Oppimistehtävä 2

Ohjausrakenteet

Tee seuraavat tehtävät. Pakkaa tiedostot yhteen .zip –pakettiin ja palauta Moodlen kautta. Kirjoita jokaisen tehtävän ratkaisu niille tarkotettuihin tiedostoihin (ot2a.js, ot2b.js, ot2c.js ja ot2d.js) kommenteilla merkattujen alueiden väliin. Älä muuta muita koodeja!

Tämän tehtäväsarjan tehtävät koskevat javascript.info:n kappaleita 2.10 - 2.13. Käytä näitä kappaleita materiaalina tehtävien tekemiseen. Voit myös käyttää allaolevaa cheatsheettia tärkeimpien pointtien kertaamiseen. Lisäksi voit tutustua aiheeseen esimerkkien avulla tutustumalla oppimistehtävän yhteydessä olevaan ot2_demot-tiedostoon, minkä sisällä olevien html-tiedostojen lähdekoodista löydät aiheeseen liittyviä skriptejä.

Cheatsheet

Ehtolauseet

- JavaScriptin ehtolauseilla voidaan vertailla eri arvoja. Vertailun todenmukaisuuden perusteella lausekke on joko tosi (boolean true) tai epätosi (boolean false). Esimerkki ehtolauseesta: 10 === 12, jonka tulos on false, sillä lauseke ei pidä paikkaansa. Vertailuihin käytettyjä operaattoreita ovat mm.
- x === y, eli onko x yhtä kuin y
- x !== y, eli onko x eri kuin y
- x < y, eli onko x pienempi kuin y
- x > y, eli onko x suurempi kuin y
- x <= y, eli onko x pienempi TAI yhtä suuri kuin y
- x >= y, eli onko x suurempi TAI yhtä suuri kuin y
- JavaScriptissä on myös vertailuoperaattorit == ja !=, mutta näille on hyvin vähän tarpeellisia käyttökohteita. Nämä operaattorit eroavat === ja !== operaattoreista siten, että ne aiheuttavat arvoille automaattisen tyyppimuunnoksen vertailua varten, mikäli arvot ovat eri tyyppiä, joten esim. '4' == 4 on true, kun taas '4' === 4 on false. Vertailuissa pidetään hyvien käytäntöjen mukaisena käyttää tripla-merkkiä (=== ja !==), sillä tyyppimunnokset saattavat aiheuttaa arvaamattomia tuloksia joissain tilanteissa, esim. 0 == '' on true, ja false == 0 on true. Jos siis käytät joskus == ja != merkkejä koodissasi, sinun täytyy olla valmis perustelemaan ratkaisusi muulle tiimille!
- Vertailuja voidaan tehdä minkä tahansa JavaScript objektien välillä, mutta jotta tulokset olisivat järkeviä, on syytä huolehtia että vertailtavat objektit ovat samaa tyyppiä

Boolean operaattorit

- Ehtolauseita voidaan yhdistää muodostaakseen monimutkaisempia ehtoja. Yhdistämiseen käytetään kolmea boolean-operaattoria: &&, eli AND, ||, eli OR ja !, eli NOT (kts. https://javascript.info/logical-operators)
- && -operaattori palauttaa true, jos lauseen molemmat tekijät ovat true
- || -operaattori palauttaa true, jos jompi kumpi lauseen tekijöistä on true
- ! -operaattori palauttaa yhden tekijän vastakohda, eli esim. !true === false

Esimerkkejä:

```
if (ika >= 18 && ika < 65)
  console.log('Henkilö on työikäinen!);
```

```
if (ika < 18 || ika >= 65)
  console.log('Henkilö EI ole työikäinen!');
```

```
if ( !(ika >= 18 && ika < 65) )
  console.log('Henkilö EI ole työikäinen!');
```

Boolean-operaattorien avulla ehtolauseita voidaan ketjuttaa mielivaltaisesti:

```
if( hlo.ika >= 18 && hlo.ika < 65 && ( hlo.kansalaisuus === 'Suomi' || hlo.tyolupa === t
rue) && !hlo.tyokyvyton)
  console.log('Henkilö voi työskennellä Suomessa!');
```

