Inleiding JavaScript

Leereenheid 4

Inleiding JavaScript

INTRODUCTIE

JavaScript maakt het mogelijk om webpagina's dynamisch te maken, en om de gebruiker meer mogelijkheden tot interactie te bieden. Een aanzienlijk gedeelte van deze cursus heeft daarom JavaScript tot onderwerp. In deze leereenheid introduceren we JavaScript. We gebruiken deze taal aanvankelijk los van webpagina's om eerst de elementen van de taal te leren. Vanaf de leereenheid over jQuery en de DOM zullen we zien hoe we JavaScript kunnen gebruiken in combinatie met webpagina's.

We gaan er bij de behandeling van uit dat u voorkennis van en ervaring met een objectgeoriënteerde taal hebt op minimaal het niveau van *Objectgeoriënteerd programmeren in Java 1.* De behandeling van de taal zal daarom kort zijn en gebeurt aan de hand van het tekstboek Eloquent JavaScript van Marijn Haverbeke.

JavaScript is een taal met veel mogelijkheden: mogelijkheden om geavanceerde dynamiek aan webpagina's te geven, maar ook mogelijkheden om slecht leesbare en slecht onderhoudbare programma's te schrijven en om fouten te maken.

We besteden daarom in dit werkboek extra aandacht aan richtlijnen en gewoonten om goed leesbare en onderhoudbare code te schrijven. U ziet daarom in de tekst af en toe programmeeraanwijzingen.

LEERDOELEN

Na het bestuderen van deze leereenheid wordt verwacht dat u

- de primitieve typen Number, Boolean en String kent
- weet hoe u het type van een expressie of variabele kunt opvragen
- weet hoe u een stringwaarde kunt converteren naar een number
- de speciale waarden `undefined`, `NaN` en `null` kent
- de keuzestructuren `if…else` en `switch` kent en kunt toepassen
- de herhalingstructuren `for` en `while` kent en kunt gebruiken op de juiste wijze

Studeeraanwijzing

Deze leereenheid is gebaseerd op de hoofdstukken 1 en 2 van het tekstboek. U kunt het tekstboek op papier bekijken (zoals het is meegeleverd met het cursusmateriaal), in de vorm van een pdf-bestand (via Studienet), of in de vorm van een interactieve website (via Studienet of via http://eloquentjavascript.net/). In de papieren en de pdf-versie zijn

de hoofdstukken extra onderverdeeld in paragrafen en is er een index. Vanuit dit werkboek zullen we steeds refereren aan hoofdstukken en paragrafen van het tekstboek in de papieren en pdf-versie. De digitale versie mist beide, maar is wel voorzien van een interactieve tool waarmee u JavaScript kunt uitvoeren. In deze leereenheid gebruikt u die tool.

In het werkboek zullen we opmerkingen maken over de paragrafen van het tekstboek. Het is de bedoeling dat u ook alle opgaven uit het tekstboek maakt. Daarnaast zullen we in het werkboek soms ook opgaven opnemen. Deze kunt u dan maken in de console van het interactieve tekstboek.

Bij het studeren is het van belang eerst de aangegeven paragrafen van het tekstboek door te nemen en dan de opmerkingen daarover in het werkboek te bestuderen.

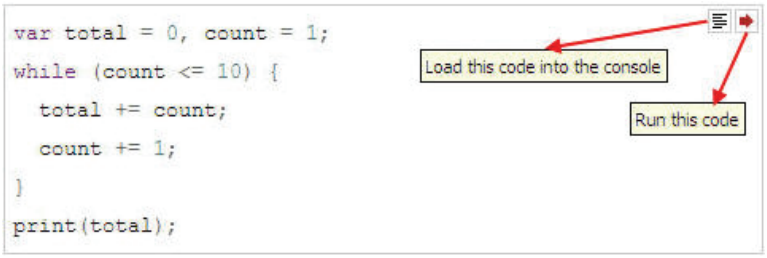
LEERKERN

1 Leren programmeren in JavaScript

Leren programmeren in een programmeertaal doet men door een tekstboek over die taal te bestuderen én door code in die taal te schrijven. Door alleen de taal te bestuderen, leert men nooit goed programmeren. Naast kennis van een programmeertaal zijn ook kennis en vaardigheden in het oplossen van problemen van groot belang. In deze cursus gaan we ervan uit dat u ook op dat gebied ervaring heeft.

