JS adv project: Widget User Interface

by vdab

JS Statements

- JS is **Hoofdlettergevoelig**:
  de variabele *objTekstBox* is verschillend van *objtekstbox* en *OBJtekstbox*...
  de methode `Focus()` zal niet werken, `focus()` daarentegen wel
- JS **negeert witruimte** (spaties en tabs)
- Elke statement afsluiten met een **punt-komma** is aan te raden maar **optioneel**
- **Block** statements worden vervat in accolades `{}`, een punt-komma is onnodig
- **Commentaar** kan op twee manieren:
  - gebruik `//` voor commentaar op 1 lijn
  - gebruik `/* */` voor commentaar gespreid over meerdere lijnen
- **Escape strings**: het **backslash** karakter (\) gevolgd door een karakter zorgt ervoor dat dat karakter steeds letterlijk genomen wordt en niet geïnterpreteerd door de parser. We noemen dit een **escape sequence** .
  Bijvoorbeeld: onderstaande stringvariabele bevat de aanhalingstekens letterlijk

```
var strTekst = "en toen zei ze \"oei, het is gevallen\"";
```

  Sommige speciale statements worden ook met een escape sequence weergegeven:

  \b backspace

  \r nieuwe regel

  \n nieuwe lijn

  \t tabulatie
- **Identifiers** zijn de termen die je mag gebruiken voor **variabelen** en **functies**:
  ze mogen beginnen met een letter, een underscore _ , of een dollarteken $.
  ze mogen niet beginnen met een getal
- bij conventie beginnen alle variabelennamen, functienamen, parameters met een kleine letter, enkel Constructors krijgen een hoofdletter
- **Sleutelwoorden** mag je niet gebruiken als variabele of fucntienaam.
  Er zijn er heel wat: `break, do, this, final,` ...
  alle sleutelwoorden worden in kleine letters geschreven

waarden

JS kent 3 **primitieve** datatypes, `boolean`, `number` en `string` met daarnaast **objecten** en `null` en `undefined`. Een primitief datatype is een eenvoudige waarde in het computergeheugen tegenover objecten die complexe gegevens zijn.

number

Er is slechts één getaltype in JS: `number`, dat infeite een 64-bit floating point getal (=double). Er is geen apart integertype.

Er bestaat een speciaal getal `NaN`, dat *geen getal* voorstelt: iets is geen getal. `NaN` is het resultaat van een `undefined` waarde of een foutieve bewerking. `NaN` is besmettelijk: het plant zich voort in een bewerking (2 * `NaN` = `NaN`) .
`NaN` kan met niets vergeleken worden, inclusief zichzelf. Om te testen voor deze waarde maak je gebruik van de speciale functie `isNaN()`.

Omdat bijna alle waarden in het programma binnenkomen als tekst - we zijn tenslotte bezig in een webpagina - zijn er enkele speciale functies die tekst omzetten in een getal:

- `Number()`: zet een waarde om in een getal (base-10). Bij problemen krijg je `NaN`
- `parseInt()`, `parseFloat()`: zetten waarden resp. om in integers of in floating point getallen. Een verschil met is dat er ook hexadecimale of octale getallen kunnen omgezet worden dmv een 'radix' argument. Bij problemen returnt de functie een `NaN`

```
var getal = Number(" 33 ") //geeft 33

var probleem = Number (3 muizen)
```

Om berekeningen te kunnen maken met getallen is er een apart Math object met heel wat functies.

string

Een `string` is een sequentie van nul of meer **Unicode** karakters vervat in enkele (apostrophe) of dubbele aanhalingstekens. Deze twee kunnen in elkaar vervat worden, zoniet moet je escape sequences gebruiken:

```
"<i>htmltekst vervat in i tag</i>";
```

```
"'alternerende types aanhalingstekens'";
"en toen zei ze: \"ik heb het laten vallen\"";
```

Om twee strings te **concateneren** gebruiken we de + operator:

```
'hello' + 'world' = 'hello world'
```

Om het aantal karakters van een string te kennen gebruiken we de `length` property:

```
var tekst = "";
tekst.length; // geeft 0
```

Om een getal in een string om te zetten gebruiken we de `String` functie:

```
var waarde = 9
isNaN(String(waarde) // geeft true
```

Een string is infeite een object en heeft een heleboel String methods laten ons toe ze te manipuleren.