If-, else if- ja else-lauseet

- If-lauseilla voidaan päättää suoritetaanko jokin koodi vai ei
- Päättelyyn käytetään vertailulauseita, jotka asetataan if-lauseen sulkeiden sisään:

```
if(x !== y) {
  console.log('Äks on erisuuri kuin yy');
}
```

- If-lausetta voidaan jatkaa myös else if- ja else-lauseilla. Jos else-lausetta käytetään, sen tulee aina olla viimeisenä. Else if-lauseita voi olla kuinka monta hyvänsä:

```
if( x < y) {
  console.log('X on pienempi kuin Y');
} else if (x > y) {
  console.log('X on suurempi kuin Y');
} else if (x === y) {
  console.log('X on yhtä suuri kuin Y');
} else {
  console.log('X on jotain ihan muuta kuin Y. Kenties eri tyyppiä..?');
}
```

- if-else-lauseita voidaan käyttää erilaisilla tyyleillä. Jos if-lauseen sisällä oleva koodi on vain yhden rivin mittainen, voidaan aaltosulkeet halutessa jättää pois. Seuraavat if-lauseet ovat ohjelmallisesti identtisiä

```
if ( x === y) {
  console.log('...');
}
```

```
if (x === y) { console.log('...'); }
```

```
if ( x === y)
   console.log('...');
```

```
if ( === y) console.log('...');
```

- Erilaisia tyylivaihtoehtoja kannattaa käyttää harkinnan mukaan. Ennen pitkään kannattaa tutustua JavaScriptin erilaisiin tyylioppaisiin
(esim. https://www.w3schools.com/js/js_conventions.asp)

? : -syntaksi

- Jos vertailulausetta käytetään saadakseen jokin vaihtoehtoinen arvo, voidaan käyttää ?: -syntaksia:

```
const alennus = hinta > 100 ? hinta * 0.1 : 0;
```

- Edellisessä lauseessa siis on 4 osaa:
- const alennus, jolle annetaan arvo
- hinta > 100, eli ehtolauseke, jolla tarkastetaan onko hinta yli 100
- ? hinta * 0.1, on lauseke jonka arvo asetetaan alennuksen arvoksi, JOS edellinen ehtolauseke oli tosi
- : 0, on lauseke jonka arvo asetetaan alennuksen arvoksi, JOS edellinen ehtolauseke oli epätosi
- Vastaavan lauseen voisi toteuttaa myös if-else lauseella seuraavasti:

```
let alennus;
if(hinta > 100) {
  alennus = hinta * 0.1;
} else {
  alennus = 0;
}
```

- ? : -syntaksia kuitenkin pidetään elegantimpana arvojen asettamisessa, sillä se kertoo heti että lausekkeella asetetaan vaihtoehtoinen arvo, sen sijaan kun taas if-else lauseiden sisällä voi olla mitä tahansa ohjelman toimintaa muuttavaa koodia. On siis suositeltavaa käyttää ? : -syntaksia silloin kuin se on mahdollista (järkevyyden rajoissa toki, kts https://javascript.info/ninja-code)

Truthy / Falsy

- On hyvä tietää, että monista muista ohjelmointikielistä poiketen, JavaScriptissä käytetään truthy ja falsy -käsitteitä. Tämä tarkoittaa sitä, että jos jotakin arvoa käytetään Boolean -kontekstissa, tehdään arvoille automaattinen tyyppimuunnos. Tämä mahdollistaa mielivaltaisten arvojen ja objektien käyttämisen mm. if-lauseissa.

