math.floor()
- Math.max()
- Math.min()
- Math.pow()
- Math.random()

Math

Math.abs()

De Math.abs() methode geeft de absolute waarde van een getal terug. De methode heeft één parameter: het getal waarvan de absolute waarde moet worden berekend.

```
let number = -5;  
let absolute = Math.abs(number);  
console.log(absolute); // 5
```

Math.ceil()

De Math.ceil() methode rondt een getal af naar het dichtstbijzijnde gehele getal. De methode heeft één parameter: het getal dat afgerond moet worden.

```
let number = 5.5;  
let rounded = Math.ceil(number);  
console.log(rounded); // 6
```

Math

Math.floor()

De Math.floor() methode rondt een getal af naar het dichtstbijzijnde gehele getal. De methode heeft één parameter: het getal dat afgerond moet worden.

```
let number = 5.5;  
let rounded = Math.floor(number);  
console.log(rounded); // 5
```

Math.max()

De Math.max() methode geeft het grootste getal uit een lijst van getallen terug. De methode heeft één parameter: een lijst van getallen.

```
let numbers = [1, 2, 3, 4, 5];  
let max = Math.max(...numbers);  
console.log(max); // 5
```

Math

Math.min()

De Math.min() methode geeft het kleinste getal uit een lijst van getallen terug. De methode heeft één parameter: een lijst van getallen.

```
let numbers = [1, 2, 3, 4, 5];
let min = Math.min(...numbers);
console.log(min); // 1
```

Math.pow()

De Math.pow() methode geeft het resultaat van een getal tot een bepaalde macht terug. De methode heeft twee parameters: het getal en de macht.

```
let number = 2;
let power = 3;
let result = Math.pow(number, power);
console.log(result); // 8
```

Math

Math.random()

De Math.random() methode geeft een willekeurig getal tussen 0 en 1 terug. De methode heeft geen parameters.

```
let random = Math.random();
console.log(random); // 0.123456789
```

Javascript vergelijkingen en logische operators

Wat zijn vergelijkingen?

Vergelijkingen zijn een soort operator die twee waardes met elkaar vergelijkt.

- == (gelijk aan)
- === (identiek aan)
- != (niet gelijk aan)
- !== (niet identiek aan)
- > (groter dan)
- < (kleiner dan)
- >= (groter dan of gelijk aan)
- <= (kleiner dan of gelijk aan)
- && (en)
- || (of)
- ! (niet)
- ? (ternary operator)

[BACK TO AGENDA PAGE](#) →

if statements

- U kunt een if statement gebruiken om een stuk code uit te voeren als een voorwaarde waar is.
- In JavaScript hebben we de volgende voorwaardelijke statements:
 - **if** – gebruikt om een stuk code uit te voeren als een voorwaarde waar is
 - **else** – gebruikt om een stuk code uit te voeren als de voorwaarde niet waar is
 - **else if** – gebruikt om een stuk code uit te voeren als een andere voorwaarde waar is
 - **switch** – gebruikt om een stuk code uit te voeren uit een aantal mogelijke opties

if statement

De if statement wordt gebruikt om een stuk code uit te voeren als een voorwaarde waar is.

Syntax:

```
if (condition) {  
    // code to be executed if condition is true  
}
```

else statement

- De else statement wordt gebruikt om een stuk code uit te voeren als de voorwaarde niet waar is.

Syntax:

```
if (condition) {  
    // code to be executed if condition is true  
} else {  
    // code to be executed if condition is false  
}
```

Voorbeeld:

```
let age = 17;  
  
if (age >= 18) {  
    console.log("Je mag naar binnen");  
} else {  
    console.log("Je mag niet naar binnen");  
}
```

else if statement

De else if statement wordt gebruikt om een stuk code uit te voeren als een andere voorwaarde waar is.

