

OHJELMOINNIN PERUSTEET

JAVASCRIPT TAULUKOT JATKUU



JS

NÄISSÄ DIOSSA:

- Alkioiden järjestäminen **.sort()** ja **.reverse()**
- Taulukon iterointi **.forEach()**, **.map()**
- Alkioiden tarkistus **.some()**, **.every()**
- Sisällön suodattaminen **.filter()**
- Spread-operaattori ...
- Muokkaaminen vai uusi taulukko?

JS

ALKIOIDEN JÄRJESTÄMINEN: AAKKOSJÄRJESTYS



.sort()-metodi järjestää taulukon numero- tai aakkosjärjestykseen.

Aakkosjärjestys on helppo.

```
let hedelmat = ["Banaani", "Appelsiini", "Omena", "Mango"];
hedelmat.sort(); // Tuloksena "Appelsiini", "Banaani", "Mango", "Omena"
```

Numerojärjestys onkin hieman hankalampi, sillä numerotkin järjestetään oletuksena aakkosjärjestykseen, jolloin esimerkiksi "4" on suurempi kuin "10".

JS

ALKIOIDEN JÄRJESTÄMINEN: PIENIMMÄSTÄ SUURIMPAAN



Onneksi voimme antaa parametrina funktion, minkä perusteella alkioita verrataan pareittain. Jos tuloksena on positiivinen luku, ensimmäinen alkio (a) kuuluu toisen alkion (b) jälkeen. Muuten jälkimmäistä ennen.

Esimerkiksi tämä järjestää taulukon pienimmästä luvusta suurimpaan.

```
let pisteet = [40, 100, 1, 5, 25, 10];  
  
pisteet.sort(function(a, b){return a - b}); //Tuloksena 1,5,10,25,40,100
```

JS

ALKIOIDEN JÄRJESTÄMINEN: SUURIMMASTA PIENIMPÄÄN



Ja tämä puolestaan suurimmasta pienimpään.

```
let pisteet = [40, 100, 1, 5, 25, 10];  
  
pisteet.sort(function(a, b){return b - a}); //Tuloksena 100,40,25,10,5,1
```

Tällä tavalla voi myös löytää helposti taulukon suurimman (tai pienimmän) arvon.

```
console.log("Suurin arvo on " + pisteet[0]);
```

JS

ALKIOIDEN JÄRJESTÄMINEN: SATTUMANVARAISESTI

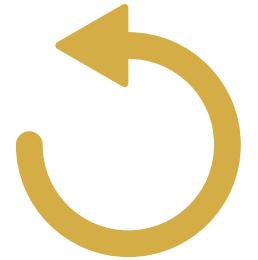


.sort()-metodia voi siis käyttää myös taulukon sekoittamiseen satunnaiseen järjestykseen. Käytössä **Math.random()** funktiomäärittely

```
let pisteet = [40, 100, 1, 5, 25, 10];  
  
pisteet.sort(function(a, b){return 0.5 - Math.random()}); // Tuloksena luvut satunnaisesti
```

JS

ALKIOIDEN JÄRJESTÄMINEN: KÄÄNTEISESTI



.reverse() -metodia käytetään silloin, kun halutaan muuttaa nykyisen taulukon alkiot käänteiseen järjestykseen.

```
let taulukko = ["banaani", "kiivi", "selleri"];
taulukko.reverse(); // Tuloksena "selleri", "kiivi", "banaani"
```

JS

TAULUKON ITEROIMINEN*

***Iterointi** tarkoittaa jonkin asian läpikäymistä vaiheittain, yleensä toistamalla samaa toimintoa jokaiselle alkioille joukossa.

.**forEach()** suorittaa **funktion** jokaiselle taulukon elementille.

```
const numerot = [1, 2, 3, 2, 1];

numerot.forEach((alkio) => console.log(alkio * alkio));
// Tulostaa jokaisen numeron kerrottuna itsellään
```

Ei ole sattumaa, että metodin alussa on **silmukoista tuttu for** - tämä metodi toistuu kuin silmukka jokaisen alkion kohdalla.

JS

ITEROINTI UUTEEN TAULUKKOON: MAP()

.map() Luo uuden taulukon ja käsittelee jokaisen elementti funktion avulla. Alla luodaan uusi taulukko, jossa on numerot-taulukon kaikki arvot kahdella kerrottuna.

```
const numerot = [1, 2, 3, 2, 1];

const tuplattu = numerot.map((alkio) => alkio * 2);
console.log(tuplattu); // [2, 4, 6, 4, 2]
```

JS

MITÄ EROA: FOREACH() VS MAP()

.forEach() suorittaa funktion jokaiselle alkioille.

.map() suorittaa funktion jokaiselle alkioille ja luo uuden taulukon tuloksista.

.forEach() palauttaa ***undefined***, joten **ei voi ketjuttaa** muiden metodien kanssa.

.map() palauttaa uuden taulukon, joten **voi ketjuttaa** esimerkiksi **.sort()** kanssa.

