Web technology

Les 11: JS - Strings, getallen, datums, arrays en loops

Kristof Michiels

Strings: methods en properties

concat()

We weten ondertussen dat we strings aan elkaar kunnen plakken met de + operator. Er is evenwel nog een andere manier. We kunnen hiervoor ook de concat method gebruiken

```
let deel1 = "The Marvelous ";
let deel2 = "Mrs. Maisel";
let samen = deel1.concat(deel2);
console.log(samen); // The Marvelous Mrs. Maisel
```

length

Strings hebben verder ook een length property waarmee we kunnen te weten komen uit hoeveel karakters de string bestaat.

```
let deel1 = "Hallo ";
let deel2 = "wereld";
let samen = deel1 + deel2;
console.log(samen.length); // 12
```

Aanhalingstekens

Wanneer we dubbele aanhalingstekens gebruiken om een string aan te maken, dan kunnen we geen dubbele aanhalingstekens gebruiken binnen de string. En omgekeerd.

```
let zin1 = "Piano's wegen zwaar";
let zin2 = '"Weet dat je niet weet" - Socrates';
console.log(zin1);
console.log(zin2);
```

Escapen

Een betere manier om om te gaan met dit probleem: het escapen van dergelijke karakters. Je doet dit door een \ teken te plaatsen voor het karakter in kwestie. Met het \ teken, ook escape karakter genoemd, kunnen we enkele of dubbele aanhalingstekens gebruiken binnen een string

```
let zin1 = 'Piano\'s wegen zwaar';
let zin2 = "\"Weet dat je niet weet\" - Socrates";
console.log(quote1);
console.log(quote2);
```

slice()

Met de slice() method kunnen we een stukje uit een string knippen. De getallen die we meegeven met de slice() method zijn de start- en eindpositie van de string die we willen uitknippen. Als we negatieve waardes gebruiken, dan wordt de positie geteld vanaf het einde van de string.

```
let s = "Met de slice() method kunnen we een stukje uit een string knippen.";
console.log(s.slice(15,31)); // method kunnen we
```

substr()

De substr() method gedraagt zich zoals slice(), met het verschil dat de 2de waarde die we meegeven de lengte van de string die we willen uitknippen. Het tweede getal is hier optioneel (net zoals bij slice). Als we het niet meegeven dan wordt geknipt tot het einde van de originele string

```
let s = "De kat krabt de krollen van de trap";
console.log(s.substr(13)); // de krollen van de trap
console.log(s.substr(13, 10)); // de krollen
```

replace()

We kunnen de replace() method gebruiken om binnen een string te zoeken en te vervangen. Replace() zoekt naar de waarde die we meegeven en geeft de string terug waarin de vervanging(en) is/zijn gebeurd.

```
let zin = "Hallo Wereld!";
let nieuweZin = zin.replace("Wereld","aan jezelf");
console.log(nieuweZin); // Hallo aan jezelf!
```

toLowerCase() en toUpperCase()

We kunnen van een string een string met enkel kleine letters maken door gebruik te maken van .toLowerCase(). Er bestaat een gelijkaardige method om alle letters tot drukletters om te vormen, namelijk .toUpperCase()

```
let zin = "Hallo Wereld!";
let nieuweZin = zin.toUpperCase();
console.log(nieuweZin); // HALLO WERELD!
```

Getallen

Number

Getallen. Niet bang zijn, we gaan er geen hogere wiskunde van maken. Getallen beschikken ook over properties en methods. Er is slechts één type getal in JS: number.

```
let leeftijd = 42;
let lengte = 1.72;
console.log(typeof(leeftijd)); // number
console.log(typeof(lengte)); // number
```

Modulus: rest bij deling door

De modulus operator (%) geeft de rest bij deling door een bepaald getal terug. Handig als je de deelbaarheid van een getal door een ander getal wil bepalen. Is 9 deelbaar door 2? Neen, want 9 % 2 = 1. Is 8 deelbaar door 4? Ja, want 8 % 4 = 0.

```
let x = 5;
let y = 2;
let z = x % y;
console.log(z); // 1
console.log(6 % 2); // 0
console.log(15 % 4); // 3
```