- Käytännössä kaikki arvot ovat truthy-arvoja, paitsi falsyksi määritellyt arvot (falsy, null, undefined, 0, NaN ja ''). Tämän kanssa täytyy siis olla hyvin tarkka, sillä esim. numeerinen 0 on falsy arvo, mutta merkkijono '0' on truthy arvo. Muita truthy arvoja on mm:
- true
- {}
- new Object()
- Infinity
- 'False'
- Jos siis jokin edellämainituista arvoista olisi esim. if-lauseen ehtona, suoritettaisiin if-lause. Katso lisää aiheesta: https://developer.mozilla.org/en-US/docs/Glossary/Truthy

Switch -lause

- Switch -lause on ohjausrakenne, jolla voidaan if-else -lausetta vastaavasti valita suoritettava koodi jonkin arvon perusteella
- Switch -lause eroaa if-else lauseista siten, että jokainen vertailu tehdään samalle arvolle
- Switch -lauseen syntaksi on seuraavanlainen:

```
switch (nimi) {
  case 'Keijo':
    console.log('Nimi on Keijo');
    break;
  case 'Kaija':
    console.log('Nimi on Kaija');
    break;
  default:
    console.log('Nimi on tuntematon');
}
```

- Esimerkissä switch-lauseelle on annettu muuttuja nimeltä "nimi". Jos muuttujan arvo on 'Keijo', suoritetaan ensimmäinen koodipolku (console.log('Nimi on Keijo');), ja jos muuttujan arvo on 'Kaija' suoritetaan toinen koodipolku. Mikäli muuttujan arvo ei ole mikään caseihin listatuista arvoista, suoritetaan default-haarassa oleva koodipolku.
- Casen loppuun on tärkeää muistaa lisätä "break;"-lause, sillä se lopettaa switch-lauseen läpikäynnin. Mikäli break-lauseen jättää pois, siirrytään koodin suorittamisen jälkeen seuraavaan caseen. Se on hieman eksoottisempi käyttötapa, eikä sille ole montaa perusteltua käyttötarkoitusta. Tuleeko sinulla jokin käyttötarkoitus mieleen?
- Switch -lausetta on erityisen perusteltua käyttää if-lauseen sijaan, mikäli kaikissa ehdoissa tarkastellaan yhtäläisyyttä yhteen ja samaan arvoon, ja jos vaihtoehtoisia koodipolkuja on paljon. Tällöin muut tiimin jäsenet ymmärtävät heti, että kaikkien koodipolkujen suoritus riippuu samasta muuttujasta, eikä heidän tarvitse lukea kaikkia if- ja else if -lauseita tämän havaitsemiseksi. Seuraavat esimerkit tekevät täysin saman asian:

```
let maa;
if (maakoodi === 358) {
  maa = 'Suomi';
} else if (maakoodi === 91) {
  maa = 'Intia';
} else {
  maa = 'Muu maa';
}
```

```
let maa;
switch (maakoodi) {
  case 358:
    maa = 'Suomi';
    break;
  case 91:
    maa = 'Intia';
    break;
  default:
    maa = 'Muu maa';
}
```

- Joissain tilanteissa on siis suositeltavampaa käyttää switch-lausetta mielummin kuin if-else -lausetta, erityisesti jos caseja on huomattavan paljon. Myöhemmin kurssilla käymme kuitenkin tätäkin elegantimman tavan suorittaa koodia ja valita arvoja. Niistä lisää tauluja (Array) käsittelevässä kappaleessa.

