JavaScript

Yleistä

- JavaScript on skriptikieli
- Ajetaan selaimessa käyttäen JavaScript-moottoria (JavaScript engine)
- Esimerkkinä JavaScript-moottorista Googlen V8 engine tai Firefoxin SpiderMonkey

Johdanto

- JavaScriptiä ajetaan isäntäympäristössä (esim. Web-selain)
- Kieltä voi käyttää missä tahansa ympäristössä, johon JavaScript-moottori on saatavilla ja integroitu (esim. NodeJs- palvelinsovellus)
- Kieli on standardoitu: ECMAScript (ECMA-262)
- JavaScript ei ole Java-kielen "alijoukko" ja sillä on itse asiassa vain vähän tekemistä Javan kanssa

> "JavaScript is a prototype-based scripting language that is dynamic, weakly typed and has first-class functions. It is a multi-paradigm language, supporting object-oriented, imperative, and functional programming styles." WikiPedia

Kehitys

- Netscapen vuonna 1995 Netscape Navigator 2 -selaimen mukana julkaisema JavaScript 1.0 -kieli mahdollisti ensimmäistä kertaa interaktiivisten HTML-dokumenttien tuottamisen.
- Hieman sen jälkeen Microsoft kehitti oman kieliversionsa, JScript-kielen.
- Vuonna 1998 julkaistiin *ECMA-262* (**ECMAScript**) 1st + 2nd edition, joka standardoi kielen.

EcmaScript-standardikieli

- **ECMAScript** on suunniteltu sovellusriippumattomaksi skriptikieleksi, jolloin se sopii minkä tahansa sovelluksen ohjelmointiin.
- Sovellusriippumattomuus saavutetaan jakamalla kieli kahteen osaan: sovellusriippumattomaan runkokieleen (core language) ja sovellusriippuvaiseen oliomalliin (DOM, Document Object Model).
- v. 2019 uusin kieliversio on EcmaScript 2019 / EcmaScript 10th version / ES10

Selaimet ja yhteensopivuus

- JavaScript jättää sovelluskohtaiset asiat jonkin sovelluskohtaisen oliomallin huoleksi

- WWW-dokumenteissa käytettävä DOM (Document Object Model) on esimerkki oliomallin liittämisestä JavaScript-kieleen.
- Esimerkki selaimesta: Firefox 4 ja uudemmat selainversiot tukevat ECMAScript-versiota 5 sisältäen tuen mm. Object.* metodeille ja strict moodille.

```
<div style="page-break-after: always;"></div>### DOM (Document Object Model)
```

Dokumenttioliomalli (DOM) määrittelee, kuinka dokumentissa olevat elementit välittävät tietoa toisilleen ja kuinka elementteihin voidaan viitata. Tämän DOM saavuttaa kuvaamalla dokumentin elementtihierarkian puurakenteen mukaisesti.

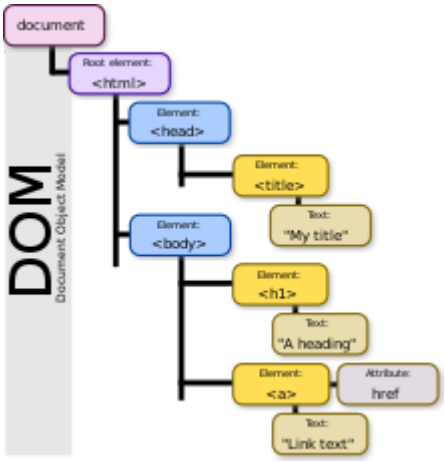
Dom-oliomalli on hierarkkinen dokumenttipuu, joka koostuu etukäteen rajoittamasta määrästä solmuja(nodes). Jokainen solmu on yksittäinen olio hierarkiassa. Dokumenttiolio on koko hierarkian ylimmällä tasolla. Jokaisella solmulla voi olla useita lapsisolmuja (child nodes, children). Vastaavasti kaikilla solmuilla on paitsi juurisolmulla (body, root node) on äitisolmu (parent node).

