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” (3rd 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?

Two different objects *can* have different properties, you can change them if needed.

One object takes one memory"slot".

b. The keyword `new`.

The "new" operator lets you create an instance of object.

Creates a blank javascript object, and then add possible properties to that object.

c. The keywod `this`.

Refers to an object. If "this" is in object, it refers to the object, if its alone it is global object same when it is within function.

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

Other one is array; other one is object.

```
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`.

```
> let { name, instrument } = musician;
< undefined
> name
< 'Sting'
> instrument
< ▶ {type: 'bass'}
```

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

```
> let { name: nameOfArtist, instrument: instrumentOfArtist} = musician;
< undefined
> nameOfArtist
< 'Sting'
> instrumentOfArtist
< ▶ {type: 'bass'}
```

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

```
> let {instrument: { type: instrumentTypeOfArtist}}=musician;
< undefined
> instrumentTypeOfArtist
< 'bass'
```

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”.

```
> let {instrument: { make: instrumentMakeOfArtist ="unknown" }} = musician;
< undefined
> instrumentMakeOfArtist
< 'unknown'
```

3. For in and for of loops. (4 * 0,5 = 2 points)

a. Explain for..in loop.

Usually in use with objects.

b. Explain for..of loop.

Usually in use with arrays or iterable objects

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

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


for (const attribute in musician) {
  console.log(`${attribute}: ${musician[attribute]}`);
}
```

d. Use for..of with an array. Give an enlightening example of your own choice.

```
const colors = ["red","blue","yellow","orange"]
colors.push("green")

for (const item of colors) {
  console.log(`The color is ${item}`);}
```

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.

```
const song = {
  time:5,

  get duration() {
    return this.time;
  },

  set duration(newTime) {
    this.time =newTime;
  },


};



song.duration=3;

console.log(song.time)
```

b. Invoke the setter and the getter.

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

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

a. What are the purposes of JSON?

JavaScript Object Notation (JSON). It is used for transmitting data in web applications.

Text-based format.

In theory JSON is a string of JavaScript object.

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

Main difference is JSON has double quotes.

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

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

```
let personString = JSON.stringify(person)
```

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

```
let person = { name: "Pentti", age: 25, country: "Finland"}


//Json object literal
//Has to be with "" except numerals
//Convert JavaScript object to Json object
let personString = JSON.stringify(person)

//Convert from Json object to JavaScript object
let reCreatedPerson = JSON.parse(personString)
```

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

a. 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 also be an integer and it sets the end of the Range. The function returns a random integer that is greater or equal to the `startRange` and less or equal to the `endRange`.

```
function getRandomIntegerFromRange (startRange, endRange) {
    return Math.round(Math.random() * (endRange - startRange) + startRange);
}


getRandomIntegerFromRange(1,100)
```

b. Create a function `getTimeDifferenceInFullDays` that returns the number of full days between to 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.

```
function getTimeDifferenceInFullDays (startDate, endDate) {
    var date1 = new Date(startDate);
    var date2 = new Date(endDate);

    var difference = date2.getDate() - date1.getDate()-1;

    return difference


}
```

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 to arguments: `name` and `age`. There should be two functions tacked on the class: `getName`, `getAge` and `sayGreeting`.

b. What is the idea of a constructor?

To give object values.

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

```
class Person {

    constructor(name, age) {
        this.name = name;
        this.age = age;
    }

    getName() {
        return this.name;
    }
    getAge() {
        return this.age;
    }

    sayGreeting() {
        console.log("Greetings, world!");
    }
}


const person1 = new Person("Pekka","20")

const person2 = new Person("Steve","55")

person1.getName()

person2.sayGreeting()
```

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.

```
class Zipvalidator {



    static isValidZip (zipCode) {
       return /^\d{5}?$/.test(zipCode);
    }
    static fixZip (zipCode) {
        let code=zipCode.toString()
        console.log(code.padStart(5,'0'));
    }

}
```

b. Use the class and call the static methods.

```
> console.log(Zipvalidator.isValidZip(5))
  false
< undefined
> console.log(Zipvalidator.isValidZip(50000))
  true
< undefined
> console.log(Zipvalidator.fixZip(5))
  00005
  undefined
< undefined
```

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

Static methods are called on class, not on object/instance. If done so, it returns “not a function”

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)

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

You use super to inherit/initialize previous constructor.

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

```
> class Person {

      constructor(name, age) {
          this.name = name;
          this.age = age;
      }

      getName() {
          return this.name;
      }
      getAge() {
          return this.age;
      }

      sayGreeting() {
          console.log("Greetings, world!");
      }
  }

  class SuperHero extends Person {
      constructor(name, age, superpower) {
        super(name, age);
        this.superpower = superpower;
      }

      useSuperPower() {
        console.log(this.superpower);
      }
    }

< undefined
> const person1 = new SuperHero("Pekka","20", "Flight")
< undefined
> const person2 = new SuperHero("Steve","55", "Super Strength")
< undefined
> person1.useSuperPower()
  Flight
< undefined
> person2.sayGreeting()
  Greetings, world!
< undefined
```

10. Revealing Module pattern and IEFE. (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?

Its function is called immediately greeter is called. It frees variables after it has run. You can also use it to create private and public variables/methods

b. What is IIFE?

Immediately Invoked Function Expression

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

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?