



Help to complete the tasks of this exercise can be found on the chapter 4 “Data Structures: Objects and Arrays” of our course book “Eloquent JavaScript” (3<sup>rd</sup> edition) by Marijin Haverbeke, but in this exercise you’ll need use Google too.

The aims of the exercise are to learn some skills working with objects and classes in JavaScript. However, the exercise does not elaborate on the rather intricate JavaScript prototypal inheritance. That is left for you as a subject of further studies.

Embed your theory answers, drawings, codes, and screenshots directly into this document. Always immediately after the relevant question. Return the document into your return box in itsLearning by the deadline.

Remember to give your own assessment when returning this document.

It’s also recommendable to use Internet sources to supplement the information provided by the course book.

The maximum number of points you can earn from this exercise is  $10 + 3 = 13$ .

### Tasks:

#### 1. Explain. (4 \* 0,25 = 1 point)

a. What is the difference between having two references to the same object and having two different objects that contain the same properties?

Kun sinulla on kaksi viittausta samaan objektiin, molemmat viittaukset jakavat saman objektin muistissa, ja muutokset vaikuttavat kumpaankin viittaukseen. Kun sinulla on kaksi erillistä objektia, jotka sisältävät samat ominaisuudet, ne ovat itsenäisiä, ja muutokset toisessa eivät vaikuta toiseen.

b. The keyword `new`.

Kun käytät `new`-avainsanaa JavaScriptissä, se luo uuden instanssin tietylle objektiluokalle. Se varaa muistista tilan uudelle objektille ja viittaa siihen. Käyttäessäsi `new`-avainsanaa, luot uuden objektin tietyn rakenteen pohjalta, kuten konstruktorifunktion tai luokan määrittelyn perusteella. Tämä mahdollistaa useiden samanlaisten objektien luomisen, joilla on yhteisiä ominaisuuksia ja metodeja, mutta jotka voivat sisältää erilaisia arvoja. `new`-avainsana auttaa organisoimaan ja luomaan dynaamisia objekteja ohjelmoinnissa.



**this** on avainsana JavaScriptissä, jota käytetään viittaamaan objektiin, jossa tietty koodi suoritetaan. **this** voi viitata eri objekteihin riippuen siitä, missä se sijaitsee koodissa ja miten funktio tai metodi on kutsuttu.

d. Is there something wrong in the examples below? Are you certain?

```
const friedmanBooks = [  
  'The Little Schemer',  
  'The Seasoned Schemer',  
];  
  
const hoyteBook = {  
  name: 'Let Over Lambda',  
  author: 'Doug Hoyte',  
};
```

## 2. Object destructuring. (4 \* 0,25 = 1 point)

You have the following object.

```
const musician = {  
  name: 'Sting',  
  realName: 'Gordon Matthew Thomas Sumner',  
  instrument: {  
    type: 'bass'  
  }  
};
```

Use object destructuring to do the following assignments.

a. Read the attributes `name` and `instrument` into the variables `name` and `instrument`.



The screenshot shows the Visual Studio Code interface. The Explorer panel on the left displays the project structure, including folders like 'JavaScript', 'Koodit', 'tehtäväsarja1', 'tehtäväsarja3', and 'T2'. The file 'JS A.js' is selected. The Editor panel shows the following JavaScript code:

```

1  const musician = {
2      name: "Sting",
3      realName: "Gordon Matthew Thomas",
4      instrument: {
5          type: "bass",
6      },
7  };
8
9  const { name, instrument } = musician;
10
11 console.log(name); // Output: Sting
12 console.log(instrument); // Output: { type: 'bass' }
13

```

The Terminal panel at the bottom shows the output of the code execution:

```

PS C:\Users\jouni\OneDrive\Desktop\Selainohjelmointi> node A.js
Sting
{ type: 'bass' }

