Objecten

Objecten in JavaScript, net zoals in andere programmeertalen, kan vergeleken worden met objecten in de echte wereld. Vergelijk het met bijvoorbeeld een fiets. Een fiets is een object en bevat de eigenschappen: merk, kleur, gewicht, aantal versnellingen,

Een JavaScript-object bevat **eigenschappen** ***(Eng. properties).*** Een eigenschap van een object kan uitgelegd worden als een variabele dat verbonden is aan het object. Objecteigenschappen definieren de tekenistieken van het object. Toegang tot deze eigenschappen van een object gebeurt via de dot-notatie (`.`):

```js
objectName.propertyName
```

Laat ons als voorbeeld een object aanmaken via het instantiëren van de `Object()` klasse. In dit object beschrijven we een persoon met een voornaam, familienaam en leeftijd:

```js
let person = new Object();
person.firstName = 'Philippe';
person.lastName = 'De Pauw - Waterschoot';
person.age = 32;
```

Het aangemaakt object bevat nu een reeks eigenschappen. Om bijvoorbeeld de leeftijd van deze persoon op te vragen schrijven we de volgende code:

```js
const age = person.age;
console.log(age); // Output: 32
```

Spreken we een eigenschap aan die niet bestaat, dan resulteert dit in de waarde `undefined` :

```js
let hairColor = person.hairColor;
console.log(hairColor); // Output: undefined
person.hairColor = 'brown';
hairColor = person.hairColor;
console.log(hairColor); // Output: brown
```

Een nieuwe eigenschap kan dynamisch aan een object toegekend worden. In dit voorbeeld kennen van een nieuwe eigenschap `hairColor` toe aan een `person` met de waarde `'brown'` .

Eigenschappen van JavaScripts objecten kunnen ook benaderd worden via de **bracket notatie**. Objecten worden hierdoor soms **associatieve arrays** genoemd, omdat elke eigenschap geassocieerd wordt met een string waarde waarmee we toegang hebben tot de corresponderende waarde. Toegang tot een eigenschap kan ook via een variabele die een waarde bevat.

```js
person['height'] = 1.72;
let height = person['height'];
console.log(height); // Output: 1.72
height = person.height;
console.log(height); // Output: 1.72
```

In dit voorbeeld voegen we een nieuwe eigenschap `height` toe aan het `person` object met de waarde `1.72` . De lengte van een persoon kan dan opgevraagd worden via de **dot (`.`) notatie** of via de **bracket (`[…]`) notatie**. Eigenschappen die:

- starten met een cijfer, bijv. `3tier`
- een spatie bevatten, bijv. `date created`
- een **koppelteken** ***(Eng. hyphen)*** bevatten, bijv. `cool-shit`

kunnen enkel aangemaakt en opgevraagd worden via de bracket notatie.

```js
person['3tier'] = 'Presentation, Logic and Data tier';
person['date created'] = Date.now();
person['cool-shit'] = 'Associate Degree in Programming makes Cool Shit!';
const quote = person['cool-shit'];
console.log(quote); // Output 'Associate Degree in Programming makes Cool Shit!'
```

Plaatsen we een object in de bracket notatie, dan zal de naam van de eigenschap gegenereerd worden op basis van de `obj.toString()` methode. Deze methode geeft de **string representatie** weer van het object.

Met de `for...in` -lus kunnen we alle eigenschappen van een object doorlopen:

```js
let msg = 'Properties of the object person\n------------------------------';
for(let i in person) {
    msg += `\n${i} = ${person[i]}`;
}
console.log(msg);
```

Dit resulteert in de volgende output:

```
Properties of the object person
-------------------------------
firstName = Philippe
lastName = De Pauw - Waterschoot
age = 32
hairColor = brown
height = 1.72
3tier = Presentation, Logic and Data tier
date created = 1569311008294
cool-shit = Associate Degree in Programming makes Cool Shit!
```

Voorgedefinieerde objecten

JavaScript bevat een hele resem **voorgedefinieerde** *(Eng. predefined)* objecten, zoals: `Boolean`, `Number`, `String`, `Math`, `Date`, `Array` … Deze objecten bevatten eigenschappen en methoden (functies) die we kunnen aanspreken. Eigenschappen kunnen gemanipuleerd worden via toewijzing of via de ingebouwd functies uit deze objecten.

`Boolean`

Het **ingebouwde** *(Eng. built-in)* `Boolean` object is een wrapper voor een boolean waarde.

> ⚠ **Opgelet**
>
> Gebruik het `Boolean` object niet in de plaats van het primitief datatype boolean.

