Oppimistehtävä 6

Tee seuraavat tehtävät. Pakkaa tiedostot yhteen .zip –pakettiin ja palauta Moodlen kautta. Kirjoita jokaisen tehtävän ratkaisu niille tarkotettuihin tiedostoihin (ot6a.js, ot6b.js, ot6c.js ja ot6d.js) kommenteilla merkatuihin alueihin. Älä muuta muita koodeja!

Tämän tehtäväsarjan tehtävät koskevat DOMin hallintaa, mikä kuuluu jo edistyneempää JavaScript ohjelmointiin. Tehtäväsarjan aiheita käsitellään javascript.info:n TOISESSA OSIOSSA (Browser: Documents, Events, Interfaces) kappaleissa 1.1. - 4.4. Käytä näitä kappaleita materiaalina tehtävien tekemiseen, mutta huomaa että kappaleet käsittelevät aihetta niin laajasti, että kaikkea tietoa et tarvitse tehtävien tekemiseen. Löydät hyviä ohjeita myös w3school:in sivuilta https://www.w3schools.com/js/js_htmldom.asp. Voit myös käyttää allaolevaa cheatsheettia tärkeimpien pointtien kertaamiseen.

Mikäli haluat syventyä vielä lisää JavaScript-ohjelmointiin, voit tutustua tehtäväsarjan mukana tulevaan ot6.html:ään nähdäksesi mihin HTML-elementteihin tehtävissä kytkeydytään, ja miten ohjelma toimii kokonaisuutena. Onnea viimeisiin tehtäviin!

Cheatsheet

Selaimet tarjoavat JavaScript -ympäristöön objektin nimeltä "document". Tätä käyttämällä voi aloittaa DOMin hallinnan. Tässä on joitakin documentin tärkeimpiä ominaisuuksia:

Elementtien haku DOMista:

```
const tekstiKenttaElem = document.getElementById('tekstikentta');
```

```
consts luokkaElems = document.getElementsByClassName('luokanNimi');
```

```
const kappaleElems = document.getElementsByTagName('p');
```

Ylläolevat esimerkit hakevat DOMista tiettyjä elementtejä:

- getElementById hakee ensimmäisen elementin, jonka id-attribuutin arvo on 'tekstikentta'.
- getElementsByClassName hakee elementit, joiden class-attribuutti sisältää luokan nimeltä 'luokanNimi'.

- getElementsByTagName hakee kaikki elementit, joiden tagi on 'p', eli HTML:ssä kirjoitettuna esim.

Arvojen luku DOMista

Joihinkin DOM-elementteihin liittyy käyttäjäinteraktioita, eli käyttäjä pystyy vuorovaikuttamaan elementin kanssa. Esimerkki tällaisista elementeistä ovat input-elementit, jotka ovat yleisimpiä elementtejä käyttäjän syötteiden antamiseen. Input-elementeillä on value-kenttä, josta saadaan kenttään kirjoitettu arvo string-muotoisena, esim:

```
const tekstiKentta = document.getElementById('tekstikentta');
const kentanArvo = tekstiKentta.value;
console.log(kentanArvo);
```

DOM elementtien lisääminen / muuttaminen

Jos verkkosivuilla olevia elementtejä pitää muuttaa, se onnistuu muutamallakin eri tavalla:

```
const container = document.getElementById('container');
const kappale = document.createElement('p');
container.appendChild(kappale);
```

Ylläoleva esimerkki hakee ensin container-nimisen elementin DOMista, sitten luo p-elementin, ja lopuksi asettaa luodun p-elementin container-elementin sisälle. Sama onnistuu helpommin myös näin:

```
document.getElementById('container').innerHTML = '<p></p>';
```

Ylläolevassa esimerkissä on syytä huomata, että innerHTML:n asettaminen muuttaa koko containerin alla olevan DOM-puun, joten esimerkissä containerin alle jää vain ja ainoastaan . Sen sijaan appendChild-funktiolla containerin sisään voidaan lisätä uusia elementtejä olemassaolevien lisäksi.