Operaties op getallen met andere datatypes

Wanneer we een waarde willen gebruiken van een datatype dat niet gemaakt is voor sommige operaties, dan probeert JavaScript het om te zetten naar een datatype dat daar wel voor geschikt is.

```
let getal = "10" + 10;
console.log(getal); // "1010"
```

NaN

Een ander concept binnen JS: NaN of Not a Number. Vertelt ons dat een waarde geen getal is. Wanneer we een wiskundige operatie proberen met een niet-numerieke string dan is het resultaat NaN. Hoewel NaN staat voor Not a Number is het data type toch een number.

```
let getal = 20 / "Robin";
console.log(getal); // NaN
```

parseInt() en parseFloat()

We kunnen numerieke strings omzetten naar getallen. Via parseInt() voor gehele getallen. Of via parseFloat() voor reële getallen. Parsing is het proces van het analyseren en verwerken van een string. De strings mogen ook niet-numerieke karakters bevatten. Zolang ze maar starten met een getal.

```
let string = "59.99 USD";
let prijs1 = parseInt(string);
let prijs2 = parseFloat(string);
console.log(prijs1 + " " + prijs2); // 59 59.99
```

Math.round()

We kunnen de Math.round() method gebruiken om een getal af te ronden. Afronden gebeurt naar boven of naar beneden, naar het dichtsbijzijnde gehele getal. Onthou dat waardes onder de 0.5 naar beneden zullen afgerond worden. Vanaf 0.5 worden ze naar boven afgerond.

```
let pi = 3.141592653589793;
let getal = Math.round(pi);
console.log("Pi is nu " + pi); // Pi is nu 3
```

Het Math object

Math is een object die verschillende constantes bevat. Zoals Math.PI. En ook methods zoals math.random(). PI is een constante, dwz een variabele die niet gewijzigd kan worden. En Math.random() geeft je een willekeurig getal tussen 0 en 1.

```
let getal = Math.PI;
let willekeurig = Math.random();
console.log(getal); // 3.141592653589793
console.log(willekeurig); // 0.3672451738322162 (bijvoorbeeld)
```

Datums

Het Date object

Een ander object, Date, laat ons werken met datums en tijd. Het verschil tssen Date en Math is dat we het woord new gebruiken om een zogenaamde instance ervan te verkrijgen. Een datum bevat een jaar, maand, dag, uur, minuut, seconde en millisecondes.

```
let datum = new Date("1995-10-21");
console.log(datum); // Mon Oct 21 1995 01:00:00 GMT+0100
```

Het Date object

Omdat ze objecten zijn hebben datums een aantal methodes die ons helpen datums te gebruiken. Laat ons zelf een datum string bouwen

```
let datum = new Date("1995-10-21");
let jaar = datum.getFullYear(); // 1995
let maand = datum.getMonth() + 1; // 10 (start met 0)
let dag = datum.getDate(); // 21 (getDay() zou ons het dagnummer geven, start met 0)
console.log(jaar + "/" + maand + "/" + dag); // 1995/10/21
```

Het Date object

We kunnen deze datums op een aantal manieren gebruiken. We kunnen hen veranderen, vergelijken, en hen vertonen.

```
let d1 = new Date(1985,10,21);
let d2 = new Date(d1.getTime());
d2.setFullYear(2019);
if (d1 < d2) {
    console.log(d1.toDateString()) + " is \"kleiner\" dan " + d2.toDateString());
} else {
    console.log(d1.toDateString()) + " is \"groter\" dan " + d2.toDateString());
} // Thu Nov 21 1985 is "kleiner" dan Thu Nov 21 2019
```

Heb je gezien hoe we d1 en d2 aanmaakten? De getTime() method geeft de tijd terug in milliseconden die verstreken zijn sinds 1970.