Het bevat een reeks van eigenschappen (`length`, `prototype` …) en methoden (`toString()`, `valueOf()` …) die meestal ook aangesproken kunnen worden via het primitief datatype boolean.

Een nieuwe instantie kan aangemaakt worden via de `new` operator:

```
const b1 = new Boolean(false);
```

De waarde die de variabele `b1` bevat is `true`, raar maar waar. Dit geldt ook voor alle andere waarden `0`, `null`, `undefined`, `''` ...

Het `Boolean` object wordt voornamelijk gebruikt bij datatype conversie (converteren van een datatype naar een andere datatype) en dit via de `Boolean()` functie:

```js
const b1 = Boolean('true'); // true
const b2 = Boolean(0); // false
const b3 = Boolean(undefined); //false
const b4 = Boolean(NaN); // false
const b5 = Boolean(1); // true
const b6 = Boolean(null); // false
```

`Number`

Een `Number` object is een wrapper rond een numerieke waarde met **dubbele precisie (64-bit)**. Instanties kunnen aangemaakt worden via de `new Number()` constructor. Een primitieve numerieke waarde kan aangemaakt worden via de `Number()` functie.

Het bevat een reeks van eigenschappen (`MIN_VALUE`, `POSITIVE_INFINITY` ...) en methoden (`parseInt()`, `parseFloat()`, `isNaN()`, `toFixed()` ...) die meestal ook aangesproken kunnen worden via het primitief datatype number.

De eigenschappen vermeld in capitalen zijn eigenlijk constanten die we kunnen aanroepen zonder dat we een instantie moeten maken van het `Number` object:

```js
const minValue = Number.MIN_VALUE; // 5e-324
const maxValue = Number.MAX_VALUE; // 1.7976931348623157e+308
const posInfinity = Number.POSITIVE_INFINITY; // Infinity
```

De `Number` function wordt gebruikt bij datatype conversies:

```js
const n1 = Number('123') // 123
const n2 = Number('12.3') // 12.3
const n3 = Number('12.00') // 12
const n4 = Number('123e-1') // 12.3
const n5 = Number('') // 0
const n6 = Number(null) // 0
const n7 = Number('0x11') // 17
```

```
const n8 = Number('foo') // NaN
const n9 = Number('100a') // NaN
const n10 = Number('-Infinity') //-Infinity
const n11 = Number(true); // 1
const n12 = Number(Date.now()); // 1569314197567
```

Een handige functie is `toFixed()` waarmee we het aantal cijfers na de komma (decimaal punt) kunnen specificeren (geeft een string waarde terug):

```js
const v = 12.678904;
const vNew1 = v.toFixed(2); // "12.68"
const vNew2 = Number(v).toFixed(2); // "12.68"
const vNew3 = new Number(v).toFixed(2); // "12.68"
```

`String`

Het `String` object is een wrapper voor het primitieve datatype string. Instanties kunnen aangemaakt worden via de `new String()` constructor. Een primitieve string waarde kan aangemaakt worden via de `String()` functie.

Het bevat een reeks van eigenschappen (`length` , `POSITIVE_INFINITY` ...) en methoden (`fromCharCode()` , `charAt()` , `concat()` , `indexOf()` , `endsWith()` , `match()` , `replace()` , `split()` , `substring()` , `trim()` ...) die meestal ook aangesproken kunnen worden via het primitief datatype string.

Een `String` object kan geconverteerd worden naar een primitieve waarde via de `String()` functie of via de `valueOf()` functie uit het `String` object:

```js
const sPrim = 'john';
const sObj = new String(sPrim);

console.log(typeof sPrim); // "string"
console.log(typeof sObj);  // "object"

const sObjToPrim1 = String(sObj); // typeof sObjToPrim1 => "string"
const sObjToPrim2 = sObj.valueOf(); // typeof sObjToPrim2 => "string"
```

Voorbeeld "verwijderen van witruimte" vooraan en achteraan de string:

```js
const greeting = '      like graphics love code make cool shit          ';
console.log(greeting);
console.log(greeting.trim());
```

Dit resulteert in de volgende output:

```
"      like graphics love code make cool shit          "
"like graphics love code make cool shit"
```

Voorbeeld “bevat een string een bepaald woord”:

```js
const paragraph = 'The quick brown fox jumps over the lazy dog. If the dog barked, was

const searchTerm = 'Fox'.toLowerCase();
const position = paragraph.toLowerCase().indexOf(searchTerm);
if(position !== -1) {
    console.log(`We have a first occurence for the search term ${searchTerm} at positio
} else {
    console.log(`No results for search term ${searchTerm}`);
}
```