Syntax:

```
if (condition1) {  
    // code to be executed if condition1 is true  
} else if (condition2) {  
    // code to be executed if the condition1 is false and condition2 is true  
} else {  
    // code to be executed if condition1 is false and condition2 is false  
}
```

Voorbeeld:

```
let age = 17;  
  
if (age >= 18) {  
    console.log("Je mag naar binnen");  
} else if (age >= 16) {  
    console.log("Je mag naar binnen, maar alleen onder begeleiding");  
} else {  
    console.log("Je mag niet naar binnen");  
}
```

Switch statement

De switch statement wordt gebruikt om een stuk code uit te voeren uit een aantal mogelijke opties.

Syntax:

```
switch (expression) {  
    case x:  
        // code block  
        break;  
    case y:  
        // code block  
        break;  
    default:  
        // code block  
}
```

break statement

De break statement wordt gebruikt om de uitvoering van een switch statement te stoppen.

Syntax:

```
switch (expression) {  
    case x:  
        // code block  
        break;  
    case y:  
        // code block  
        break;  
    default:  
        // code block  
}
```

default statement

De default statement wordt gebruikt om een stuk code uit te voeren als geen van de case statements waar is.

Syntax:

```
switch (expression) {  
    case x:  
        // code block  
        break;  
    case y:  
        // code block  
        break;  
    default:  
        // code block  
}
```

Verschillende soorten loops

Er zijn verschillende soorten loops in JavaScript.

- **for** – wordt gebruikt om een stuk code uit te voeren een bepaald aantal keren
- **for/in** – wordt gebruikt om een stuk code uit te voeren voor elk eigenschap in een object
- **for/of** – wordt gebruikt om een stuk code uit te voeren voor elk element in een array
- **while** – wordt gebruikt om een stuk code uit te voeren zolang een voorwaarde waar is
- **do/while** – wordt gebruikt om een stuk code uit te voeren zolang een voorwaarde waar is

for loop

De for loop wordt gebruikt om een stuk code uit te voeren een bepaald aantal keren.

Syntax:

```
for (statement 1; statement 2; statement 3) {  
    // code block to be executed  
}
```

Verschillende soorten loops

for/in loop

De for/in loop wordt gebruikt om een stuk code uit te voeren voor elk eigenschap in een object.

Syntax:

```
for (key in object) {  
    // code block to be executed  
}
```

for/of loop

De for/of loop wordt gebruikt om een stuk code uit te voeren voor elk element in een array.

Syntax:

```
for (variable of object) {  
    // code block to be executed  
}
```

Verschillende soorten loops

while loop

De while loop wordt gebruikt om een stuk code uit te voeren zolang een voorwaarde waar is.

Syntax:

```
while (condition) {  
    // code block to be executed  
}
```

do/while loop

De do/while loop wordt gebruikt om een stuk code uit te voeren zolang een voorwaarde waar is.

Syntax:

```
do {  
    // code block to be executed  
}  
  
while (condition);
```

Verschillende soorten loops

break statement

De break statement wordt gebruikt om de uitvoering van een for, for/in, for/of, while of do/while statement te stoppen.

Syntax:

```
for (let i = 0; i < 5; i++) {  
  if (i === 3) {  
    break;  
  }  
  console.log(i);  
}
```

continue statement

De continue statement wordt gebruikt om de uitvoering van de huidige iteratie van een for, for/in, for/of, while of do/while statement te stoppen en gaat verder met de volgende iteratie.

Syntax:

```
for (let i = 0; i < 5; i++) {  
  if (i === 3) {  
    continue;  
  }  
  console.log(i);  
}
```

Javascript Classes

- Classes zijn een manier om objecten te maken in JavaScript. Classes zijn een soort template voor objecten. Classes worden gebruikt om objecten te maken met dezelfde eigenschappen en methodes.
- Classes worden gemaakt met de **class** keyword. Classes hebben een constructor. De constructor wordt gebruikt om objecten te maken met dezelfde eigenschappen en methodes.
- Classes worden gebruikt om objecten te maken met dezelfde eigenschappen en methodes.

Syntax:

```
class ClassName {  
    constructor() { ... }  
    method_1() { ... }  
    method_2() { ... }  
    method_3() { ... }  
}
```

Constructor

De constructor wordt gebruikt om objecten te maken met dezelfde eigenschappen en methodes.

