Objects

IIKG1002/IDG1011 – Front-end web development

Johanna Johansen
johanna.johansen@ntnu.no

- used to create models of the real world; e.g., people, cars, trees, hotels
- are grouping together a set of variables and functions
- variables are **properties**
  - properties → characteristics of obj. from the real world
- functions are **methods**
  - methods → how people (or other things) interact with an object in the real world

Creating objects

- Two ways of creating objects
  - the literal notation and object constructor notation
- **literal notation**
  - create first the object, then add properties and methods to it

```
let hotel = {}; // creates an empty object

// adds properties to the object
hotel.name = 'Thon';
hotel.rooms = 40;
hotel.booked = 25;
 // adds a method to the object
hotel.checkAvailability = () => {
   return this.rooms - this.booked;
};
```

The dot notation

- we add properties/methods to the object by using the **dot notation**

```
hotel.name = 'Thon';
```

  - the name of the object, followed by period, then the name of the property/method
  - the period is known as the **member operator**
    - the property/method on its right is a member of the object itself

Creating objects

- **literal notation**
  - creating the object with properties and methods
- ***this*** keyword used inside the method refers to the containing object; i.e., the `hotel` object

```
let hotel = {
    name = 'Thon',
    rooms = 40,
    booked = 25,
    checkAvailability = () => {
      return this.rooms - this.booked;
      // 40 - 25
    }
};
```

Name/value pairs

```
let hotel = {
  name = 'Thon',
  ...
}
```

- The object contains name/value pairs;
  - names are referred to as keys
  - key → name
  - value → 'Thon'
- Name/value pairs used a lot in programming
  - HTML → attribute names and values

    `<p class=“fruit”>peach</p>`
  - CSS → property names and values

    `.fruit {color: pink;}`

Name/value pairs

- In JavaScript
  - Variables
    - you give them a name when you declare them
    - you assign them a value that can be a string, number, or Boolean

```
let hotel = 'Thon';
```

  - Arrays
    - have a name and a group of values
      - each item in the array is a name/value pair because each value is associated with an index

```
let hotels = ['Thon', 'Scandic', 'Grand', 'Radisson Blu'];
```

Name/value pairs

- In JavaScript
  - Named functions
    - have a name
    - the value is the set of statements to be run when the function is called

```
function updateMessage() {
  let el = document.getElementById('welcomeMessage');
  el.textContent = message;
}
```

Creating objects

- **object constructor notation**
  - uses the *new* keyword and the *Object()* constructor function
  - the *Object()* function is part of the JavaScript language, used to create objects
  - all constructor functions start with a capital letter
  - similarly to the literal notation, we add properties/methods using the *dot notation*
  - *this* keyword

    `return this.rooms – this.booked`

    is similar to

    `return hotel.rooms – hotel.booked`

```
/**
  * creating an empty object with the literal
  notation:
  * var hotel = {};
  */

var hotel = new Object(); // creates an empty
object


// adds properties to the object
hotel.name = 'Thon';
hotel.rooms = 40;
hotel.booked = 25;
 // adds a method to the object
hotel.checkAvailability = () => {
   return this.rooms - this.booked;
};
```

Updating an object

- We use the same method for both the literal and the constructor notation
- updating is done in the same way as with adding properties/methods
  - by using the **dot notation,** but giving a new value
  - e.g., to change the value of the name property of the hotel object

```
hotel.name = 'Radisson Blu';
```

- NOTE: if the object does not have the property you are trying to update, it will be added to the object

```
hotel.pool = true;
```

```
let hotel = {
   name = 'Radisson Blu',
   rooms = 40,
   pool = true,
   booked = 25,
   checkAvailability = () => {
      return this.rooms -this.booked;
   }
};
```

Updating an object

- by using the **square bracket syntax**
  - only for the properties, but not the methods

```
hotel['name'] = 'Radisson Blu';
```

  - looks very similar to how you access the items in an array
    - instead of using an index number, you are using the name of the property

- more complex examples (Flanagan, 2020, p. 131)

```
let empty = {};                        // an empty  object
let point = { x: 0, y: 0};             // numeric values
let p2 = { x: point.x, y: point.y+1 }; // more complex values
let book = {
  'main title': 'JavaScript',          // the property name includes spaces
  'sub-title': 'The Definitive Guide', // the property name includes hyphens
  for: 'all audiences',                // for is a reserved word, with no quotes
  author: {                            // the value of this property is itself an object
    firstname: 'David',
    surname: 'Flanagan'
  }
};
```