Mikäli elementtiin ei tarvitse lisätä muuta kuin tekstiä, voit myös käyttää seuraavaa syntaksia:

```
document.getElementById('tekstikentta').innerText = 'Tekstikentän sisältö';
```

Ylläoleva esimerkki lisää pelkkää tekstiä haetun elementin sisälle, mikä on suositeltava tapa muuttaa elementin sisältöä, jos sisältö on pelkkää tekstiä.

Eventtien kuuntelu

DOM laukaisee jatkuvasti monenlaisia "eventtejä". Eventit ovat kömpelösti suomennettuna tapahtumia, joihin halutessa voidaan reagoida. Esimerkkejä tällaisista tapahtumista on mm. hiiren klikkaus, tekstikentän arvon muuttuminen, näppäimistön painallus tms. Eventtejä täytyy ns. "kuunnella", jotta tällaisiin tapahtumiin voidaan reagoida. DOMin elementeille voidaan antaa kuuntelijoita (listener), jotka seuraavat jonkin tietyn eventin toteutumista, ja kun kyseinen elementti laukaisee kyseisen eventin, kutsutaan koodaajan määritteleмää funktiota. Sanotaan esim. että meillä on nappi, jonka id on 'nappula'. Napin painalluksesta voidaan kutsua funktiota seuraavalla tavoin:

```
document.getElementById('nappula').addEventListener('click', () => {
  console.log('Nappia painettiin');
});
```

Ylläolevassa esimerkissä siis haetaan nappi id:llä 'nappula', ja kyseiselle napille asetetaan event listener eventille 'click', jolloin toisena parametrina annettua funktiota kutsutaan vain silloin kun nappi laukaisee 'click' eventin, eli kun sitä klikataan.

Tässä on esimerkki, miten saadaan käyttäjän syöte otettua koodissa vastaan:

```
const tekstikentta = document.getElementById('tekstikentta');
tekstikentta.addEventListener('input', event => {
  const tekstikentanArvo = event.target.value; //Myös tekstikentta.value toimii
  console.log(tekstikentanArvo);
});
```

Ylläoleva esimerkki kutsuu funktiota joka kerta kun tekstikenttään kirjoitetaan jotakin ja 'input' -event laukaistaan. Vaihtoehtona input-eventille voitaisiin käyttää esim. 'change' -eventtiä, joka laukaistaan sillon kun tekstikenttää on muutettu ja käyttäjä kohdistaa kursorin johonkin muualle.

Saat myös eventin kaikista hiiren liikkeistä 'mousemove' eventillä:

```
document.getElementById('container').addEventListener('mousemove', e => {
  console.log('Hiiren sijainti: ', e.offsetX, e.offsetY);
});
```

Huomaa, että event listenerejä voidaan rekisteröidä myös HTML:n puolella, esim:

```
<button onclick="painaNappia()">Paina</button>
```

Tällöin oletetaan, että skripteihin on luotu painaNappia -funktio, jota kutsutuaan napin painalluksesta. Tämä on tavanomaisempi tapa rekisteröidä event listenerejä, mutta jotta sinun ei tarvitsisi koskea tehtävän html-tiedostoon, voit luoda kaikki event listenerit addEventListener -funktiolla.

Attribuuttien lisääminen

Voit asettaa DOM elementeille attribuutteja muuttamalla saman nimisen kentä arvoa haetusta elementistä. Esim. hidden-attribuutti piilottaa kyseisen elementin näkyvistä:

```
const paneeli = document.getElementById('piilopaneeli');
paneeli.hidden = true;
```

Ylläoleva esimerkki piilottaa paneelin näkyvistä.

Tehtävät

a)

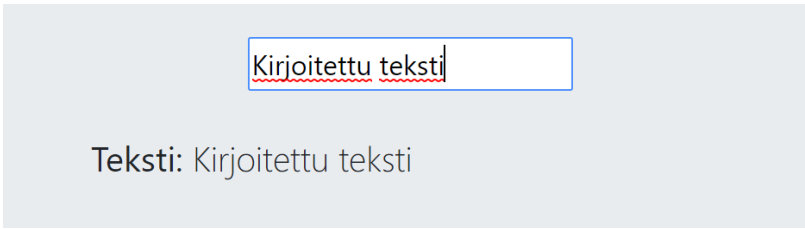
Tee ohjelma, joka ottaa käyttäjän syötteen vastaan ja kirjoittaa sen DOMiin.