```

b. Read the attributes `name` and `instrument` into the variables `nameOfArtist` and `instrumentOfArtist`.



```
Terminal Help
JS B.js U X JS C.js U JS D.js U
JavaScript > Koodit > tehtäväsarja3 > T2 > JS B.js > ...
1  const musician = {
2      name: "Sting",
3      realName: "Gordon Matthew Thomas Sumner",
4      instrument: {
5          type: "bass",
6      },
7  };
8
9  const { name: nameOfArtist, instrument: instrumentOfArtist } = musician;
10
11 console.log(nameOfArtist);
12 console.log(instrumentOfArtist);
13

PORTS PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
● PS C:\Users\jouni\OneDrive\Desktop\Selainohjelmointi> node "c:\Users\jouni\OneDrive\Desktop\Selainohjelmointi\B.js"
Sting
{ type: 'bass' }
○ PS C:\Users\jouni\OneDrive\Desktop\Selainohjelmointi> 
```

c. Read the type of the instrument into a variable `instrumentTypeOfArtist`.



```
Terminal Help
JS C.js U x JS D.js U
JavaScript > Koodit > tehtäväsarja3 > T2 > JS C.js > ...
1  const musician = {
2      name: "Sting",
3      realName: "Gordon Matthew Thomas Sumner",
4      instrument: {
5          type: "bass",
6      },
7  };
8
9  // Object destructuring to read the 'type' property into
10 const {
11     instrument: { type: instrumentTypeOfArtist },
12 } = musician;
13
14 console.log(instrumentTypeOfArtist); // 'bass'
15
```

PORTS PROBLEMS OUTPUT **TERMINAL** DEBUG CONSOLE

```
● PS C:\Users\jouni\OneDrive\Desktop\Selainohjelmointi> node "c:\Users\jouni\OneDrive\Desktop\Selainohjelmointi\T2\JS\C.js"
○ bass
PS C:\Users\jouni\OneDrive\Desktop\Selainohjelmointi>
```

d. Read the make of the instrument into a variable `instrumentMakeOfArtist`. If the attribute is missing from the current object, give it a default value "unknown".



```
Terminal Help
JS D.js U x
JavaScript > Koodit > tehtäväsarja3 > T2 > JS D.js > ...
1  const musician = {
2    name: "Sting",
3    realName: "Gordon Matthew Thomas Sumner",
4    instrument: {
5      type: "bass",
6    },
7  };
8
9  // Object destructuring with a default value
10 const {
11   instrument: { make: instrumentMakeOfArtist = "unknown"
12 } = musician;
13
14 console.log(instrumentMakeOfArtist); // 'unknown' (since
15
```

PORTS PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
● PS C:\Users\jouni\OneDrive\Desktop\Selainohjelmointi> node "c:\Users\jouni\OneDrive\Desktop\Sela
unknown
○ PS C:\Users\jouni\OneDrive\Desktop\Selainohjelmointi> 
```

### 3. For..in loop, Object.entries, Object.keys, and Object.values. (4 \* 0,5 = 2 points)

a. Explain for..in loop.

silmutka on JavaScript-ohjelmointikielen silmukkatyyppi, joka käy läpi objektin ominaisuudet, kuten avaimet tai indeksit, ja suorittaa sille määritellyn koodin kullekin ominaisuudelle. Tämä silmutka on hyödyllinen objektien tietojen käsittelyssä ja läpikäynnissä.

b. Explain static methods Object.entries, Object.keys, and Object.values.



JavaScriptissä **Object.entries**, **Object.keys** ja **Object.values** ovat staattisia

metodeja, jotka liittyvät JavaScript-objekteihin. Nämä metodit mahdollistavat objektin ominaisuuksien, kuten avainten ja arvojen, käsittelyn ja käsittelytoimenpiteiden suorittamisen helposti ja tehokkaasti.

**Object.entries** palauttaa objektin avaimet ja niihin liittyvät arvot, **Object.keys** palauttaa objektin avaimet ja **Object.values** palauttaa objektin arvot. Näitä metodeja voidaan käyttää monipuolisesti objektien sisällön tutkimiseen ja manipulointiin JavaScript-ohjelmoinnissa.

c. Use `for..in` with the object musician above. Log the attribute names and attribute values on the console. For example, when it is the turn of the attribute `realName`, the following text should be printed:

`realName = Gordon Matthew Thomas Sumner`



```
Terminal  Help
JS D.js  U  JS C.js  U x
JavaScript > Koodit > tehtäväsarja3 > T3 > JS C.js > ...
1  const musician = {
2    stageName: "Sting",
3    realName: "Gordon Matthew Thomas Sumner",
4    instrument: "Bass guitar, vocals",
5    genre: "Rock",
6  };
7
8  for (const attribute in musician) {
9    console.log(`${attribute} = ${musician[attribute]}`);
10 }
11
```

PORTS PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

- PS C:\Users\jouni\OneDrive\Desktop\Selainohjelmointi> node "c:\Users\jouni\OneDrive\Desktop\Selainohjelmointi\JavaScript\Koodit\tehtäväsarja3\T3\JS C.js"
unknown
- PS C:\Users\jouni\OneDrive\Desktop\Selainohjelmointi> node "c:\Users\jouni\OneDrive\Desktop\Selainohjelmointi\JavaScript\Koodit\tehtäväsarja3\T3\JS C.js"
stageName = Sting
realName = Gordon Matthew Thomas Sumner
instrument = Bass guitar, vocals
genre = Rock
- PS C:\Users\jouni\OneDrive\Desktop\Selainohjelmointi>

d. Use the musician object from the task 2 again. Implement also another musician object. Put them both into an array. Give examples of the use of the static methods `Object.entries`, `Object.keys`, and `Object.values` with the object array you just created.





