a. Oliot (Object)

Muuttujien ja taulukoiden avulla pystymme tallentamaan ohjelman ajon aikana yksittäisiä ja useita tietoja. Oliot tuovat tähän oman lisänsä. Niiden avulla voi kertoa millaisesta tiedosta on kyse. Edellisessä kappaleessa tallensimme kurssi tietoja taulukkoon ja havaitsimme että se ei ole kovin selkeä tapa. Sama ongelma olisi esimerkiksi lentotietojen kanssa. Olisi hyvä kertoa mitä tieto tarkoittaa, jos samaan tietorakenteeseen tulee erilaisia tietoja. Kurssi tietojen kohdalla näitä voisivat olla esimerkiksi kurssin nimi, opintopisteet, kurssinumero. Tämä onnistuu olioissa helposti:

```
var kurssi = {    nimi: "JavaScript",
                  opintopisteet: 5.0,
                  kurssinumero: "SWDXX00X"
};
```

Nyt olion tietojäsenet eli ominaisuuksien nimet kertovat selkeästi mitä oliossa olevat tiedot tarkoittavat. Tässä onkin yksi tärkeä syy miksi aikanaan oliot kehitettiin ohjelmointikieliin. Muitakin syitä toki on. Yllä olevassa kurssi oliossa on kolme ominaisuutta (property). Esimmäisenä on nimi, jonka arvoksi on annettu ”JavaScript”. Huomaa, että ominaisuuden arvo annetaan kaksoispisteen (:) jälkeen, eikä sijoistusoperaattrorin eli = merkin jälkeen. Muutoinhan voi ajatella, että ominaisuus on ikään kuin olion sisällä oleva ”muuttuja”. Seuraava ominaisuus annetaan kenttäeroittimen eli pilkun (,) jälkeen. Kurssi oliomme näyttää siis seuraavalta:

kurssi

nimi: JavaScript
opintopisteet: 5.0
kurssinumero: SWDXXX00X

Huomaa myös, että olio määritellään aaltosulkujen sisälle:

```
var olionNimi = {};
```

Olion sisältämiä arvoja voimme tulostaa seuraavaan tapaan:

```
document.write(kurssi.nimi);
document.write(kurssi.opintopisteet);
```

Tyhjän olion voi siis luoda yllä olevalla tavalla. Myös let avainsanaa voi käyttää olion luonnissa aivan kuten taulukon ja muuttujankin kohdalla.

Jos siis haluaisimme luoda uuden tyhjän olion, jonka nimi olisi henkilö, onnistuisi se lauseella:

```
var henkilo = {};
```

Tämän jälkeen voisimme lisätä olioomme dynaamisesti niin paljon ominaisuuksia kuin tarvitsemme. Esimerkiksi:

```
henkilo.etunimi = "Maija";
```

Nyt oliomme näyttäisi seuraavalta:

Voisimme lisätä myös sukunimen olioon:

```
henkilo.sukunimi = "Poppanen";
```

Nyt oliomme näyttäisi alla olevalta:

Myös henkilö oliosta tietojen tulostaminen vaikkapa web selaimen konsoli-ikkunaan onnistuisi:

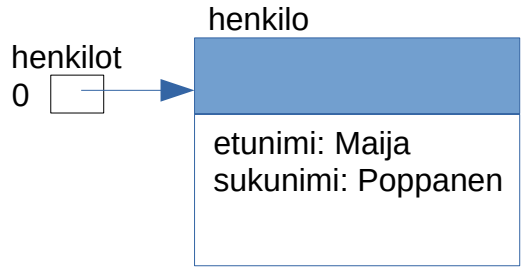
```
console.log(henkilo.etunimi +" " + henkilo.sukunimi);
```

Oliot ovat siis JavaScriptissä dynaamisia, toisinkuin useimmissa olio-ohjelmointikielissä, kuten Javassa tai C#:ssa. Muitakin eroja on. Emme tarvinneet tehdä ensin luokkaa, voidaksemme sen jälkeen luoda olioita. JavaScriptissä loimme olioita suoraan. Tässä mielessä JavaScriptissä oliot on toteutettu eri tavalla.

JavaScriptin olioita voi tietysti tallentaa eli sijoittaa taulukon sisälle. Tämä onnistuu esimerkiksi lisäämällä henkilö oliomme henkilot taulukkoon ensimmäiseksi:

```
var henkilot = [henkilo];
```

Nyt on meillä henkilot nimisessä taulukossa olio. Taulukon tietorakenteen voisi kuvata seuraavalla kuvalla:



Kuva 1. Henkilöt taulussa oleva henkilö olio.

