JAVASCRIPT ADVANCED - ЧАСТ 2

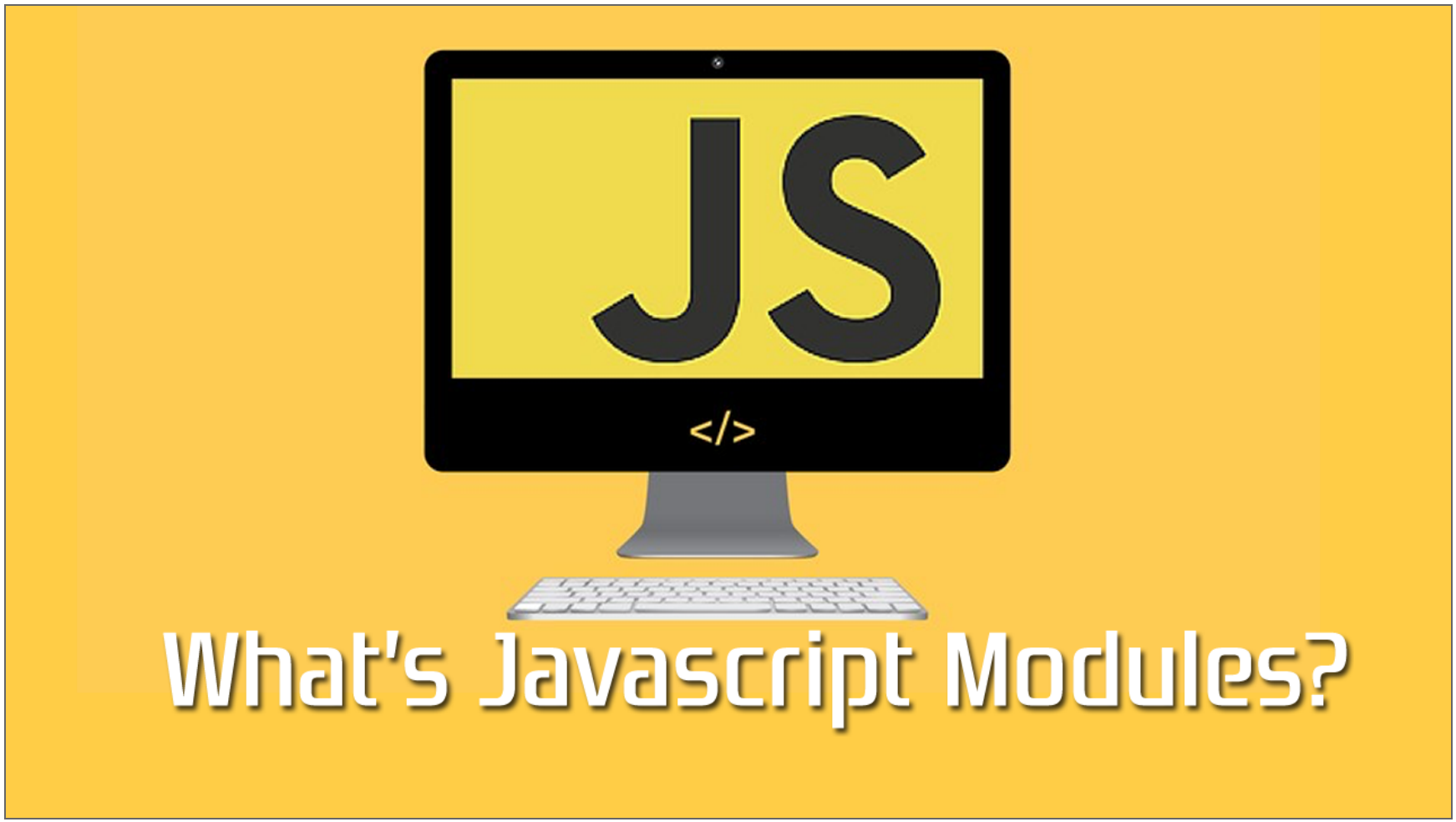
MODULES, MAP, WEAKMAP, SET, WEAKSET, ITERABLE AND ITERATOR

СЪДЪРЖАНИЕ

ДА ПРИПОМНИМ

КАКВО Е PROTOTYPE И PROTOTYPE CHAIN?