```
JS D.js U x
JavaScript > Koodit > tehtäväsarja3 > T3 > JS D.js > ...
14   };
15
16   // Put both musician objects into an array
17   const musiciansArray = [musician1, musician2];
18
19   // Example of using Object.entries with the array
20   console.log("Using Object.entries:");
21   for (const musician of musiciansArray) {
22     const entries = Object.entries(musician);
23     console.log(entries);
24   }
25
26   // Example of using Object.keys with the array
27   console.log("\nUsing Object.keys:");
28   for (const musician of musiciansArray) {
29     const keys = Object.keys(musician);
30     console.log(keys);
31   }
32
33   // Example of using Object.values with the array
34   console.log("\nUsing Object.values:");
35   for (const musician of musiciansArray) {
36     const values = Object.values(musician);
37     console.log(values);
38   }
39
```

PORTS PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
[ 'stageName', 'Sting' ],
[ 'realName', 'Gordon Matthew Thomas Sumner' ],
[ 'instrument', 'Bass guitar, vocals' ],
[ 'genre', 'Rock' ]
]
[
  [ 'stageName', 'Prince' ],
  [ 'realName', 'Prince Rogers Nelson' ],
  [ 'instrument', 'Various instruments, vocals' ],
  [ 'genre', 'Funk, Pop, R&B' ]
]
```

Using Object.keys:

```
[ 'stageName', 'realName', 'instrument', 'genre' ]
[ 'stageName', 'realName', 'instrument', 'genre' ]
```

Using Object.values:

```
[
  'Sting',
  'Gordon Matthew Thomas Sumner',
  'Bass guitar, vocals',
  'Rock'
]
[
  'Prince',
  'Prince Rogers Nelson',
  'Various instruments, vocals',
  'Funk, Pop, R&B'
]
```

PS C:\Users\jouni\OneDrive\Desktop\Selainohjelmointi>



**4. Getters and setters. (2 subtask answered gives 0,5 points, 3 subtasks answered gives 1 point)**

- a. Create an object `song`. It has one attribute called `name`. It has a getter (a virtual attribute) called `duration`, and a setter that is also called `duration`. The getter returns the duration of the in minutes and seconds, and the setter can be used to set it.
- b. Invoke the setter and the getter.



```
Terminal Help
JS D.js JS AB.js U X
JavaScript > Koodit > tehtäväsarja3 > T4 > JS AB.js > ...
1 // A ja B Kohta samassa
2 const song = {
3   name: "My Song",
4   _duration: 0,
5
6   get duration() {
7     const minutes = Math.floor(this._duration / 60);
8     const seconds = this._duration % 60;
9     return `${minutes}:${seconds}`;
10  },
11
12  set duration(value) {
13    const [minutes, seconds] = value.split(":").map(Number);
14    this._duration = minutes * 60 + seconds;
15  },
16 };
17
18 // tässä kohta B
19
20 // Aseta kesto (3 minuuttia ja 45 sekuntia)
21 song.duration = "3:45";
22 // Hae kesto
23 console.log(song.duration); // Tulostaa '3:45'
24
```