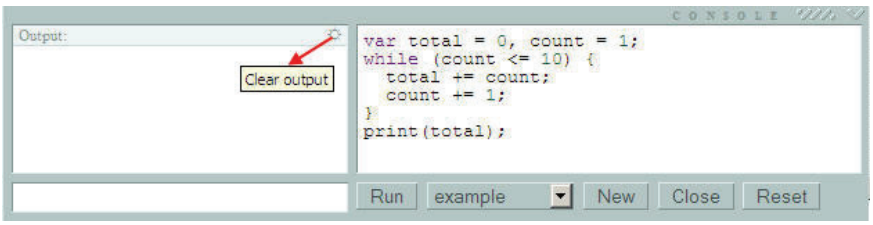
Console

De interactieve versie van het tekstboek (u moet dan kiezen voor de HTML-versie) bevat een console die het mogelijk maakt de voorbeeldprogramma's te draaien of zelf opgaven te maken of programma's te schrijven en te draaien. U ziet in de interactieve versie aan de rechterkant van de code of opgave twee icoontjes waarmee de console geopend kan worden, zie figuur 4.1.



FIGUUR 4.1 Stukje code met twee icoontjes

Als u op het linker icoontje klikt, wordt de code in de console geladen en deze wordt getoond aan de onderkant van het scherm.



FIGUUR 4.2 Console met de programmatekst

De console kent drie belangrijke elementen. Het eerste is het output window links waar de uitvoer van het programma getoond wordt.

Het tweede is de regel onder het output window die u kunt gebruiken om bijvoorbeeld een JavaScript expressie in te typen. Nadat u op Enter gedrukt heeft, wordt het resultaat van uw regel weergegeven als het een geldige expressie is, anders wordt een exception getoond. Met de pijltjestoetsen omhoog en omlaag kunt u voorgaande of volgende regels terughalen.

Het laatste element is het rechterdeel van de console dat u kunt gebruiken voor code die uit meerdere regels bestaat. Met Run kunt u de code draaien waarvan het resultaat weer links verschijnt.

2 **Chapter 1: Introduction**

Hoewel de syntaxis van Java en JavaScript wel overeenkomsten vertoont, zijn de twee talen zeer verschillend. Java is een statisch getypeerde taal waarvoor compilers beschikbaar zijn; JavaScript is een dynamisch getypeerde taal die als onderdeel van een webapplicatie geïnterpreteerd en uitgevoerd wordt door een browser.

Statisch getypeerd: koppeling van type aan variabele

Statisch getypeerd houdt in dat elke *variabele* een vast *type* is. Het type van een variabele kan dus niet tijdens de uitvoering van het programma veranderen. Bij het declareren van een variabele wordt vastgelegd van welk type hij is, en dat type houdt de variabele gedurende zijn hele leven.

Dynamisch getypeerd: koppeling van type aan waarde

Dynamisch getypeerd houdt in dat elke *waarde* een *type* heeft. Het type van een variabele is afhankelijk van de waarde die er aan is toegekend. Omdat de waarde van een variabele tijdens z'n levensduur kan veranderen, kan ook z'n type tijdens z'n levensduur veranderen. Bij een dynamisch getypeerde taal is het niet nodig bij declaratie van een variabele vast te leggen van welk type hij is: dat wordt bepaald door de waarde die aan de variabele wordt toegekend.

Een statisch getypeerde taal heeft onder andere als voordeel dat veel fouten snel zichtbaar worden (namelijk tijdens compileren); een dynamisch getypeerde taal heeft onder andere als voordeel dat casten (expliciet aangeven dat bijvoorbeeld een waarde van type `Object` behandeld moet worden alsof het van het type `String` is) nooit nodig is: robuustheid tegenover flexibiliteit.

