Ήχος στο Διαδίκτυο

JavaScript Objects, Classes and Functions

Χρυσούλα Αλεξανδράκη
Τμήμα Μουσικής Τεχνολογίας και Ακουστικής
Ελληνικό Μεσογειακό Πανεπιστήμιο

What are JS Objects?

- **Objects** are **variables** that can store both **values** and **functions**
- **Variables** reflect object properties
- **Functions** (aka **methods)** reflect object behaviour
- Example:

```
const person = {
    firstName: "John",
    lastName : "Doe",
    age      : 50,
    sayHello : function() {
            return "Hello I am " + this.firstName + " " + this.lastName;
    }
};
console.log(person.sayHello());
```

Object Constructors

- Sometimes we need to create many objects of the same **type**.
- To create an **object type** we use an **object constructor function**.
- It is considered good practice to name constructor functions with an upper-case first letter.
- Example:

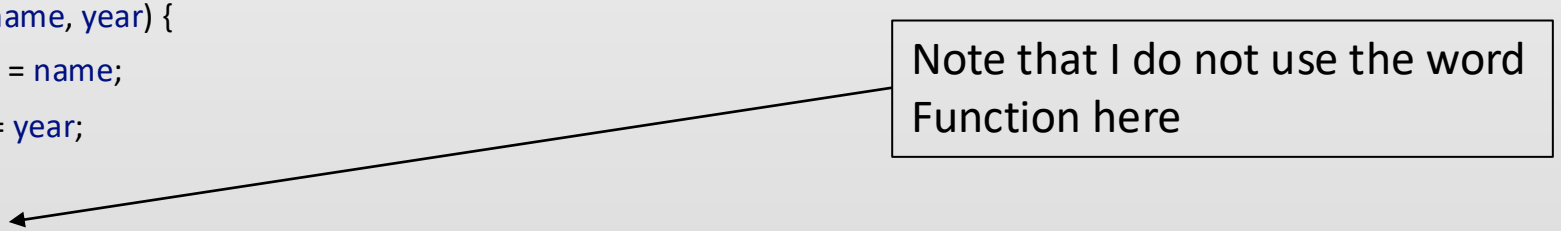
```
// Constructor function for Person objects
    function Person(first, last, age) {
        this.firstName = first;
        this.lastName = last;
        this.age = age;
        this.sayHello = function() {
            console.log(`Hello, my name is ${this.name} and I am ${this.age} years old.`);
        }
    }
    // Create 2 Person objects
    const myFather = new Person("John", "Doe", 50, "blue");
    const myMother = new Person("Sally", "Rally", 48, "green");
    myFather.sayHello();
    myMother.sayHello();
```

JS Classes

- Class is a prototype of objects in other words it defines the type of objects, in terms which attributes and methods objects will have
- Example

```
class Car {
    constructor(name, year) {
        this.name = name;
        this.year = year;
    }
    age() {
        // date = new Date(); // This will not work
        const date = new Date(); // This will work
        return date.getFullYear() - this.year;
    }
}

const myCar = new Car("Ford", 2014);
console.log("My car is " + myCar.age() + " years old");
```



Why do we have JS Classes?

- JavaScript (since 1995) is **prototype-based**, not class-based.
- Originally, objects were created using constructor functions
- Classes were introduced in **ES6 (2015)**.
  - They were added for:

    1. **Cleaner syntax**
    2. **Easier mental model (especially for Java/C++ programmers)**
    3. **Better structure for large applications**
    4. **Clearer inheritance syntax**

JS Functions

- Functions are one of the fundamental building blocks in JavaScript
- Every JS function is a Function object, and therefore has all properties and methods of this object
- In JS, Function are ***first-class objects***, which means, that they can be:
  - Passed as arguments to other functions
  - Returned from other functions
  - Assigned to variables and properties
  - They can have properties and methods like any other object
- The only difference of a function from an object is that functions can be called

Function Declarations vs Function Expressions

- JS Functions may be defined in two ways
  1. Standard declaration

```
function multiply1(a, b) {
    return a * b;
}
```

  2. Function Expression
     - Functions stored in variables **do not need names**.
     - The **variable name is used** to call the function.

```
const multiply2 = function(a, b) {
    return a * b;
};
```

Function expression ends with semicolon
Don't forget: this is a variable definition!!

```
console.log(multiply1(4, 5)); // Output: 20
console.log(multiply2(4, 5)); // Output: 20
```

Function Declarations vs Function Expressions

- In JavaScript, **function declarations** and **function expression** refer to different ways of defining functions
  - They have different syntax
  - They both work the same when you call them.
  - The difference is when they become available in your code.