Dokumenttipuun rakenne muistuttaa hyvin pitkälti oikeaa puuta:

```
Puussa on juuri; dokumenttipuussa on runko (body, root node)
Puussa on oksia; dokumenttipuussa on lapsielementtejä (children, child node)
Puussa on lehtiä; dokumenttipuussa on lehtiä (nodes), joilla ei ole enää omia la
```

Esimerkki:

```
<!DOCTYPE html>
<html>
<head>
    <title>My title</title>
</head>
<body>
    <h1>A heading</h1>
    <a href="">link text</a>
</body>
</html>
```



W3C määrittelee DOM-suosituksessa metodit ja ominaisuudet rajapintoina.

JavaScript-koodin suorittaminen

- JavaScript-koodi ajetaan WWW-selaimeen sisäänrakennetulla JavaScript-moottorilla.
- JavaScript koodia ei tarvitse kääntää mitenkään, vaan se suoritetaan tulkkaamalla.
- Selain suorittaa koodin ja tulos on nähtävillä heti virheineen.

Selainten JavaScript-moottorit

Suosituimpien selainten JavaScript-moottorit:

- SpiderMonkey is Mozilla's JavaScript engine written in C/C++. It is used in various Mozilla products, including Firefox, and is available under the MPL2.
- V8 is an open source high-performance JavaScript engine, written in C++ and developed by Google and used in Google Chrome and Chromium based web browsers. V8 can also run within a standalone application like node.js.
- Chakra is a JavaScript engine developed by Microsoft for its Internet Explorer 9 (IE9) web browser. While developed originally as closed source proprietary software, on January 2016 it was released as open source software under the MIT license. A modified version is also used by the Edge browser shipped with Windows 10. But the new Edge browser that ships with Windows 10 in the first half of 2020 is based in Chromium and also use Chromium's javaScript engine (V8).

Ajo selaimessa

JavaScript-koodi ajetaan selaimeen sisäänrakennetulla moottorilla. Moottorit ovat nykyään erittäin tehokkaita.

Samalla voidaan ladata script-tagin avulla myös muiden kirjoittamia skriptejä (Open Source pohjaisia) tai yrityksen sisällä tuotettuja omia kirjastoja. Omat kirjastot kirjoitetaan yleensä funktioina (functions) tai luokkina (objects).

Niitä jaellaan erillisissä tiedostoissa (.js-loppuinen), jossa pelkästään JavaScriptiä

Javascript koodia lisätään HTML -sivuille tagiparien `<script>` ja `</script>` avulla. JavaScript koodia voidaan sijoittaa mihin kohtaan dokumenttia tahansa, mutta yleisesti sijoitetaan juuri ennen `</body>` -lopputagia, koska siinä vaiheessa HTML -tiedosto on jo luettu ja DOM on pystyssä. Eli Javascript -koodi pystyy lukemaan tai muokkamaan sitä.

On myös mahdollista lisätä kokonaan Javascript -tiedostoa tällä lohkolla `<script src="JStiedosto.js"></script>`. Kun yleisesti kyseessä on Javascript kirjastoa, joka ei valttamatta lue tai muokkaa DOMia, tälläisen lohko upotetaan HTML-sivuille `<head>` -tagien väliin.

Javascriptin kielen johdanto

Syntaksi

JavaScript-kielen perussyntaksi on hyvin samankaltaista kuin C- tai Java-kielissä, joista suurin osa kielen avainsanoista (keywords) on periytynyt. (Avainsana on varattu sana, jota ei voi käyttää esimerkiksi muuttujien tai funktioiden nimissä.)

JavaScript-ohjelma rakentuu seuraavista osista:

- Lauseet
- Operaattorit
- Lausekkeet

Koodin kirjoittaminen

JavaScript-kielen **lause** on yhden rivin mittainen, ja se **päättyy puolipisteeseen**:

`document.writeln("Tulostaa dokumenttiin!<BR>");`

Puolipiste ilmaisee lauseen päättymisen. Yksittäinen lause voidaan kirjoittaa myös ilman puolipistettä

`console.log("Tulostaa selaimen konsolille")`

JavaScript-kieltä voi kirjoittaa millä tahansa ASCII- tekstieditorilla tai HTML-editorilla. Jotkut HTML-editorit tuottavat automaattisesti JavaScript-koodia.