Syntax:

```
class ClassName {  
    constructor() { ... }  
}
```

Javascript Inheritance

Inheritance is een manier om classes te maken die van andere classes afstammen. Classes die van andere classes afstammen, kunnen de eigenschappen en methodes van de andere classes gebruiken.

Classes die van andere classes afstammen, kunnen de eigenschappen en methodes van de andere classes gebruiken.

Syntax:

```
class ChildClass extends ParentClass {  
    constructor() {  
        super();  
    }  
}
```

Javascript Static Methods

Static methods worden gebruikt om methodes te maken die niet van objecten worden aangeroepen.

Static methods worden gebruikt om methodes te maken die niet van objecten worden aangeroepen.

Syntax:

```
class className {  
    static method_1() { ... }  
    static method_2() { ... }  
    static method_3() { ... }  
}
```

Voorbeeld:

```
class Animal {  
    constructor(name) {  
        this.name = name;  
    }  
    speak() {  
        console.log(this.name + ' makes a noise.');//  
    }  
    static eat() {  
        console.log('Eating...');//  
    }  
}  
  
Animal.eat();
```

Asynchrone JavaScript

Asynchrone JavaScript is een manier om JavaScript code uit te voeren zonder dat de rest van de code wacht op de uitvoering van de asynchrone code.

Syntax: `setTimeout(function, milliseconds);`

Voorbeeld: `setTimeout(function () {
 console.log("Hello World");
}, 3000);`

Syntax: `setInterval(function, milliseconds);`

Voorbeeld: `setInterval(function () {
 console.log("Hello World");
}, 3000);`

Asynchrone JavaScript

Syntax:

```
clearInterval(interval);
```



Voorbeeld:

```
let interval = setInterval(function () {  
    console.log("Hello World");  
}, 3000);
```



```
clearInterval(interval);
```

Javascript DOM (Document Object Model)

- Met de HTML DOM kan JavaScript alle elementen van een HTML-document openen en wijzigen.
- De HTML DOM is een standaard objectmodel voor HTML.
- Het HTML DOM model wordt opgebouwd als een boomstructuur van objecten.
- Met het HTML DOM kunnen JavaScript-programmeurs HTML-elementen en -attributen toevoegen, verwijderen en wijzigen.

Wat je gaat leren?

- De inhoud van html manipuleren.
- CSS wijzigen
- DOM events
- HTML elementen toe te voegen en te verwijderen

HTML Elements vinden

JavaScript biedt verschillende methoden om DOM-elementen te selecteren en te manipuleren.

Er zijn verschillende manieren:

- HTML Elementen vinden op `id` = **getElementById()**: Selects an element by its ID.
- HTML Elementen vinden op `naam` = **getElementsByName()**: Selects elements by their name attribute.
- HTML Elementen vinden op `tag naam` = **getElementsByTagName()**: Selects elements by their tag name.
- HTML Elementen vinden op `class naam` = **getElementsByClassName()**: Selects elements by their class name.
- HTML Elementen vinden op `css selectors` = **querySelector()**: Selects the first element that matches a CSS selector.

Bijvoorbeeld: HTML Element vinden op id: we gebruiken : **getElementsById()**

html

```
<p id="myElement">Oude tekst</p>
```

js

```
const element = document.getElementById("myElement");
```

html

```
<p class="intro">A</p>
```

js

```
const list = document.getElementsByClassName("intro");
```

Werken met attributen en stijlen (DOM)

In HTML hebben elementen attributen en stijlen. Met JavaScript kan je die lezen, aanpassen, of verwijderen.

- `setAttribute()`: Sets the value of an attribute.
- `getAttribute()`: Gets the value of an attribute.
- `removeAttribute()`: Removes an attribute.
- `style`: Gets or sets the inline styles of an element.

js:

```
// Set an attribute
element.setAttribute('data-custom', 'value');

// Get an attribute
var attrValue = element.getAttribute('data-custom');

// Remove an attribute
element.removeAttribute('data-custom');

// Set the style of an element
element.style.backgroundColor = 'blue';
```

JavaScript Event

Inleiding

In deze deel gaan we kijken naar JavaScript evenementen. We gaan kijken naar de verschillende soorten evenementen en hoe we deze kunnen gebruiken in onze applicaties.