.forEach() riippuen käytetystä funktiosta saattaa muokata alkuperäistä taulukkoa.

.map() ei koskaan muokkaa alkuperäistä taulukkoa.

JS

SISÄLLÖN TARKISTUS

.some() tarkistaa, läpäisevätkö **jotkut** alkiot ehdon. Palauttaa **true** tai **false**

```
const taulukko = [1, 2, 3, 4, 5];

// Tarkista, sisältääkö taulukko alkioita, jotka ovat suurempia kuin 3
const onkoSuurempiaKuinKolme = taulukko.some((alkio) => alkio > 3);

console.log(onkoSuurempiaKuinKolme); // Tuloste: true
```

.every() tarkistaa, läpäisevätkö **kaikki** alkiot ehdon. Palauttaa **true** tai **false**

```
// Tarkista, ovatko kaikki taulukon alkiot alle kuusi
const ovatkoKaikkiAlleKuusi = taulukko.every((alkio) => alkio < 6);

console.log(ovatkoKaikkiAlleKuusi); // Tuloste: true
```

JS

SISÄLLÖN SUODATTAMINEN

.filter() palauttaa kaikki ehtoa täyttävät alkiot uutena taulukkona.

```
const numerot = [1, 2, 3, 2, 1];  
  
const kaikkiYliYhden = numerot.filter((n) => n > 1); // [2, 3, 2]
```

JS

SPREAD-OPERAATTORI: TAULUKOIDEN YHDISTÄMINEN

... eli kolmea pistettä kutsutaan spread-operaattoriksi. Sillä on käytöä muuallakin, mutta taulukoissa sitä voi käyttää varsinkin kahteen asiaan.

Ensimmäinen on taulukoiden yhdistäminen uuteen taulukkoon. Kolme pistettä laitetaan alkuperäisen taulukon nimen eteen.

```
const kvartaali1 = ["tammikuu", "helmikuu", "maaliskuu"];
const kvartaali2 = ["huhtikuu", "toukokuu", "kesäkuu"];
const kvartaali3 = ["heinäkuu", "elokuu", "syyskuu"];
const kvartaali4 = ["lokakuu", "marraskuu", "joulukuu"];

const vuosi = [...kvartaali1, ...kvartaali2, ...kvartaali3, ...kvartaali4];
console.log(vuosi.join(", "));
/* Tuloste: tammikuu, helmikuu, maaliskuu, huhtikuu, toukokuu, kesäkuu, heinäkuu, elokuu, syyskuu, lokakuu, marraskuu, joulukuu */
```

JS

SPREAD-OPERAATTORI: TAULUKON KOPIOIMINEN

Toinen käyttötarkoitus on taulukon kopioiminen. Näin luodun kopion arvot eivät muudu, vaikka alkuperäisen taulukon arvot muuttuvat.

```
const numerot = [1, 2, 3];
let kopio1 = numerot;
let kopio2 = [...numerot];
console.log(numerot); // [ 1, 2, 3 ]
console.log("kopio 1:", kopio1); // [ 1, 2, 3 ]
console.log("kopio 2:", kopio2); // [ 1, 2, 3 ]

numerot[0] = 0;

console.log(numerot); // [ 0, 2, 3 ]
console.log("kopio 1:", kopio1); // [ 0, 2, 3 ]
console.log("kopio 2:", kopio2); // [ 1, 2, 3 ]
```



JS

MUOKKAAMINEN VAI UUSI TAULUKKO?

Osa metodeista muokkaa alkuperäistä taulukkoa ja osa ei.

Osan 1 dioissa esitellyistä metodeista melkein kaikki ovat alkuperäistä taulukkoa muuttavia (*eng mutating*).

Ei-muuttavat (*eng non-mutating*) metodit eivät muuta alkuperäistä taulukkoa, vaan palauttavat jotain muuta, joko totuusarvon, numeron tai jos palauttavat taulukon, palauttavat uuden version.

Ohjelmaa tehdessä on tärkeää miettiä kannattaako alkuperäistä taulukkoa muokata vai täytyykö se säilyttää sellaisenaan. Usein halutaan suojella alkuperäistä versiota, koska sitä voidaan käyttää muuallakin.

Muuttavia metodeja
ovat muun muassa:

- .pop()**
- .push()**
- .reverse()**
- .shift()**
- .unshift()**
- .sort()**
- .splice()**

Ei-muuttavia metodeja
ovat muun muassa:

- .length**
- .join()**
- .indexOf()**
- .lastIndexOf()**
- .map()**
- .some()**
- .every()**
- .filter()**
- .forEach()**

+

... eli spread-operattori

JS

EIKÄ SIINÄ VIELÄ KAIKKI

Näissä dioissa on käyty läpi vain osa taulukkometodeista.

Voit tutustua muihin taulukkometodeihin esimerkiksi

- [W3Schools sivulla](#)
- [Mozillan dokumentaatiossa](#)