For-, while- ja do...while -silmukat

- Silmukat, eli loopit, ovat yksi (proseduraalisen) ohjelmoinnin oleellisimmista rakenteista. Ne ovat ohjausrakenteita, jotka suorittavat niille annetun koodin useaan kertaa peräkkäin. Esimerkiksi pelimoottoreissa käytetään game-looppia, joka suorittaa koodia jokaisella silmukalla, joka kerta päivittäen ruudulle piirrettävän kuvan, laskien fysiikkalaskennat seuraavalle ruudulle, tarkastaen onko käyttäjä painanut jotakin näppäin jne. (Nice to know: yksinkertaistettuna pelien FPS (frames per second), kertoo kuinka monta kertaa sekunnissa game-loop käydään läpi)
- TÄRKEÄÄ! Loopit ovat yksi yleisimpiä tapoja saada ohjelma jumiin tai kaatumaan! Jos siis tehtäviä tehdessä sinun nettiselaimesi jumittuu, ei ole syytä huoleen. Sulje selain, jos muu ei auta, ja tarkista mikä koodissa meni pieleen. Tämä tarkoittaa yleensä sitä, että loopin EHTOLAUSE toteutuu aina, eikä ohjelma siksi lähde loopista koskaan pois.
- Ehtolauseella määritellään se, että suoritetaanko looppi (uudestaan). Se voi olla esim. "counter < suoritustenMaara". Tällaisessa loopissa on syytä huolehtia, että counter-muuttujan arvoa kasvatetaan jokaisella kierroksella, jotta se kasvaa ennen pitkää yhtä suureksi kuin suoritustenMaara.
- JavaScriptissa looppeja on mm. for -loop, while -loop ja do...while -loop. Niiden syntaksi on seuraavanlainen:

```
for ( i = 1; i <= 10; i++ ) {
  console.log(i + '. tuloste');
}
```

```
let counter = 1;
while (counter <= 10) {
  console.log(counter + '. tuloste');
  counter++;
}
```

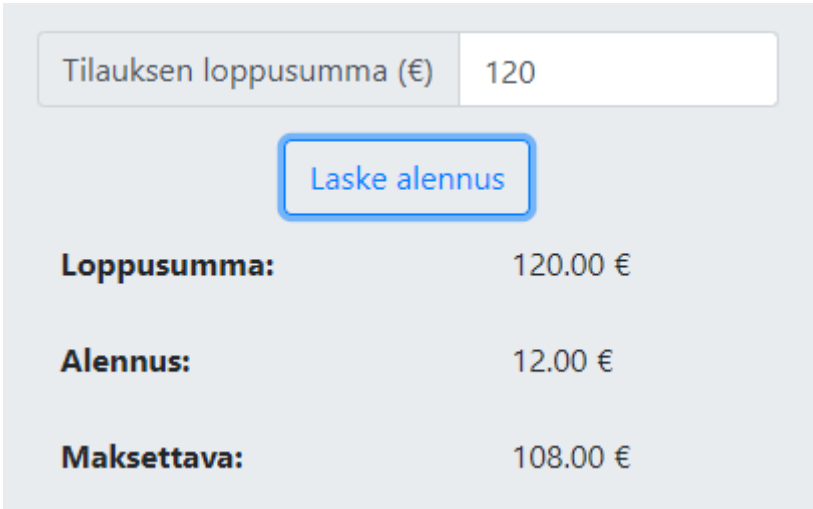
```
let counter = 1;
do {
  console.log(counter + '. tuloste');
  counter++;
} while (counter < 10);
```

- Kaikki ylläolevat esimerkit tekevät saman asian, eli tulostavat konsolille 10 kertaa "1. tuloste", "2. tuloste", jne.
- for -lauseessa on 3 osaa: Ensimmäinen osa "i = 1" on alustusosio, missä voidaan alustaa loopissa käytettävän muutujan arvo. Seuraava osio, "i < 10", on ehtolause, jonka tulee olla tosi jotta looppi suoritettaisiin (uudestaan). Viimeinen osio on "i++", joka suoritetaan loopin lopuksi, ja siinä voidaan esim. kasvattaa alustetun muuttujan arvoa. Jos alustus olisi siis esim. for(i = 0; i < 10; i--), niin muutujan i -arvoa pienennettäisiin joka kierroksella, eikä muuttujan arvo koskaan rikkoisi ehtolausetta: näin looppi jäisi ikuisesti pyörimään. Joidenkin ohjelmointikielien ja -ympäristöjen kanssa tällainen virhe saattaisi aiheuttaa koneen täydellistä jumittautumista ja crashia, mutta selainympäristössä suoritettavalla JavaScriptilla tästä ei ole pelkoa.
- while -lauseelle annetaan ainoastaa ehtolause, joten sen rikkomisesta täytyy huolehtia loopin sisällä. Esimerkissä looppia ennen on alustettu muuttuja "counter" arvolla 0, jota kasvatetaan jokaisella kierroksella yhdellä. Ennen kierroksen alkamista tarkistetaan, toteutuuko ehto, ja ehdon toteutuessa aloitetaan uusi kierros.
- do...while -lause on lähes identtinen while-lauseen kanssa, mutta erona on se, että tarkistus tehdään aina loopin päätteeksi. Tämä tarkoittaa käytännössä sitä, että do...while -lause suoritetaan AINA vähintään kerran, kun taas while-lausetta ei suoriteta kertaakaan, jos ehtolause ei ensimmäiselläkään kerralla toteudu. Jos siis "counter" alustettaisiinkin arvolla 10, niin esimerkin while-lausetta ei suoritettaisi, mutta do...while -lause suoritettaisiin kerran.