Arrays

Arrays

Een array is een bijzonder type variabele, die verschillende waarden van een type of zelfs verschillende types kan bevatten. Arrays bieden een geweldige manier om gerelateerde waarden op te slaan.

```
let dingen = ["raindrops","roses","whiskers","kittens"];
console.log(dingen); // ["raindrops","roses","whiskers","kittens"]
```

Arrays

We creëren een array door waarden binnen vierkante haakjes toe te kennen aan een variabele. We kunnen zelfs lege vierkante haakjes gebruiken om zo een lege array aan te maken.

```
let dingen = [];
console.log(dingen); // []
```

Index posities

Hoe kunnen we die waarden opvragen? Arrays hebben posities of index posities. Dus, om een bepaalde waarde op te vragen gaan we het overeenkomstige indexgetal tussen vierkante haakjes meegeven. Deze index posities starten bij 0.

```
let dingen = ["raindrops","roses","whiskers","kittens"];
console.log(dingen[3]); // "kittens"
```

toString()

Arrays zijn objecten. Dit betekent dat we kunnen gebruik maken van een reeks methods. Hier: een method toString() die de array omvormt tot een komma-gescheiden string. Hetzelfde geldt voor join() als we het een string als separator meegeven.

```
let dingen = ["raindrops","roses","whiskers","kittens"];
console.log(dingen.toString()); // "raindrops,roses,whiskers,kittens"
let vruchten = ["banaan", "sinaasappel", "appel", "mango"];
let energie = vruchten.join("-");
console.log(energie); // banaan-sinaasappel-appel-mango
```

length

Zoals bij strings is er een eigenschap (of property) die ons vertelt hoeveel elementen een array telt.

```
let dingen = ["raindrops","roses","whiskers","kittens"];
console.log(dingen.length); // 4
```

Arrays: index

Niet enkel kunnen we een waarde door zijn index opvragen. We kunnen ze ook veranderen. Bovendien kunnen we ook waarden toevoegen met een index nummer. De index telt wel vanaf 0!

```
let dingen = ["raindrops","roses","whiskers","kittens"];
dingen[0] = "kettles";
console.log(dingen); // ["kettles","roses","whiskers","cats"]
dingen[4] = "tall hat";
console.log(dingen); // ["kettles","roses","whiskers","cats","tall hat"]
```

Arrays: split()

We hebben reeds strings uit arrays gecreëerd. We kunnen ook arrays maken uit een string. We kunnen de split() method met een separator gebruiken om een string op te splitsen in een array van zgn substrings.

```
let sentence = "raindrops and roses and whiskers and kittens";
let dingen = sentence.split(" and ");
console.log(dingen); // ["raindrops","roses","whiskers","kittens"]
```

Arrays: push()

We kunnen de push() method gebruiken om een nieuwe waarde aan het einde van de array toe te voegen. Heeft hetzelfde resultaat als een waarde toekennen aan de index van dingen.length

```
let dingen = ["raindrops","roses","whiskers","kittens"];
dingen.push("kettles");
console.log(dingen); // ["kettles","roses","whiskers","kittens","kettles"]
```

Arrays: pop()

Waarden verwijderen uit arrays? De pop() method verwijdert de laatste waarde uit de array. Als we deze toekennen aan een variabele, dan ontvangt deze variabele de verwijderde waarde.

```
let dingen = ["raindrops","roses","whiskers","kittens"];
let ding = dingen.pop();
console.log(dingen); // ["raindrops","roses","whiskers"];
console.log(ding); // kittens
```

Arrays: shift()

Wat als we de eerste waarde willen verwijderen? Dat doen we met de shift() method. Shift() verwijdert het eerste element uit de array en verhuist alle andere waarden naar een lagere index.

```
let dingen = ["raindrops","roses","whiskers","kittens"];
dingen.shift();
console.log(dingen[0]); // roses
```

Arrays: unshift()

De unshift() method is het tegenovergestelde van shift(). Het voegt een waarde toe aan het begin van de array. Alle andere waarden krijgen een hoger indexgetal. We kunnen unshift() ook gebruiken met meerdere waardes tegelijk.

```
let dingen = ["raindrops","roses","whiskers","kittens"];
dingen.unshift("guitars");
console.log(dingen); // ["guitars","raindrops","roses","whiskers","kittens"];
```

Arrays: splice()