- **Dot notation** or square bracket syntax?
  - we use the *dot notation* when the property name is a **legal identifier**
    - an *identifier* is simply a name
    - identifiers are used to name constants, variables, properties, functions, and classes
    - the identifier must begin with either a letter, an underscore (_), or a dollar ($) sign, subsequent characters can be letters, digits, underscores, or dollar signs

```
object.identifier
```

- Dot notation or **square bracket syntax**?
  - `...`
  - *square bracket syntax* is used
    - if the property name includes spaces, punctuation characters, is a number, or
    - if the property name is not static, but is itself the result of a computation
  - with the *bracket syntax*, the property name is evaluated as an expression and converted into a string

    ```
    object[expression]
    ```

  - the *bracket syntax* <u>cannot</u> be used to update/add methods

```
let book = {
  'main title': 'JavaScript',
  'sub-title': 'The Definitive Guide',
  for: 'all audiences',
  author: {
    firstname: 'David',
    surname: 'Flanagan'
  }
};

book.edition = 7; // Create an “edition“ property of book
book['main title'] = 'ECMAScript'; // Change the “main title” property
```

(Flanagan, 2020, p. 133)

Deleting properties

- we use the *delete* keyword

```
delete hotel.name;
```

- to clear the value of a property, use a blank string

```
hotel.name = '';
```

Accessing objects

- Objects can be access by using the *dot notation* and *square bracket syntax*
  - the same rules apply as for the case of updating/adding objects

```
let hotelName = hotel.name; // 'Radisson Blu'

let hotelName = hotel['name']; // 'Radisson Blu'

let roomsFree = hotel.checkAvailability(); // 15
```

```
let hotel = {

   name = 'Radisson Blu',

   rooms = 40,

   pool = true,

   booked = 25,

   checkAvailability =
   function() {

      return this.rooms -
      this.booked;

   }

};
```

Exercise

- Try out in your text editor the example from the page 112, Duckett syllabus book
  - source code: https://javascriptbook.com/code/c03/
- add, update and delete values of the *hotel* object
- open the Developer tools and see what happened with the *class attribute* of the *paragraph elements* with the *id attribute* of 'pool' and 'gym.'

- The *object literal notation* is good to use to work with individual objects
  - storing / transmitting data between applications
  - global and configuration objects that set up information for the page
- For cases where you need to create multiple objects within the same page, use the **object constructor notation**

Object constructor notation

- to create several objects representing similar things
  - they will have the same names for the properties and methods, but with different values
- a constructor is a function
  - the name of the constructor function usually begins with a **capital letter**
- use a function as a template for creating many objects

Object constructor notation

- very similar to creating a function that needs information (**parameters**)
- the parameters are set as values for the properties in the object

```
function Hotel(name, rooms, booked) {
    this.name = name;
    this.rooms = rooms;
    this.booked = booked;

    this.checkAvailability = () => {
        return this.rooms - this.booked;
    };
}
```

Object constructor notation

```
function Hotel(name, rooms, booked) {
   this.name = name;
   this.rooms = rooms;
   …
```

- ***This*** keyword is used instead of the object name
  - indicates that the property/method belongs to the object that this function creates

Object constructor notation

- when we call the constructor function preceded by the *new* keyword we create a new object
  - is an **instance** of object constructor
  - e.g., we create two hotels

```
let firstHotel = new Hotel('Thon', 40, 25);
let secondHotel = new Hotel('Radisson Blu', 120, 77);
```

  - similar with calling functions that need information, we give values (**arguments**) to be used for the properties of each hotel.

Object constructor notation

- Once we have created an object, we can **add/delete/update** its properties/methods similarly as for the objects created with the literal notation
  - we only add/remove properties/methods from the instance and not all other objects created with that function

Storing data

- We can store data in JavaScript using variables, arrays, and objects
  - in variables we can store one piece of information
    - that value can be retrieved by using the name of the variable
  - with arrays and objects we can store multiple pieces of information
    - arrays
      - the order of the information is important, because the items in an array are assigned a number (an index)
      - to retrieve an item we use the index number; e.g. `hotel[1]`;
      - if the order of the information matters, use arrays
    - objects
      - we access items via a key (property/method name) and the dot / square bracket notation; e.g., `hotel.name;` or `hotel['name']`;
      - the key needs to be unique
    - we can combine arrays, with objects to create complex data structures; we can have arrays in objects and objects in arrays, i.e., one array as the value of an object