Event(interactie)

Een evenement is een actie die plaatsvindt in een applicatie. Denk bijvoorbeeld aan het klikken op een knop, het verplaatsen van een muis of het indrukken van een toets op het toetsenbord. Een evenement kan ook een actie zijn die plaatsvindt buiten de applicatie, zoals het laden van een webpagina.

In JavaScript kunnen we een functie aan een evenement koppelen. Deze functie wordt dan uitgevoerd wanneer het evenement plaatsvindt. We kunnen bijvoorbeeld een functie koppelen aan het klikken op een knop. Wanneer de gebruiker op de knop klikt, wordt de functie uitgevoerd.

Eventnaam – Wat zijn de acties?

- `click` Gebruiker klikt op een element (één klik).
- `dblclick` Gebruiker dubbelklikt op een element (twee snelle klikken).
- `keydown` Een toets wordt ingedrukt (op het moment dat je drukt).
- `input` De waarde van een inputveld verandert terwijl de gebruiker typt/plakt/wist (direct, live).
- `submit` Een formulier wordt verstuurd (via submit-knop of Enter).

Event-methoden in JS:

Als je met events werkt, gebruik je vooral deze kern-methoden:

Op elementen / document / window

- `addEventListener()` → event koppelen
- `removeEventListener()` → event loskoppelen
- `dispatchEvent()` → zelf een event “afvuren”

Op het event object (in je handler)

- `preventDefault()` → default actie stoppen
- `stopPropagation()` → bubbling/capturing stoppen
- `stopImmediatePropagation()` → ook andere handlers op hetzelfde element stoppen

Klikken op een knop

html

```
<button id="btn">Klik op mij</button>
```

javascript

```
const btn = document.getElementById("btn");
```

```
btn.addEventListener("click", function (event) {
```

```
    btn.textContent = "Geklikt!";
```

Klik op mij



JavaScript evenementen

Event listeners

Een event listener is een stukje JavaScript dat je aan een HTML-element koppelt. Het zorgt ervoor dat JavaScript blijft “luisteren” of “wachten” tot een bepaalde gebeurtenis(Event) plaatsvindt op dat element. Zodra die gebeurtenis gebeurt, wordt automatisch een functie uitgevoerd. Die functie heet de event handler en bevat de code die bepaalt wat er precies moet gebeuren.

Event listener: addEventListener(...) (koppelt)
Event handler = de functie die uitgevoerd wordt

html

```
<button id="btn">Klik op mij</button>
```

javascript

```
const btn = document.getElementById("btn");
// 1) Event handler: deze functie wordt uitgevoerd bij het event
function handleClick() {
  btn.textContent = "Je hebt geklikt!";
}
// 2) Event listener: koppelt het click-event aan de event handler
btn.addEventListener("click", handleClick);
```

Wat gebeurt er?

1. De browser “luistert” naar het click-event op de knop (event listener).
2. Als je klikt, voert JavaScript de event handler handleClick() uit.
3. De tekst van de knop verandert.

JavaScript evenementen

Event handlers

Een event handler is een functie die bepaalt wat er moet gebeuren wanneer een bepaalde gebeurtenis (event) plaatsvindt. Het is dus de code die wordt uitgevoerd als reactie op een actie, zoals een klik op een knop, het typen in een invoerveld of het verzenden van een formulier.

Je koppelt een event handler aan een HTML-element via een event listener. Zodra de gebruiker de actie uitvoert op dat element, wordt de event handler automatisch uitgevoerd. Met andere woorden: een event handler is de “reactie-functie” die jouw gewenste gedrag uitvoert wanneer het event gebeurt.