PORTS PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
PS C:\Users\jouni\OneDrive\Desktop\Selainohjelmointi> node "c:\Users\jouni\OneDrive\Desktop\Selainohjelmointi\3:45"
3:45
PS C:\Users\jouni\OneDrive\Desktop\Selainohjelmointi> 
```



c. Explain the differences between normal object methods and these getters and setters.

Normaalit objektimetodit ovat toimintoja, jotka suoritetaan, kun niitä kutsutaan, ja ne voivat tehdä monenlaisia asioita. Getterit ja setterit ovat erityisiä metodeja, jotka liittyvät yhteen objektin ominaisuuteen. Getterit palauttavat arvon kuin ominaisuuden lukemisen, ja setterit mahdollistavat arvon asettamisen kuin ominaisuuden asettamisen. Gettereitä ja settereitä käytetään usein datan hallintaan ja antavat mahdollisuuden suorittaa lisätoimintoja arvon hakuun ja asettamiseen.

### 5. Working with JSON. (4 \* 0,5 = 2 points)

a. What are the purposes of JSON?

(JavaScript Object Notation) on yksinkertainen tietojen muotoilutapa, joka on tarkoitettu tietojen vaihtamiseen ja tallentamiseen. Sen tärkeimmät tarkoitukset ovat:

1. **Tietojen siirtäminen verkossa:** JSONia käytetään usein tietojen siirtämiseen sovellusten ja palvelimien välillä verkossa. Se on kevyt ja helppo lukea sekä kirjoittaa, mikä tekee siitä suositun tiedonvaihtoformaatin.
2. **Tiedon tallentaminen:** JSONia voidaan käyttää tietojen tallentamiseen, esimerkiksi konfiguraatietietojen, käyttäjäasetusten tai tietokantadatan tallentamiseen. Se on ihmisen luettavissa ja helppo käsitellä ohjelmallisesti.
3. **API-vasteet:** Useimmat verkkopalvelut palauttavat JSON-muotoisia vastauksia, kun pyydetään tietoja RESTful API -kutsuilla. Tämä mahdollistaa helpon tiedon käsittelyn sovelluksissa.
4. **Rakenteellinen tiedonsiirto:** JSON mahdollistaa monimutkaisten tietorakenteiden, kuten objektien ja taulukoiden, luomisen. Tämä tekee siitä hyödyllisen monenlaisissa sovelluksissa.

Yhteenvetona JSONin päätarkoitus on helppo tietojen muotoilu ja vaihtaminen ohjelmien ja palvelimien välillä sekä tietojen JSON tallentaminen ja käsittely eri ohjelmointiympäristöissä.

b. There are few differences between JavaScript objects and JSON. List and explain them.

On olemassa muutamia eroja JavaScript-objektien ja JSONin (JavaScript Object Notation) välillä:

1. **Kirjoitusasu:** JSON on tiukemmin määritelty muotoilutapa kuin JavaScript-objektit. JSONissa kaikki ominaisuudet ja arvot on ympäröity lainausmerkeillä. Esimerkiksi **"nimi": "John"** JSONissa, kun taas JavaScript-objekteissa voit käyttää **nimi: "John"**.



2. **Toiminnot:** JSON ei tue toimintoja tai funktioita. JavaScript-objektit voivat sisältää toimintoja (metodeja), kun taas JSON voi sisältää vain staattisia tietoja.
3. **Ylimääräiset ominaisuudet:** JavaScript-objekteissa voit lisätä ja poistaa ominaisuuksia dynaamisesti. JSON on staattinen ja sen rakenne on kiinteä, eikä siinä voi olla ylimääräisiä tai tuntemattomia ominaisuuksia.
4. **Kommentit:** JSON ei tue kommentteja. JavaScript-koodissa voit lisätä kommentteja selventämään koodin toimintaa.
5. **Merkkijonot:** JSON vaatii merkkijonojen käyttämistä ominaisuuden nimen ympärillä ja arvojen kohdalla. JavaScript-objekteissa voit käyttää myös muita datatyppejä ominaisuuden niminä, kuten numeroita ja symboleja.
6. **Pilkkumerkit:** JSON vaatii pilkkujen (,) käyttämistä erottimena eri ominaisuuksien välillä. JavaScript-objekteissa voit jättää pilkut pois viimeisestä ominaisuudesta.
7. **Tyhjät arvot:** JSON tukee **null**-arvoa, kun taas JavaScript-objekteissa voi olla ominaisuuksia, joiden arvo on **undefined**.

Yhteenvedona, JSON on tiukemmin määritelty tietojen muotoilutapa, joka on tarkoitettu tietojen siirtämiseen ja tallentamiseen, kun taas JavaScript-objektit ovat dynaamisia tietorakenteita, jotka voivat sisältää monenlaisia tietoja ja toimintoja. JSONia käytetään yleisesti tiedonvaihtoon ja tallentamiseen, kun taas JavaScript-objektit ovat keskeinen osa ohjelmakoodia JavaScript-sovelluksissa.

c. Serialize the object `person` below to a string containing a JSON object literal.