Dit resulteert in de volgende output:

```
"We have a first occurence for the search term fox at position 16"
```

Voorbeeld “splijten van woorden op basis van characters”:

In dit voorbeeld splijten we een string op basis van een spatie (`' '`). Elke woord vormt een elment op een specifieke index in de resulterende array.

```js
const quote = 'Like Graphics Love Code Make Cool Shit';
const quoteArray = quote.split(' ');
for(let i = 0; i < quoteArray.length; i++) {
    console.log(quoteArray[i]);
}
```

Dit resulteert in de volgende output:

```js
"Like"
"Graphics"
"Love"
"Code"
"Make"
"Cool"
"Shit"
```

Math

Met het `Math` object kunnen we mathematische berekeningen uitvoeren. Het bevat contanten (eigenschappen: `PI` , `E` , `SQRT2` ...) en methoden (functies: `abs` , `sin()` , `atan()` , `floor()` , `round()` , `ceil()` , `log()` , `pow()` , `random()` ...).

De trigoniometrische functies, zoals `sin()` , `cos()` e.d., gebruiken een argument uitgedrukt in radialen. Om de sinus van een hoek, uitgedrukt in graden, implementeren we de volgende code:

```js
const angleDegrees = 45;
const angleRadians = Math.PI/180*angleDegrees;
const vSin = Math.sin(angleRadians);
console.log(`The sinus of ${angleDegrees} degrees is ${vSin}.`); // Output "The sinus o
```

Voorbeeld "random integer waarde tussen twee getallen:

```js
function getRandomInteger(min, max) {
    const minV = Math.ceil(min);
    const maxV = Math.floor(max);
    return Math.floor(Math.random() * (max - min + 1)) + min;
}

const v = getRandomInteger(10, 80);
console.log(`The generated random value is ${v}.`); // Example output: "The generated r
```

De `Math.random()` genereert een decimaal getal tussen `0` en `1` , waarbij de waarde `1` niet wordt meegerekend, maar de waarde `0` wel. We berkenen eerst de plafondwaarde van het agrument `min` (bijv.: `10.6` wordt `11`), vervolgens de vloerwaarde van het argument `max` (bijv.: `90.4` wordt `90`) en tenslotte geven we een positief geheel random getal terug aan de aanvrager.

`Date`

Met het `Date` object kunnen we een instantie maken (lees een object) voor een specifieke datum (incl. tijd) en dit platformonafhankelijk. Het Date object bevat een getal uitgedrukt in milliseconden sinds **1 January 1970 UTC** (epoch time).

Een **date object** kan aangemaakt worden via:

- een instatie van het `Date` object te maken via de constructur en `new` keyword
- statische methode (geen instantie van een object nodig) `parse()` bijv.: `const startOfJS = Date.now('04 Dec 1995 00:12:00 GMT');`

Het bevat een reeks van eigenschappen (`length` …) en methoden (`getDate()` , `getSeconds()` , `getTime()` , `setYear()` , `setHours()` , `toISOString()` , `toLocalString()` , `toTimeString()` …).

Custom Object

Een **op maat gemaakt object** ***(Eng. custom object)*** is een object dat niet behoort tot de voorgedefinieerde objecten. In een custom object beschrijven we de eigenschappen en methoden die inherent zijn aan dit nieuw object.

Beschrijven we een persoon als object in een programmeertaal, dan bevat deze:

- eigenschappen
  - haarkleur
  - lengte
  - gewicht
  - voornaam
  - …
- functies (acties)
  - haarkleur veranderen
  - sporten
  - studeren
  - …

Methoden zijn acties die een persoon in kwestie kan ondernemen of die door een andere partij kan ondernomen worden. Door een actie kan één of meerdere eigenschappen gemanipuleerd worden.

Voorheen definieerde we alle eigenschappen apart die allen van toepassing waren op eenzelfde persoon. Het is stukken beter om al deze eigenschappen te omsluiten door een custom object. Het custom object is toegankelijk door één aanroep, waardoor tevens alle onderliggende eigenschappen en methoden aanspreekbaar zijn.