Tehtävät

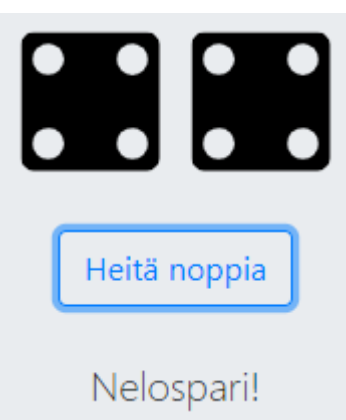
a)

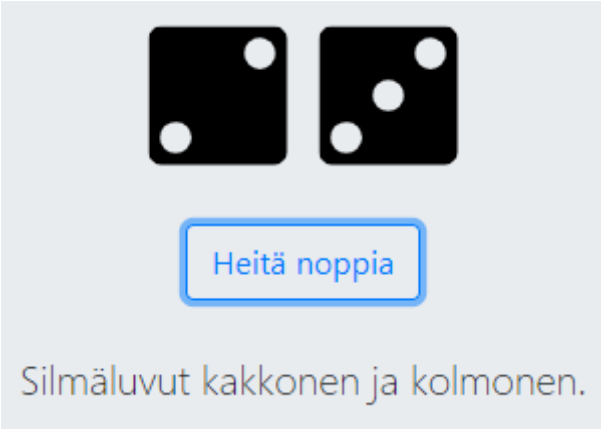
Tee ohjelma, joka laskee asiakkaan saaman alennuksen euroina, kun syöttötietona annetaan tilauksen loppusumma. Alennusta annetaan 10 prosenttia, jos tilaus on suurempi kuin 100 euroa.



b)

Tee ohjelma, joka "heittää" kaksi noppaa, kun käyttäjä painaa nappia. Mikäli silmäluvut ovat samat, ohjelman pitäisi tulostaa, mikä pari on kyseessä. Mikäli silmäluvut ovat eri, pitäisi ohjelman tulostaa ne erikseen (kts. kuva).

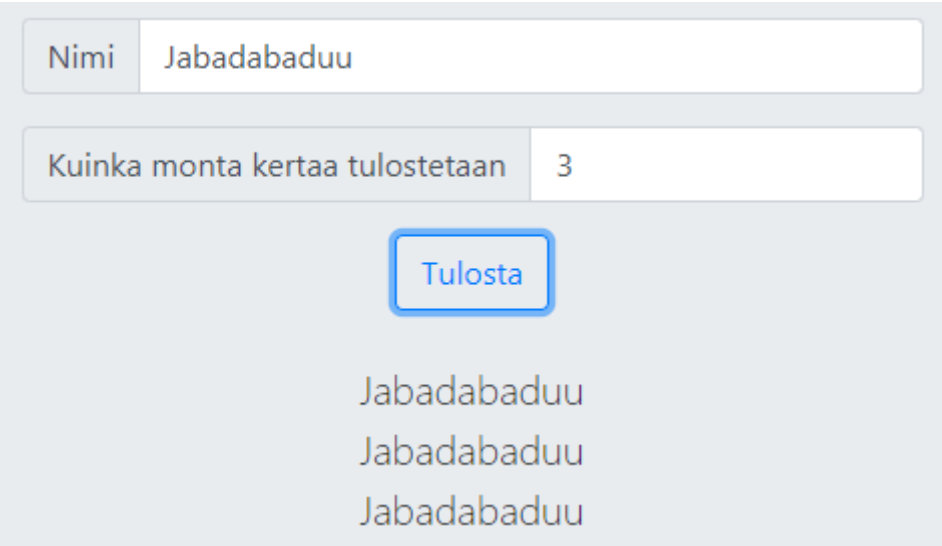




VINKKI: Katso demo_4.html

c)

Tee ohjelma, joka tulostaa ohjelman käyttäjän antaman nimen niin monta kertaa kuin hän haluaa.