- Käyttäjän syöte-kentän id: 'teksti-input'

- Tulostus-kentän id: 'teksti-output'

Vinkki:

- Saat elementit haettua getElementById -funktiolla
- Saat arvon luettua elementin value-kentästä
- Saat arvon näkyviin asettamalla sen elementin innerText-kentän arvoksi



b)

Tee ohjelma, joka näyttää vain välilehdeltä valitun sivun ja merkitsee välilehden aktiiviseksi.

Ohjelmassa on kolme välilehteä ja niiden sisältämät kolme sivua. Välilehtien ja sivujen id:t ovat seuraavat:

- Koti-välilehtinapin id: 'koti-tab'
- Yhteystiedot-välilehtinapin id: 'yhteystiedot-tab'
- Lomake-välilehtinapin id: 'lomake-tab'
- Koti-välilehtisivun id: 'koti-page'
- Yhteystiedot-välilehtisivun id: 'yhteystiedot-page'
- Lomake-välilehtisivun id: 'lomake-page'

Sinun tulee näyttää vain haluttu sivu (XXX-page) ja piilottaa muut sivut, kun käyttäjä on painanut jotakin välilehtinappia (XXX-tab).

Saat välilehtinapin visuaalisesti aktiiviseksi, kun lisäät sen luokkaan 'active'. Tämä onnistuu esim:

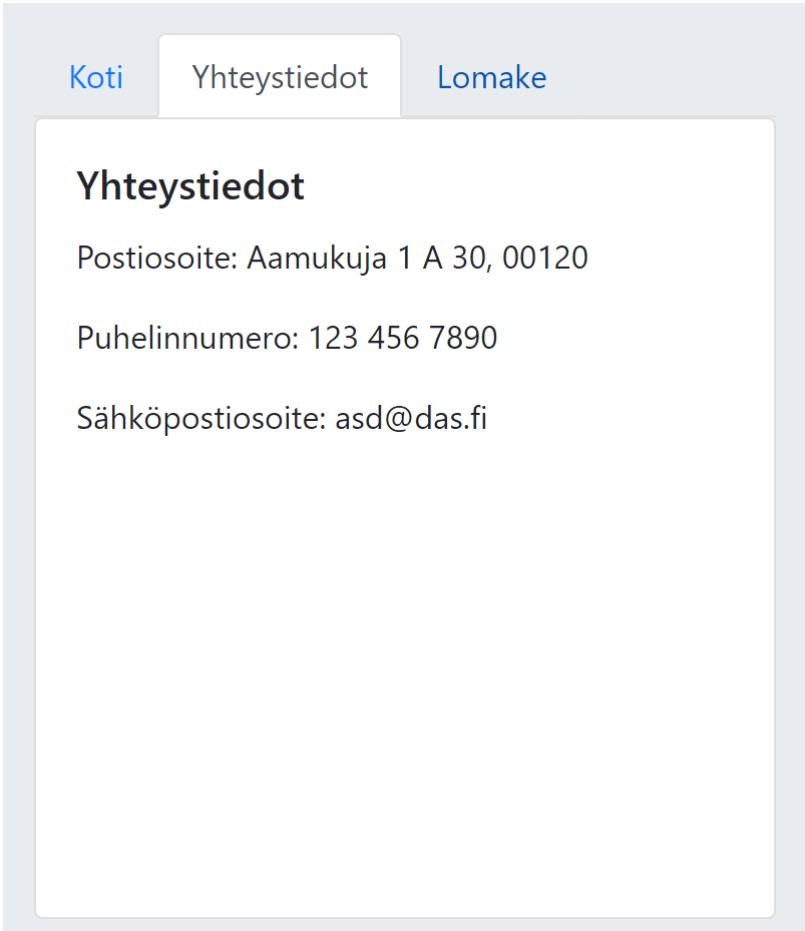
```
document.getElementById('koti-tab').classList.add('active');
```

Vastaavasti saman luokan voi poistaa näin:

```
document.getElementById('koti-tab').classList.remove('active');
```

Vinkki:

- Piilota ei-toivotut sivut käyttämällä hidden-attribuuttia
- Voit halutessasi käyttää tauluja elementtien läpikäyntiin, siten että vaikka tabeja olisi lukemattomia määriä, ohjelma toimisi ilman lisäkoodausta



c)

Toteuta tehtävässä olevaan kuvaan parallax-efekti: Liikuta kuvaa hiukan siihen suuntaan missä hiiri on. Kun hiiri on kuvan keskellä, on myös kuva keskitettynä. Kun hiiri on kuvan vasemmassa yläreunassa, on kuva hieman (esim. 20% keskipisteestä kulmaan olevasta vektorista) vasempaan yläreunaan vietynä jne. Tarkoitus siis on, että kuva liikkuu hiiren mukana, mutta vain hieman. Lisäksi kuvan tarvitsee liikkua ainoastaan, kun hiirtä liikutetaan kuvan päällä.

Tarvittavat elementit:

- Kuvan containerin id: 'parallax-container'
- Kuvan oma id: 'parallax-image'

Vinkki:

- Voit käyttää mousemove -eventtiä kuvan sijainnin päivittämiseen. Aseta tämä listeneri containerille.
- Saat containerin leveyden ja korkeuden offsetWidth ja offsetHeight -kentistä
- Saat hiiren sijainnin listenerin eventistä: e => e.offsetX ja e.offsetY -kentistä
- Saat kuvan sijainnin asetettua käyttämällä style.top ja style.left -kenttiä. Esim:

```
document.getElementById('parallax-image').style.top = '200px';
```

- Huomaa, että voit joutua antamaan top- ja left-kenttiin arvon merkkijonomuodossa, johon lisätty loppuu 'px'!
- Käytännössä tehtävässä on container, jonka koko on n. 400x400 pikseliä. Containerin sisällä on kuva, jonka koko on 727x648 pikseliä. Toivottu toiminnallisuus olisi sellainen, missä kuva liikkuu hiiren mukana sen verran vähän, että se peittää koko ajan containerin 400x400 alueen.

- Jos eventtien logiikka ei tunnu avautuvan, voit logittaa eri arvoja console.log-funktiolla, ja tarkastella niitä selaimesi kehittäjän työkaluista. Huomaa että tehtävän ei tarvitse olla täydellinen, jotta se hyväksytään!

d)

Tee ohjelma, jolla voi luoda ja poistaa postauksia.

Alkutilassa ohjelma näyttää vain "Tee postaus"-napin.

Kun käyttäjä painaa napista, se piilotetaan, ja tilalla näytetään "Tee postaus" -paneeli, mihin käyttäjä voi kirjoittaa otsikon ja postauksen. Jos käyttäjä tallentaa postauksen, luodaan siitä postikortti, joka näytetään postaus containerissa. Jos käyttäjä painaa "Peruuta", ei uutta postausta luoda. Sekä "Tallenna", että "Peruuta" -nappien painalluksen jälkeen "Tee postaus" -paneeli suljetaan, ja "Tee postaus" -nappi näytetään jälleen.

Kun "Tee postaus" -paneeli suljetaan, sen kenttien arvot tyhjennetään, joten aina kun uutta postausta aletaan tehdä, kentät ovat valmiiksi tyhjiä.