Nog enkele bemerkingen:

- enkele aanhalingstekens kunnen variabelen niet interpreteren zoals in php
- JS heeft geen apart data type voor een enkel karakter zoals in C of Java
- om te weten of ze identiek zijn kan je twee strings vergelijken worden met de `==` operator

boolean

Er zijn uiteraard slechts twee `boolean` waarden: `true` en `false.`
Er is ook de functie `Boolean()` die 'falsy' en 'truthy' waarden evalueert tot een resulterende `boolean`

```
  false == false // returnt true
  Boolean(null) // returnt false
```

de waarden `false`, `null`, `undefined`, een lege string, het getal 0, geven steeds `false` in deze functie, de rest geeft `true`.

null

De waarde `null` staat voor *geen waarde, geen object*. Als een variabele de waarde `null` bevat, weet je dat het een ongeldige waarde is.

Door een variabele de waarde `null` te geven, vernietig je de variabele.

undefined

De waarde `undefined` wordt gereturnt *als een variabele gedeclareerd is, maar geen waarde gekregen heeft*. `undefined` krijg je ook als een object niet gevonden kan worden. De `void` operator returnt ook `undefined`.

```
null == undefined; // returnt true
null === undefined; //returnt false
var v   // returnt undefined
var tekort = document.getElementById('afwezig'); // returnt undefined
void 0 === undefined;   // returnt true
```

objecten

Alle andere waarden in JS zijn objecten, inclusief functies.

Een **object** is een container voor **naam=waarde** paren, andersgenoemd **properties** (eigenschappen). De *naam* is een `string`, de *waarde* eender welk datatype uitgezonderd `undefined`. Die properties kunnen dan bereikt worden ofwel met de . notatie, ofwel als in een *associatief array*, met vierkante haakjes en de naam van de property in aanhalingstekens:

```
image.width
image["width"]
```

JS bevat een aantal ingebouwde objecten zoals het Math, Date object.

Objecten kunnen aangemaakt worden, maar sommige properties van objecten zijn zelf een object, bijvoorbeeld `document.cookie`

```
var o = new Object();
o.x = 4.5;
o.y = 0;
```

zoals je hierboven merkt kan je zelf arbitraire eigenschappen bedenken en die eender welke waarde geven.

Zo zijn properties eigenlijk variabelen die een waarde bevatten. Bevat die variabele echter zelf een functie (zie hieronder) dan spreken we van een **method** en dan schrijven we de eigenschap met **()** erachter:

```
document.form.button.focus      //is de eigenschap focus en returnt true of false
document.form.button.focus()    //is de method focus die de focus op de knop zet
```

expressies en operatoren

Een **expressie** is een JS statement dat geëvalueerd wordt tot een resultaat. Een expressie die bestaat uit een letterlijke waarde is die waarde zelf, maar waarden kunnen ook vergeleken worden of er kan een bewerking mee gedaan worden. Voorbeelden van expressies:

```
"javascript is OK"      // een letterlijke expressie
tekst.length    // het aantal karakters in een string
i++     // de variabele i verhogen met 1
i>j     // twee variabelen vergelijken
```

Expressies maken gebruik van **operatoren**:

toewijzingsoperatoren:

Naast de normale toewijzing operator = bestaan er nog enkele 'shortcuts'

=	x = y	normale toewijzing
x += y	x = x + y	

berekeningsoperatoren

+	1 + 2 = 3 1 + "1" = "11"	som indien één van de waarden een string is wordt dit een concatenation operator
-	1 - 2 = -1 "1" -"2" = -1	verschil indien mogelijk gebeurt er impliciete conversie van andere types
*	1 * 2 = 2 "3" * "2" = 6	vermenigvuldiging indien mogelijk gebeurt er impliciete conversie van andere types
/	1 / 2 = 0.5 1 / 0 = NaN	deling indien één van de waarden 0 is, dan evalueert de expressie tot `NaN`
%	12 % 5 = 2	Modulo Geeft de rest van een deling
++	als x = 3 dan x++ geeft x = 3 ++x geeft x = 4	Stijging voegt 1 toe voor of na de waarde
--	als x = 3 dan x-- geeft x = 3 --x geeft x = 2	Daling trekt 1 af voor of na de waarde
-	-y	Negatie Geeft de negative waarde van

vergelijkingsoperatoren

==	3 == "3"	True indien gelijk, niet noodzakelijk van hetzelfde type
===	x === y	True indien gelijk en van hetzelfde type