Komenttia voidaan lisätä `//` -merkkeillä. Se on rivin mukainen komentti. Eli kun merkki käytetään, sen jälkeen kaikki on komentti rivin loppuun asti. Seuraava rivi on taas normaali.

```
i = 1; // Tämä on komentti. i=2; Tätä ei suoritetaan
// Koko rivi on komentti
```

Tietotyypit

JavaScript-kielen tietotyypit jaetaan:

- Alkeistietotyyppeihin (engl. *Primitives*). Esim. numerot ja boolean.
- Oliotietotyyppeihin (engl. *Objects*). Esim. Date, Math, Array ja itse tehdyt oliot.

Alkeistietotyypit

Alkeistietotyypien muuttujat määritellään `let` -, `var` - tai `const` - avainsanoilla. Muuttujan näkyvyys (engl. *Scope*) vaihtelee sen mukaan missä ja miten se on määritelty (1, 2 ja 3).

> "primitives: In JavaScript, a primitive (primitive value, primitive data type) is data that is not an object and has no methods." MDN

On olemassa 7 alkeistietotyyppiä:

- Numerot - **Number**
- Isonumerot - **BigInt**
- Merkkijonot - **String**
- Boolean arvot - **Boolean**
- Symbolit - **Symbol**
- Ja kaksi yksiarvoiset datatyypit:
  - **null**
  - **undefined** (määrittelemätön)

Numerot

```
1 // Kokonaisluvut
2.3 // Desimaaliluvut
1 / 0 // Infinity. Infinity on numero JavaScriptissa
1 / 'a' // NaN (Not a Number). NaN on numero JavaScriptissa
typeof 1 // number
typeof 2.3 // number
typeof Infinity // number
typeof NaN // number
```

Isonumerot

`BigInt` on tarkoitettu esittamaan isoja numeroita, isompia kuin `Number` pystyy esittamaan. `BigInt` määrittellään lisäämällä `n` numeron perään. `BigInt` ja `Number` eivät ole yhteensopivia.

```
1n
9007199254740992n
1 + 1n // Virhe!!!
(2 ** 127) + 1 == (2 ** 127) // true. Number ei ole riittävää tärkkää isoilla
numeroilla
(2n ** 127n) + 1n == (2n ** 127n) // false
```

Merkkijonot

```
'abc' // Heittomerkkiä (Single quotes)
"abc" // Lainausmerkkiä (Double quotes)
`abc` // Backticks
'i = ' + i // Vanha tapa
`i = ${i}` // Uusi tapa
typeof 'abc' // string
```

Boolean arvot

```
typeof true // boolean
typeof false // boolean
```

null

```
typeof null // object!!! Se on tuttu virhe, mutta ei korjataan
yhteensopivuuden syistä.
```

undefined (määrittelemätön)

```
typeof undefined // undefined

let m; // m muuttujan on olemassa, mutta se on määrittelemättä
if (m === undefined) {
    // nämä lauseet suoritetaan
} else {
    // Näitä lauseita ei suoritetaan
}

// tai

if (typeof x === 'undefined') {
    // nämä lauseet suoritetaan
} else {
    // Näitä lauseita ei suoritetaan
}
```

Symbolit

Symbolit käytetään harvoin.

```
typeof Symbol('id') // symbol
```

null vs. undefined

Null on nolla ja undefined on määrittelemätön eli muuttajalla ei ole mitään asetettua arvoa

```
typeof null // object
typeof undefined // undefined
null === undefined // false
null == undefined // true
```

Oliot

Olio on monimutkaisempi tietotyyppi, joka voi sisältää muita tietotyyppia. Sillä on yleisesti omia *ominaisuuksia* ja *metodeja*. Oliotyypit:

- Array
- Date
- Math
- RegExp
- Function
- ...

```
typeof Math // object
typeof [1,2,3] // object
```

Oliot voidaan määrittyä eri tavalla:

```
// literal notation
let henkilo = {
    nimi: 'Seppo',
    sukunimi: 'Suomalainen',
    pituus: 187
}

let henkilo2 = new Object();
henkilo2.nimi = 'Sami';
henkilo2.sukunimi = 'Suomalainen';
henkilo2.pituus = 187;

// Voidaan käyttää myös hakasuluja:
henkilo2['pituus'] = 192;

let taulukko = [1,2,3];

let nyt = new Date();
let pvm = nyt.getDate() + "." + (nyt.getMonth()+1) + "." + nyt.getYear();
```