```
let person = {name: "Pentti", age: 22, country: "Finland"};
```

Voit serialisoida (muuntaa JSON-muotoon) **person**-objektin käyttämällä **JSON.stringify()**-funktiota JavaScriptissa.

esimerkki:



```
Terminal  Help
JS D.js  JS AB.js  U  JS esimerkejä.js ●
JavaScript > Koodit > JS esimerkejä.js > ...
1  let person = { name: "Pentti", age: 22, country: "Finlan
2  let jsonString = JSON.stringify(person);
3
4  console.log(jsonString);
5
6
```

Tämä koodi muuntaa person-objektin JSON-merkkijonoksi. jsonString sisältää nyt JSON-muotoisen objektiliitteen:

```
JS D.js  JS AB.js  U  JS esimerkejä.js 3 ●
JavaScript > Koodit > JS esimerkejä.js
1  {"name":"Pentti","age":22,"country":"Finland"}
2
3
```

d. Deserialize the JSON object literal back to another JavaScript object.

Voit deserialisoida (muuntaa JSON-muodon takaisin JavaScript-objektiksi) käyttämällä **JSON.parse()**-funktiota JavaScriptissä.

esimerkki:



```
Terminal Help
JS D.js JS AB.js U JS esimerkejä.js
JavaScript > Koodit > JS esimerkejä.js > ...
1 let jsonString = '{"name":"Pentti","age":22,"country":"Fi
2 let person = JSON.parse(jsonString);
3
4 console.log(person);
5
6
```

PORTS PROBLEMS OUTPUT **TERMINAL** DEBUG CONSOLE

PS C:\Users\jouni\OneDrive\Desktop\Selainohjelmointi>

Tämä koodi muuntaa JSON-merkkijonon takaisin JavaScript-objektiksi. **person**-objekti sisältää nyt samat tiedot kuin alkuperäinen **person**-objekti

```
Terminal Help
JS D.js JS AB.js U JS esimerkejä.js 2
JavaScript > Koodit > JS esimerkejä.js
...
1 { name: 'Pentti', age: 22, country: 'Finland' }
2
3
4
5 /*const musician = {
6   realName: "Gordon Matthew Thomas Sumner",
```

PORTS PROBLEMS **2** OUTPUT **TERMINAL** DEBUG CONSOLE

PS C:\Users\jouni\OneDrive\Desktop\Selainohjelmointi>

## 6. Working with some common JavaScript library objects. (2 \* 0,5 = 1 point)

- Create a function `getRandomIntegerFromRange`. It accepts two arguments. The argument `startRange` should be an integer and it sets the start of the Range. The argument `endRange` should



random integer that is greater or equal to the `startRange` and less or equal to the `endRange`.

```
Terminal Help
JS A.js U x
JavaScript > Koodit > tehtäväsarja3 > T6 > JS A.js > ...
1  function getRandomIntegerFromRange(startRange, endRange) {
2      if (!Number.isInteger(startRange) || !Number.isInteger(endRange)) {
3          throw new Error("Both arguments must be integers");
4      }
5
6      if (startRange > endRange) {
7          throw new Error("startRange must be less than or equal to endRange");
8      }
9
10     const min = Math.ceil(startRange);
11     const max = Math.floor(endRange);
12
13     return Math.floor(Math.random() * (max - min + 1)) + min;
14 }
15
16 // Example usage:
17 const randomNum = getRandomIntegerFromRange(1, 10);
18 console.log(randomNum); // This will print a random integer between 1 and 10
19
PORTS PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
● PS C:\Users\jouni\OneDrive\Desktop\Selainohjelmointi> node "c:\Users\jouni\OneDrive\Desktop\Selainohjelmointi\T6\JS A.js"
1
○ PS C:\Users\jouni\OneDrive\Desktop\Selainohjelmointi> 
```





b. Create a function `getTimeDifferenceInFullDays` that returns the number of full days between two dates. It accepts two arguments. The argument `startDate` is the start date of the period. The function `endDate` is the end date of the period. Use `Date` and `Math` objects.

```
Terminal  Help
JS A.js  U  JS B.js  U X
JavaScript > Koodit > tehtäväsarja3 > T6 > JS B.js > ...