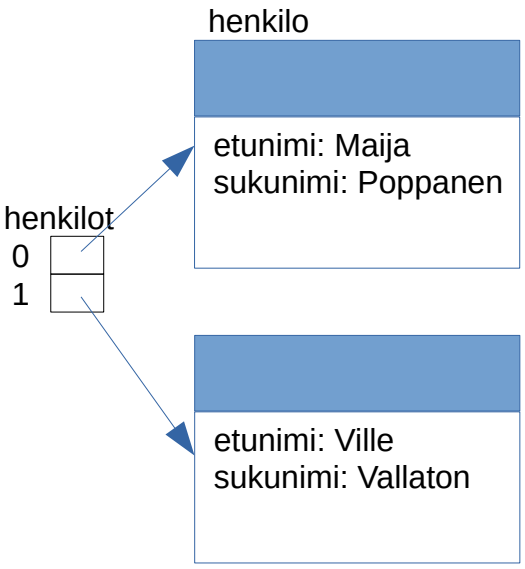
Tulostus onnistuisi taulukosta toistorakenteen avulla:

```
for (var i=0; i<henkilot.length; i++) {

     document.write(henkilot[i].etunimi + " " +
          henkilot[i].sukunimi + "<br />");
}
```

Voisimme lisätä henkilöt taulukkoon toisenkin olion seuraavalla tavalla:

```
var henkilo2 = { etunimi: "Ville",
                 sukunimi: "Vallaton"
};

henkilot[1] = henkilo2;
```

Nyt henkilot taulukon tietorakennetta voisi kuvata seuraavanlaisella kuvalla:



Kuva 2. Henkilöt taulussa olevat henkilö oliot.

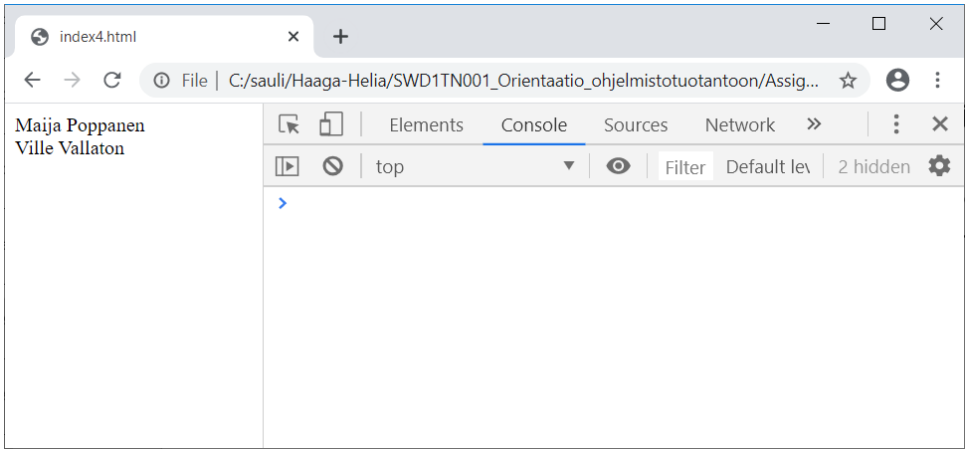
Henkilöt taulukossa on nyt siis kaksi oliota. Niitä voisi tallentaa (sijoittaa, assign) useita vastaavalla tavalla. Nyt toistorakennettakin alkaa tarvita, vaikkapa taulukossa olevien olioiden sisältäminen tietojen tulostuksessa tms. Valmis ohjelma näyttäisi seuraavalta:

```
function aja() {
    var henkilo = {};
    henkilo.etunimi = "Maija";
    henkilo.sukunimi = "Poppanen";
    var henkilot = [henkilo];

    var henkilo2 = {etunimi: "Ville",
                    sukunimi: "Vallaton"
                   };

    henkilot[1] = henkilo2;


    for (var i=0; i<henkilot.length; i++) {
        document.write(henkilot[i].etunimi + " " + henkilot[i].sukunimi+"<br />");
    }
}
```

Tarvittava html tiedosto olisi oltava samassa kansiossa ohjelma1.js tiedoston kanssa. Kyseinen html tiedosto olisi alla olevan näköinen:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title></title>
    <meta charset="utf-8">
</head>
<body>
    <button onclick="aja()">Käynnistä aja funktio</button>
    <script src="ohjelma1.js"></script>
</body>
</html>
```

Ja Chromessa tulostus näyttäisi alla olevalta kun painonappulaa on klikattu:



Olioita voi tietysti määritellä suoraan taulukkoon:

```
var tuotteet= [{nimi: "Takki", hinta: 170.5, alvprosentti: 24},
               {nimi: "Pipo", hinta: 10.5, alvprosentti: 24},
               {nimi: "Hanskat", hinta: 55.5, alvprosentti: 24},
               {nimi: "Sukat", hinta: 34, alvprosentti: 24}
             ];
```

Taulukon läpikäynti onnistuu for –rakenteella seuraavaan tapaan:

```
var output = "";
for (var i=0; i<tuotteet.length; i++) {

    output = output + (tuotteet[i].nimi + " " +
        tuotteet[i].hinta + "<br />");
}
document.getElementById('result').innerHTML = output;
```

Ensimmäisellä rivillä luodaan output muuttuja, jonka arvoksi annetaan heti tyhjä merkki eli `""`. Siinä on oltava jokin arvo, sillä toisto rakenteessa tarvitaan lisätä output muuttujassa olemassa olevaan arvoon lisää tekstiä. Jollei siellä ole valmiina jotain arvoa, kaatuu ohjelman ajon aikaiseen virheeseen. Seuraavana on toistorakenne, jossa muodostetaan henkilot taulukossa olevista neljästä oliosta merkkijono output muuttujaan rivi riviltä. Loppu tulos näyttää seuraavalta:

```
Takki 170.5
Pipo 10.5
Hanskat 55.5
Sukat 34
```

Lopuksi on lause, jossa
`document.getElementBy('result').innerHTML = output;`
lauseella tulostetaan tämä tulos web-selaimeen. Jotta tämä onnistuu, on html tiedostossa oltava vaikkapa jonka id nimi on result. Document olion getElementById() metodi pystyy sen silloin löytämään ja lisäämään sinne output muuttujassa olevan tekstin.

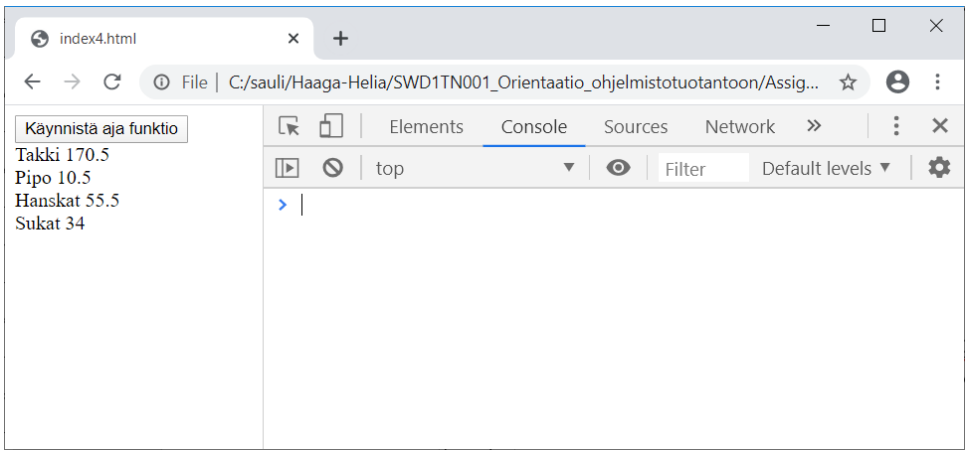
JavaScript ohjelma näyttää tässä vaiheessa seuraavalta:

```
function aja() {
    var tuotteet = [{nimi: " Takki", hinta: 170.5, alvprosentti: 24},
                    {nimi: " Pipo", hinta: 10.5, alvprosentti: 24},
                    {nimi: " Hanskat", hinta: 55.5, alvprosentti: 24},
                    {nimi: " Sukat", hinta: 34, alvprosentti: 24}
                ];
    var output = "";
    for (var i=0; i<tuotteet.length; i++) {
        output = output + (tuotteet[i].nimi + " " + tuotteet[i].hinta + "<br />");
    }
    document.getElementById('result').innerHTML = output;
}
```

Tarvittava html tiedosto on oltava samassa kansiossa ohjelma1.js tiedoston kanssa. Kyseinen tiedosto näyttää seuraavalta:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title></title>
    <meta charset="utf-8">
</head>
<body>
    <button onclick="aja()">Käynnistä aja funktio</button>
    <script src="ohjelma1.js"></script>
    <div id='result'></div>
</body>
</html>
```

Lopputulos näyttää Chromessa seuraavalta:

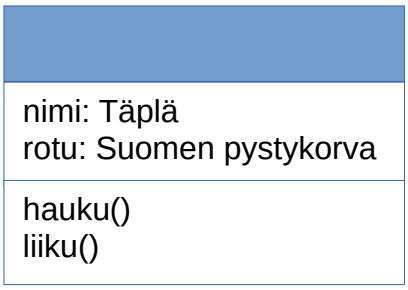


b. Metodit olioissa

Olioihin voi lisätä tarvittaessa myös funktioita, joita olioiden yhteydessä kutsutaan metodeiksi. Olion ”funktioiden” ideana on pystyä toteuttamaan olioihin ominaisuuksien ohella myös toiminnallisuutta. Jos esimerkiksi haluaisi toteuttaa koira olion, jolla on nimi, syntymävuosi ja rotu ominaisuuksina. Tämän koira olion toiminnallisuuksina voisi olla hauku, liiku. Koira oliomme pystyisi siis haukkumaan ja liikkumaan. Tätä kaikkea pystyy myös soveltamaan muuhunkin ja niin paljon tehdäänkin. Jos toteuttaisimme koira oliomme näyttäisi se seuraavalta:

```
var koira = {nimi: "Täplä",
             rotu: "Suomen pystykorva",
             syntymavuosi: 2019,
             hauku: function() {
                  return " Hau-hau!";
             },
             liiku: function(x) {
                  x++;
                  return x;
             }
};
```

Luokkakaaviona koira olio näyttäisi seuraavan kuvan mukaiselta:



Kuva 3. Koira olio luokkakaaviona.

JavaScript ohjelma näyttäisi seuraavalta:

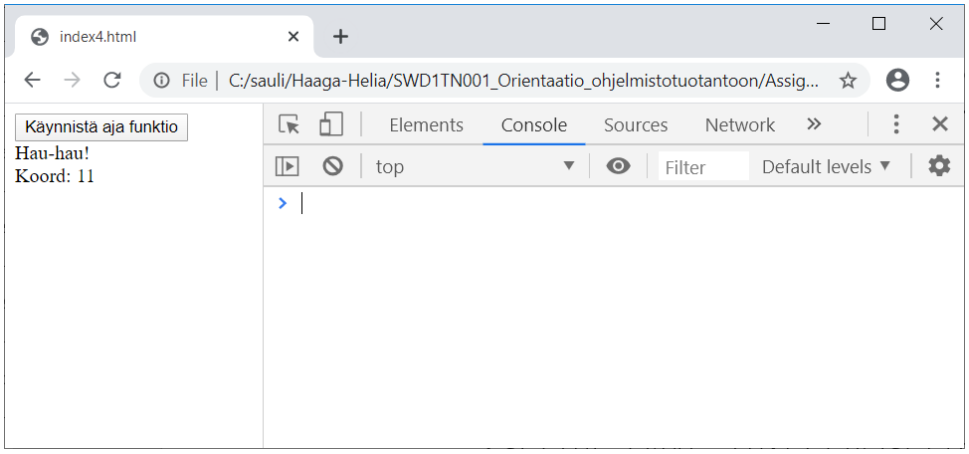
```
function aja() {
    var koira = {nimi: "Täplä",
            rotu: "Suomen pystykorva",
            syntymavuosi: 2019,
            hauku: function() {
                return " Hau-hau!";
            },
            liiku: function(x) {
                x++;
                return x;
            }
    };
    document.getElementById('result').innerHTML=koira.hauku() + "<br />";
    document.getElementById('result').innerHTML+="Koord: " + koira.liiku(10);
}
```

Html tiedosto puolestaan olisi seuraavanlainen:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title></title>
    <meta charset="utf-8">
</head>
<body>
    <button onclick="aja()">Käynnistä aja funktio</button>
    <script src="ohjelma1.js"></script>
    <div id='result'></div>
</body>
</html>
```

Line 10, Column 22 — 12 Lines INS UTF-8 HTML Spaces:

Sen olisi oltava samassa kansiossa ohjelma1.js tiedoston kanssa. Lopputulos Chromessa näyttäisi seuraavalta kun painonappulaa on klikattu:



Saimme siis luotua olion, jolla on koiramaisia ominaisuuksia ja se pystyy tekemään jotain. Samaan tapaan voisimme luoda muitakin oliota, joilla on reaalimaailman vastine. Olioita käytetään myös muihin tarkoituksiin. Edellä olemme viime aikoina tehneet JavaScript tiedostoon aina aja() funktion. Tämän nimen olisimme voineet aina korvata sovellutuksen nimellä, esimerkiksi koiraSovellus(). Olisimme lisäksi voineet tehdä siitä olion, jonka sisällä on esimerkiksi aja() funktio, ja muita funktioita joita sovellutuksessa tarvitaan:

```
var koiraSovellus = {aja: function() {
            var koira = {nimi: "Täplä",
            rotu: "Suomen pystykorva",
            syntymavuosi: 2019,
            hauku: function() {
                return " Hau-hau!";
            },
            liiku: function(x) {
                x++;
                return x;
            }
        };
        document.getElementById('result').innerHTML=
            koira.hauku() + "<br />";
        document.getElementById('result').innerHTML+=
            "Koord: " + koira.liiku(10);
    }
};
```

Nyt voisimme käynnistää sovellutuksemme lauseella:

```
koiraSovellus.aja();
```

Painonappulasta käynnistäminen onnistuisi html sivulla:

```
<button onclick="koiraSovellus.aja()">Käynnistä aja funktio</button>
```

Nämä muutokset näyttäisivät JavaScript ohjelmassa seuraavalta:

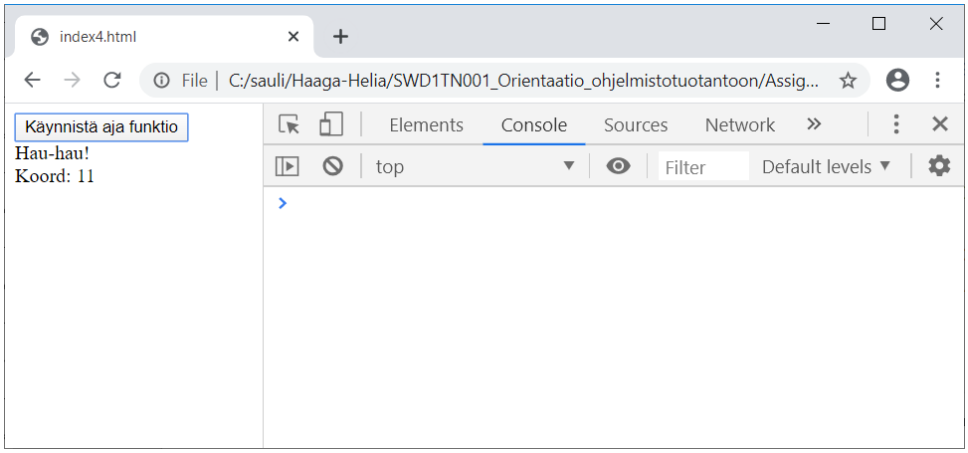
```
1  var koiraSovellus = {aja: function() {
2              var koira = {nimi: "Täplä",
3          rotu: "Suomen pystykorva",
4          syntymavuosi: 2019,
5          hauku: function() {
6              return " Hau-hau!";
7          },
8          liiku: function(x) {
9              x++;
10             return x;
11         }
12         };
13         document.getElementById('result').innerHTML=koira.hauku() + "<br />";
14         document.getElementById('result').innerHTML+="Koord: " +
           koira.liiku(10);
15     }
16 };
17
```

Html tiedosto näyttäisi seuraavalta:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title></title>
    <meta charset="utf-8">
</head>
<body>
    <button onclick="koiraSovellus.aja()">
        Käynnistä aja funktio</button>
    <script src="ohjelma1.js"></script>
    <div id='result'></div>
</body>
</html>
```