Operaatiot

- `+` Yhteenlasku. Toimii myös merkkijonoilla yhdistäen merkkijonot toisiinsa.
- `-` Vähennyslasku. Jos operandit eivät ole numeerisia, ne yritetään muuttaa numeerisiksi ennen operaatiota.
- `*` Kertolasku. Jos operandit eivät ole numeerisia, ne yritetään muuttaa numeerisiksi ennen laskuoperaatiota.
- `/` Jakolasku. Toimii reaalilukuna, vaikka operaattorit olisivatkin kokonaislukuja.
- `%` Jakojäännös. Laskee kahden luvun jakojäännöksen, ja toimii myös reaaliluvuille.
- `**` Potenssi
- `%` Jakojaannos
- `=` Sijoitusoperaatio

```
2 + 2 // Addition
2 - 2 // Substraction
2 * 2 // Multiplication
2 / 2 // Division
2 ** 2 // Exponentiation
2 % 2 // Remainder / Modulus
i = 2 // Asigment
a = b = c = 2 // Chained asigment
'abc' + 'd' // Concatenation
```

Unaariset operaatiot

- `++` Lisää lukua yhdellä

- `--` Vähentää lukua yhdellä
- `-` Negaatio vaihtaa numeron etumerkki

```
i++ // Post Increment by 1
i-- // Post Decrement by 1
++i // Pre Increment by 1
--i // Pre Decrement by 1
i = 1
i = -i // i = -1

// Post increment vs Pre increment
i = 1
j = i++ // j=1, i=2
i = 1
j = ++i // j=2, i=2
```

Ehdollinen operaatori (Ternary operator)

Sijoitus voidaan tehdä ehdollisella operaattorilla (if/else):

```
let on_posit = (x > 0) ? true : false
```

Se on sama kuin:

```
if (x > 0)
    on_posit = true;
else
    on_posit = false;
```

Binäärisiä operaatioita vastaavat unaariset operaatiot:

- `summa = summa + i` vs. `summa += i`
- `summa = summa - i` vs. `summa -= i`
- `tulo = tulo * x` vs. `tulo *= x`
- `jako = jako / x` vs. `jako /= x`
- jne...

```
i += d // i = i + d
i -= d // i = i - d
i *= d // i = i * d
i /= d // i = i / d
i **= d // i = i ** d
i %= d // i = i % d
```

Loogiset operaatiot

- `a && b` : **AND** eli looginen **ja**. `a && b` palauttaa arvon *true* vain, jos molemmat operandit ovat *true*.
- `a || b` : **OR** eli looginen **tai**. `a || b` palauttaa arvon *true*, jos vähintään toinen operandeista on *true*.
- `!a` : **NOT** eli looginen **ei** (on unaarinen). `!a` palauttaa arvon *true*, jos a on *false* ja paulauttaa *false* toisaalta.

```
true && false // false
true && true // true
true || false // true
false || false // false
```

a && b	true	false
true	true	false
false	false	false

a || b	true	false
true	true	true
false	true	false

bittitason operaatiot

- `&` bittitason **JA** (*AND*)
- `|` bittitason **TAI** (*OR*)
- `^` bittitason **joko/tai** (*XOR*, exclusive or)
- `~` bittitason **EI** (*NOT*)
- `<<` siirto vasemmalle (*shift left*)
- `>>` siirto oikealle (*shift right*)
- `>>>` siirto oikealle nollalla (*shift right*)

```
// Bitwase operators: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Bitwise_Operators
1 & 0 // AND
1 | 0 // OR
1 ^ 0 // XOR
~0 // NOT
1 << 2 // Left shift
1 >> 2 // Right shif
-1 >>> 1 // Zero-fill right shift
```

Lauserakenteet

JavaScript sisältää seuraavat ehtolauseet ja silmukat (Java-kielen perintönä):

- if (ehtolause "jos")
- while (silmukat)
- for (silmukat).
- break ja continue toistojen katkaisuun silmukoissa
- Lisäksi try/catch virheenkäsittelyyn hallitusti

Ehtolause – if else

Perus if-else rakenne on:

```
if (ehtolauseke) lause;
else lause;

// tai

if (ehtolauseke)
    lause;
else
    lause;
```

Huoma seuraavat:

- Ehtolauseke on sellainen lauseke, joka palauttaa *true* tai *false*.
- Sulut ovat **pakollista**. Ehtolauseke on oltava sulkujen sisällä. Jos ehtolauseke on monimutkaista on mahdollista lisätä vielä lisää sulkuja
- Kirjaimien koolla on merkitystä. Eli *If* tai *IF* ovat väärin
- `else` -osa on **ehdollista**. Ei tarvitse käyttää sitä jos ei ole tarvetta.
- *lause* on yhdenrivinen JavaScriptin lause. Jos tarvitaan lisää lauseita, korvataan sen lohkolla:

  ```
  if (ehtolauseke) {
      lause;
      lause;
  }
  else {
      lause;
      lause;
  }
  ```

- Kaarisulujen jälkeen ei tarvitse käyttää puolipisteetta.
- Kaarisulujen sijainti ei ole pysyvä. Eri kehittäjiä käytävät niitä eri tavalla:

```
if (ehtolauseke) {
    lause;
    lause; }
else {
    lause;
    lause; }

// on sama kuin

if (ehtolauseke)
{
    lause;
    lause;
}
else
{
    lause;
    lause;
}
```

- Sisennys ei ole myöskään pakollista, mutta se auttaa lukemaan koodia.
- Samalla tavalla rivinvaihdot eivät ole pakollista, mutta ovat hyvin suositelttava.

Esimerkki:

```
if (ika < 0)
    console.log("Ikä ei voi olla negatiivinen!");
else
    if (ika >= 0 && ika<18)
        console.log("Et ole vielä työiässä!");
    else
        if (ika >= 18 && ika <= 65)
        { // HUOM! Kaarisulkujen tärkeys!
            if (ika >= 40)
                console.log("Olet keski-ikäinen!");
            else
                console.log("Olet nuori työikäinen!");
        }
        else
            console.log("Olet eläkeläinen!");
```

while-silmukka

Perus while-silmukan rakenne on:

```
while (ehtolauseke) {
    lause;
}
```

- Samanlaisia saantoja kuin if-ehtolausella (kirjainkoko, sulut, jne...)
- lause suoritetaan niin kauan kuin ehto on totta. Joten ehdon on jossain vaiheessa oltava väärä. Muuten se on ääretön silmukka (ääretön silmukka pysäyttää ikuisesti ohjelman, kuin ei voi suorittaa mitä muuta.)

Esimerkki:

```
let x = 1;
let summa = 0;

while (x <= 10) {
    summa += x;
    x++;
    console.log(`summa on ${summa} ja x on ${x}`);
}
```

for-silmukka

Perus for-silmukan rakenne on:

```
for (alustavalause; ehtolauseke; jälkilause) {
    lause;
}
```

- Samanlaisia saantoja kuin edellisissa ehtolauseilla (kirjainkoko, sulut, jne...)
- *alustavalause* suoritetaan ennen kaikkea ja vain yksi kerta.
- Jo *ehtolauseke* on totta, suoritetaan se päälause/päälohko. Sen jälkeen suoritetaan se jälkilause. Tehdään toinen kierros siihen asti, että ehto on väärä
- Eli taas on mahdollista luoda vahingossa jonkun ääretön silmukka

Esimerkki:

```
let summa = 0;
for (let i = 1; i <= 10; i++) {
    summa += i;
    console.log(`Kierros ${i}: summa = ${summa}`);
}
```

do while -silmukka

Samanlainen kuin while -silmukka, mutta ehtolauseke on lopussa. Eli ehto tarkistetaan kierroksen jälkeen.

```
do {
    lause;
} while (ehtolauseke) {
```

Esimerkki:

```
let i = 0;
do {
    console.log(`Kierros: ${i++}`);
} while (i <= 5);
```

with lause

Voidaan muuttaa olion ominaisuuksia ja viitataan ominaisuuteen ilman this-viitettä

```
let henkilo = {
    nimi: 'Aku',
    ika: 61
}
with (henkilo) {
    ika += 1;
}
```

Varatut sanat

Seuraavat sanat ovat varattuja ja ei saa käytää omissa koodissa tunnisteena (esim. muuttuja)

- await
- break
- case
- catch
- class
- const
- continue
- debugger
- default
- delete
- do
- else

- enum
- export
- extends
- false
- finally
- for
- function
- if
- import
- in
- instanceof
- new
- null
- return
- super
- switch
- this
- throw
- true
- try
- typeof
- var
- void
- while
- with
- yield