2  // Convert both dates to milliseconds since the Unix epoch
3  const startMillis = startDate.getTime();
4  const endMillis = endDate.getTime();
5
6  // Calculate the time difference in milliseconds
7  const timeDifferenceMillis = endMillis - startMillis;
8
9  // Calculate the number of full days by dividing by milliseconds in a day
10 const fullDays = Math.floor(timeDifferenceMillis / (24 * 60 * 60 * 1000));
11
12 return fullDays;
13 }
14
15 // Example usage:
16 const startDate = new Date("2023-10-01");
17 const endDate = new Date("2023-10-10");
18 const daysDifference = getTimeDifferenceInFullDays(startDate, endDate);
19 console.log(daysDifference); // This will print the number of full days
20

PORTS  PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
PS C:\Users\jouni\OneDrive\Desktop\Selainohjelmointi> node "c:\Users\jouni\OneDrive\Desktop\Selainohjelmointi\T6\JS B.js"
9
PS C:\Users\jouni\OneDrive\Desktop\Selainohjelmointi> 
```



**7. Initializing an object with a JavaScript class. (2 \* 0,5 = 1 point)**

- a. Create a class called `Person`. The class has a constructor that accepts two arguments: `name` and `age`. There should be two functions tacked on the class: `getName`, `getAge` and `sayGreeting`. Create 2 objects of the class. Call some methods.



```
JavaScript > Koodit > tehtäväsarja3 > T7 > JS A.js > Person > getAge

1  class Person {
2      constructor(name, age) {
3          this.name = name;
4          this.age = age;
5      }
6
7      getName() {
8          return this.name;
9      }
10
11     getAge() {
12         return this.age;
13     }
14
15     sayGreeting() {
16         console.log(
17             `Hello, my name is ${this.name} and I am ${this.age} years old.
18         );
19     }
20 }
21
22 // Luodaan kaksi objektia
23 const person1 = new Person("Alice", 30);
24 const person2 = new Person("Bob", 25);
25
26 // Kutsutaan joitakin metodeita
27 console.log(person1.getName()); // Tulostaa "Alice"
28 console.log(person2.getAge()); // Tulostaa 25
29 person1.sayGreeting(); // Tulostaa "Hello, my name is Alice and I am
30 person2.sayGreeting(); // Tulostaa "Hello, my name is Bob and I am 25
31
```

PORTS PROBLEMS OUTPUT **TERMINAL** DEBUG CONSOLE

```
PS C:\Users\jouni\OneDrive\Desktop\Selainohjelmointi> node "c:\Users\jouni\OneDrive\Desktop\Selainohjelmointi\Java
9
PS C:\Users\jouni\OneDrive\Desktop\Selainohjelmointi> node "c:\Users\jouni\OneDrive\Desktop\Selainohjelmointi\Java
Alice
25
Hello, my name is Alice and I am 30 years old.
Hello, my name is Bob and I am 25 years old.
PS C:\Users\jouni\OneDrive\Desktop\Selainohjelmointi> node "c:\Users\jouni\OneDrive\Desktop\Selainohjelmointi\Java
Alice
25
Hello, my name is Alice and I am 30 years old.
```



Konstruktorin idea on alustaa luokan luomien objektien alkutila tai tila. Se suoritetaan aina, kun uusi objekti luodaan luokasta. Konstruktori asettaa objektin ominaisuudet alkuperäisiin arvoihin tai tiloihin, jotka ovat luokan käsiteltävissä. Tämä mahdollistaa objektin yksilöllisten ominaisuuksien alustamisen, kun se luodaan luokasta.

**8. Creating a utility with a JavaScript class. (2 \* 0,5 = 1 point)**

a. Create a class ZipValidator. It has two static methods: isValidZip and fixZip. The first static methods accept a zipCode and checks that it contains only numbers, and that it contains exactly five numbers. It returns true or false. The second static method accepts an argument zipCode. If the argument has a length less than five characters, the method prefixes it with leading zeros. The method returns a valid zipCode. Use the class and call the static methods.



