

# JavaScript ohjelmointi

## *Valmistautuminen Pikku eläintarha -tehtävään*

MARGIT TENNOSAARIN MATERIAALISTA  
LAURA JÄRVISEN MUOKKAAMA

# Pikku Eläintarha

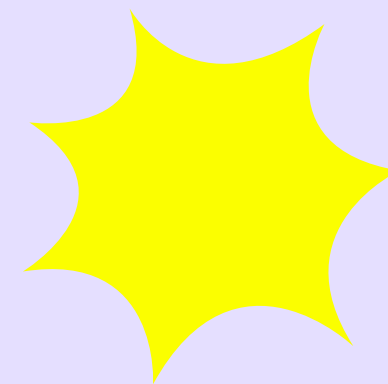
Tehtävä yhdistelee tähän asti opittuja asioita DOM-muokkaamisesta taulukoihin ja olioihin.

Seuraavissa dioissa esitellään tehtävässä käytettävät metodit ja muut käsitteet.

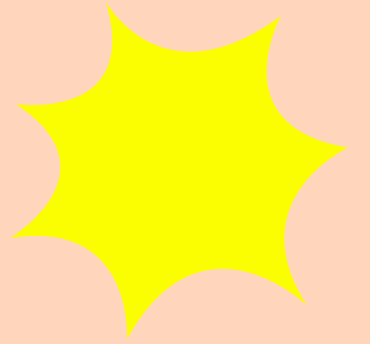
Suurin osa on jo tuttuja



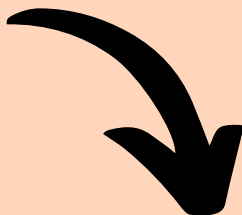
muutama uusia.

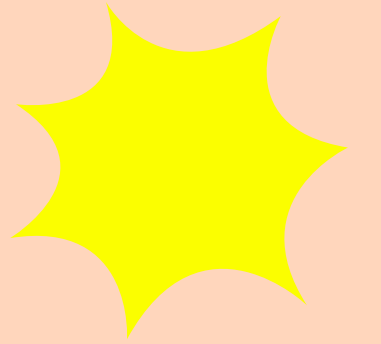


# Käsitteet 1



## Callback

- Callback-funktio on ohjelmoinnissa funktio, joka annetaan toiselle funktiolle parametrina, ja jonka toivottu toiminto ajetaan myöhemmin juuri siellä, missä kutsu (“call”) tapahtuu. Callbackien ydinajatus on siis korkeamman tason abstrahointi: teet jonkin operoinnin (esim. syötteen läpikäynnin, asynkronisen tapahtuman käsittelyn tai vaikkapa DOM-päivityksen) ja annat samalla parametriksi ketjun viimeisen toiminnon, eli callbackin.
- Esimerkki 



- filter käy läpi jokaisen taulukon alkion.
- Callbackina annettu **fruit => fruit.includes("i")** palauttaa true aina, kun merkkijono sisältää kirjaimen "i".

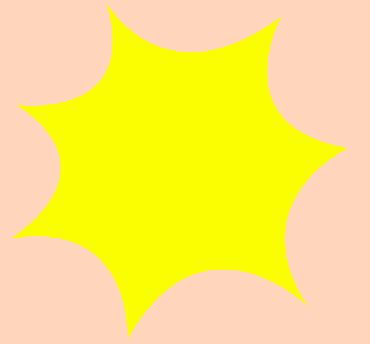
```
const fruits = ["omena", "banaani", "kirsikka", "appelsiini", "luumu"];

// Haluamme valita vain ne hedelmät, joissa on kirjain "i"
const withI = fruits.filter((fruit) => fruit.includes("i"));

console.log(withI); // ["banaani", "kirsikka", "appelsiini"]
```

- Käytämme myöhemmin callback-funktioita osana *asynkronista koodia*.

# Käsitteet 2



## Renderöinti

- Renderöinti (*engl. rendering*) tarkoittaa prosessia, jossa raakadata (kuten skriptikoodi) muutetaan visuaaliseksi esitykseksi näytöllä.
- Renderöinti on siis se vaihe, jossa sovelluksen data ja logiikka muutetaan visuaaliseksi. Siksi on yleistä sisällyttää niiden funktioiden, jotka vastaavat asioiden esittämisestä, nimeen sana *render*.