Actie: gebruiker klik op de knop

Event handler: de functie die wordt uitgevoerd

html

```
<button id="btn">Klik op mij</button>
```

javascript

```
const btn = document.getElementById("btn");

// Event handler (functie)
function handleClick() {
  btn.textContent = "Geklikt!";
}

// Event listener koppelt het event aan de handler
btn.addEventListener("click", handleClick);
```

JavaScript evenementen

Event object

Een event object is een object dat informatie bevat over het evenement. Een event object wordt doorgegeven aan de event handler. De event handler kan de informatie uit het event object gebruiken om te bepalen wat er moet gebeuren.

Actie: gebruiker typt zijn naam in een formulier (input)

- Event (actie): input (elke keer dat de gebruiker typt)
- Event handler: functie die de naam toont
- Event object: is een object dat informatie bevat over het evenement >> we gebruiken e.target.value(de tekst die de gebruiker typt)

html

<label>

Naam:

<input id="naam" type="text" placeholder="Typ je naam..." />

</label>

<p id="resultaat">Naam: ...</p>

javascript

```
const naamInput = document.getElementById("naam");
const resultaat = document.getElementById("resultaat");
```

// Actie/event: gebruiker typt in het input veld

```
naamInput.addEventListener("input", function (e) {  
  
    // e is het event object  
    // e.target is het input element  
    // e.target.value is de tekst die de gebruiker typt  
    resultaat.textContent = "Naam: " + e.target.value;  
});
```

Websites om JavaScript events te leren

- <https://developer.mozilla.org>
- <https://javascript.info>
- <https://www.w3schools.com>

- **DOM Manipulatie**
- **createElement** De methode createElement maakt een nieuw element aan. Het element wordt niet automatisch aan de DOM toegevoegd. Je moet het element eerst aan de DOM toevoegen voordat het zichtbaar is op de pagina.
- **createTextNode** De methode createTextNode maakt een nieuw tekstknooppunt aan. Het tekstknooppunt wordt niet automatisch aan de DOM toegevoegd. Je moet het tekstknooppunt eerst aan de DOM toevoegen voordat het zichtbaar is op de pagina.
- **appendChild** De methode appendChild voegt een kindknooppunt toe aan een elementknooppunt. Het kindknooppunt wordt toegevoegd aan het einde van de lijst met kindknooppunten.
- **insertBefore** De methode insertBefore voegt een kindknooppunt toe aan een elementknooppunt. Het kindknooppunt wordt toegevoegd voor een bestaand kindknooppunt.
- **removeChild** De methode removeChild verwijdert een kindknooppunt van een elementknooppunt. Het verwijdert het kindknooppunt dat is opgegeven als argument.
- **replaceChild** De methode replaceChild vervangt een kindknooppunt van een elementknooppunt. Het vervangt het kindknooppunt dat is opgegeven als eerste argument door het kindknooppunt dat is opgegeven als tweede argument.
- **cloneNode** De methode cloneNode maakt een kopie van een elementknooppunt. Het maakt een kopie van het elementknooppunt dat is opgegeven als argument. Het maakt een kopie van het elementknooppunt en alle kindknooppunten van het elementknooppunt. Het maakt geen kopie van de eventlisteners die zijn toegevoegd aan het elementknooppunt.
- **hasChildNodes** De methode hasChildNodes controleert of een elementknooppunt kindknooppunten heeft. Het retourneert true als het elementknooppunt kindknooppunten heeft, anders retourneert het false.
- **firstChild** De property firstChild retourneert het eerste kindknooppunt van een elementknooppunt. Het retourneert null als het elementknooppunt geen kindknooppunten heeft.
- **lastChild** De property lastChild retourneert het laatste kindknooppunt van een elementknooppunt. Het retourneert null als het elementknooppunt geen kindknooppunten heeft.

