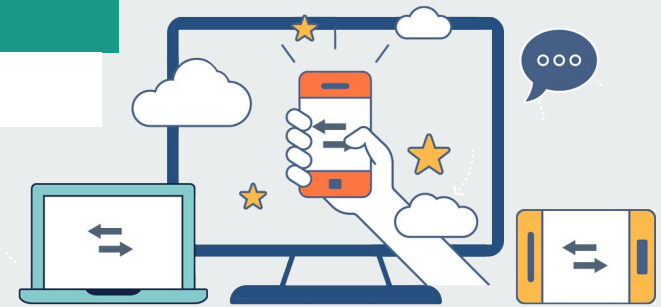
-Lecture 3-

Chapter 2 – **Advanced Javascript Concepts**

Part I

Adil **CHEKATI**, PhD

adil.chekati@univ-constantine2.dz



Prerequisites

- ❑ Basic JavaScript Proficiency.
- ❑ Basic Knowledge of Functions and Scope.
- ❑ Object-Oriented Programming

JavaScript-Advanced

Objectives

- ➔ Understand some advanced JavaScript concepts
- ➔ Understand higher-order functions and callbacks
- ➔ Identify JavaScript iterators and "this" management.

1. Nested data structures

Objects within objects

Nesting allows you to place objects inside other objects at multiple levels.

Example :

```
var schools = {
 IT: {
 address: "North Ave NW, Atlanta, GA 30332",
 phoneNumber: "(404) 894-2000",
 dateEstablished: "October 13, 1885"
 }
};
var gtObject = schools.IT;
```

1. Nested data structures

To access the internal object, we can use dot notation: with the variable **gtObject** we can access the object's attributes:

```
gtObject.address; // returns "North Ave NW, Atlanta, GA 30332"
gtObject.phoneNumber; // returns "(404) 894-2000"
gtObject.dateEstablished; //returns "October 13, 1885"
// We can directly use schools
schools.IT.address;
schools.IT.phoneNumber;
schools.IT.dateEstablished;
```

1.1 Complex objects

Example :

```
var instructorData = {
    name: "Tim",
    favoriteHobbies: ["Sailing", "Hiking", "Coding"]
   };
```

The access method is as follows:

```
instructorData.favoriteHobbies[0]; // returns "Sailing"
```

1.1 Com

Example combining objects and arrays :

```javascript
var instructorData = {
    name: "Elie",
    additionalData: {
     instructor: true,
     favoriteHobbies: ["Playing Chess", "Tennis", "Coding"],
     moreDetails: {
      basketballTeam: "NYJ",
      numberOfSiblings: 3,
      isYoungest: true,
      hometown: {
       city: "West Orange",
       state: "NJ",
      },
    citiesLivedIn: ["Seattle", "Providence", "New York"]
     }
    }
   };
instructorData.name; // "Elie"
instructorData.additionalData.instructor; // true
instructorData.additionalData.favoriteHobbies[2]; // "Coding"
instructorData.additionalData.moreDetails.fbasketballTeam; // "NYJ"
instructorData.additionalData.moreDetails.hometown.state; // "NJ"
instructorData.additionalData.moreDetails.citiesLivedIn[1]; //"Providence"
```

*The use of complex objects is **common practice** in JAVASCRIPT.*

1.2. Accessing and modifying the contents of nested objects

- **Dot Notation** is useful when attribute names are known.

- If you don't know the exact names of the attributes, use **Bracket Notation**.

1.2. Accessing and modifying the contents of nested objects

```javascript
var programmingLanguages = {
    java: {
     yearCreated: 1995,
     creator: "James Gosling"
    },
    javaScript: {
     yearCreated: 1995,
     creator: "Brendan Eich"
    }
}

function addProgrammingLanguage(nameOfLanguage, yearLanguageCreated, creatorOfLanguage){
    programmingLanguages[nameOfLanguage] = {
     yearCreated: yearLanguageCreated,
     creator: creatorOfLanguage
    }
}
    addProgrammingLanguage("ruby", 1995, "Yukihiro Matsumoto");
```

In the following example, we'd like to write a function that adds an attribute (in this case **nameOfLanguage**) to the programmingLanguages embedded object.

2. Higher order functions

- JS functions are a fundamental part. They can accept any type of parameter types: (**strings, numbers, booleans, arrays objects**).
- Functions can also be passed as parameters to other functions!
- A function whose parameters include another function is called a "**higher order**" function. This is specific to JavaScript.
- The function in parameter is called a **Callback function**.