```
Terminal Help
JS A.js ...T6 U JS B.js U JS A.js ...T7 U JS A.js ...T8 U X
JavaScript > Koodit > tehtäväsarja3 > T8 > JS A.js > ZipValidator > fixZip
1 class ZipValidator {
2     static isValidZip(zipCode) {
3         // Tarkistetaan, että zipCode sisältää tasan viisi n
4         const regex = /^d{5}$/;
5         return regex.test(zipCode);
6     }
7
8     static fixZip(zipCode) {
9         // Jos zipCode on alle viiden merkin pituinen, lisät
10        while (zipCode.length < 5) {
11            zipCode = "0" + zipCode;
12        }
13        return zipCode;
14    }
15 }
16
17 // Kutsutaan staattisia metodeita
18 const zip1 = "12345";
19 const zip2 = "54321";
20 console.log(ZipValidator.isValidZip(zip1)); // Tulostaa
21 console.log(ZipValidator.isValidZip(zip2)); // Tulostaa
22
23 const shortZip = "123";
24 const fixedZip = ZipValidator.fixZip(shortZip);
25 console.log(fixedZip); // Tulostaa "00123"
26
PORTS PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
PS C:\Users\jouni\OneDrive\Desktop\Selainohjelmointi> node "c:\Users\jouni\OneDrive\Desktop\Selain
o true
true
00123
PS C:\Users\jouni\OneDrive\Desktop\Selainohjelmointi> 
```



b. What is the difference between static methods and normal (instance) methods?

Staattiset (static) metodit eroavat normaaleista (instance) metodeista seuraavasti:

- **Staattiset metodit:** Ne liittyvät luokkaan eivätkä vaadi luokasta luotuja objekteja. Niitä kutsutaan suoraan luokan nimellä, kuten **Luokka.metodi()**. Staattiset metodit eivät pääse käsiksi luokan luomien objektien ominaisuuksiin tai tilaan, koska niillä ei ole pääsyä objektin **this**-kontekstiin.
- **Normaalit (instance) metodit:** Ne liittyvät luokasta luotuihin objekteihin. Niitä kutsutaan objektin kautta, kuten **objekti.metodi()**. Normaalit metodit voivat käyttää objektin omia ominaisuuksia ja tilaa, koska niillä on pääsy objektin **this**-kontekstiin.

Toisin sanoen, staattiset metodit ovat liittyneet itse luokkaan, kun taas normaalit metodit ovat liittyneet luokasta luotuihin objekteihin ja voivat toimia niiden tilan kanssa.

## 9. Extending JavaScript classes. (2 subtask answered gives 0,5 points, 3 subtasks answered gives 1 point)

a. Create a class `SuperHero`. Inherit it from the class `Person`. The constructor accepts an additional argument: `superpower`. There is also a function tacked on the class: `useSuperPower`. (In our case it is enough to just log the `superpower` on the console as a string)



```

Terminal  Help
JS A.js ...T6 U  JS B.js  U  JS A.js ...T7 U  JS A.js ...T8 U  JS A.js ...T9 U X
JavaScript > Koodit > tehtäväsarja3 > T9 > JS A.js > ...

1  class Person {
2      constructor(name, age) {
3          this.name = name;
4          this.age = age;
5      }
6
7      sayHello() {
8          console.log(
9              `Hello, my name is ${this.name} and I am ${this.age} years old.`
10             );
11        }
12    }
13
14    class SuperHero extends Person {
15        constructor(name, age, superpower) {
16            super(name, age); // Kutsutaan ylläluokan konstruktoria
17            this.superpower = superpower;
18        }
19
20        useSuperPower() {
21            console.log(`I am using my superpower: ${this.superpower}`);
22        }
23    }
24
25    // Luodaan SuperHero-objekteja
26    const hero1 = new SuperHero("Superman", 30, "Flight");
27    const hero2 = new SuperHero("Spider-Man", 25, "Web-slinging");
28
29    // Kutsutaan metodeita
30    hero1.sayHello(); // Tulostaa "Hello, my name is Superman and I am 30 years old."
31    hero1.useSuperPower(); // Tulostaa "I am using my superpower: Flight"
32
33    hero2.sayHello(); // Tulostaa "Hello, my name is Spider-Man and I am 25 years old."
34    hero2.useSuperPower(); // Tulostaa "I am using my superpower: Web-slinging"

```

PORTS PROBLEMS OUTPUT **TERMINAL** DEBUG CONSOLE