3 Chapter 2, paragrafen 1 t/m 5

Number
Boolean
String

Het tekstboek behandelt in deze paragrafen het type *Number* (er is geen onderscheid tussen gehele en gebroken getallen; er zijn geen typen zoals Integer of Double), het type *Boolean* en het type *String*. In JavaScript zijn strings reeksen karakters tussen enkelvoudige of dubbele aanhalingstekens.

typeof

Bij de operatoren wordt *typeof* behandeld. Deze operator is belangrijk omdat hiermee van iedere expressie of variabele het type opgevraagd kan worden. Merk op dat deze operator de gevonden typen zonder hoofdletter aangeeft.

Keyword var

Variabelen worden gedeclareerd met behulp van het *keyword var*.

Programmeeraanwijzing: Declareer variabelen aan het begin
Het is een goed gebruik om in JavaScript alle variabelen helemaal in het begin van het programma te declareren.

Aan het eind van deze paragraaf laten we u in een opgave zien waarom dat belangrijk is.

Keyword const

Naast het keyword var kent JavaScript ook het keyword *const*. Dat gebruikt u voor constanten. Later in deze cursus zult u daar gebruik van maken. Het is de gewoonte om de naam van een constante met hoofdletters te schrijven.

Om de werking van een JavaScript-programma beter te begrijpen, zullen we soms gebruikmaken van geheugendiagrammen.

Primitieve typen
Kopie van de waarde

Number, Boolean en String zijn *primitieve typen*. Een variabele bestaat uit een naam en een waarde. Bij een toekenning van de waarde van de ene variabele aan een andere variabele krijgt de laatste variabele een *kopie van de waarde* als de waarde een primitief type heeft: bij het veranderen van de waarde van de kopie verandert de oorspronkelijke waarde niet mee, zoals figuur 4.3 laat zien.

FIGUUR 4.3 Geheugendiagram met toekenningsopdrachten

4 **Chapter 2, paragrafen 6 en 7**

print
show

Paragraaf 6 behandelt *print* en *show*. Binnen de console van het tekstboek kunnen deze functies gebruikt worden, maar niet in een normaal JavaScript-programma. Het zijn dus functies die exclusief zijn voor deze console.

Paragraaf 7 behandelt de basis-controlestructuren `while`, `for` en `if`. De syntaxis lijkt veel op die van Java. Voor het juiste gebruik van deze structuren is niet alleen de syntaxis van belang, maar ook kennis van probleemoplossen en algoritmiek. We gaan ervan uit dat u al over deze kennis beschikt.

Number

De inleiding van paragraaf 7 toont de functie `Number` waarmee een string geconverteerd kan worden naar een number.

```
var theNumber = Number(prompt("Pick a number", ""));
```

prompt

De functie *prompt* levert een string die door functie `Number` wordt omgezet naar een number. Het type van variabele `theNumber` is dan ook number. Als de string geen getal representeert, is het resultaat de speciale waarde `NaN`: Not a Number. De functie `prompt` is een functie van JavaScript (en dus niet alleen in de console beschikbaar).

parseInt
parseFloat

JavaScript kent nog twee andere functies om een string naar een number te converteren. Functie *parseInt* converteert de waarde van een string naar een integer. De functie kan een string als argument meekrijgen, of een string en een grondtal (er wordt uitgegaan van grondtal 10 als er geen grondtal wordt meegegeven). Bij `parseInt(<string>,10)` worden dus eventuele decimalen afgekapt. De functie *parseFloat* converteert de waarde van een string naar een floating point getal. In beide gevallen is het type `Number` (want JavaScript kent niet zoiets als Integers of Doubles).

if

Paragraaf 7.3 behandelt de keuzestructuur met *if*. Helaas is de behandeling onvolledig doordat er steeds maar één statement wordt uitgevoerd, afhankelijk van de conditie van de *if*. Als er meerdere statements uitgevoerd moeten worden, moeten deze statements, net als in Java, tussen { en } gezet worden.

Programmeeraanwijzing: Accolades bij een if
Zet altijd { en } om de statements bij een if, ook als het maar om één statement gaat. Door deze richtlijn zal het onderhoud van een programma gemakkelijker zijn.

