JavaScript Objects

Code examples available at https://github.com/lassehav-oamk/js-primer

Understanding the concept of objects in JS is a key skill for any JS programmer.

Object is a collection of properties and each property has a name and a value. The property value can be of any primitive type or a function. Object is different from a basic variable that whereas a basic variable can hold only a single value, an object can hold multiple values idenfied by names – these are known as properties.

1. Creating new objects

Object is referenced and accessed through a variable. Object can be created in different ways, below is a n example demonstrating two different ways:

```
let car = new Object();
car.manufacturer = "Mercedes Benz";
car.model = "E";
car.type = "Sedan";

let anotherCar = {
  manufacturer: "Audi",
  model: "A4",
  type: "Estate"
};
```

objects/1-basics.js

2. Adding properties

Object properties can be added any time as seen already in the above example.

Object properties are added simply by using the dot notation with property name and assigning some value to it.

3. Access properties

The object properties are accessed via dot notation or bracket notation as seen in example below where object properties are printed to console.

```
console.log(car.type); // Sedan
console.log(anotherCar['manufacturer']); // Audi
```

objects/1-basics.js

4. Iterate or loop over all object properties

There are couple of ways how to iterate or loop over all properties in objects.

The for..in loop (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/for...in) works as seen below:

```
// Iterate over properties
for(prop in car)
{
  console.log(prop);
}
```

objects/1-basics.js

Exercise 1 (objects/exercises/1-iterate-object.properties.js): Create your own vehicle object, which has the following properties

- type – truck
- tireCount – 6
- *capacity* – 5 tons

Then use the for .. in loop to print out all the property names and their values of your object.

Bonus Exercise 1.1: Create an object to describe a person. The object should have the necessary properties to contain information of name, sex, birthday and profession. Create a function, which prints out the information of a person object.

Bonus Exercise 1.2: Modify you person object so that it can contain information of the parents of the person – mother and father. Create a function, which takes a person object as a parameter and prints out information of the parents.

5. Constructor functions

In addition to the ways of creating new objects seen in chapter 1 you can define new objects, by using constructor functions. After defining the object, you can create an instance of it by using the keyword *new*.

Example below.

```
function Animal(type, speed, isCute)
{
  this.type = type;
  this.speed = speed;
```

```
  this.isCute = isCute;
}

let alligator = new Animal('alligator', 'slow', false);
let dog = new Animal('dog', 'fast', true);
```

objects/2-constructor-functions.js

Exercise 2 (objects/exercises/2-constructor-function.js): Create a car object definition and then create two instances of it – one for Audi and one for Mercedes. In addition use the for..in loop to list print all the properties to the console.

6. Functions or methods in an object

Method is a function associated with an object.

You can define object methods with following syntax.

```
let example = {
  foo: "bar",
  sum: function(a, b) {
    return a + b;
  }
}

console.log(example.foo);
console.log(example.sum(4, 5));
```

objects/3-object-methods.js

For objects created with constructor functions the process is almost similar. It is done by simply assigning a new property to *this* and the property value is a function. See an example below.

```
function AnotherAnimal(type)
{
  this.type = type;
  this.printWhoAmI = function() {
    console.log(this.type);
  }
}

let dog = new AnotherAnimal('Dog');
let bird = new AnotherAnimal('Bird');
```

```
dog.printWhoAmI();
bird.printWhoAmI();
```
objects/3-object-methods.js

Exercise 3 (objects/exercises/3-object-methods.js): Create a new *Car* object, which has the following properties

- Brand – string, name of the car manufacturer. For example 'BMW'.
- Registration – string, unique registration code of the car. For example 'ABC-123'
- Speed – integer, current speed of the car. Initial value 0.

The object should have also two methods

- increaseSpeed – method increases the speed of the car by 5
- displaySpeed – prints the car registration number and its current speed

Bonus Exercise 3.1: Create an Animal constructor function as seen in the example *objects/2-constructor-functions.js*, add a property called *strength*. Next create a function *Battle* to battle another Animal given as parameter to the function. Each Animal object should have the *Battle* function in it, so it should be a method of the Animal.
The Battle function should use the *strength* properties to decide which animal wins the battle. Bigger is better. Add a small random factor to the battle so that it is not just a comparison, which animal has greater strength.