Line 5, Column 1 — 13 Lines INS UTF-8 HTML Spaces: 4

Lopputulos Chomessa sovellutusta ajettaessa näyttäisi alla olevalta:



Olioita ja funktioita käytetään siis nykyisin ohjelmoinnissa paljon muuhunkin. Niillä rakennetaan koko sovellutus ja sen sisäinen rakenne.

c. JSON (JavaScript Object Notation)

JSON on tärkeä tietorakenne formaatti datan siirtämisessä Internetin yli tai erilaisissa sovellutuksissa data formaattina. Nimensä mukaisesti kyse on JavaScript oliosta. Kun haluaa luoda JSON tiedoston tai sovellutuksen ajon aikasen JSON tietorakenteen, on sen noudatettava tiettyjä sääntöjä:

1) Datan on oltava nimi ja sen arvo pareja (esim, sukunimi, Mattila) ,
2) Data pitää olla erotettuna toisistaan pilkulla (esim {“sukunimi”: “Mattila”, “etunimi”: “Matti”}),
3) Aaltosulut pitää olla olion ympärillä,
4) Taulukon ympärillä on hakasulut.

Siis aivan kuten olemme jo tehneetkin JavaScriptin taulukoissa ja olioissa edellä. Sen lisäksi JSON olion nimet (ominaisuudet, properties) pitää olla merkitty siten, että niiden ympärillä on lainaus-merkit. Esimerkiksi:

```
{"tuotteet":[
    {"nimi": "Takki", "hinta": 170.5, "alvprosentti": 24},
    {"nimi": "Pipo", "hinta": 10.5, "alvprosentti": 24},
    {"nimi": "Hanskat", "hinta": 55.5, "alvprosentti": 24},
    {"nimi": "Sukat", "hinta": 34, "alvprosentti": 24}
]}
```

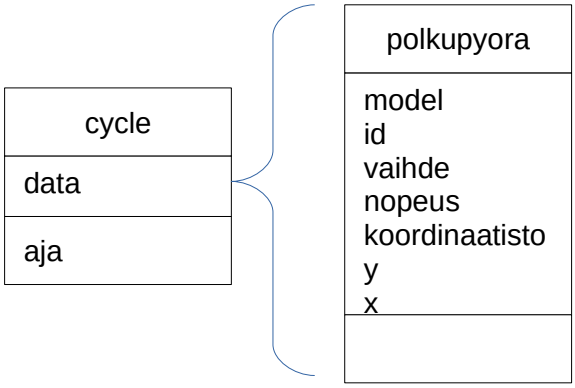
Lisäksi JSON tietotyypit ovat:

Number	(kokonaisnumerot ja desimaalinumerot)
String	(teksti)
Boolean	(totuusarvo, eli true tai false)
Null	(ei arvoa, sitä ole tai sitä ei ole annettu)
Object	(olio)
Array	(taulukko)

Siis käytännössä samat kuin JavaScriptissä muutenkin.

d. Sovellus olio ja data oliot

JavaScriptissä on mahdollista käyttää oliota sovellutuken ohjelmoinnissa. Sen sisälle voi ohjelmoida olioita ja metodeita ja tietysti myös tarvittavia tietorakenteita. Nekin tietysti omiin olioihinsa. Siis eräänlaisia data olioita, joita sovelluksena toimiva olio pystyy käyttämään. Esimerkkinä tästä voisi olla vaikkapa pyörä (cycle) olio, joka ”osaa” ajaa eteenpäin, jarruttaa, vaihtaa vaihdetta ja kiihdyttää. Lisäksi tarvittaisiin tiedot tästä pyörästä, kuten malli, pyörän tunnus (id), päällä oleva vaihde, käytettävä koordinaatisto ja sen mukainen sijainti piste jne. Tällainen pyörä olio voisi alla vaikkapa alla olevan kuvan 1 mukainen luokkakaavio:



Kuva 1. cycle olio

Yllä olevassa `cycle` nimisessä oliossa on yksi muuttuja, jonka nimi on `data`. Myös metodeja on vielä tässä vaiheessa vain yksi eli `aja()`. Niitä voisi ohjelmoida lisääkin. Esimerkiksi voisi tehdä omat metodinsa kyydyttämiseen, jarruttamiseen ja vaihteiden vaihtamiseen jne. Oliossa oleva `data` muuttujan arvoksi voi antaa vaikkapa toisen olion, esimerkiksi polkupyora olion. Tämä antaa meille mahdollisuuden muuttaa pyörän data-arvoja. Tämä onnistuu lauseella: `data: polkupyora`. Nyt voimme luoda `polkupyora` olion JavaScriptin Globaalin namespace:en. Voisimme muuttaa myös JSON tietorakenteen avulla `data` muuttujan arvon haluamaksemme. Nyt kuitenkin teemme sen `polkupyora` oliolla, joka yllä olevan kuvan mukaisesti on JavaScript koodina seuraavanlainen:

```
var polkupyora = {
    malli: "mtb",
    id : "1",
    vaihe: 2,
    nopeus: 0,
    koordinaatisto: "WGS 84",
    y: 60.20409030673413,
    x: 24.935754768274744
};
```