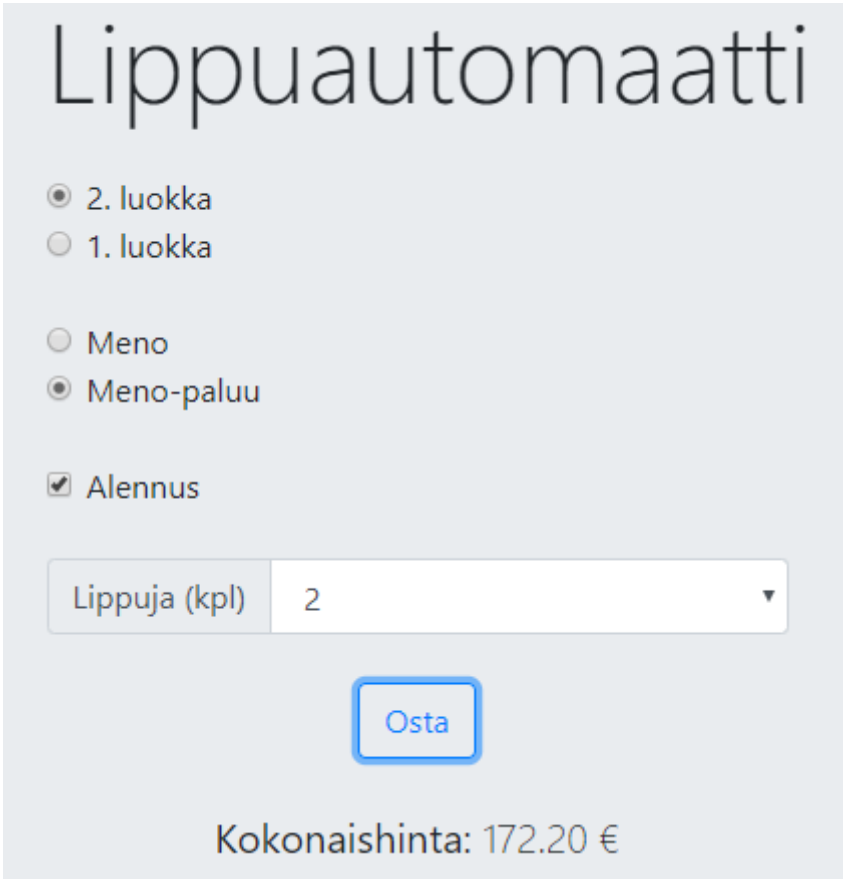
d)

Tee ohjelma Helsinki-Pietari välisen Allegro –junan lipun hinnan määrittämistä varten. Matkalipun hinta määräytyy siten, että ensimmäisen luokan menolippu maksaa 140,61 euroa, toisen luokan menolippu puolestaan 86,10 euroa. Meno-paluu –lippujen hinnat ovat kaksi kertaa menolipun hinta.

Opiskelijat ja eläkeläiset saavat toisen luokan liput puoleen hintaan. Lisäksi asiakkaan pitäisi voida ostaa kerralla monta samanlaista lippua.

Syöttötietoina annetaan siis matkustusluokka, matkustajien lukumäärä ja yhden vai kahdensuuntainen matka sekä tieto mahdollisesta alennuksesta. Tietojen syöttö pitäisi olla käyttäjälle mahdollisimman helppo. Tulostietona ohjelma tulostaa lippuostoksen kokonaishinnan.

Lomake-esimerkki ohjelmasta voit katsoa mallia.



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