De uitwerking van exercise 2.5 luidt dan met deze richtlijn:

```
var answer = prompt("You! What is the value of 2 + 2?", "");
if (answer == "4"){
  alert("You must be a genius or something.");
}
else {
  if (answer == "3" || answer == "5") {
    alert("Almost!");
  }
  else {
    alert("You're an embarrassment.");
  }
}
```

break

Bij paragraaf 7.4 willen we de kanttekening maken dat het gebruik van *break* ernstig is af te raden. Gebruik een `for`-lus als het aantal herhalingen bekend is, een `while`-lus als dat niet bekend is maar afhangt van een conditie. Door de conditie netjes te formuleren (vaak met behulp van een extra boolean) ontstaat veel beter leesbare en begrijpbare code die qua performance niets onderdoet voor de versie met `break`.

In plaats van de gebruikte programma's in paragraaf 7.4 is dit een betere versie:

```
var teller = 21, DELER = 7;
while ((teller % DELER) != 0){
  teller += 1;
}
print(teller);
```

We gebruiken hier ook de conventie om een constante met hoofdletters te schrijven. JavaScript kent het begrip constante echter niet. Een versie met expliciet gebruik van een boolean om de lus te stoppen luidt:

```
var teller = 21, DELER = 7, klaar = ((teller % DELER) != 0);
while (!klaar){
  teller += 1;
  klaar = ((teller % DELER) != 0);
}
print(teller);
```

OPGAVE 4.1
Maak exercise 2.6 uit het tekstboek, maar zonder gebruik te maken van `break`. Denk ook aan het juiste gebruik van de `if`.

5 Paragrafen 2.8 t/m 2.10

Paragraaf 2.8 toont een aanvulling op de typen uit de vorige paragrafen en allerlei eigenaardigheden van de taal JavaScript. We geven hier nog wat extra uitleg maar ook richtlijnen, zodat u zoveel mogelijk deze eigenaardigheden kunt vermijden.

undefined

Als we een variabele declareren maar niet initialiseren, zijn het type en de waarde van deze variabele *undefined*, wat we in een diagram aangeven met een ?, zie figuur 4.4.

FIGUUR 4.4 Een variabele met de waarde `undefined`

null

Om expliciet aan te geven dat een variabele *geen* waarde heeft, kunnen we, net als in Java, *null* gebruiken. Het bijzondere is dat de variabele dan van het type `object` is. Op objecten en variabelen van het referentietype komen we in de volgende leereenheden terug.

JavaScript past bij het evalueren van expressies automatische type-conversie toe. Dat lijkt op het eerste gezicht mooi, maar is in de praktijk een belangrijke bron van fouten. Bij vergelijkingen kunt u daarom het beste gebruik maken van `===` en `!===`, die zowel de waarden als de typen vergelijken. Deze operatoren vergelijken niet alleen de waarden maar ook de typen van de expressies aan weerszijden.

Programmeeraanwijzing: Gebruik geen automatische typeconversie
Zorg ervoor dat u deze automatische conversies nooit gebruikt. Dat doet u door bij een vergelijking === te gebruiken in plaats van == en bij de ontkenning daarvan !== in plaats van !=.

NaN

Bij rekenkundige operatoren treedt ook automatische conversie op waarop u wellicht niet bedacht bent. De conversie van een string naar een number kan, indien de string geen nummer voorstelt, mislukken. Hiervoor gebruikt JavaScript zoals we al zagen de waarde *`NaN`*. Als we de volgende code laten uitvoeren:

```
var a = "abc";
var b = "pq12";
var c = "45";
print(5+a);
print(5*a);
print(5*b);
print(5*c);
```

is het resultaat:

```
5abc
NaN
NaN
225
```

Omdat de operator `+` ook gedefinieerd is voor strings, zal bij de eerste printopdracht `5` naar een string geconverteerd worden en `abc` erachter geplakt worden.