# DOM-manipulaatio 1



## `getElementById(id)`

- Hakee dokumentista sen elementin, jonka id-attribuutti vastaa annettua merkkijonoa. Palauttaa ensimmäisen löytyvän elementin tai null jos sellaista ei ole

# DOM-manipulaatio 2



**`createElement(tagName)`**

- Luo uuden HTML-elementin annetuilla tag-nimellä (esim. "li", "button"), mutta ei vielä sijoita sitä dokumenttiin.

# DOM-manipulaatio 3



## `appendChild(node)`

- Lisää annetun DOM-solmun (node) vanhemman loppuun lapsisolmuksi.  
Käytetään esim. `<ul>`-listaan uuden `<li>`-elementin lisäämiseen.

# DOM-manipulaatio 4



## `querySelectorAll(selector)`

- Palauttaa staattisen NodeList-kokoelman kaikista lapsista, jotka vastaavat **CSS-valitsinta**. Esim. kaikki elementit, joiden luokka on *.animals*
- NodeList on array-like, mutta ei täysi taulukko
  - Sisältää length-ominaisuuden ja voit hakea alkioita `item(index)` tai indeksoinnilla `[index]`.
- Iterointi onnistuu:
  - `forEach`-metodilla (nykyselaimet)
  - `for...of`-silmukalla
  - Muuntamalla taulukoksi: `Array.from(nodeList)` tai spread-operaattorilla `[...nodeList]`

# DOM-manipulaatio 5



**`addEventListener(event, function)`**

- Rekisteröi tapahtumakuuntelijan tietylle tapahtumatyypille ("click", "input", "change" jne.) kutsujassa, kutsuu funktiota. Mahdollistaa dynaamisen reagoinnin käyttäjän toimintaan.

# DOM-manipulaatio 6



## **innerHTML**

- HTML-elementin ominaisuus, jolla voi lukemalla hakea ja kirjoittamalla asettaa elementin sisällä olevan HTML-koodin (myös teksti ja tagit).

# DOM-manipulaatio 7



## **textContent**

- Ominaisuus, jolla luet tai asetat elementin pelkän tekstisisällön (HTML-tageja ei tulkita).
- Turvallinen tapa näyttää käyttäjästä peräisin olevaa tekstiä.

# DOM-manipulaatio 8



## **classList**

- Ominaisuus (DOMTokenList), jonka kautta hallitset elementin CSS-luokkia.
- Sisältää metodit kuten *add()*, *remove()*, *toggle()* ja *contains()*.

# DOM-manipulaatio 9



**add()**

- Yleisimmin käytetty **element.classList.add("luokka")**-kontekstissa: lisää HTML-elementille CSS-luokan.

# DOM-manipulaatio 10



## `preventDefault()`

- Tapahtumakäsittelijässä (***event***) kutsuttu metodi, joka estää selaimen oletustoiminnon esim. linkin klikkauksen, lomakkeen lähetyksen.

# DOM-manipulaatio 10



## event

- Sisältää tiedot tapahtuman tyypistä (**event.type**), lähteestä (**event.target**), ajankohdasta ja muista kontekstisidonnaisista tiedoista.
- Eli sen avulla saat selville *mitä tapahtui, missä, milloin* ja voit hallita tapahtuman kulkua.
- **event** välitetään callback-funktiolle aina, kun tapahtuma käynnistyy.
- Ilman event-oliota emme voisi tehdä juurikaan dynaamista reagointia selaimen tapahtumiin.
- Voit nimetä parametrin vapaasti (esim. e tai evt), mutta yleensä kutsutaan event tai lyhennettynä e.

# Taulukko ja -iteraatiometodit 1



## **filter(callback)**

- Se käy läpi alkuperäisen taulukon alkion kerrallaan:
  - Jokaiselle alkiollese kutsuu annettua *callback*-funktiota.
  - Jos *callback(element)* palauttaa **true**, alkio sisällytetään uuteen taulukkoon.
  - Jos se palauttaa **false**, alkio ohitetaan.
- Lopputuloksena saat uuden taulukon, jossa on vain ne alkiot, jotka “läpäisivät” testin.
- Esimerkki



# Taulukko ja -iteraatiometodit 1



**filter(callback)**

```
const numbers = [1, 2, 3, 4, 5];  
  
// Haluamme uuden taulukon, jossa on vain parilliset numerot:  
const even = numbers.filter(n => n % 2 === 0);  
  
console.log(even); // [2, 4]
```

# Taulukko ja -iteraatiometodit 2



## `forEach(callback)`

- Taulukon metodi, joka kutsuu jokaiselle taulukon alkiolle annettua callback-funktiota.
- Se on tarkoitettu tilanteisiin, joissa haluat tehdä sivuvaikutuksia, kuten DOM-elementtien luontia, konsoliin tulostusta, muuttujien päivitystä, etkä tarvitse uutta taulukkoa.
- Palauttaa **undefined**, joten sitä ei voi ketjuttaa.