```

● PS C:\Users\jouni\OneDrive\Desktop\Selainohjelmointi> node "c:\Users\jouni\OneDrive\Desktop\Selainohjelmointi\JavaScript\Koodit\T9\A.js"
Hello, my name is Superman and I am 30 years old.
I am using my superpower: Flight
Hello, my name is Spider-Man and I am 25 years old.
I am using my superpower: Web-slinging
● PS C:\Users\jouni\OneDrive\Desktop\Selainohjelmointi> node "c:\Users\jouni\OneDrive\Desktop\Selainohjelmointi\JavaScript\Koodit\T9\A.js"
Hello, my name is Superman and I am 30 years old.
I am using my superpower: Flight
Hello, my name is Spider-Man and I am 25 years old.
I am using my superpower: Web-slinging
○ PS C:\Users\jouni\OneDrive\Desktop\Selainohjelmointi>

```



b. How do you make certain that also the initializations defined in the constructor of the inherited class `Person` are done?

varmistaa, että ylliluokan **Person** konstruktori suoritetaan perityssä **SuperHero**-luokassa käyttämällä **super(name, age)** -riviä **SuperHero**-luokan konstruktorissa. Tämä kutsuu ylliluokan konstruktoria ja varmistaa, että **name** ja **age**-ominaisuudet alustetaan oikein myös **SuperHero**-objekteissa.

c. Create 2 objects of the class. Call some methods.

Tein kaksi **SuperHero**-objektia, **hero1** ja **hero2**, ja kutsuttiin niiden metodeita **sayHello** ja **useSuperPower**. Näin varmistetaan, että sekä **Person**-luokan että **SuperHero**-luokan metodit toimivat oikein perinnän kautta.

## 10. Revealing Module pattern and IIFE. (4 \* 0,5 = 2 points)

```
const greeter = (function () {  
  let greeting = 'Hello';  
  const exclaim = msg => `${msg}!`;  
  const greet = name => exclaim(`${greeting} ${name}`);  
  const salutation = (newGreeting) => {  
    greeting = newGreeting;  
  };  
  return {  
    greet: greet,  
    salutation: salutation,  
  };  
})();
```

Look at the code above.

a. What is the idea of the code? What is the extra value it produces to JavaScript?

Tämän koodin idea on luoda moduuli käyttäen paljastavaa moduulipohjaa (Revealing Module pattern). Moduulipohjassa määritellään yksityisiä ja julkisia toimintoja ja ominaisuuksia. Yksityisiä ovat **greeting**-muuttuja ja **exclaim**-funktio, jotka eivät ole suoraan saavutettavissa ulkopuolelta. Julkisia ovat **greet**- ja **salutation**-funktiot, jotka ovat ulkopuolelta saavutettavissa ja tarjoavat rajapinnan moduulin toiminnallisuuteen.





Tämä moduulipohja tuottaa lisäarvoa JavaScriptille, koska se mahdollistaa tietyn toiminnallisuuden piilottamisen yksityiseksi, mikä auttaa välttämään nimiavaruuskonflikteja ja tekee koodista organisoitua ja helpommin ymmärrettävää.

b. What is IIFE?

IIFE tarkoittaa "Immediately Invoked Function Expression". Se on JavaScriptin toimintotapa, jossa funktio määritellään ja kutsutaan välittömästi sen määrittelyn jälkeen. IIFE:itä käytetään yleisesti moduulien luomiseen ja suojaamaan muuttujia ja toimintoja, jotka eivät ole globaalisti näkyviä.

c. Use object greeter. Call its functions. Try to read and set the greeting attribute without calling a method. What do you notice?

```
Terminal  Help
JS A.js ...T6 U  JS B.js  U  JS A.js ...T7 U  JS A.js ...T8 U  JS A.js ...T9 U  JS esimerke
JavaScript > Koodit > JS esimerkejä.js
1 console.log(greeter.greet("Alice")); // Tulostaa "Hello Alice"
2 greeter.salutation("Hi"); // Asettaa tervehdyksen uudeksi "Hi"
3 console.log(greeter.greet("Bob")); // Tulostaa "Hi Bob!"
4 console.log(greeter.greeting); // Tulostaa undefined, koska greeting ei ole määritelty
5
```

d. Look at the object the function returns. Use property value shorthands to make it a bit less verbose. Do you lose anything when using the shorthands?

Voi käyttää property value shorthand -syntaksia (lyhennettyä syntaksia) objektin ominaisuuksien määrittelyssä, joka tekee koodista hieman vähemmän verbose (tavallisesti lyhyempiä ja helpommin luettavia). Tässä on päivitetty versio greeter-objektista käyttäen property value shorthand -syntaksia:



JS A.js ...T6 U JS B.js U JS A.js ...T7 U JS A.js ...T8 U JS A.js ...T9 U JS esimerkejä.

JavaScript > Koodit > JS esimerkejä.js > ...

```
1  const greeter = (function () {  
2      let greeting = "Hello";  
3      const exclaim = (msg) => `${msg}!`;  
4      const greet = (name) => exclaim(`${greeting} ${name}`);  
5      const salutation = (newGreeting) => {  
6          greeting = newGreeting;  
7      };  
8      return {  
9          greet,  
10         salutation,  
11     };  
12 })();  
13
```