Yllä oleva `polkupyora` olio oltaisiin myös voitu luoda `let` avainsanaa käyttäen `var` avainsanan sijaan. Tämä koskee kaikkia alla olevian kohtia, joissa on käytetty `var` avainsanaa. Ne voitaisiin korvata uudemmalla muuttujien luontiin eli määrittelyyn tarkoitetulla `let`:llä. Seuraavaksi voimmekin tehdä yllä olevan kuva 1 mukaisen `cycle` olion:

```
let cycle = {
    data : polkupyora,
    aja : function() { }
};
```

Cycle oliomme sisältää nyt kuva 1:n suunnitelman mukaisesti olevan `data` olion ja `aja()` metodin. Aja metodi on määritelty olion sisäiseksi `aja` nimiseksi muuttujaksi, jolle onkin annettu arvoksi nimetön funktio (anonymous function). Sitä kutsutaan anonyymiksi funktioksi, koska sillehän ei ole annettu nimeä funktio rungossa. Metodille nimi annetaankin nyt siis lauseella: `aja : function() { }`, jolloin metodin nimeksi tulee `aja()`. Data oliosta on puolestaan viittaus `cycle` muuttujan ulkopuolella Globaalissa namespacessa olevaan `polkupyora` olioon, joka on luonteeltaan data-olio. Se siis sisältää vain dataa eikä siinä ole tarvittu metodeja. Seuraavaksi voimme alkaa katsomaan, mitä `aja()` metodin pitäisi osata tehdä. Ajaminen polkypyörällä tarkoittaa paikasta toiseen siirtymistä. Ohjelmassamme siis itseasiassa riittäisi, että muutamme polkupyörämme sijainti koordinaatteja uuteen pisteeseen. Esimerkiksi muuttamalla `polkupyora` olion `y` ja `x` koordinaatteja vaikkapa koordinaattipisteeseen 61.5,24.7. Tarvisemme siis antaa `aja()` metodille uudet sijainti koordinaatit, johon haluamme siirtää pyörämme. Metodia on siis pystyttävä kutsumaan siten, että sille annetaan uudet sijainti koordinaatit pyörälle. Tämä onnistuu lauseella: `aja(61.5,24.7)`. Metodimme muuttuu siis muotoon:

```
aja : function(uusi_Y, uusi_X) { }
```

Seuraavaksi voimmekin kirjoittaa lauseet metodimme sisälle lauseiksi, jotka suoritetaan metodin kutsumisen yhteydessä (eli kun metodi ajetaan käyntiin). Pyörämme sijainti koordinaatit löytyvät `data` oliosta, joka osoittaa `polkupyora` olion. Sen sisältä löytyvät `y` ja `x` muuttujat. Niiden arvoja haluamme pystyä muuttamaan `aja()` metodin avulla. Se onnistuu lauseilla:

```
this.data.y = uusi_Y;
this.data.x = uusi_X;
```

`this` kertoo lausessa, että halutaan käyttää `cycle` olion sisällä olevaa `data` oliota. Ei siis esimerkiksi `cycle` olion ulkopuolella olevaa saman nimistä `data` oliota – jos siis sellainen sattuisi olemaan. Näin siis varmistamme, että lauseessa käytetään täsmälleen sitä oliota, jota haluamme käyttää. Tämän jälkeen kerromme pisteen jälkeen haluamamme muuttujan tai olion nimen, eli tässä tapauksessa `data` olion nimen. Nyt olemmekin sitten päässeet `data` olion avulla suoraan `polkupyora` olion sisälle. Seuraavaksi voimmekin kertoa, mitä muuttujaa tai oliota haluamme käyttää. Haluamme viitata `y` ja `x` muuttujiin, joka onnistuukin lauseella: `this.data.y`. Nyt pääsemmekin asettamaan niille haluamamme data-arvot lauseilla:

```
this.data.y = uusi_Y;
this.data.x = uusi_X;
```

Valmis `aja()` metodimme näyttäisi tässä vaiheessa seuraavanlaiselta:

```
aja : function(uusi_Y,uusi_X) {
       this.data.y = uusi_Y;
       this.data.x = uusi_X;
 }
```

Ja kun se on lisätty `cycle` oliomme sisälle metodiksi, tilanne näyttää alla olevan kaltaiselta:

```
var cycle = {
     data : polkupyora,
     aja : function(uusi_Y,uusi_X) {
         this.data.y = uusi_Y;
         this.data.x = uusi_X;
     }
};
```

Nyt oliomme sisällä olevan `aja()` metodin kutsuminen `cycle` olion ulkopuolelta, onnistuu lauseella:

```
cycle.aja(61.5,24.7)
```