De splice() method kan waarden toevoegen, vervangen en verwijderen. Splice() neemt twee getallen en daarna eventueel een variabel aantal waarden.

```
let fruit = ["banaan", "sinaasappel", "appel", "mango", "kiwi"];
fruit.splice(2, 2); // Op positie 2, verwijder 2 items
console.log(fruit); // ["banaan", "sinaasappel", "kiwi"]
fruit.splice(2, 1, "citroen", "kiwi"); // 2 items bij en 1 verwijderen op index 2
console.log(fruit); // ["banaan", "sinaasappel", "citroen", "kiwi"]
fruit.splice(2, 0, "peer", "pruim"); // 2 items bij op index 2, geen verwijderen
console.log(fruit); // ["banaan", "sinaasappel", "peer", "pruim", "citroen", "kiwi"]
```

Arrays: sort()

We kunnen de sort() method gebruiken om een array in alfabetische volgorde te sorteren. Als we de volgorde willen omkeren kunnen we reverse() gebruiken op de array.

```
let dingen = ["raindrops","roses","whiskers","kittens"];
dingen.sort();
console.log(dingen); // ["kittens","raindrops","roses","whiskers"]
dingen.sort().reverse()
console.log(dingen); // ["whiskers", "roses", "raindrops", "kittens"]
```

Loops

Loops?

Programmeurs zijn graag lui. Ze hebben gelijk :-) En ze houden hun code graag zo kort en leesbaar mogelijk. Dit doet pijn aan de ogen

```
let getallen = [3,7,14,15];
getallen[0]++;
getallen[1]++;
getallen[2]++;
getallen[3]++;
console.log(getallen); // [4,8,15,16]
```

De for-loop

We kunnen loops gebruiken om deze stukjes code zo vaak te herhalen als we nodig achten. In de haakjes na de for plaatsen we een control variabele, een voorwaarde en een manier om de voorwaarde false te maken op een bepaald punt.

```
let string = "";
for(let i = 0; i < 5;i++) {
    string += i;
}
console.log(string) // 01234
```

De for-loop

Gaan we nu terug naar ons eerste voorbeeld, dan krijgen we het volgende. De loop zal starten op 0 en zal duren tot i niet meer kleiner is dan de length van de array. Handig, want dit komt exact overeen met de index (en dus het aantal elementen) van de array.

```
let getallen = [3,7,14,15];
for(var i = 0; i < getallen.length; i++) {
getallen[i]++;
}
console.log(getallen); // [4,8,15,16]
```

Loopen met het in keyword

Er is een andere manier om met for door de array te loopen, door gebruik te maken van het in keyword.

```
let dingen = ["raindrops","roses","whiskers","kittens"];
for(let ding in dingen) {
console.log(dingen[ding]);
}
// raindrops roses whiskers kittens
```

De while-loop

De while loop, een andere soort loop , loopt door een groep statements zo lang een specifieke conditie waar is. We gebruiken het while keyword om while loops te creëren. Dit soort loops hebben geen controle variabelen, dus we moeten er dus zelf voor zorgen dat de voorwaarde vals wordt.

```
let getallen = [3,7,14,15,17,18];
let i = 0;
while (i < getallen.length) {
    getallen[i]++;
    i++;
}
console.log(getallen); // [4, 8, 15, 16, 18, 19]
```

De do while-loop

Een lichte variatie op de while loop is de do-while loop. Een do-while loop loopt minstens één keer, zelfs indien de voorwaarde vals is.

```
let getallen = [3,7,14,15,17,18];
let i = 0;
do {
    getallen[i]++;
    i++;
}
while(i < getallen.length);
```

Break

Soms willen we uit de loop starten voor deze normaliter zou afgelopen zijn. We kunnen hiervoor het break keyword gebruiken.

```
for(var i = 0; i < 5; i++) {
    if (i === 1) {
        break;
    }
    console.log("Tellen tot " + i); // tellen tot 0
}
```

Continue

Als we een specifieke iteratie willen stoppen, maar toch verder gaan met de loop, dan kunnen we gebruik maken van continue.

```
for(var i = 0; i < 5; i++) {
    if (i == 1) {
        continue;
    }
    console.log("Tellen tot " + i); // Tellen tot 0 Tellen tot 2 Tellen tot 3 Tellen tot 4
}
```

Webtech - JS - les11 - kristof.michiels01@ap.be