Aanmaak nieuwe objecten

Object constructur

Custom objecten kunnen in JavaScript aangemaakt worden met een **Object constructor** in combinatie met het `new` keyword. Er wordt als het waren een instatie gemaakt van het `Object()` object. De instantie wordt opgeslagen in een variabele. Eigenschappen kunnen hier dynamisch aan toegekend worden via de dot (`.`) notatie.

```js
let person = new Object();
person.firstName = 'Philippe';
person.surName = 'De Pauw - Waterschoot';
person['666'] = 'The Devil';
person.quote = 'Learning by doing';
person.toString = function() {
  return `${ this.firstName } ${ this.surName }`;
}
console.log(person.toString());
```

Voegen we de onderstaande code toe aan het voorbeeld:

```js
let person1 = person;
person1.firstName = 'Evelien';
console.log(person.toString());
```

De regel `let person1 = person;` kopieert alle inhoud naar een nieuwe variabele `person1`. De variabelen `person` en `person1` verwijzen naar dezelfde geheugenruimte. Dit betekent dat wanneer we bijv. via de variabele `person1` de eigenschap `firstName` een andere waarde geven, deze nieuwe waarde ook toegekend is aan de corresponderende eigenschap in de `person` variabele.

Object initalizers

Objecten kunnen ook aangemaakt worden via **object initialisers** of **object literals** (gelijkaardig met een JSON object). . Een object initaliser is een expressie dat de initalisatie van een object beschrijft. Binnen de curly brackets (`{ ... }`) worden de eigenschappen en functies gedefinieerd. De eigenschappen en functies kunnen beschreven worden via een identifier, een nummer, bracket (`[...]`) notatie of een string en worden gescheiden door een komma (`,`).

```js
const person = {
    firstName: 'Philippe',
    surName: 'De Pauw - Waterschoot',
    666: 'The Devil',
    "quote": 'Learning by doing',
    toString: function() {
        return `${ this.firstName } ${ this.surName }`;
    },
    completeTask(task) {
        return `Task "${task}" completed!`;
    }
};
console.log(person.toString()); // Output: Philippe De Pauw - Waterschoot
console.log(person.completeTask('Buy some food')); // Output: Task "By some food" comple
```

In het voorbeeld definiëren we een persoon met de variabele `person` . Binnen de curly brackets (`{...}`) omschrijven we de persoon met specifieke eigenschappen en acties (die we kunnen uitvoeren).

Het `this` keyword refereert naar het huidige object waarin deze code wordt geschreven. Dit betekent dat we met `this` de persoon kunnen aanspreken. Hierdoor kunnen we bijv. in de functie `toString()` eigenschappen aanspreken binnen het object via `this.firstName` …

```js
const car = {
    color: 'red',
    wheels: 4,
    engine: {
      cylinders: 6,
      size: 3.6,
            toString: function() {
                return this.color;
            }
    },
```

```
};
console.log(car.engine.toString()); // Output: undefined
```

In het voorbeeld met de `car` variabele willen we in de `engine` eigenschap (zelf een object) een functie `toString()` specifiëren die tracht een eigenschap `color` uit het bovenliggend niveau aan te sprekene via het `this` keyword. Dit is niet mogelijk omdat `this` betrekking heeft op de `engine` en niet op de `car`.

Constructor function

Als alternatief kunt u een object maken met deze twee stappen:

Definiëren object via de **constructor function** dan gebruiken we als eerste letter een hoofdletter bijv. `function Post() { ... }`. Bevat de naam van deze functie meerdere woorden, dan begint elk woord met een hoofdletter bijv. `function InstructionGuide() { ... }`. Een instatie van een constructor function gebeurt via het `new` keyword.

Wensen we een object type te creëren voor posts (bevat een titel, synopsis en het volledige verhaal), dan schrijven we de volgende functie:

```js
function Post(title, synopsis, body) {
  this.title = title;
  this.synopsis = synopsis;
  this.body = body;
}
```

Het `this` keyword heeft betrekken tot het object in dit geval op het object type `Post`. Eigenschappen kunnen aan dit object toegekend worden via **this.propertyName** bijv. `this.firstName`. De waarden aan deze eigenschappen worden toegekend gebasseerd op de waarden uit de argumenten.

We kunnen nu een instantie maken van het object type `Post` als volgt:

```js
const post1 = new Post('SpaceX assembleert en toont eerste Starship prototype', 'SpaceX
```

De titel van `post1` kan opgevraagd worden via `post1.title` of `post1['title']`. Nieuwe eigenschappen kunnen dynamisch aan de variabele `post1` toegekend worden. Bijvoorbeeld:

```js
post1.createdAt = Date.now();
post1.likes = 1256;
```

Veronderstellen we dat een post een categorie bevat, dan kunnen we deze categorie toekennen via een corresponderend argument:

```js
function Category(name, description) {
  this.name = name;
  this.description = description;
}
const c1 = new Category('Nieuws', 'Nieuws uit de ruimte!');

function Post(title, synopsis, body, category) {
  ...
  this.category = category;
}
const p1 = new Post(..., c1);
```

De naam van de categorie die gekoppeld is aan een post `p1` kunnen bevragen via `p1.category.name`.