Seuraavaksi siis tarvitsemme jonkin keinon, minkä avulla voimme ajaa `cycle` ”polkupyörämme” haluamaamme paikkaan. Jokin toinen olio ratkaisisi tämän tehtävän erittäin hyvin. Tämän uuden olion nimi voisi olla jotain sellaista, joka kertoo mihin sitä tarvitaan. Olion nimi voisi olla vaikkapa `polkupyoraApp`, minkä tyylisesti tässä roolissa tai tehtävässä olevia olioita melko yleisesti nimetään. Seuraavaksi meidän on päätettävä, haluammeko jo olemassa olevan `cycle` olion olevan uuden `polkupyoraApp` olion sisällä vai sen ulkopuolella. Molemmat lähestymistavat olisivat sinällään täysin mahdollisia JavaScriptissä. JavaScript ohjelmoinnissa toisaalta on vallalla ohjelmointi tyyli (best practice), jossa JavaScriptin Globaaliin namespace:en ei haluta luoda liikaa olioita, vaan ne halutaan olevan tehtävän sovellutus olion sisällä. Siis meidän tapauksessa `polkupyoraApp.` Näin haluamme mekin tässä siis toimia:

```
var polkupyoraApp = {
         var cycle = {
            data : polkupyora,
            aja : function(uusi_Y,uusi_X) {
                this.data.y = uusi_Y;
                this.data.x = uusi_X;
            }
         };
};
```

Seuraavaksi voisimme lisätä `aja()` metodin kutsun, eli siis ajaa kyseisen metodin käyntiin ajettavaksi. Tarvitsemme siis nyt kirjoittaa lauseen `cycle.aja(61.5,24.7)` johonkin sopivaan paikaan, josta sen saa ajettua. Siihen tarvitaan tietysti funktio eli metodi, koska teemme tämän

tietysti `polkupyoraApp` olion sisällä. Voisimme tehdä sen suoraan anonyymilla funtiolla, sillä sen tyyppiset funktioa ajetaan käyntiin heti kun sellainen tulee lauseessa ajonaikasesti suoritettavaksi. Toisaalta sovellutuksemme luonteeseen kuuluu, että pyörällä ajetaan silloin kun halutaan siirtyä paikasta toiseen. Haluamme siis tavallaan ”käynnistää” pyörän halutessamme ja aja sillä sitten sen jälkeen. Sähköpyörässä tämä pitäisi kirjaimellisesti paikkansakin. Voisimme siis tehdä `polkupyoraApp` olion sisälle vaikkapa `run()` nimisen metodin, jota kutsumalla ikään kuin saisimme pyörämme ”käyntiin”. Jos laitamme koko jo tekemämme cycle `olion` tämän `run()` metodin sisälle, onnistuisi tämä ”käynnistäminen” niin lähelle kuin se on ohjelmoinnissa mahdollista:

```
var polkupyoraApp = {
    run: function() {
        var cycle = {
            data : polkupyora,
            aja : function(uusi_Y,uusi_X) {
                this.data.y = uusi_Y;
                this.data.x = uusi_X;
            }
        };
    }
};
```

Nyt meillä on keino ajaa sovellutuksessa vaikkapa sen sisällä olevan `cycle` olion metodeja. Esimerkiksi `aja()` metodia. Sopiva paikka olisi esimerkiksi heti `cycle` olion luontilauseen jälkeen.

```
var polkupyoraApp = {
    run: function() {
         var cycle = {
            data : polkupyora,
            aja : function(uusi_Y,uusi_X) {
                this.data.y = uusi_Y;
                this.data.x = uusi_X;
            }
         };
         cycle.aja(61.5,24.7);
    }
};
```

Pyörän saisimme puolestaan ”käyntiin” ajamalla lauseen, jossa kutsutaan `polkupyoraApp` olion `run()` metodia:

```
polkupyoraApp.run();
```

Nyt meidän pitäisi löytää jokin sopiva paikka tälle lauseelle. Voisimme laittaa sen ohjelmassamme heti polkupyoraApp olion perään, jolloin se suoritettaisiin aina:

```
var polkupyoraApp = {
    run: function() {
         var cycle = {
            data : polkupyora,
            aja : function(uusi_Y,uusi_X) {
                this.data.y = uusi_Y;
                this.data.x = uusi_X;
            }
         };
         cycle.aja(61.5,24.7);
    }
};
polkupyoraApp.run();
```

Tietysti tarvitsisimme html tiedoston, pystyäksemme ajamaan JavaScript sovellutustamme web selaimessa. Html tiedosto voisi olla alla olevan kaltainen tiedosto:

```
<!DOCTYPE html>
<html>
     <head>
           <meta charset="utf-8">
           <title>Polkupyörä App</title>
           <script type="text/javascript" src="main.js">
           </script>
     </head>
     <body>

     </body>
</html>
```

JavaScript tiedosto on siis oltava tässä tapauksessa talletettuna `main.js` nimiseen tiedostoon ja sijaittava samassa kansiossa kuin on yllä oleva html tiedostokin. Tästä eteenpäin ohjelmointi tehtävämme etenisi siten, että joutuisimme suunnittelemaan, miten näytämme tulevalle käyttäjälle, että pyörämme myös siirtyi uuteen koordinaattipisteeseen (61.5,24.7). Voisimme esimerkiksi näyttää tulevalle sovellutuksen käyttäjälle pyörän uudet sijainti koordinaatit. Html tiedostoon tarvitsemme elementin tähän takoitukseen. Se voisi olla vaikkapa seuraava elementti, jolla on oma yksikäsitteinen eli uniikki id nimi: `<div id="result"></div>`. Html tiedosto siis näyttäisi tässä vaiheessa seuraavanlaiselta:

```
<!DOCTYPE html>
<html>
     <head>
           <meta charset="utf-8">
           <title>Polkupyörä App</title>
           <script type="text/javascript" src="main.js">
           </script>
     </head>
     <body>
         <div id="result"></div>
     </body>
</html>
```