De operator `*` is een pure numerieke operator en dus zal JavaScript pogen de string te converteren naar een Number. Dat lukt voor de variabelen `a` en `b` niet maar wel voor `c`.

isNaN

Om te testen of een variabele de waarde `NaN` heeft, gebruikt u functie *`isNaN`*. Merkwaardigerwijs is van een variabele met de waarde `NaN` het type gelijk aan Number.

Ten slotte behandelen we waarom het zinvol is alle variabelen direct aan het begin van een programma te declareren. Gegeven is het volgende programmafragment:

```
var a = 3;
print(a,": ",typeof a);
print(i,": ",typeof i);
var i = 5;
print(i,": ",typeof i);
```

Hoisting

Door dit programma uit te voeren, kunnen we een ander bijzonder aspect van JavaScript aantonen, *hoisting* genaamd. De eerste print-opdracht doet precies wat u verwacht en toont `3: number`. Wellicht zou u bij de tweede printopdracht een foutmelding verwachten, immers `i` moet dan nog gedeclareerd worden. De opdracht toont echter `undefined: undefined`. De laatste printopdracht toont `5: number`.

Wat er achter de schermen gebeurt is het volgende: de JavaScript-interpreter leest alle code en 'ziet' dat er onder andere een variabele `i` gedeclareerd wordt en direct wordt gebruikt in een toekenning. Bij de uitvoering van het programma maakt de interpreter er deze code van:

```
var a = 3,
    i;
print(a,": ",typeof a);
print(i,": ",typeof i);
i = 5;
print(i,": ",typeof i);
```

Scope

De declaratie van `i` wordt als het ware omhoog gehesen (hoisting betekent hijsen), maar de toekenning blijft op zijn oorspronkelijke plaats. Dit betekent dat variabelen in tegenstelling tot in Java een *scope* hebben die begint op de eerste regel van de code (dat geldt overigens alleen voor variabelen die buiten functies om worden gedeclareerd). Dit is dan ook de reden dat het goed is om alle variabelen direct aan het begin van een programma te declareren: u herinnert u zelf er daarmee aan dat de variabelen bekend zijn vanaf het begin van de code.

Als u als laatste opdracht

```
print(q,": ",typeof q);
```

toevoegt aan het bestaande programma, ziet u een foutmelding omdat `q` niet gedeclareerd is of in een toekenningsopdracht voorkomt.

6 Extra

Operator ?

Voor een eenvoudige `if…else` kent JavaScript ook een conditionele ternaire *operator* `?`.

```
<Conditie>  ?  <expressie 1> : <expressie 2>
```

Meestal kunnen we hierdoor wat kortere en beter leesbare code schrijven. Een voorbeeld:

```
var geslacht = 'M',
    naam = 'Jansen',
    aanhef = 'Geachte ' +
      (geslacht==='M' ? 'heer ' : 'mevrouw ') +
       naam;
```

switch

Wanneer u een aantal maal `else if` in uw conditionele statement gebruikt, is het overzichtelijker om over te stappen op het *`switch`*-statement dat JavaScript ook kent. De syntax van het `switch`-statement is:

```
switch (expression) {
   case label1:
      statements1
      break;
   case label2:
      statements2
      break;
   ...
   case labelN:
      statementsN
      break;
   default:
      statements_def
      break;
}
```

De werking ervan is dat `expression` wordt geëvalueerd. Wanneer het resultaat gelijk is aan `label1` worden `statements1` en verder uitgevoerd, tot er een `break` staat. Wanneer het resultaat gelijk is aan `label2` worden `statements2` en verder uitgevoerd. Een `break` zorgt ervoor dat niet automatisch alle volgende statements ook worden uitgevoerd: de executie van het `switch` statement stopt bij de `break`.

De `break` is optioneel: wanneer hij in een `case`-gedeelte ontbreekt, wordt alles wat in de volgende `case`-gedeelten staat ook uitgevoerd, tot de eerste `break`.

7 Samenvatting

We definiëren variabelen met behulp van het keyword `var`; we kunnen geen type specificeren. JavaScript leidt dat type af op basis van toekenningsopdrachten. We declareren onze variabelen en onze constanten helemaal aan het begin van het programma. Constanten declareren we met het keyword `const`. We schrijven de naam van constanten met hoofdletters.