Voorbeeld:

```
<script>
    var createNewElement = function() {
        let newElement = document.createElement('p');
        let newTextNode = document.createTextNode('Dit is een nieuwe paragraaf');
        newElement.appendChild(newTextNode);
        let articleElem = document.querySelector('article');
        articleElem.appendChild(newElement);
    }

    window.onload = function () {
        let articleElem = document.querySelector('article');
        articleElem.addEventListener('click', createNewElement, false);
    }
</script>

<article>
    <p>Dit is een paragraaf</p>
</article>
```

DOM Traversing

- **parentNode** De property parentNode retourneert het elementknooppunt van het ouderknooppunt van een elementknooppunt.
- **childNodes** De property childNodes retourneert een HTMLCollection van alle kindknooppunten van een elementknooppunt.
- **firstChild** De property firstChild retourneert het eerste kindknooppunt van een elementknooppunt. Het retourneert null als het elementknooppunt geen kindknooppunten heeft.
- **lastChild** De property lastChild retourneert het laatste kindknooppunt van een elementknooppunt. Het retourneert null als het elementknooppunt geen kindknooppunten heeft.
- **nextSibling** De property nextSibling retourneert het volgende broerknooppunt van een elementknooppunt. Het retourneert null als het elementknooppunt geen volgende broerknooppunt heeft.
- **previousSibling** De property previousSibling retourneert het vorige broerknooppunt van een elementknooppunt. Het retourneert null als het elementknooppunt geen vorige broerknooppunt heeft.

Voorbeeld:

```
<script>
    var traverseDOM = function() {
        let articleElem = document.querySelector('article');
        let childNodes = articleElem.childNodes;
        for (let i = 0; i < childNodes.length; i++) {
            console.log(childNodes[i]);
        }
    }

    window.onload = function () {
        let articleElem = document.querySelector('article');
        articleElem.addEventListener('click', traverseDOM, false);
    }
}

</script>

<article>
    <p>Dit is een paragraaf</p>
    <p>Dit is een paragraaf</p>
    <p>Dit is een paragraaf</p>
</article>
```

DOM CSS

- **style** De property style retourneert een object met alle CSS eigenschappen van een elementknooppunt.
- **className** De property className retourneert een string met alle CSS klassen van een elementknooppunt.
- **classList** De property classList retourneert een DOMTokenList met alle CSS klassen van een elementknooppunt.
- **id** De property id retourneert een string met de id van een elementknooppunt.
- **getAttribute** De methode getAttribute retourneert de waarde van een attribuut van een elementknooppunt.
- **setAttribute** De methode setAttribute stelt de waarde van een attribuut van een elementknooppunt in.
- **removeAttribute** De methode removeAttribute verwijdert een attribuut van een elementknooppunt.
- **hasAttribute** De methode hasAttribute controleert of een attribuut van een elementknooppunt bestaat. Het retourneert true als het attribuut bestaat, anders retourneert het false.

Voorbeeld:

```
<script>
    var changeCSS = function() {
        let articleElem = document.querySelector('article');
        articleElem.style.backgroundColor = 'red';
        articleElem.style.color = 'white';
        articleElem.style.padding = '20px';
        articleElem.style.border = '1px solid black';
    }

    window.onload = function () {
        let articleElem = document.querySelector('article');
        articleElem.addEventListener('click', changeCSS, false);
    }
</script>

<article>
    <p>Dit is een paragraaf</p>
    <p>Dit is een paragraaf</p>
    <p>Dit is een paragraaf</p>
</article>
```

Voorbeeld getAttribute en setAttribute:

```
<script>
    var changeCSS = function() {
        let articleElem = document.querySelector('article');
        articleElem.style.backgroundColor = 'red';
        articleElem.style.color = 'white';
        articleElem.style.padding = '20px';
        articleElem.style.border = '1px solid black';
        articleElem.setAttribute('data-color', 'red');
        console.log(articleElem.getAttribute('data-color'));
    }

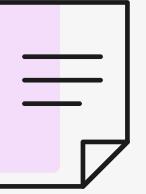
    window.onload = function () {
        let articleElem = document.querySelector('article');
        articleElem.addEventListener('click', changeCSS, false);
    }

</script>

<article>
    <p>Dit is een paragraaf</p>
    <p>Dit is een paragraaf</p>
    <p>Dit is een paragraaf</p>

</article>
```

Referenties



- [MDN Web Docs](#)
- [W3Schools DOM](#)
- [W3Schools document](#)
- [W3Schools elements](#)
- [W3Schools HTML](#)
- [W3Schools CSS](#)
- [W3Schools nodes](#)
- [W3Schools traversing](#)