!=	x != y	Niet gelijk. Verschillende types worden gerpobeerd gelijk te stellen
!==	x !== y	strikt niet gelijk. False indien niet van hetzelfde type
>		groter dan
<		kleiner dan
>=		ten minste
<=		ten hoogste

logische operatoren

&&	expr1 && expr2	Logische AND
||	expr1 || expr2	Logische OR
!expr	!expr	Logische NOT

string operatoren

+	"text" + "text"	concatenation
+=	str1 += str2	shorthand voor str1 = str1 + str2

andere operatoren

in

the `in` operator:
geeft `true` als de linkse waarde een *property* is van het rechtse *object*

instanceof

the `instanceof` operator:
evalueert of een objectvariabele een instance is van een object

typeof

the `typeof` operator:
evalueert het type van een variabele. Mogelijke returnwaarden zijn "number", "string", "boolean", "object", "function", "undefined" .
Niet perfect want ook arrays en `null` worden als object getypeerd...

void

the `void` operator:
wordt gebruikt om een returnwaarde op `undefined` te zetten, zodat de expressie niets teruggeeft. Typisch bij een javascript statement in de `href` van een `a` element

new

the `new` operator:
wordt gebruikt bij een constructor van een object

conditionele operator

var = (test) ? true : false

tekst = (G>10) ? 'groot' : 'klein'

variabelen

Een variabele is een naam die een waarde krijgt. JS reserveert een stukje geheugen om de waarde - gekoppeld aan de variabelenaam - te bewaren, tijdelijk. De variabele krijgt een waarde of het resultaat van een bewerking toegewezen.
In JS is het ook mogelijk een functie aan een variabele toe te wijzen.

```
i = 0;
var totaal = 23 *7;
var flag = true;
var voorlopig = undefined;
var lijst = ['jan','piet','pol'];
var mijnObject.spreek = function() { alert(this.naam) };
```

typing

Typing gaat over de vraag welke soort waarde de `var` bevat: een tekst, een getal, een true/false, een object?

Javascript is een 'losjes getypeerde' taal. Javascript kent data types maar ze worden niet streng toegepast. In tegenstelling tot 'sterk getypeerde' talen zoals C++ of Java zijn een aantal dingen mogelijk:

- Een variabele wordt geen data type toegekend tijdens de declaratie, dus kan je er eender welke waarde aan toekennen
- dat betekent niet dat de variabele niet getypeerd is: afhankelijk van de waarde krijgt de variabele een datatype toegekend

declaratie

Om een variabele te gebruiken moet je hem **declareren** met het `var` sleutelwoord:

```
var i;
var totaal;
var voornaam;
```

omzetting

We merkten eerder op dat JS veel **impliciete** conversies doet: zo zullen in `'3'*'2'` de `string`'s omgezet worden naar `number` om de vermenigvuldiging te doen slagen. Dat gebeurt niet altijd, vooral als de bewerking onduidelijk is zoals bij een optelling. Soms moeten we dus zelf proberen een typeconversie te doen.

Omzetting naar Number

- met `parseInt(), parseFloat()`
- met `Number()`

Omzetting naar een String

- met `toString()`
- met `String()`

Omzetting naar boolean

- *type-cast* met `Boolean()`

scope

De **scope** van een variabele is het **bereik** binnen het programma waarin de variabele geldt.

Een ***global*** variable is overal bereikbaar (binnen de geheugenruimte van één webpagina) in tegenstelling tot een ***local*** variable die enkel bereikbaar is binnen de functie waarin hij gedeclareerd wordt.

eval()

De globale functie `eval()` evalueert - en voert dus eventueel uit - een Javascript expressie. Deze functie is niet geassocieerd met een bepaald object.