JavaScript onderscheidt de volgende primitieve typen, zoals tabel 4.1 laat zien.

TABEL 4.1 Primitieve typen

type	*bijzonderheden*
Number	zowel integers als doubles
String	karakters tussen enkele of dubbele aanhalingstekens
Boolean	mogelijke waarden true en false

Daarnaast kent JavaScript een aantal bijzondere waarden, zoals tabel 1.2 laat zien.

TABEL 4.2 Bijzondere waarden

waarde	*bijzonderheden*
undefined	een variabele is wel gedeclareerd maar heeft nog geen waarde
NaN	een variabele stelt geen geldig getal voor
null	een variabele van het referentietype wijst nergens naar

JavaScript kent een vergelijkbare syntaxis als Java voor keuzestructuren (`if` en `switch`) en voor herhalingsstructuren (`for` en `while`).

JavaScript is een dynamisch getypeerde taal en voert automatische typeconversies uit.

ZELFTOETS

1 Schrijf een programma dat van een willekeurig aantal getallen het gemiddelde berekent en toont. Het programma moet steeds om een getal vragen en als de gebruiker als getal 777 invoert, moet de invoer stoppen en de berekening worden uitgevoerd. Het getal 777 moet niet meegenomen worden in de berekening.

Maak eerst een analyse van het probleem voordat u gaat programmeren. Gebruik zinvolle namen voor uw variabelen en houdt u zoveel mogelijk aan de richtlijnen van deze leereenheid, bijvoorbeeld door de juiste lusstructuur te kiezen. Test uw programma onder andere door direct 777 in te voeren en wijzig het indien dat nodig is.

TERUGKOPPELING

1 Uitwerking van de opgaven

4.1 De code is:

```
var klaar = false,
    answer = "";
while (!klaar){
  answer = prompt("You! What is the value of 2 + 2?", "");
  if (answer == "4"){
    alert("You must be a genius or something.");
    klaar = true;
  }
  else {
    if (answer == "3" || answer == "5"){
      alert("Almost!");
    }
    else {
      alert("You're an embarrassment.");
    }
  }
}
```

2 Uitwerking van de zelftoets

1 Voor het uitrekenen van een gemiddelde moeten we de ingevoerde getallen optellen en delen door het aantal. We gebruiken hiervoor de variabelen `invoer`, `som` en `aantal`.

Om te zorgen dat het invoeren stopt, gebruiken we een `while`-lus (we weten immers niet hoe vaak de lus doorlopen zal worden) waarin we testen of het ingevoerde getal ongelijk is aan 777, waarvoor we de constante `STOPGETAL` gebruiken. Om te kunnen vergelijken moeten we voor het begin van de lus de invoer al een keer lezen.

Als de invoer ongelijk is aan het stopcriterium, verhogen we het aantal met `1` en tellen we het ingevoerde getal op bij de som van de reeds ingevoerde getallen, en vragen we ten slotte om nieuwe invoer.

Na afloop kan het gemiddelde worden berekend, maar we moeten wel rekening houden met het feit dat direct `777` ingevoerd is. In dat geval is de waarde van aantal nog `0`. Hierop kunnen we testen en afhankelijk van deze test kunnen we het gemiddelde uitrekenen.

Om te kunnen rekenen en te vergelijken gebruiken we `Number` om de ingevoerde string te converteren naar een number.

```
var invoer = "",
    aantal = 0,
    som = 0,
    gemiddelde = 0;
const STOPGETAL = 777;
print("Dit programma berekent het gemiddelde van de door u
ingevoerde getallen");
invoer = Number(prompt("Voer een getal in, 777=stoppen",""));
while(invoer != STOPGETAL){
  som = som + invoer;
  aantal += 1;
  invoer = Number(prompt("Voer een getal in, 777=stoppen",""));
}
if (aantal > 0){
  gemiddelde = som/aantal;
  print("Het gemiddelde = " + gemiddelde);
}
else {
  print("Het gemiddelde kon niet bepaald worden");
}
```