MODULES



ЗАЩО ДА ИЗПОЛЗВАМЕ МОДУЛИ?

- Лесни за поддръжка (Maintenability)
- Ползват локален обхват на видимост (scope)
- Преизползваеми са

COMMONJS И AMD

- ES5 няма вградени модули в езика.
- CommonJS
- AMD (Require.js)

“Writing Modular JavaScript With AMD, CommonJS & ES Harmony” by Addy Osmani

AMD ПРИМЕРИ

```
define(function () {
    return {
        addToCart: function () {
            //logic for adding to cart
        }
    }
})
```

```
define(["./cart"], function(cart) {
        return {
            addToCart: function() {
                cart.add(this);
            }
        }
    }
);
```

COMMONJS ПРИМЕРИ

```
function myModule() {
  this.hello = function() {
    return 'hello!';
  }

  this.goodbye = function() {
    return 'goodbye!';
  }
}

module.exports = myModule;
```

```
var myModule = require('myModule');

var myModuleInstance = new myModule();
myModuleInstance.hello(); // 'hello!'
myModuleInstance.goodbye(); // 'goodbye!'
```

MODULE BUNDLERS

- Webpack
- Browserify
- JSPM

ЗАЩО ИЗПОЛЗВАМЕ MODULE BUNDLERS?

- Комбинира всички модули в един файл
- Позволява използването на различни module формати

ES 6 МОДУЛИ

- Синтаксис подобен на CommonJS
- Поддържат асинхронно зареждане на модулите
- Подходящи за статичен анализ на кода и оптимизации

NAMED EXPORTS

- Множество наименувани exports

```
export const sqrt = Math.sqrt;
export function square (x) {
    return x * x;
};
export function cube(x) {
    return x * x * x;
};
```

```
import { square, cube } from './lib';
console.log(square(11)); // 121
console.log(cube(4)); // 64
```

```
import * as lib from './lib';
console.log(lib.square(11)); // 121
console.log(lib.cube(4)); // 64
```

DEFAULT EXPORT

- Един export

```
export default function square(x) {
    return x * x;
}
```

```
import square from './lib';
console.log(square(11)); // 121
```

ВАЖНО ЗА ES 6 МОДУЛИ

- Import и export могат да бъдат в top level
- Декларацията на import подлежи на hoisting

MAP И WEAKMAP

Двойка ключ - стойност

СЪЗДАВАНЕ НА MAP

```
const map = new Map([[1, 'one'], [2, 'two']]);

console.log(map.get(1)); // 'one'
```

МЕТОДИ И СВОЙСТВА НА MAP

- **Map.prototype.clear()**
- **Map.prototype.delete(key)**
- **Map.prototype.entries()**
- **Map.prototype.keys()**
- **Map.prototype.values()**
- **Map.prototype.get(key)**
- **Map.prototype.has(key)**
- **Map.prototype.set(key, value)**
- **Map.prototype.forEach(callbackFn[, thisArg])**
- **Map.prototype.size**

РАЗЛИКА МЕЖДУ WEAKMAP И MAP

WeakMap позволява ключовете му да бъдат garbage collected

```
const map = new Map();
const weakmap = new WeakMap();

(function(){
    const a = {x: 12};
    const b = {y: 12};

    map.set(a, 1);
    weakmap.set(b, 2);
})();
```

РАЗЛИКА МЕЖДУ OBJECT И MAP?

- Object може да приема само стрингове за ключ
- Map има size property
- Map имплементира iterable портокола

SET И WEAKSET

Структура от данни, която позволява запазване на уникални стойности

СЪЗДАВАНЕ НА SET

```
const set = new Set([1, 2, 3, 3, 2]);

console.log(set.has(1); // true
```

МЕТОДИ И СВОЙСТВА НА SET

- **Set.prototype.clear()**
- **Set.prototype.delete(value)**
- **Set.prototype.entries()**
- **Set.prototype.keys()**
- **Set.prototype.values()**
- **Set.prototype.has(value)**
- **Set.prototype.add(value)**
- **Set.prototype.forEach(callbackFn[, thisArg])**
- **Set.prototype.size**

ЗА КАКВО МОЖЕ ДА ИЗПОЗЛВАМЕ SET?