Bonus Exercise 3.2: Use the code from above bonus exercise and enhance it so that a view is presented to the user in a web browser, which allows the user to define the values for animals using form elements and a tournament length in rounds. Then there should be a button start a battle. The result of each battle should be stored and presented. The tournament should end and winner should be declared when there has been as many rounds as specified in the tournament length.

7. ES6 Class syntax for objects

JavaScript does not have similar object oriented class features as C++ for example as it is an so called prototype based language. Each object has an prototype object, which is sort of inheritance, since the properties and methods of the prototype object are available to the 'child' object as well.

From a programmer viewpoint building inheritance with protype objects in JS is a bit complex.

To make programmers life easier in EcmaScript 6 (the standard on which JavaScript is based on) version a new class syntax was introduced. With the class syntax you can build similar class constructs to C++ even though under the hood the original JS prototype based approach is still used to implement the classes.

Below is an example of a class and object created from the class. A class has a constructor method, which is invoked when you create an object from the class. The constructor method can use *this* keyword to bind properties to the object.

Class methods are defined without keyword function. Class methods can have parameters like every other function.

```
class Car {
  constructor(brand, registration)
  {
    this.brand = brand;
    this.registration = registration;
    this.speed = 0;
  }

  increaseSpeed()
  {
    this.speed += 5;
  }

  displaySpeed()
  {
    console.log(this.registration + ", speed " + this.speed);
  }
}


let audi = new Car("audi", "abc-123");
audi.displaySpeed();
audi.increaseSpeed();
audi.displaySpeed();
```

objects/4-classes-es2015.js

Exercise 4 (objects/exercises/4-classes.js):
Create a class for *Food* which allows the user to set the type of food and quantity. The class should have a method, which can be used to eat the food and quantity will decrease accordingly and a method to be able to describe what type of food it is.

With the following code

```
let banana = new Food('Banana', 4);
```

```
banana.whatIsThis();
banana.eatOne();
banana.eatOne();
banana.eatOne();
banana.eatOne();
banana.eatOne();
```

objects/exercises/4-classes-ex1.js

You should get this output

```
Banana
Slurp! One banana eaten. 3 remaining
Slurp! One banana eaten. 2 remaining
Slurp! One banana eaten. 1 remaining
Slurp! One banana eaten. 0 remaining
Sorry, no more bananas remaining!
```

Exercise 5: Create a class for *Refrigerator* which is capable of storing food. The class should have couple of different methods.

- putFood – stores food in refrigerator
- getAndEatFood – gets and consumes the specified food and amount from refrigerator, indicates if there is no such food available, indicates if last food of the type was eaten
- getContents – display list of what food and how many is inside the refrigerator

Fill your refrigerator with different Foods and then eat them. Validate the everything works as intended.

The following code

```
let r = new Refrigerator();
let apple = new Food('Apple', 2);
let bananas = new Food('Banana', 3);
r.putFood(apple);
r.putFood(bananas);
r.getContents();
r.getAndEatFood('Apple');
r.getAndEatFood('Apple');
r.getAndEatFood('Banana');
r.getAndEatFood('Apple');
r.getContents();
```

Should give about the following output:

```
-----------
| Apple 2
| Banana 3
-----------
Slurp! One apple eaten. 1 remaining
Slurp! One apple eaten. 0 remaining
Slurp! One banana eaten. 2 remaining
Sorry, no such food in this refrigerator!
-----------
| Banana 2
-----------
```

8. Inheritance with ES6 Classes

Inheritance is supported by class syntax with keyword *extends*.

In the previous example there was a class *Car* with some properties and methods. To create a new class which inherits the methods and properties from this class *exteds* keyword must be used.

The following example demonstrates how to create an *RaceCar* class, which inherits, or exteds, the *Car* class.

```
class RaceCar extends Car {
  constructor(type, brand, registration)
  {
    super(brand, registration);
    this.type = type;
  }

  makeNoise()
  {
    console.log('Brrum brrum');
  }

  increaseSpeed()
  {
    this.speed += 10;
  }
}

let formula = new RaceCar("F1", "Ferrari", "F-1");
formula.makeNoise();
formula.displaySpeed();
formula.increaseSpeed();
formula.displaySpeed();
```

objects/5-inheritance-es2015.js

In the example above notice the *super* method call in the constructor. The *super* executes the constructor of the class, which this class exteds and therefore it should be the first to be called in constructor.

Notice also the *increaseSpeed* method, which is an overriding method since there was a method with same name in the *Car* class. The *increaseSpeed* method in *RaceCar* increases speed more than the original method in *Car* class.