2. Higher order functions

In the following example, the **sendMessage** function accepts a *String* and a *function* as parameters.
The **sendMessage** function will return the result of the function passed to it as parameter, with the message (*string*) as argument:

```
function sendMessage(message, fn){
        return fn(message);
}
sendMessage("FREE PALESTINE", console.log); // FREE PALESTINE
sendMessage("FREE PALESTINE", alert); // FREE PALESTINE is alerted
sendMessage("We Stand with Ghaza!", prompt); //value from prompt is returned
sendMessage("Do you know Abu Obaida?", confirm);//true or false is returned
```

2. Higher order functions

- It is important to differentiate between a referenced function and an invoked function.
- In **sendMessage("FREE PALESTINE", console.log)**; console.log is a function referenced but not yet not yet invoked. Nothing is written to the console until the function is invoked.
- A function passed as a parameter to a **higher-order** function must be referenced only by its name (which is why we don't see the function's parameters in the invocation).
- It is within the **higher-order** function that the referenced function will be invoked (invoked with parameters between ( )) .
- In the previous example, the line **return fn(message)**; shows the invocation of the function, message will be displayed on the console.

2.1 Anonymous functions as parameters

We can pass an anonymous function as a parameter.

```
sendMessage("Hello World", function(message){
    // message refers to the string "Hello World"
    console.log(message + " from a callback function!");
}); // Hello World from a callback function!
```

2.1 Anonymous functions as parameters

The previous example is equivalent to:

```
var myFunction = function(message){
    // message refers to the string "Hello World"
    console.log(message + " from a callback function!");
    };

sendMessage("Hello World", myFunction);
```

*In JS, **anonymous** functions are frequently passed as parameters.*

2.2 Why higher-order functions?

Usefulness: Code reuse.

In the previous example, we'd have to write a lot of code to get the same result.

Higher-order functions mean we don't have to write separate functions for each case. as illustrated:

```javascript
function sendMessageWithConsoleLog(message){
    return console.log(message);
}

function sendMessageWithAlert(message){
    return alert(message);
}

function promptWithMessage(message){
    return prompt(message);
}

function confirmWithMessage(message){
    return confirm(message);
}

function sendMessageWithFromCallback(message){
    return console.log(message + " from a callback function!");
}
```

2.2 Why higher-order functions?

Instead of writing 5 different functions, we can write just one and pass it another function as a parameter.

This function is called a **callback function.**

Exemples:

```javascript
function add(a,b){
    return a+b;
}

function subtract(a,b){
    return a-b;
}

function math(a,b,callback){
    return callback(a,b);
}

math(1,4,add); // returns 5
math(5,5,subtract); // returns 0
```

2.2 Why higher-order functions?

Test:

Write an **each** function that accepts 2 parameters: an array and a callback function. The function must loop over the array and apply the callback function to each of its elements.

```
function each(){
    // your code HERE !
}

each([1,2,3,4], function(val){
console.log(val);
});
```

2.2 Why higher-order functions?

Solution:

```
function each(array, fn){
    for(var i=0; i< array.length; i++){
        fn(array[i]);
    }
}
```

3 JavaScript iterators

Predefined JS methods for processing arrays, used to optimize code optimize code size and readability.

We'll look at the iterators: **forEach, map, filter, reduce.**

3 JavaScript iterators

a) forEach:

Its first parameter is a callback function with a maximum of 3 arguments: *value, index and array.*

```
var arr = [4,3,2,1];
   arr.forEach(function(val,index,arr){
   console.log(val);
});

// 4
// 3
// 2
// 1
```

3 JavaScript iterators

a) forEach:

Its first parameter is a callback function with a maximum of 3 arguments: *value, index and array.*

forEach always returns **undefined.**

```
var doubledValues = arr.forEach(function(val,index,arr){
    return val*2;
    });

doubledValues; // undefined
```

3 JavaScript iterators

b) map:

Unlike forEach, map returns an array of the values returned in the callback.

The callback function is the same as forEach: *value, index and array* (in that order).

```
var arr = [1,2,3,4];
arr.map(function(val, index, array){
return val * 2;
}); // [2,4,6,8]


var tripledValues = arr.map(function(val,index,arr){
return val*3;
});
tripledValues; // [3,6,9,12]
```

3 JavaScript iterators

b) map:

Another way of using map to double array values:

```
function doubleArray(arr){
    // return the result of arr.map
    return arr.map(function(val){
    // return a new array with each value doubled
      return val *2;
    });
}

doubleArray([2,4]); // [4,8]
```

3 JavaScript iterators

d) reduce:

The aim of reduce is to reduce an array to a single value of any type (*string*, *boolean*, *object*, *array etc.*)

There are 4 arguments to the callback function: the first 3 are identical to those of map or forEach, but not all are mandatory; the 4th, called start, previous, is an accumulation of the results of the various iterations.

3 JavaScript iterators

d) reduce:

Example:

```
var arr = [2,4,6,8];
arr.reduce(function(acc,next){
return acc + next;
},5);
/*
1st iteration, acc = 5 & next = 2; callback returns 5 + 2 = 7.
2nd iteration, acc = 7 & next = 4; callback returns 7 + 4 = 11.
3rd iteration, acc = 11 & next = 6; callback returns 11 + 6 = 17
last iteration, acc = 17 & next = 8; callback returns 17 + 8 = 25.
```

4 'this' keyword

this is a JavaScript keyword indicating an object.

On the Chrome console, if you type this, the result is the window object.

But in general, the value indicated by the this keyword depends on where it's used in the code.