Union

```
const a = new Set([1,2,3]);
const b = new Set([4,3,2]);
const union = new Set([...a, ...b]);
```

ЗА КАКВО МОЖЕ ДА ИЗПОЗЛВАМЕ SET?

Intersection

```
const a = new Set([1,2,3]);
const b = new Set([4,3,2]);
const intersection = new Set(
    [...a].filter(x => b.has(x)));
```

ЗА КАКВО МОЖЕ ДА ИЗПОЗЛВАМЕ SET?

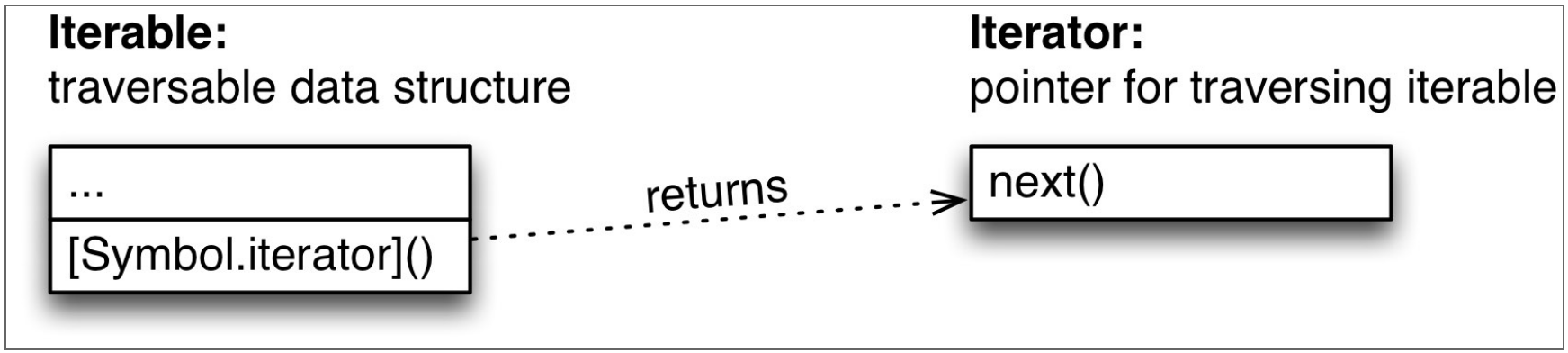
Difference

```
const a = new Set([1,2,3]);
const b = new Set([4,3,2]);
const difference = new Set(
    [...a].filter(x => !b.has(x)))
```

РАЗЛИКА МЕЖДУ WEAKSET И SET

WeakSet позволява ключовете му да бъдат garbage collected

ITERATION PORTOCOL



ITERATION

- Нов механизъм за обхождане на данни
- Iterable - структура от данни, която може да бъде обхождана
- Iterator - pointer, за обхождане на елементите в структурата от данни

ПРИМЕР ЗА ITERABLE СТОЙНОСТИ

- Arrays
- Strings
- Maps
- Sets
- DOM data structures
- arguments

ПРИМЕР ЗА КОНСТРУКЦИИ, КОИТО ИЗПОЗЛВАТ ITERATION ПОРТОКОЛА

- Destructoring
- for-of
- Array.from
- Spread operator
- Конструкторите на Map и Set
- Spred operator

ИМПЛЕМЕНТИРАНЕ НА СОБСТВЕН ITERABLE ОБЕКТ

```
const iterable = {
    [Symbol.iterator]() {
        let step = 0;
        const iterator = {
            next() {
                if (step <= 2) {
                    step++;
                }
                switch (step) {
                    case 1:
                        return { value: 'hello', done: false };
                    case 2:
                        return { value: 'world', done: false };
                    default:
                        return { value: undefined, done: true };
                }
            }
        };
        return iterator;
    }
};
```

БЕЗКРАЙНИ ITERABLES

```
const iterable = {
    [Symbol.iterator]() {
        let n = 0;
        return {
            next() {
                return { value: n++, done: false }
            }
        };
    }
};
```

ВЪПРОСИ?

VAR THANK = "YOU!";

names: "Найден Найденов и Евдокия Йорданова"
emails: **" nayden.naydenov@sap.com** & **evdokia.yordanova@sap.com "**