In het volgende voorbeeld definiëren we een persoon via de `Person()` object constructor die vier argumenten bevat. De waarden van de argumenten kennen we toe aan de corresponderende eigenschappen. We voegen aan dit object ook een methode `toString()` toe waarmee we een string representatie kunnen teruggeven van dit object.

```js
function Person(firstName, surName, _666, quote) {
  this.firstName = firstName;
  this.surName = surName;
  this['666'] = _666;
  this.quote = quote;
  this.toString = function() {
    return `${ this.firstName } ${ this.surName }`;
  }
}
const person1 = new Person('Philippe', 'De Pauw - Waterschoot', 'The Devil', 'Learning
console.log(person1.toString());
```

We maken twee instantie van het object type `Person()` als volgt:

```js
const person1 = new Person('Philippe', 'De Pauw - Waterschoot', 'Learning by doing');
const person2 = new Person('Evelien', 'Rutsaert', 'Together we are strong');
```

`Object.create` methode

Objecten kunnen ook aangemaakt worden d.m.v. de `Object.create()` methode en dit op basis van een bestaand object literal (zonder dat een object constructor nodig is).

```js
const Person = {
  type: 'person',
  displayType: function() {
    return this.type;
  }
}

const p1 = Object.create(Person);
p1.type = 'lecturer';
console.log(p1.displayType()); // Output: lecturer

const p2 = Object.create(Person);
p2.type = 'student';
console.log(p2.displayType()); // Output: student
```

Getters en setters

Een **getter** is een methode dat de waarde van een specifieke eigenschap ophaalt. Een **setter** is een methode dat een waarde toekent aan een specifieke eigenschap. De getter en setter methoden kunnen toegepast op alle voorgedefinieerde objecten en custom objecten.

```js
var person = {
  firstName: 'Philippe,
  earn: 0,
  get earnings() {
    return this.earn;
  },
  set earnings(x) {
    this.earn = x;
  }
};

console.log(person.earn); // Output: 0
```

```
person.earnings = 2000;
console.log(person.earnings); // Output: 2000
console.log(person.earn); // Output: 2000
```

De volgende code illustreert hoe getters en setters de functionaliteit van een Date object kan uitgebreid worden met een `year` eigenschap voor alle instanties. De volgende statements definiëren een getter en setter voor de `year` eigenschap:

```js
let d = Date.prototype;
Object.defineProperty(d, 'year', {
  get: function() { return this.getFullYear(); },
  set: function(y) { this.setFullYear(y); }
});
```

De vogende statements gebruiken de getter en setter in een Date object:

```js
let now = new Date();
console.log(now.year); // Output: 2022
now.year = 2023;
console.log(now.getFullYear()); // Output: 2023
```

Om meerdere getters en setters te definiëren kunnen de de `defineProperties` methode aanspreken uit het `Object` object:

```js
let obj = { a: 0 };

Object.defineProperties(obj, {
    'b': { get: function() { return this.a + 1; } },
    'c': { set: function(x) { this.a = x / 2; } }
});
```

Verwijderen van eigenschappen

In JavaScript kunnen we eigenschappen (die niet overgeërfd zijn) verwijderen door gebruik te maken van de `delete` operator. De volgende code toont hoe we een eigenschap kunnen verwijderen:

```js
let person = new Object();
person.firstName = 'Philippe';
person.blah = 'bleuh';

delete person.blah;
console.log('blah' in person); // Output: false;
```

Indien het `var` keyword niet werd gebruikt bij een globale variabele, dan kunnen we deze variabele verwijderen met de `delete` operator:

```js
g = 17;
delete g;
```

Vergelijken van objecten

Objecten zijn van het **referentie** ***(Eng. reference)*** type. Twee objecten met dezelfde inhoud of structuur zijn **nooit gelijk aan elkaar**. Enkel objecten met dezelfde referentie zijn gelijk aan elkaar:

```js
const person1 = { firstName: 'Philippe' };
const person2 = { firstName: 'Philippe' };
console.log(person1 == person2); // Output: false
console.log(person1 === person2); // Output: false

const person3 = person2;
console.log(person3 == person2); // Output: true
console.log(person3 === person2); // Output: true

person2.firstName = 'Evelien';
console.log(person3.firstName); // Output: "Evelien" instead of "Philippe"
```

← Functies

Geïndexeerde collecties →

Laatst bijgewerkt: 8/9/2022 13:39:31