The 4 ways of using the '**this**' keyword:

a) default binding/global context,
b) implicit binding,
c) explicit binding,
d) new binding

4 'this' keyword

a) Default Binding/ Contexte Global:

In the global context, this refers to the window object

```
var thing = this;
thing; // window

function outer() {
    return this;
}
outer(); // window
```

*For other modes, the value of this has a different value when used **inside** a function.*
*Its value is assigned according to **where and how** the function is called.*

4 'this' keyword

b) Implicit/Object

In an object, this indicates the nearest parent object.

Example:

```javascript
person.sayHi(); // Hi Omar
person.determineContext(); // true
person.dog.sayHello(); // Hi undefined
person.dog.determineContext(); // false
```

```javascript
var person = {
    firstName: "Omar",
    sayHi : function (){
      return "Hi " + this.firstName
    },

    determineContext: function() {
      return this===person;
    },

    dog: {
    sayHello : function (){
      return "Hi " + this.firstName
    },

    determineContext: function() {
      return this===person;
    },
    }
}
```

4 'this' keyword

c) Explicit binding

In this case, the user indicates what this corresponds to, using 3 methods: *call, apply or bind.*

```
var Omar = {
    firstName: "Omar",
    sayHi : function (){
        return "Hi " + this.firstName
    }
}

var ali = {
    firstName: "Ali",
    }

Omar.sayHi(); // Hi Omar
Omar.sayHi.call(ali); // Hi Ali
```

4 'this' keyword

d) new binding

Used as a class constructor in object-oriented languages.

5 JavaScript and Object Oriented

JavaScript is not originally an OO language.

It has been given characteristics that make it make it similar to an OO language (like Java or C++) :
classes, class constructor and inheritance.

/ the following part is replaced by notion of class in ES6 in part 2 of chapter 2 */*

5 JavaScript and Object Oriented

5.1 'new' keyword

new allows you to create instances of object classes based on a constructor, which is a a special function in JS.

The use of new and the constructor reduces the amount of and avoids code repetition.

In the following example, there's a lot of code repetition:

```
var car1 = {
 make: "Honda",
 model: "Accord",
 year: 2002
}

var car2 = {
 make: "Mazda",
 model: "6",
 year: 2008
}

var car3 = {
 make: "BMW",
 model: "7 Series",
 year: 2012
}

var car4 = {
 make: "Tesla",
 model: "Model X",
 year: 2016
}
```

5 JavaScript and Object Oriented

5.2 Constructor function

This is an ordinary JS function, and should conventionally begin with an uppercase letter (good practice) to indicate that it's a constructor function.

```
function Car(make, model, year){
    this.make = make;
    this.model = model;
    this.year = year;
}
```

To construct instance objects, use **new:**

```
var probe = new Car('Ford', 'Probe', 1993);
var cmax = new Car('Ford', 'C-Max', 2014);
probe.make; // Returns "Ford"
cmax.year; // Returns 2014
```

5 JavaScript and Object Oriented

Role of the new keyword :

- Create an empty object that is referenced by **'this'** in the constructor function.

- Automatic addition of 'return this' at the end of the constructor function (no need to explicitly return a value) explicitly return a value.

5 JavaScript and Object Oriented

5.3 Classes

In JavaScript, the concept of class didn't exist; it was "imitated" to resemble classes in OO (existing in later versions of JavaScript), using constructor functions and invoked by new.

Inheritance is not explicit either, but it is ensured by the notion of prototype (for attributes and static methods) to borrow properties from one constructor and use them in another.

Inheritance can be created by using call.

5 JavaScript and Object Oriented

5.3 Classes

Example:

```javascript
function Vehicle(make,model,year){
    this.make = make;
    this.model = model;
    this.year = year;
}

function Motorcycle(make,model,year,motorcycleType){
    Vehicle.call(this,make,model,year)
    this.motorcycleType = motorcycleType;
}

var moto = new Motorcycle("Kawasaki", "Ninja 500", 2006, "Sports")
```

Lab Exercises Submission Guidelines

- **Deadline:**
  At the end of each Lab session (no later than Saturday at 23:59)
  To: adil.chekati@univ-constantine2.dz

- **File's Name to be submitted:**
  CAW_Lab%_Gr%_NAMEPair1_NAMEPair2.zip
  Example : "CAW_Lab1.part1_Gr1_CHEKATI_BOUZENADA.zip"

Textbook

➔ All academic materials will be available on:

Google Drive.
E-learning platform of Constantine 2 University.
Google Classroom.

aoa5lne

Google Classroom

SCAN ME!

References

➔ **Book:**
Haverbeke, Marijn - *Eloquent JavaScript: A Modern Introduction to Programming*- (2019)

Online Resource:
Mozilla Developer Network-"JavaScript Guide"
(https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide)

Next Lecture

-Lecture 4-

Chapter 2 – **Advanced JavaScript concepts**

Part II

Adil **CHEKATI**, PhD

adil.chekati@univ-constantine2.dz

Questions, & comments...

adil.chekati@univ-constatine2.dz