Hoisting

- Function declarations can be called before they are defined.
- Function expressions can not be called before they are defined.
- **Function declarations** are "hoisted" to the top of their scope. This means you can call a function before it is defined in the code:

```
let sum = add(2, 3); // Will work

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

- But ---

```
let sum = add(2, 3); // ⛔ Will generate error

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

Arrow Functions

- **Arrow Functions** allow a shorter syntax for **function expressions**.
- You can skip the **function** keyword, the **return** keyword, and the **curly brackets**
- An arrow function is always written as a function expression
  - Therefore, cannot be used before defined
- Syntax:

```
const add = (a, b) => a * b;
```

OR

```
const add = (a, b) => {
        return a + b;
};
```

JS Callback Functions

- A **callback function** is a function passed as an argument into another function.
- A **callback function** is intended to be **executed later**.
- Later is typically when a specific event occurs or an asynchronous operation completes
- Two Types:
  1. Synchronous callbacks → executed immediately
  2. Asynchronous callbacks → executed at a later time

Παράδειγμα Synchronous Callback

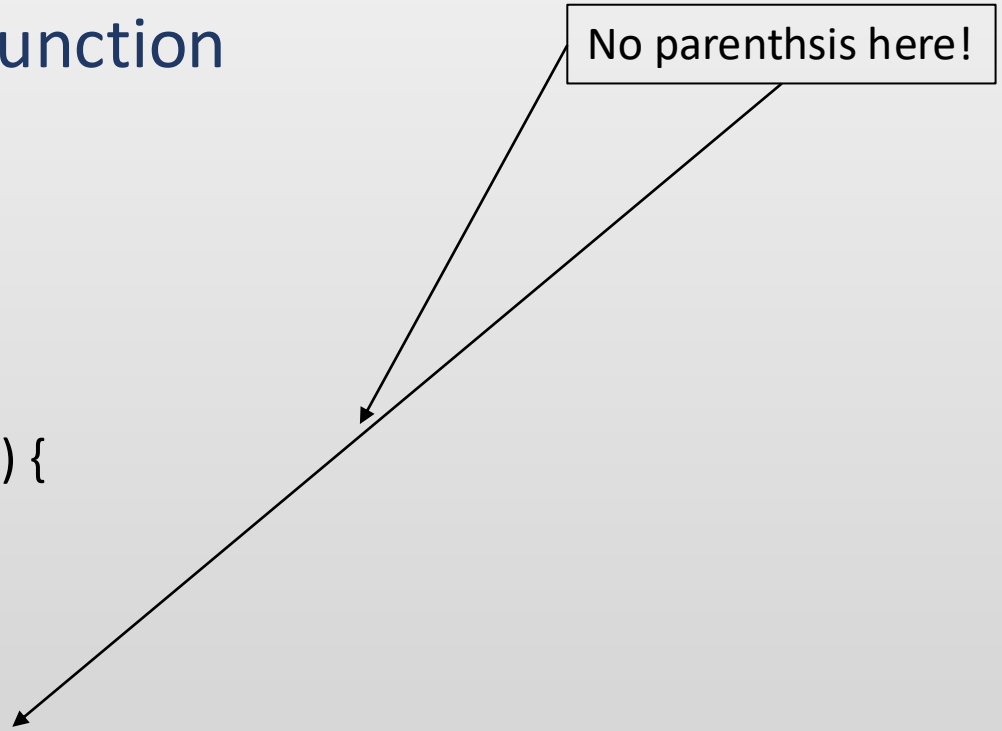
- myDisplayer() is used as a callback function

```
function myDisplayer(something) {
  console.log(something);
}


function myCalculator(num1, num2, myCallback) {

    let sum = num1 + num2;
    myCallback("to athroisma einai " + sum);
}

myCalculator(5, 5, myDisplayer);
```



Asynchronous Callbacks

- Do not disturb the execution of the main program
- They called is delayed until, e.g.:
  - An event has been triggered
  - Data, e.g. an audio stream, has been fetched
  - A file has been loaded
  - The results of a database query has been returned

```
function fetch (data, callback) {
    //do sth to fetch data
}


function display (data) {
    console.log(data);
}
fetch("some data", display);
```

Asynchronous callback after a time interval

- Delayed callback

```
console.log("Start");

function myFunction() {
   console.log("Inside myFunction");
}

setTimeout(myFunction, 2000);
console.log("End");
```

```
Start
End
Inside myFunction
»
```

HTML DOM API

- Reference:
  - https://developer.mozilla.org/en-US/docs/Web/API/HTML_DOM_API
- Κληρονομικότητα του HTML <audio> element:
  - HTMLElement
    - HTMLMediaElement
      - HTMLAudioElement
        - Audio
- Οι περισσότερες ιδιότητες και μέθοδοι για Audio() object interaction υπάρχουν στο HTMLMediaElement
  - https://developer.mozilla.org/en-US/docs/Web/API/HTMLMediaElement