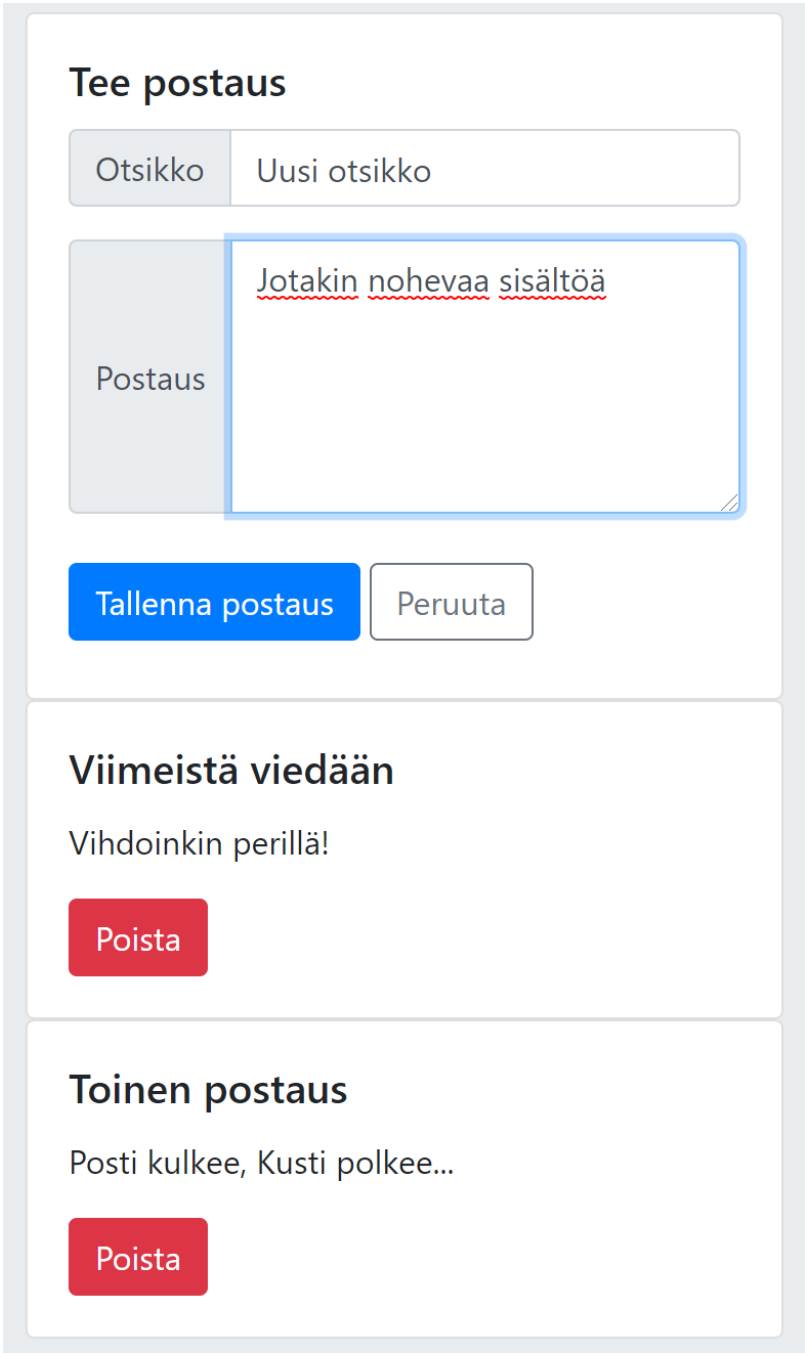
Jos käyttäjä painaa postikortissa olevaa "Poista" -näppäintä, poistetaan kyseinen postaus listasta.

Tehtävässä tarvittavien elementtien id:t ovat:

- "Tee postaus" -napin id: 'lisaa-postaus-button'
- "Tee postaus" -paneelin id: 'lisaa-postaus-panel'
- "Tallenna postaus" -napin id: 'tallenna-postaus-button'
- "Peruuta postaus" -napin id: 'peruuta-postaus-button'
- Otsikko -tekstikentän id: 'otsikko-input'
- Postaus -tekstikentän id: 'postaus-input'
- Postaus -containerin id: 'postaus-container'

Vinkki:

- Luo jokaisen uuden postauksen yhteydessä uusi objekti, joka tallennetaan tauluun.
- Voit käyttää array-funktioita taulukon käsittelyyn ja yhdistää postaus-objektit HTML:ksi (käytä getPostCard-funktiota)
- Käytä postaus-containerin innerHTML -kenttää lisätäksesi postikortteja sivulle



Palautuksen tila

Suorituskerran numero	Tämä on suorituskerta 1
Palautuksen tila	Ei suorituskertoja
Arvioinnin tila	Ei arvioitu
Viimeksi muokattu	-
Palautuksen lisätiedot	▸ Kommentit (0)

Lisää palautus

Muokkaa palautustasi

NAVIGOINTI

Työpöytä
- Sivuston etusivu
- Omat opintojaksoni
- Opintojaksokategoriat
- Moodle-alustan tilaus (Opettajille)

ASETUKSET

- Opintojakson ylläpito

Oppaat- ja ohjeet - Kysy eTuutorilta - Moodle-tuki