# Taulukko ja -iteraatiometodit 3



**push(element)**

- Lisää uuden alkion taulukon **loppuun**.


# Taulukko ja -iteraatiometodit 3



**sort(compareFunction)**

- Järjestää taulukon alkiot käyttäen annettua vertailevaa funktiota.
- Jos taulukon **alkioina on olioita**, **sort()-metodille pitää määritellä miten kahta olioita tulisi vertailla**

```
animals.sort((a, b) => a.name.localeCompare(b.name, "fi"));
```

*Tästä lisää* 

# Merkkijono- ja muokkausmetodit 1

**localeCompare(anotherString, locale)**

- Vertailee merkkijonoja kielikohtaisesti; palauttaa negatiivisen luvun, jos kutsujamerkkijono tulee ennen toista, positiivisen jos jälkeen, ja 0 jos ne ovat identtiset.
- Toimii suomalaisten erikoismerkkien (Ä, Ö) kanssa, kun **locale** arvoksi annetaan “fi”.

# Merkkijono- ja muokkausmetodit 2

## `toLowerCase()`

- Muuttaa merkkijonon kaikki kirjaimet pieniksi. Hyödyllinen case-insensitive-hausta varten: verrattaessa ei tarvitse huomioda isoja ja pieniä kirjaimia erikseen.

# Merkkijono- ja muokkausmetodit 3



## `parseInt()`

- Muuntaa merkkijonon kokonaisluvuksi Jos merkkijono ei ala numerolla, palauttaa NaN.
- Tässä tehtävässä poistonapin data-index-attribuutin merkkijono muutetaan luvuksi, jotta sen avulla voidaan poistaa kyseisen indeksin alkio.

# Merkkijono- ja muokkausmetodit 4

`trim()`

- Poistaa merkkijonon alusta ja lopusta kaikki whitespace-merkit (välilyönnit, rivinvaihdot jne.). Auttaa varmistamaan, ettei tyhjästä merkeistä tule virheitä esim. hakutermissä tai syötekentässä.

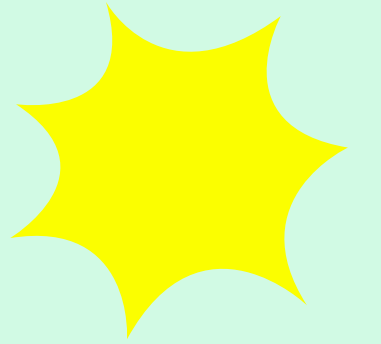
# Merkkijono- ja muokkausmetodit 5



## `includes()`

- Merkkijonometodina: tarkistaa, sisältääkö merkkijono annetun alimerkkijonon.
- Taulukkometodina: tarkistaa, löytyykö taulukosta tietty arvo.
- Palauttaa **true** tai **false**

# Miksi lista tulee tyhjentää?



**Tehtävässä pyydetään tyhjentämään lista ennen päivitystä -miksi?**

- Listan tyhjentäminen ennen uuden sisällön lisäämistä:
  - Estää kaksoiskappaleet ja virheelliset listaukset.
  - Pitää käyttäjäkokemuksen selkeänä.
  - Parantaa DOM:n suorituskykyä ja hallittavuutta.
  - Pitää huolen, että aina on juuri oikea määrä ja oikeat eläimet näkyvissä.

# Sovella!

Pikku eläintarha tehtävä löytyy GitHubista:

[https://github.com/bc-web-ohjelmistokehitys/WP25K-JS/tree/main/07\\_viikko/pikku\\_el%C3%A4intarha](https://github.com/bc-web-ohjelmistokehitys/WP25K-JS/tree/main/07_viikko/pikku_el%C3%A4intarha)

Pikku eläintarhan HTML-pohja on jo valmiina, puuttuu “vain” JavaScript ja CSS.