Nyt kun meillä on html elementti ja sillä oma `id` nimensä `result`, jota ei ole millään toisella elementillä kyseisessä htm tiedostossa. Voimme löytää tämän `result` nimisen `div` elementin JavaScript ohjelmastamme käsin etsimällä sen `document` olion `getElementById()` metodin avulla:

```
document.getElementById("result").innerHTML
```

Kun `getElementById()` metodi on löytänyt `div` elementin, voimme näyttää tässä `div` elementissä haluamamme tiedon `innerHTML` ominaisuuden avulla. JavaScript tiedosto näyttää nyt tässä vaiheessa seuraavanlaiselta:

```
var polkupyoraApp = {
    run: function() {
         var cycle = {
            data : polkupyora,
            aja : function(uusi_Y,uusi_X) {
                this.data.y = uusi_Y;
                this.data.x = uusi_X;
            }
         };
         document.getElementById("result").innerHTML =
             cycle.aja(61.5,24.7);
    }
};
polkupyoraApp.run();
```

Nyt tietysti meidän on ohjelmoitava `aja()` metodimme palauttamaan pyörän sen hetkisen sijainnin paluuarvonaan. Tämän saa tehtyä `return` lauseella `aja()` metodin viimeisenä rivinä. PolkupyoraApp näyttää siis tässä vaiheessa seuraavanlaiselta:

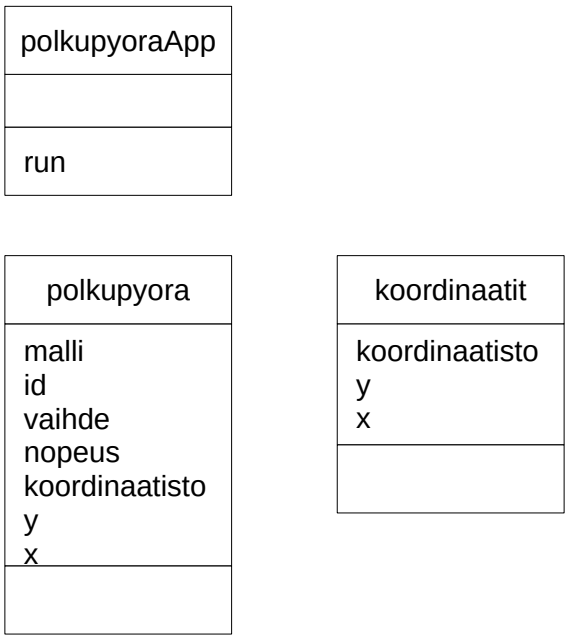
```
var polkupyoraApp = {
    run: function() {
         var cycle = {
            data : polkupyora,
            aja : function(uusi_Y,uusi_X) {
                this.data.y = uusi_Y;
                this.data.x = uusi_X;
                return this.data.x + "," +  this.data.x;
            }
         };
         document.getElementById("result").innerHTML =
             "Uusi sijaintipiste on " + cycle.aja(61.5,24.7);
    }
};
polkupyoraApp.run();
```

Toinen vaihtoehto `polkupyoraApp` sovellutuksemme ajamiseksi, olisi antaa tulevan käyttäjän käynnistää se itse. Se tietysti onnistuu painonappulan avulla: `<button id="aja" onclick="polkupyoraApp.run()">Aja</button>`. Myös uudet koordinaatit voisi kysyä käyttäjältä tekstilaatikolla tai vaihtoehtoisesti antaa JSON tietorakenteena, mikä onkin alla olevan kuvan 2 `koordinaatit` data oliossa otettukin huomioon yhtenä vaihtoehtona. Tässä tapauksessa uudet koordinaatit voisi antaa `aja()` metodille seuraavalla tavalla: `cycle.aja(koordinaatit.y, koordinaatit.x)`

Tässä tarvitaan JavaScriptin Globaaliin namespace:en uusi koordinaatit niminen data olio:

```
var koordinaatit = {
    "koordinaatisto": "WGS 84",
    "y": 61.5,
    "x": 24.7
};
```

Sovellutuksemme näyttää tässä vaiheessa luokkakaaviona esitettynä alla olevan kuva 2:n kaltaiselta.



Kuva 2. Polkupyörä applikaatio (sovellus).

Ja koko JavaScript tiedosto on alla olevan kaltainen:

```
var polkupyora = {
    malli: "mtb",
    id : "1",
    vaihe: 2,
    nopeus: 0,
    koordinaatisto: "WGS 84",
    y: 60.20409030673413,
    x: 24.935754768274744
};

var koordinaatit = {
    "koordinaatisto": "WGS 84",
    "y": 61.5,
    "x": 24.7
};

var polkupyoraApp = {
    run: function() {
        var cycle = {
            data : polkupyora,
            aja : function(uusi_Y,uusi_X) {
                this.data.y = uusi_Y;
                this.data.x = uusi_X;
                var output = this.data.malli +" polkupyörä " +  + "
                      " + this.data.id + " on paikassa " +
                      this.data.y  + ", " + this.data.x+ " ja nopeus
                      on " + this.data.nopeus + " km/h.";
                return output;
            }
        };
        document.getElementById('result').innerHTML =
            cycle.aja(koordinaatit.y, koordinaatit.x);
    }
};
```